Deep Learning

Technologies and Applications



Edited by: Gerard Prudhomme







ARCLER PRESS

Deep Learning Technologies and Applications

Deep Learning Technologies and Applications

Edited by:

Gerard Prudhomme



www.bibliotex.com

Deep Learning Technologies and Applications

Gerard Prudhomme

www.bibliotex.com
email: info@bibliotex.com

e-book Edition 2018

ISBN: 978-1-98460-177-3 (e-book)

This book contains information obtained from highly regarded resources. Reprinted material sources are indicated. Copyright for individual articles remains with the authors as indicated and published under Creative Commons License. A Wide variety of references are listed. Reasonable efforts have been made to publish reliable data and views articulated in the chapters are those of the individual contributors, and not necessarily those of the editors or publishers. Editors or publishers are not responsible for the accuracy of the information in the published chapters or consequences of their use. The publisher assumes no responsibility for any damage or grievance to the persons or property arising out of the use of any materials, instructions, methods or thoughts in the book. The editors and the publisher have attempted to trace the copyright holders of all material reproduced in this publication and apologize to copyright holders if permission has not been obtained. If any copyright holder has not been acknowledged, please write to us so we may rectify.

Notice: Registered trademark of products or corporate names are used only for explanation and identification without intent of infringement.

© 2018 Arcler Press

In Collaboration with Arcler Press. Originally Published in printed book format by Arcler Press with ISBN 978-1-77361-274-4

DECLARATION

Some content or chapters in this book are open access copyright free published research work, which is published under Creative Commons License and are indicated with the citation. We are thankful to the publishers and authors of the content and chapters as without them this book wouldn't have been possible.

ABOUT THE EDITOR



GERARD PRUDHOMME

Gerard I. Prudhomme has a graduate degree (M.S.) for Computer Science from University College London (UCL). He has also worked as a software programmer and tech writer for different Fortune 500 companies, and studied at UCL, Harvard, and Oxford.

TABLE OF CONTENTS

List of Contributors ..... *xiii*

Preface ..... *xix*

Chapter 1 What is Deep Learning? 1

- Learning a Network 3
- Analyzing a Network 3
- New Analyzation 4
- References 10

Chapter 2 Deep Learning For Population Genetic Inference 13

- Abstract 13
- Introduction 14
- Results 20
- Discussion 34
- Methods 37
- Acknowledgments 47
- Author Contributions 48
- References 48

Chapter 3 Deep Learning Automates The Quantitative Analysis of Individual Cells In Live-Cell Imaging Experiments 57

- Abstract 57
- Introduction 58
- Results 65
- Discussion 75
- Materials And Methods 80
- Acknowledgments 82
- Author Contributions 82
- References 83

Chapter 4 Accurate de Novo Prediction of Protein Contact Map By Ultra-Deep Learning Model 91

- Abstract 91
- Introduction 93
- Results 95
- Conclusion and Discussion 122
- Method 125
- Author Contributions 132
- References 133

Chapter 5 Predicting Virtual World User Population Fluctuations With Deep Learning 139

- Abstract 139
- Introduction 140
- Methods 141
- Experimental Results 147
- Discussion and Conclusion 148
- Author Contributions 149
- References 150

Chapter 6 Recognition Of Prokaryotic and Eukaryotic Promoters Using Convolutional Deep Learning Neural Networks 157

- Abstract 157
- Introduction 158
- Materials And Methods 161
- Results And Discussion 165
- Conclusion 170
- Author Contributions 170
- References 171

Chapter 7 Intrusion Detection System Using Deep Neural Network For In-Vehicle Network Security 177

- Abstract 177
- Introduction 178
- Related Work 180
- Proposed Technique 183
- Experimental Results 190
- Conclusion 194

- Acknowledgments 195
- Author Contributions 195
- References 195

Chapter 8 Event Recognition Based On Deep Learning In Chinese Texts 201

- Abstract 201
- Introduction 202
- Materials And Methods 206
- Results 216
- Discussion 223
- Acknowledgments 224
- Author Contributions 224
- References 225

Chapter 9 Imputation For Transcription Factor Binding Predictions Based on Deep Learning 229

- Abstract 229
- Introduction 230
- Models 232
- Results 233
- Discussion 247
- Materials And Methods 248
- Acknowledgments 253
- Author Contributions 254
- References 254

Chapter 10 Deep Learning of Orthographic Representations In Baboons 259

- Abstract 259
- Introduction 260
- Results 265
- Analysis 269
- Discussion 273
- Conclusion 276
- Methods 276
- Acknowledgments 279
- Author Contributions 279
- References 279

Chapter 11 Large-Scale Transportation Network Congestion Evolution Prediction Using Deep Learning Theory 285

- Abstract 285
- Introduction 286
- Methodology 288
- Numerical Study 296
- Conclusion 303
- Author Contributions 304
- References 304

Citations 309

Index 313

LIST OF CONTRIBUTORS

Gerard I. Prudhomme M.Sc.

Sara Sheehan
Department of Computer Science, Smith College, Northampton, Massachusetts, United States of America
Computer Science Division, UC Berkeley, Berkeley, California, United States of America

Yun S. Song
Computer Science Division, UC Berkeley, Berkeley, California, United States of America
Department of Statistics, UC Berkeley, Berkeley, California, United States of America
Department of Integrative Biology, UC Berkeley, Berkeley, California, United States of America
Department of Mathematics, University of Pennsylvania, Philadelphia, Pennsylvania, United States of America
Department of Biology, University of Pennsylvania, Philadelphia, Pennsylvania, United States of America

David A. Van Valen
Department of Bioengineering, Stanford University, Stanford, California, United States of America

Takamasa Kudo
Department of Chemical and Systems Biology, Stanford University, Stanford, California, United States of America

Keara M. Lane
Department of Bioengineering, Stanford University, Stanford, California, United States of America

Derek N. Macklin
Department of Bioengineering, Stanford University, Stanford, California, United States of America

Nicolas T. Quach
Department of Bioengineering, Stanford University, Stanford, California, United States of America

Mialy M. DeFelice
Department of Bioengineering, Stanford University, Stanford, California, United States of America

Inbal Maayan
Department of Bioengineering, Stanford University, Stanford, California, United States of America

Yu Tanouchi
Department of Bioengineering, Stanford University, Stanford, California, United States of America

Euan A. Ashley
Department of Genetics, Stanford University, Stanford, California, United States of America
Department of Cardiovascular Medicine, Stanford University, Stanford, California, United States of America

Markus W. Covert
Department of Bioengineering, Stanford University, Stanford, California, United States of America
Department of Chemical and Systems Biology, Stanford University, Stanford, California, United States of America

Sheng Wang
Toyota Technological Institute at Chicago, Chicago, Illinois, United States of America

Siqi Sun
Toyota Technological Institute at Chicago, Chicago, Illinois, United States of America

Zhen Li
Toyota Technological Institute at Chicago, Chicago, Illinois, United States of America

Renyu Zhang
Toyota Technological Institute at Chicago, Chicago, Illinois, United States of America

Jinbo Xu
Toyota Technological Institute at Chicago, Chicago, Illinois, United States of America

Young Bin Kim
Interdisciplinary Program in Visual Information Processing, Korea University, Seoul, Korea

Nuri Park
Department of Computer and Radio Communications Engineering, Korea University, Seoul, Korea

Qimeng Zhang
Interdisciplinary Program in Visual Information Processing, Korea University, Seoul, Korea

Jun Gi Kim
School of Games, Hongik University, Seoul, Korea

Shin Jin Kang
School of Games, Hongik University, Seoul, Korea

Chang Hun Kim
Department of Computer and Radio Communications Engineering, Korea University, Seoul, Korea

Ramzan Kh. Umarov
King Abdullah University of Science and Technology, Thuwal, Saudi Arabia

Victor V. Solovyev
Softberry Inc., Mount Kisco, United States of America

Min-Joo Kang
The Department of Electronics Engineering, Ewha W. University, Seoul, Republic of Korea

Je-Won Kang
The Department of Electronics Engineering, Ewha W. University, Seoul, Republic of Korea

Yajun Zhang
National Natural Science Foundation of China (NSFC), Shanghai University, School of Computer Engineering and Science, Shanghai, China

Zongtian Liu
National Natural Science Foundation of China (NSFC), Shanghai University, School of Computer Engineering and Science, Shanghai, China

Wen Zhou
National Natural Science Foundation of China (NSFC), Shanghai University, School of Computer Engineering and Science, Shanghai, China

Qian Qin
Department of Bioinformatics, School of Life Sciences and Technology, Tongji University, Shanghai, China

Jianxing Feng
Department of Bioinformatics, School of Life Sciences and Technology, Tongji University, Shanghai, China

Thomas Hannagan
Laboratoire de Psychologie Cognitive, Aix-Marseille University, & CNRS (Centre National de la Recherche Scientifique), Marseille, France

Johannes C. Ziegler
Laboratoire de Psychologie Cognitive, Aix-Marseille University, & CNRS (Centre National de la Recherche Scientifique), Marseille, France

Ste´phane Dufau
Laboratoire de Psychologie Cognitive, Aix-Marseille University, & CNRS (Centre National de la Recherche Scientifique), Marseille, France

Joe¨l Fagot
Laboratoire de Psychologie Cognitive, Aix-Marseille University, & CNRS (Centre National de la Recherche Scientifique), Marseille, France

Jonathan Grainger
Laboratoire de Psychologie Cognitive, Aix-Marseille University, & CNRS (Centre National de la Recherche Scientifique), Marseille, France

Xiaolei Ma
School of Transportation Science and Engineering, Beijing Key Laboratory for Cooperative Vehicle Infrastructure, Systems, and Safety Control, Beihang University, Beijing, China
Jiangsu Province Collaborative Innovation Center of Modern Urban Traffic Technologies, SiPaiLou #2, Nanjing, China

Haiyang Yu
School of Transportation Science and Engineering, Beijing Key Laboratory for Cooperative Vehicle Infrastructure, Systems, and Safety Control, Beihang University, Beijing, China
Jiangsu Province Collaborative Innovation Center of Modern Urban Traffic Technologies, SiPaiLou #2, Nanjing, China

Yunpeng Wang
School of Transportation Science and Engineering, Beijing Key Laboratory for Cooperative Vehicle Infrastructure, Systems, and Safety Control, Beihang University, Beijing, China
Jiangsu Province Collaborative Innovation Center of Modern Urban Traffic Technologies, SiPaiLou #2, Nanjing, China

Yinhai Wang
Department of Civil and Environmental Engineering, University of Washington, Seattle, Washington, United States of America

Shusen Zhou
School of Information and Electrical Engineering, Ludong University, Yantai, Shandong, China

Qingcai Chen
Shenzhen Graduate School, Harbin Institute of Technology, Shenzhen, Guangdong, China

Xiaolong Wang
Shenzhen Graduate School, Harbin Institute of Technology, Shenzhen, Guangdong, China

Da Liu
School of Economics and Management, North China Electric Power University, Beijing, China

Ming Xu
School of Natural Resources and Environment, University of Michigan, Ann Arbor, MI, United States of America

Dongxiao Niu
School of Economics and Management, North China Electric Power University, Beijing, China

Shoukai Wang
School of Economics and Management, North China Electric Power University, Beijing, China

Sai Liang
School of Natural Resources and Environment, University of Michigan, Ann Arbor, MI, United States of America

PREFACE

Deep learning tools could very well end up with a substantially higher standard of accuracy and reliability in the recognition of physical objects, in some instances more advanced than individual overall human performance. Deep learning is an outlet of understanding, or perhaps studying, that takes advantage of a number of levels of non-linear processor jobs to discover how you can make representations of highly effective daily processes unswervingly from computer data.

The first chapter refers to deep learning. Chapter 2 shows that when provided with genomic variance computer data from a variety of people, calculating the chance of complicated populace hereditary designs can often be improbable. Chapter 3 looks at how live-cell imaging provides you with started out a thrilling range into the function cellular heterogeneity performs in vibrant, subsistence devices.

Chapter 4 looks at how protein contacts provide you with crucial information and facts for the comprehension of protein frameworks. Chapter 5 suggests a structure for foretelling updates in electronic community end user behavior. Chapter 6 looks at how precise computational recognition of promoters continues to be an issue.

Chapter 7 shows that an innovative intrusion detection system (IDS) making use of a deep neural network (DNN) is offered to improve the safety of in-vehicular system. Chapter 8 looks at how event identification is easily the most basic and also crucial job in event-based all-natural vocabulary processing devices. Chapter 9 looks at getting a grasp on the cell-specific merging designs of transcription factors.

Chapter 10 looks at what exactly is the source of our capability to understand orthographic information. Chapter 11 displays comprehending the simplest way to understand blockage at one area.

Editor

CHAPTER 1

WHAT IS DEEP LEARNING?

Gerard I. Prudhomme M.Sc.

Deep learning should be considered to apply more than a few difficulties. Not to mention on the architectural structures of systems of neurons manufactured using a great number of beds. The training programs of the Deep learning devices are successfully done on a massive predefined set of computer data that has already been categorized.

The, for the most part, most adequate versions of Deep Learning apply to devices such as a sophisticated driving vehicle assistance program, an Advanced Driver Assistance Systems (ADAS). The correctness of the deep learning strategy would depend on the huge amount of computer data normally used to work with the machine. Deep learning may be utilized to aid with the verification of spoken communication, illustrations or photos, and the identification of holistic written communication.

These kinds of a neural system convolutes the characteristics identified with the contribution computer data. Convolutionary neural networks, also called a CNN, are the most widely used of the Deep Learning architectures. The neural systems are properly organized in levels composed of a collection of interrelated nodes.

And, as a consequence, it makes use of 2D convolutional levels. This architectural structure is, for this reason, flawlessly custom-made for the handling of 2D computer data, along the lines of illustrations or photos.

The first of all filtration systems are often very simple and easy features, along the lines of illumination as well as borders. And, they, as a consequence, shift to a lot more highly developed forms and sizes due to the fact the levels develop. Filtration systems are carried out on each and every illustration dedicated towards the end result.

These are done at completely different resolutions, together with the production of each and every convolved graphic is commonly used in the form of contribution for the subsequent level. A computer data handling progression originating from a elaborate convolution neural system for undertaking a picture verification course of action is a regular utilization of deep learning.

As an alternative for generating a convolutional neural system on your own, it most certainly is frequently used to make use of a pre-entrained product to completely on its own acquire capabilities from a brand new dataset. This particular technique, sometimes known as transfer learning, is a reasonable answer for making use of Deep Learning without the need of an enormously huge computer data collection or perhaps lengthy instruction. The working out of a convolutional neural network, or CNN, ought to have an enormously huge level of computer data because doing so would be smart as it allows you to gain knowledge of an incredible number of loads, for a conventional graphic verification situation.

GPU speed to a great extent helps to accomplish coaching. Coaching a Deep Learning product usually takes a considerable amount of time, from four or five days to three or four weeks. GPUs do away with the period of time anticipated to teach a system as well as, as an element of a picture verification dilemma, is effective in reducing the coaching period from a couple of days to a couple of hours.

As a matter of fact, deep learning provides you with collected positive results in the past years. With the inclusion of being able to do better than human beings at the recreational game of “go,” deep learning has advanced. That is, regardless of whether the greatest participant on the planet, in accordance with present ratings, and in spite of everything, would seem in a position to do better than this synthetic intelligence system.

Deep learning and also neural systems are, at the moment, in style in the area of automated understanding. NVIDIA, Microsoft, and Google provide you with libraries, to assist in their utilization. The awareness of the "go" game has its own advanced features, despite the presence of not at all hard guidelines. You will discover a few advances related to areas of go, against a few problems, an infinitely smaller amount for the time being.

LEARNING A NETWORK

The time when the very first trend of great interest toward deep learning from an important part of the educational universe emerged was in the nineties. Their investigation previously had demonstrated the emergence of numerous local minima of total error. The knowledge of a neural network is completed by the machine by describing the weighting of the ideas of each and every neuron.

Developing these types of loads a small amount will surely have an excellent effect on the forecasting of the system. At the same time, in theory, absolutely nothing could possibly give good reason for the achievements of neural networks. Remember, neural networks happen to be the most important piece of equipment behind deep learning.

The occurrence of these kinds of local minima would mean that, as soon as the performance of the technique is finalized, the weighting is possibly not perfect. By altering a number of rates, it might be likely that you can significantly decrease the overall mistakes. To make a choice of this weighting, every single one of the calculations work on the system on computer data in which the end product is well known.

To provide an example, you may have an audio along with the words and phrases to which it harmonizes. The significant difference matches the malfunctions prepared by the system. The purpose of the educational calculations would be to achieve the, in general, bare minimum number of malfunctions.

ANALYZING A NETWORK

This attribute threatens the overall performance of systems, considering the fact that the nearby bare minimum, after understanding the concepts,

are often not even close to being at the level of the universal bare minimum. At this point in time, theoretical technical analysis of neural networks appears to have been put in place on systems of a number of neurons. Each one of these regional minima are so therefore contained in a great number which makes them quite far-away from each other.

This behavior patterns harmonizes in physics compared to that of spin glasses. Spin glasses are metallic alloys with a selection of magnetic contaminants randomly put into the alloy. The strength of the substance is dependent upon the arrangement of the contaminants. That highlights a great number of regional minima which have been not even close to the universal minimum amount.

NEW ANALYZATION

The more substantial the system, the more these types of factors are collected around the universal minimum amount. The sole theoretical outcome on the market, up until not too long ago, was that a number of neural systems are equivalent specifically to the spin glasses. Without a doubt, deep learning indicates you need to apply an incredible multitude of these types of neurons, a number of tens of millions.

The consequence of a development is, due to this fact, by no means not even close to the universal minimum amount. At the same time, the consequence acquired by the staff supervisor of the artificial intelligence lab of Facebook demonstrates, on the other hand, that for an incredible multitude of neurons, the malfunction functionality provides you with a kind of a Funnel: nearby minima are incredibly in close proximity to the universal minimum amount.

It happens to be, for this reason, plain and simple to break free from it, following the downhill path, when it comes to a miscalculation. Stated more explicitly, studying algorithms unify at crucial areas. Research workers have demonstrated that almost all these types of crucial areas are in reality saddle stitches rather than minima.

They correspond to a smooth region, with climbing and as a consequence sinking orientations. On the whole, real minima are equivalent to cuvettes, orientations which only boost the miscalculations. On the other hand, they are rather uncommon, which makes them in close proximity to the best achievable amount.

Each one of these components come across their local arrangement simpler and easier, with a bare minimum of power. In physical terms, neural systems, for this reason, then can no longer correspond to “spin funnels”, with an increase of sympathetic varieties. The power of the arrangement will fluctuate without any warning, without really supplying an area minimum amount.

At the same time, contemporary systems might end up being a lot more complicated when compared with those analyzed, to stay away from over-learning. That is they match up education computer data at the same time nicely. On the other hand they may have a problem familiarizing themselves with computer data then that is simply not an important part of it.

These types of outcomes make certain that secrets and techniques along the lines of stochastic gradient descent (SGD) could help you. As the miscalculation performance of a neural system is nearly convex. Having said that, the theoretical chemical composition, along with the physics, of the matter, by this time, suggest a compartment of arithmetical equipment to provide an understanding of the framework associated with these systems.

The above-mentioned details are, for all intents and purposes, good sized systems of synthetic neurons, explained in a clear fashion in greater detail below, that have to do with substantial amounts of samples of a job to be performed. Deep learning is a side branch of machine learning, on its own a side branch of artificial intelligence. These types of programs gain knowledge of on their own how you can complete a task.

The above-mentioned details are, more often than not, probabilistic forecasts. Work processes in many cases are predictive responsibilities in accordance with abundant sensory computer data. As an illustration you may have a program figuring out everytime that item or perhaps individual is on a picture or even exactly what term is mentioned in an sound element.

On the other hand the course of action is often rather universal which enables you to be just as assist in some other areas. As an illustration you may have a program foretelling whether or not a consumer who may have a particular user profile check will out a particular advertising

campaign. You may also have a program foretelling whether or not a consumer who may have a particular user profile will most likely enjoy a particular motion picture.

Additionally, you may have a program foretelling whether or not a consumer who may have a particular user profile will most likely decide to buy an item. Or, alternatively, you may have a program foretelling the subsequent term in a region of textual content. Most of these challenges are comprised of awfully complicated nonlinear forecast services.

It happens to be similar to an infant baby who slowly gets to know how to recognize, as well as have an understanding, of the universe. He does this by opening up his eyes, as well as ears. Not to mention that he is also subjected to a gargantuan sensory stream.

And, additionally, we discover how you can utilize guidelines utilizing massive quantities of information and data. The deep element in deep learning is the fact that we design these types of remarkably complicated features by the arrangement of several levels. Therefore, there is certainly the deepness of quite simple parametric nonlinear services.

Artificial intelligence studies have, during days gone by, been based around a couple of completely different opening beliefs. One methodology, in the beginning, wanted to recreate the natural rational organizational behavior capability. That was referred to as symbolic and traditional artificial intelligence.

The researchers persuaded themselves that that the whole thing would certainly drift very easily from that point. The supplementary methodology desired, as an alternative, to get motivation from what happened to have been recognized with regard to low-level working of the human brain. These were imitating neural systems with connective and sub-symbolic artificial intelligence. From the very beginning, this particular technique payed attention to adjustment as well as an unbiased gaining of knowledge from sensory illustrations.

Within machine learning, deep learning could very well be set apart from shallower methods that endeavor to understand merely just one functioning level. Machine learning, as well as deep learning, are the heirs associated with the secondary prevailing attitude in artificial intelligence. On top of that it, by natural means, focused on its own in the

direction of a probabilistic management of difficulties.

A large amount of improvements in irregular methods came up with, and resulted with, eclipsing deep-seated neural system investigation for a ten year period. That was before their gradual rejuvenation started out in 2006. That, more importantly, resulted in their present booming commercial achievements.

Except that these may provide different types of proper neurons, made easier than ever to a tremendous degree when it comes to the advanced features of organic cellular material. Deep synthetic neural systems are encouraged by organic neural systems not to mention their lined association in certain parts of the human brain, which include the optical process. However, this is specifically created.

The work that is designed to get an indication of the prophecy. This difficult to understand performance is achieved by the arrangement of a number of features in accordance with the levels of the system. Mathematically, these types of systems have the capacity to get an indication of a problematic nonlinear performance of an contribution vector.

If you would like to reduce the system forecast miscalculation on almost all the coaching illustrations then you would utilize this method. The guidelines coming from all the levels are fully optimized simultaneously, by a simple yet effective slope lineage method. A standard level calculates a linear change of its very own contribution vector, and after that a fundamental nonlinearity, a hyperbolic tangent.

Architectural design particulars: amount, dimensions, level categories, and so on and so forth, as well as the methods, as well as secrets and techniques, normally used in further optimize the guidelines, should always be very carefully modified to the character of the trouble as well as the significant amount of computer data on the market.

This is done to generate more and more massive as well as complicated systems on more and more substantial quantities of information and data. Another essential expenditure is an inverse risk in downscaling. An up-to-date phenomenon in deep learning is to almost always focus on progressively more crucial issues, as well as, you will discover, intrinsic problems.

The understanding of the concepts of versions capable of producing abundant sensory information and facts are, in spite of everything, not even close to attaining the high quality of the identification designs. Having the ability to be informed about programs of really good high quality regardless of whether there is certainly just a couple samples of a job. As an illustration you may have a program procreation of credible illustrations or photos with regards to graphic identification.

Personally I think there is, certainly in spite of everything, a considerable amount of development at this time there. The sources of revelations in the shape of neuronal structural design of sophisticated organizational behavior practical knowledge are advancing. This is done in accordance with an abundant multi-sensory depiction to facilitate understanding.

This is done for each and every principle, and continues to be an accessible difficulty. Last, but not least, there is the fact that most recent accomplishments are stemming from important improvements in the methods of working out these types of deep networks. Therefore, the problems of their optimization continues to be not even close to becoming a resolved issue.

Therefore, gaining knowledge of of more advantageous graphical stylizations of percepts, that happen to be a lot more significant. In terms of how much the very important element is involved, I will be for the most part enthusiastic about mastering the ways to learn very important concepts that make it possible for self-supporting work. At the present moment I, at the same time, want to make an effort to have enough knowledge to safely and effectively educate designs.

I would like to do that with an extremely sheer number of aesthetic classes. As an illustration you may have a program working with vocabularies of a trillion different phrases. With regards to the large areas of practical application, I will be especially excited about the idea of redesigning as well as identification of face gestures.

For the most part this is associated with the emotional baggage they communicate. We need to do that with the purpose of improving upon our knowledge of the organizational abilities as well as with normal daily functioning of the organic human brain, an organic neural system. Also, naturally occurring verbal communication modeling and taking

advantage of deep learning, a computer-based neural system, to improve neuro-imaging computer data technical analysis secrets and techniques.

Technological innovation and development, in the event that it is utilized as it should be and benefited from, in many cases can make improvements to the day-to-day lives of everyone. On the other hand, due to the fact that pretty much every significant technological innovation advancement of humankind, from the time of the creation of flint as well as the management of flames, a pair of additional negative outcomes should be expected as well.

These are the ones that you will have to give consideration to. On the one hand, the enticement to make use of these new styles of methods is enticing, but bear in mind the upheavals partnered with the modification of financial paradigm which they will lead to. This is most definitely the way things work with AI. Existential things to consider which often come from implausible unrealistic science fiction do not have to draw attention away from us from reasonable, as well as warranted, medium-term issues.

I fantasize about the most cherished moment that will arise in the event that human beings will demonstrate a sufficient amount of brain power to refrain from eliminating one another. I also fantasize, as well, of a day, where, alongside one another, we will be able to mobilize and make sure that the generation, and also selling, of arms of conflict as a final point stops to be a financially rewarding human activity. In both instances equally, the solutions and information will never be technological. on the other hand they will be related to world economics, politics, as well as interpersonal relations. Is it possible that human beings really have the need for weaponry enhanced by technology?

Now, I am not scared of an AI turning out to be self-supporting to the stage of having to be in charge, as well as making a choice on its very own to do away with human beings. That is usually depicted by off the wall science fiction. On the other hand, a large number of AI research workers have have currently signed up for an application to not approve AI armaments.

They are aiming to defend against a brand new challenge when dealing with these weapons right now. The simple truth is that for a long period of time people in general, or perhaps human institutions, are going

to always be driving the series from above. Nevertheless, it really would be smart to put a brake on their warlike capability.

That being said, we should always make it a habit to lessen this change. In a similar way that a great number of wagon drivers appear to have been compelled to come across a brand new career following the appearance of the steam engine, very much the same mutations could very well be anticipated with improvements in AI. In terms of how much the world economic bench mark change is related, personally I think we are now at the start of a brand new industrial revolution, the reason for that could very well be AI.

How will you enforce, deliver, invest again, as well as reallocate, the prosperity manufactured by overall performance benefits associated with mechanization? If you read further, just as before, it will likely be with political affiliations, global financial as well as communal desires and demands, that we make decisions concerning contemporary society. The major difference between these two types of circumstances would be determined by information about how you will make a choice.

Because of the fact that ever increasing numbers of jobs are going to be carried out by self-supporting programs, to what sort of modern society do we need to head out to? Will we have a utopian modern society wherein almost all humanity might have the assurance of a comparatively relaxing functional life and much more time to dedicate their selves to the things which thoroughly appeal to all of them? Or perhaps it will be a dystopian modern culture which is where a couple of elites are now living in richness, despite the fact that the majority of people are now living in precariousness whenever they are substantially less effective when compared with a automatic robot?

REFERENCES

1. Emergence of simple-cell receptive field properties by learning a sparse code for natural images, Bruno Olhausen, Nature 1996.
2. ImageNet Classification with Deep Convolutional Neural Networks, Alex Krizhevsky, Ilya Sutskever, Geoffrey E Hinton, NIPS 2012.
3. "Improving neural networks by preventing co-adaptation of feature detectors." Hinton, Geoffrey E., et al. arXiv preprint arXiv:1207.0580 (2012).

4. Joint Learning of Words and Meaning Representations for Open-Text Semantic Parsing, Antoine Bordes, Xavier Glorot, Jason Weston and Yoshua Bengio (2012), in: Proceedings of the 15th International Conference on Artificial Intelligence and Statistics (AISTATS)
5. Learning Hierarchical Features for Scene Labeling, Clement Farabet, Camille Couprie, Laurent Najman and Yann LeCun, IEEE Transactions on Pattern Analysis and Machine Intelligence, 2013.
6. "Proceedings of the Twelfth International Conference on Artificial Intelligence and Statistics (AISTATS 2009)," in Proceedings of the Twelfth International Conference on Artificial Intelligence and Statistics (AISTATS 2009), 2009.
7. The Manifold Tangent Classifier, Salah Rifai, Yann Dauphin, Pascal Vincent, Yoshua Bengio and Xavier Muller, in: NIPS'2011.

CHAPTER 2

Deep Learning for Population Genetic Inference

Sara Sheehan[1,2], Yun S. Song[2,3,4,5,6]

[1] Department of Computer Science, Smith College, Northampton, Massachusetts, United States of America

[2] Computer Science Division, UC Berkeley, Berkeley, California, United States of America

[3] Department of Statistics, UC Berkeley, Berkeley, California, United States of America

[4] Department of Integrative Biology, UC Berkeley, Berkeley, California, United States of America

[5] Department of Mathematics, University of Pennsylvania, Philadelphia, Pennsylvania, United States of America

[6] Department of Biology, University of Pennsylvania, Philadelphia, Pennsylvania, United States of America

ABSTRACT

Given genomic variation data from multiple individuals, computing the likelihood of complex population genetic models is often infeasible.

To circumvent this problem, we introduce a novel likelihood-free inference framework by applying deep learning, a powerful modern technique in machine learning. Deep learning makes use of multilayer neural networks to learn a feature-based function from the input (e.g., hundreds of correlated summary statistics of data) to the output (e.g., population genetic parameters of interest). We demonstrate that deep learning can be effectively employed for population genetic inference and learning informative features of data. As a concrete application, we focus on the challenging problem of jointly inferring natural selection and demography (in the form of a population size change history). Our method is able to separate the global nature of demography from the local nature of selection, without sequential steps for these two factors. Studying demography and selection jointly is motivated by *Drosophila*, where pervasive selection confounds demographic analysis. We apply our method to 197 African *Drosophila melanogaster* genomes from Zambia to infer both their overall demography, and regions of their genome under selection. We find many regions of the genome that have experienced hard sweeps, and fewer under selection on standing variation (soft sweep) or balancing selection. Interestingly, we find that soft sweeps and balancing selection occur more frequently closer to the centromere of each chromosome. In addition, our demographic inference suggests that previously estimated bottlenecks for African *Drosophila melanogaster* are too extreme.

INTRODUCTION

With the advent of large-scale whole-genome variation data, population geneticists are currently interested in considering increasingly more complex models. However, statistical inference in this setting is a challenging task, as computing the likelihood of a complex population genetic model is a difficult problem both theoretically and computationally.

In this paper, we introduce a novel likelihood-free inference framework for population genomics by applying *deep learning*, which is an active area of machine learning research. To our knowledge, deep learning has not been employed in population genomics before. A recent survey article [1] provides an accessible introduction to deep learning, and we provide a high-level description below. Our general goal in this paper is to demonstrate the potential of deep learning as a powerful

framework for population genetic analysis that can allow accurate inference of previously intractable models.

As a concrete example, we consider models of non-equilibrium demography and natural selection, for which multi-locus full-likelihood computation is prohibitive, and apply deep learning to the challenging problem of jointly inferring demographic history and selection (see [2] for a recent review of this topic). One reason why this problem is difficult is that demography (for example, a bottleneck in population size) and selection can leave similar signals in the genome. Untangling the two factors directly has rarely been attempted; most methods that estimate selection try to demonstrate robustness to demographic scenarios, rather than estimating demographic parameters jointly with selection. The demographic models considered in this paper are restricted to a single population with time-varying effective population size, but the overall framework presented here can be applied to more general demographic models. We anticipate many algorithmic, modeling, and application directions that could emerge from this proof of concept.

Our focus on the joint inference of demography and selection is motivated by *Drosophila*, where previous demographic estimates [3] may have been confounded by pervasive selection. The reverse has occurred as well, with selection estimates being confounded by demography [4]. See [5] for a more thorough discussion of the role selection plays in the *Drosophila* genome. To test our method, we simulate data under a variety of realistic demographies for an African population of *Drosophila melanogaster*. For each demographic history, we simulate many regions under different selection parameters. We then apply our tailored deep learning method using hundreds of potentially informative summary statistics. From the output of this trained network, we demonstrate that parameters can be accurately inferred for test datasets, and interpret which statistics are making the biggest contributions. After training our deep network, we also apply it to analyze 197 *Drosophila melanogaster* genomes from Zambia, Africa [6] to learn about their effective population size change history and selective landscape.

A summary of related works

Several machine learning methods have been developed for selection only, often focusing on classifying the genome into neutral versus selected

regions. Examples include methods based on support vector machines (SVMs) [7–9] or boosting [10–12]. Often these methods demonstrate robustness to different demographic scenarios, but do not explicitly infer demography.

Many methods have been developed to infer ancestral population size changes, including PSMC [13], diCal [14, 15], and MSMC [16]. These methods make the underlying assumption that selection does not significantly bias the results. This is perhaps somewhat true for humans, but is not a valid assumption for many species such as *Drosophila*, for which selection seems ubiquitous throughout the genome.

Few previous works have addressed both population size changes and selection. Galtier *et al.*[17] developed a likelihood-based method for distinguishing between a bottleneck and selection. They applied their method to *Drosophila* data to conclude that a series of selective sweeps was more likely than a bottleneck, but did not explicitly infer parameters for both selection and demography. In their method, Galtier *et al.* assumed that demographic events affect the entire genome, whereas selection is a local phenomenon. In contrast, Gossmann *et al.* [18] estimated the effective population size locally along the genome, and reported that it is correlated with the density of selected sites. To make our results as easily interpretable as evolutionary events as possible, we estimate global effective population size changes.

Approximate Bayesian Computation (ABC) [19, 20] is a likelihood-free inference method based on simulating datasets and comparing their summary statistics. (A more detailed description of the framework is provided below.) This approach has been used to study various complex population genetic models (e.g., [21–23]) for which likelihood computation is prohibitive. Partly due to several influential theoretical works [24–31], the popularity of ABC has grown rapidly over the past decade. ABC's main advantages are that it is easy to use and is able to output a posterior distribution. There are a few challenges, however: 1) ABC suffers from the "curse of dimensionality," with decreasing accuracy and stability as the number of summary statistics grows [32]. 2) ABC typically uses a rejection algorithm, so the simulated datasets are not used optimally, and often a very large number are required. 3) It is difficult to interpret the output of ABC in terms of which statistics were the most informative for each parameter. 4) Although ABC can be used

very effectively for continuous parameter inference, it would be difficult to use ABC for a complex scenario that involved joint inference of continuous parameters and categorical distributions. To our knowledge, ABC has thus not been previously applied to the problem of jointly inferring demography (using continuous effective population sizes) and selection (using discrete classification). The work of Bazin *et al.* [33] estimates demographic and selection parameters using a hierarchical ABC approach, though not in an explicit joint framework.

We see our deep learning method as a complimentary approach to, rather than a direct replacement of, other likelihood-free inference methods such as ABC or SVM. Deep learning has several advantages such as making full use of datasets, elegantly handling correlations between summary statistics, and producing interpretable features of the data. It also produces an "inverse" function from the input summary statistics to the population genetic parameters of interest. However, in contrast to ABC, it does not provide a posterior distribution.

A brief introduction to deep learning

Deep learning has its beginnings in neural networks, which were originally inspired by the way neurons are connected in the brain [34]. Neural networks have been studied for over 60 years and a huge body of literature exists on the topic. A neural network is typically used to learn a complex function between inputs (data) and outputs (*response variables*) in the absence of a model, so it can be thought of as a generalized regression framework. While standard regression and classification methods involve fitting linear combinations of *fixed* basis functions, a neural network tries to *learn* basis functions (usually non-linear) appropriate for the data. A neural network architecture consists of multiple layers of *computational units* (nodes), with connections between the layers but not between nodes within a layer. Within a layer, each node computes a transformation (usually non-linear) of the outputs from the previous layer. The phrase "deep learning" refers to algorithms for learning *deep* neural network architectures with many hidden layers.

The *universal approximation theorem* [35, 36] states that any continuous function on compact subsets of $\mathbb{R}^n$ can be uniformly approximated by a feed-forward neural network with a single hidden layer, provided that the number of nodes in the hidden layer is sufficiently large

and the transformation (called the activation function) associated with each node satisfies some mild conditions. However, it can be challenging to learn the weights of such a network and to interpret the hidden layer. So as learning problems became more complex, it was desirable to train networks with more hidden layers. Since their introduction over 30 years ago, deep architectures have proved adept at modeling multiple levels of abstraction. However, they were notoriously difficult to train since their objective functions are non-convex and highly non-linear, and the level of non-linearity increases with the number of layers in the network.

A major breakthrough was made in 2006 when Hinton and Salakhutdinov [37] showed that a deep feed-forward neural network can be trained effectively by first performing unsupervised "pretraining" one layer at a time, followed by supervised fine-tuning using a gradient-descent algorithm called *back-propagation* [38]. (Simply put, pretraining provides a good initialization point for non-convex optimization.) They applied their learning algorithm to dimensionality reduction of images and achieved substantially better results than PCA-based methods.

Following the work of Hinton and Salakhutdinov, deep learning has been applied to various challenging problems in computer science over the last several years, making groundbreaking progress. Deep learning broke long-standing records for accuracy that had been set by approaches based on hand-coded rules. Well-known examples include automatic speech recognition (transforming spoken words into typed text) [39, 40] and computer vision (automatically classifying images into different categories and tagging objects/individuals in photos) [41].

Many variations have been developed, including *dropout*, which attempts to learn better and more robust features of the data (see [42, 43]). Deep learning has also been applied to problems in neuroscience [44] and computational biology [45–48], but has not been used for population genetics before. We demonstrate here that deep learning can accommodate more complex models than existing techniques, and can provide a complimentary approach to existing likelihood-free inference in population genetics. For example, deep learning could be used to select optimal statistics for ABC (as demonstrated in [49]), or another method.

ABC background

Rejection-based methods have been used since the late 1990's (see [19, 50]) to estimate population genetic parameters when the likelihood is difficult to compute. Early improvements to ABC quickly helped make it a popular method for a variety of scenarios (see [20] for a good introduction). ABC works by simulating many datasets under a prior for the parameters of interest. Then these datasets are reduced to a vector of summary statistics that are ideally informative for the parameters. The summary statistics that are closest to the summary statistics for the target dataset are retained, and the corresponding parameters used to estimate the desired posterior distributions. ABC has many advantages, including ease of use and the output of a posterior distribution. However, there are a few shortcomings, which various works have sought to address.

In its simplest form, ABC does not naturally handle correlated or weakly informative summary statistics, which can add noise to the data. To tackle this problem, many methods for dimensionality reduction or wisely selecting summary statistics have been proposed (see [26, 29, 31, 51, 52], and [53] for a dimensionality reduction comparison). However, simple reductions cannot always learn subtle relationships between the data and the parameters. Expert pruning of statistics helps some methods, but given the lack of sufficient statistics, valuable information can be eliminated. This reduction is similar in spirit to the work presented here, although there are many algorithmic and application differences.

A second issue with ABC is the rejection step, which does not make optimal use of the datasets that are not retained. The more statistics and parameters used, the more datasets must be simulated and rejected to properly explore the space, making the interaction between these two issues problematic. To address the problem of rejecting datasets, different weighting approaches have been proposed (see [28] for a good example of how the estimation error changes as fewer datasets are rejected). The idea is to keep more datasets, but then weight each retained dataset by its distance to the target dataset. However, few approaches utilize all the datasets in this way, and the most popular implementation of ABC (*ABCtoolbox*, [54]) typically still rejects most of the simulated datasets by default.

One final issue with ABC is the black-box nature of the output. Given the distances between the simulated datasets and the target dataset, and the posterior, there is no clear way to tell which statistics were the most informative.

RESULTS

We first describe the simulated dataset we generated to investigate joint inference of demographic history and selection. In the Methods section, we describe how to transform each region (as well as each region of the real data) into a set of 345 summary statistics, and also detail our deep learning algorithm. In what follows, we present the results of our method in a variety of different contexts.

Simulating data

To create a simulated dataset that is appropriate for our scenario of interest, we first define the response variables we would like to estimate. For simplicity, we consider piecewise-constant effective population size histories with three epochs: a recent effective population size N_1, a bottleneck size N_2, and an ancient size N_3. Further, we define a genomic region as belonging to 4 different selection classes: no selection (neutral), positive directional selection (hard sweep), selection on standing variation (soft sweep), and balancing selection. See [55] for a more complete analysis of the different types of selection in *Drosophila*. To accurately reflect that demography affects the entire genome while selection affects a particular region, we simulate many genomic regions under the same demographic history, but the selection class for each region is chosen independently. See Fig 1 for a simplified illustration of the data.

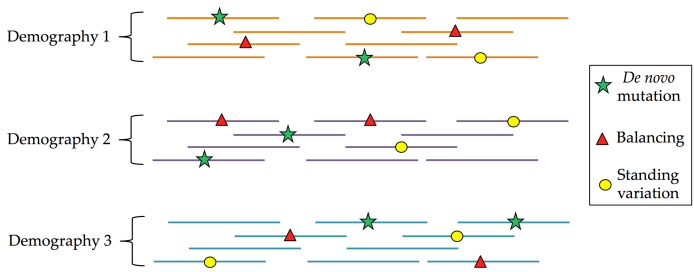


Figure 1: Input data for the demography and selection scenario.

For each demographic history (bottleneck), we simulated many different genomic regions. Each region can either have no selection (neutral), one site with an *de novo*mutation under positive selection (hard sweep), one site under balancing selection, or one standing variant under positive selection (soft sweep).

To simulate data, we used the program msms [56]. To make our simulated data as close to the real data as possible, we simulated $n = 100$ haplotypes, and correspondingly downsampled the Zambia *Drosophila melanogaster* dataset to match. We repeated the following procedure 2500 times. First we selected three population sizes for the demographic model, then simulated 160 regions with these sizes, 40 for each selection scenario. Each region was 100 kb, with the selected site (if present) occurring randomly in the middle 20 kb of the region. We used a baseline effective population size $N_{\text{ref}} = 100{,}000$, a per-base, per-generation mutation rate $\mu = 8.4 \times 10^{-9}$ [57], and a per-base, per-generation recombination rate r equal to μ, as inferred by PSMC [13]. This ratio of recombination to mutation rate is somewhat lower than what has been inferred in other works (see [58, 59]), so we also simulated a testing dataset with a higher recombination rate, described below. However, it is likely that there is significant variation in this ratio across the genome [60]. We used a generation time of 10 generations per year. Based on the population size change results of PSMC, we chose the time change-points for demographic history to be $t_1 = 0.5$ and $t_2 = 5$ in coalescent units. Scaled effective population size parameters $\lambda_i := N_i/N_{\text{ref}}$ and their prior distributions are below:

- Recent effective population size scaling factor: $\lambda_1 \sim \text{Unif}(3, 14)$.
- Bottleneck effective population size scaling factor: $\lambda_2 \sim \text{Unif}(0.5, 6)$.
- Ancient effective population size scaling factor: $\lambda_3 \sim \text{Unif}(2, 10)$.

For the selection classes, the different types are shown below:

- **Class 0, neutral**: no selection, neutral region.
- **Class 1, hard sweep**: positive selection on a *de novo* mutation (i.e., hard sweep). For the selection coefficient, we used $s \in \{0.01, 0.02, 0.05, 0.1\}$, with 10 regions for each value of s.

The onset of selection was chosen to be 0.005 in coalescent units, which provided sweeps at a variety of different stages of completion at the present time. We discarded datasets where the frequency of the selected allele was 0 (early loss of the beneficial mutation due to drift), but for a large fraction of datasets, the beneficial allele had not yet fixed.

- **Class 2, soft sweep**: positive selection on standing variation (i.e., soft sweep). The selection coefficients and selection start time were chosen as in the hard sweep scenario, but now the initial frequency of the beneficial allele was chosen to be 0.001.
- **Class 3, balancing**: heterozygote advantage, balancing selection. The selection start time and selection coefficients were chosen in the same fashion as Class 1.

Using this strategy, we simulated 2500 different demographic histories, with 160 regions for each one, for a total of 400,000 datasets. To build 160 regions for each demography, we simulated 40 datasets for each of the selection classes above.

Results for demography and selection on simulated data

Of the 400,000 total datasets, we used 75% for training and left out 25% for testing. In Table 1, we show the effective population size results for a network with 3 hidden layers respectively with 25, 25, and 10 nodes (5 layers total including input and output). The best results were found when we averaged the statistics for all of the datasets (160 "regions") with the same demography, then ran this average through the trained network.

Table 1: Deep learning results for population sizes under the scenario with demography and selection, using a network with 3 hidden layers of sizes 25, 25, and 10

Deep learning predictions	N_1 error	N_2 error	N_3 error
Average stat prediction	0.051	0.074	0.487
Final prediction	0.098	0.077	0.569
Neutral regions prediction	0.072	0.083	0.566

doi:10.1371/journal.pcbi.1004845.t001

We evaluate the results in three ways, using the relative error of the estimates: $|N_{est} - N_{true}|/N_{true}$. First, for each test demography, we average the statistics for each dataset, and then run these values through the training network. Second, for each test demography, we run the datasets through the network one by one, then average the predictions. Finally, we perform the second procedure, but with only the regions we classified as neutral. We note that the most ancient size (N_3) is always the least accurately estimated.

Table 2 shows the results of our three different size prediction methods (see Methods) for an example demography. For each parameter, we include a range between the 2.5th and 97.5th quantiles. To further investigate uncertainty in our estimates, we also include a violin plot of the three population sizes, shown in Fig 2A. In this example (and more generally), our point estimate of the most ancient size N_3 is the least accurate, but the most recent size N_1 has the largest uncertainty. When considering a single dataset, there is not always a clear winner among our three prediction methods. Overall, the average statistic method is usually the most accurate, followed by the neutral regions method.

Table 2: Results for an example demography, with the true population sizes shown in the first row

	N_1	N_2	N_3
True sizes	878,675	235,199	713,001
Average stat prediction	872,541	264,817	600,650
Final prediction	840,784	243,180	622,359
Quantiles (2.5th, 97.5th)	(659231, 954631)	(209797, 279496)	(597402, 649887)
Standard deviation	74,477	37,238	54,165
Neutral regions prediction	873,289	237,686	619,943

doi:10.1371/journal.pcbi.1004845.t002

The rows below each dotted line are the three different size predictions for this dataset. *Average stat*: average the statistics for each region within the same demography, then run these averages through the trained network. *Final*: run the statistics for each region through the trained network, then average the results. *Neutral regions*: only average the results for the regions we predict as neutral.

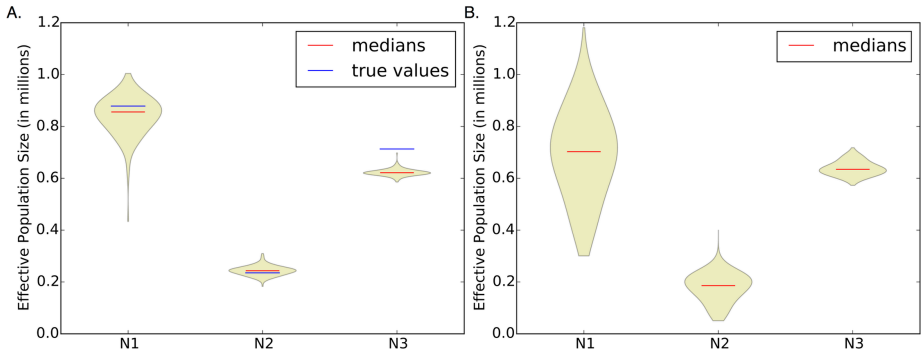


Figure 2: Violin plots of population size estimates for (A) an example simulated dataset and (B) the real dataset.

In both cases, we have the largest uncertainty for the most recent population size. For the simulated dataset, the true population sizes are shown in blue, and we note that our point estimate of the most ancient population size is the least accurate.

To analyze the selection results, we calculated a confusion matrix in Table 3 to show which datasets of each class were classified correctly, or misclassified as belonging to a different class. The rows of a confusion matrix represent the true class of each dataset, and the columns represent the predicted or called class. An ideal confusion matrix would have all 1's (100%) on the diagonal, and 0's off the diagonal. Our most frequent error is classifying hard sweep datasets as neutral (row 2, column 1 of the confusion matrix). We hypothesized that this was either because selection occurred anciently and quickly, or because selection occurred recently and the sweep was not yet complete. To test this, we examined the results conditional on the frequency of the selected allele at present. The results shown in Table 4 suggest that in fact many of the sweeps were not complete, which is why the regions sometimes appeared neutral. Regardless, this type of false negative error is arguably preferable to falsely calling many regions to be under selection when they are in fact neutral.

Table 3: Confusion matrix for the selection predictions, in the demography and selection scenario

	Called Class			
True Class	**Neutral**	**Hard Sweep**	**Soft Sweep**	**Balancing**
Neutral	**0.9995**	0.0002	0.0003	0.0000
Hard Sweep	0.1434	**0.8333**	0.0032	0.0201
Soft Sweep	0.0096	0.0010	**0.9891**	0.0003
Balancing	0.0301	0.0356	0.0056	**0.9287**

doi:10.1371/journal.pcbi.1004845.t003

Each row represents the datasets that truly belong to each selection class. Each column represents the datasets that were actually classified as each selection class. Ideally we would like all 1's down the diagonal, and 0's in the off-diagonal entries. The largest number in each row is shown in boldface. We can see that neutral datasets are the easiest to classify, and sometimes regions under selection (hard sweeps in particular) look neutral as well (first column). The overall percentage of misclassified datasets was 6.2%.

Table 4: Hard sweep results, broken down by the present-day frequency (f) of the selected allele (which we would not know for a real dataset)

		Called Class			
f		**Neutral**	**Hard Sweep**	**Soft Sweep**	**Balancing**
Low	(18.4% of datasets)	**0.6820**	0.3137	0.0009	0.0035
Moderate	(12.5% of datasets)	0.0988	**0.8220**	0.0000	0.0793
High	(69.1% of datasets)	0.0084	**0.9734**	0.0044	0.0138

doi:10.1371/journal.pcbi.1004845.t004

We defined "low" frequency as $f \in [0, 0.3)$, "moderate" frequency as $f \in [0.3, 0.7)$, and "high" frequency as $f \in [0.7, 1]$. We can see that if the frequency of the selected allele is low, the region is often misclassified as neutral, since the selective sweep is not yet complete. However, if the frequency is moderate or high, the dataset is usually classified correctly as a hard sweep.

We also wanted to test the impact of unsupervised pretraining using autoencoders (see Methods). Tables 5 and 6 compare the results for a randomly initialized network to a network initialized using autoencoders. These results demonstrate that pretraining is very effective. Due to the non-convex nature of the optimization problem, random initialization is most likely finding a poor local minima for the cost function.

Table 5: Relative error on the test dataset, for a deep network with 6 hidden layers

Initialization Type	N_1 error	N_2 error	N_3 error
Random	0.429	0.421	0.710
Autoencoder	0.061	0.166	0.577

doi:10.1371/journal.pcbi.1004845.t005

For the results in the first row, the weights of the entire network were initialized randomly, then optimized. In the second row, the weights were initialized using autoencoders for each layer. The positive impact of unsupervised pretraining is clear; random initialization causes the optimization to get stuck in a local minima.

Table 6: Confusion matrix for the selection predictions, compared between random initialization (top) and autoencoder initialization (bottom), for a deep network with 6 hidden layers

		Called Class		
True Class	Neutral	Hard Sweep	Soft Sweep	Balancing
		Random Initialization		
Neutral	**1.000**	0.000	0.000	0.000
Hard Sweep	**0.978**	0.007	0.000	0.015
Soft Sweep	**1.000**	0.000	0.000	0.000
Balancing	**1.000**	0.000	0.000	0.000
		Autoencoder Initialization		
Neutral	**1.000**	0.000	0.000	0.000
Hard Sweep	0.145	**0.831**	0.004	0.021
Soft Sweep	0.011	0.001	**0.987**	0.000
Balancing	0.030	0.028	0.001	**0.941**

doi:10.1371/journal.pcbi.1004845.t006

Again, ideally we would like all 1's down the diagonal, and 0's in the off-diagonal entries. The largest number in each row is shown in boldface. When the network is initialized randomly, almost every dataset is classified as neutral; the network has not really learned anything meaningful from the input data. The overall percentage of misclassification is 74.8% for random initialization, while it is only 6.1% for autoencoder initialization.

Lastly, we wanted to assess the impact of training the deep network on data that was created under different conditions than the testing data. In particular, we simulated testing data where the ratio of the recombination rate to the mutation rate was equal to 4 (instead of 1 as in our previous results). This rather large difference in the testing data negatively impact the results, but much more so for selection than for the effective population sizes. This is perhaps due to the dependence of

selection inference on LD and haplotype statistics, which would be more affected by recombination rate variation.

We also simulated testing data with bottlenecks that were more extreme than the training data, to assess the impact of a misspecified range of demographic models. Specifically, the most recent population size was outside the range of the training data for all the testing datasets. In contrast to the previous testing scenario, here the population size results are somewhat negatively impacted (largely for the most recent size), but the selection results are only impacted slightly. In general, identifying model misspecification for continuous population genetic parameters can be done as follows: if the predicted value of a *normalized* continuous parameter is outside the range [0,1], then the testing/real data parameter is outside the range of the training/simulated data parameter. This simple criterion can help users identify when their simulated data needs to be modified to provide accurate results for their real data.

Results on real *Drosophila melanogaster* data

We then ran the real *Drosophila melanogaster* data through our trained network just like any other test dataset. We considered only the chromosome arms 2L, 2R, 3L, and 3R in this study. We partitioned each chromosome arm into 20 kb windows and ran our method on five consecutive windows at a time, sliding by 20 kb after each run. Classification was performed on the middle 20 kb window. The population size predictions for some of these regions (roughly 14%) were outside the range of the simulated data, so we discarded them from all further analysis. These regions tended to be near the telomeres and centromeres, where assembly is often poor.

For the demographic history, the population size results are shown in Table 7. The first row (average statistic method) is our best estimate, which we also plot and compare with other histories in Fig 3. Our history is close to the PSMC result, although more resolution would be needed for a proper comparison. We based our time change points (t_1 and t_2) on PSMC, so it would be surprising if these histories were notably inconsistent. The expansion after the bottleneck is roughly consistent with previous results citing the range expansion of *Drosophila melanogaster* (out of sub-Saharan Africa) as beginning around 15,000 years ago [61]. This range expansion likely led to an effective population size increase like the one we infer. Our expansion time is also consistent with bottleneck

estimates in non-African *Drosophila melanogaster* populations, such as the Netherlands population investigated by Thornton and Andolfatto [58]. In Fig 2B, we include a violin plot of the population size estimates from each region. We have the largest uncertainty for the most recent population size.

Table 7: Population size results for African *Drosophila* **from Zambia,** rounded to the nearest hundred

Prediction	N_1 (recent)	N_2 (bottleneck)	N_3 (ancestral)
Average stat prediction	544,200	145,300	652,700
Final prediction	694,000	178,400	638,300
Quantiles (2.5th, 97.5th)	(359300, 1004300)	(63800, 265000)	(593200, 694900)
Standard deviation	170,400	53,800	27,100
Neutral regions prediction	635,900	170,700	646,700

doi:10.1371/journal.pcbi.1004845.t007

The first row (predictions based on averaging the statistics for each region in the *Drosophila* genome) represents our best estimate of the population sizes.

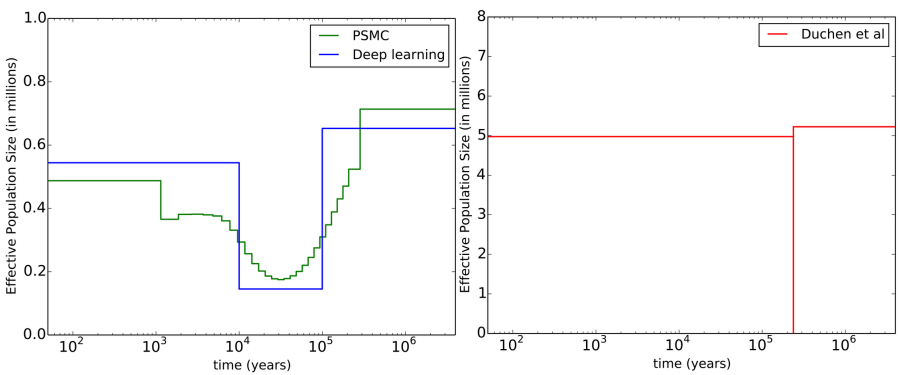


Figure 3: Comparison between demographic histories. On the left in green is PSMC [13], run on the entire genome for a subset of the data (n **= 20).**

In blue is our history from the first line of Table 7. On the right in red is the history from Duchen *et al.* [3] (note the change in the y-axis scale). The difference between our estimates and these previous estimates is likely due in large part to their assumption of a very short (1000 generations) bottleneck. Given this timing constraint, only a very severe bottleneck could fit the data. A more gradual bottleneck seems more realistic, although we do not have a simple explanation for why there was a bottleneck at all.

The number of windows classified as Neutral, Hard Sweep, Soft Sweep, and Balancing Selection are 1101, 2455, 179, 404, respectively. If we restrict our analysis to regions classified with probability greater than 0.9999, then we find 42 hard sweeps, 34 soft sweeps, and 17 regions under balancing selection. We also include a plot, Fig 4, of where the selected regions fall throughout the genome (restricted to regions with a probability at least 0.95). Interestingly, soft sweeps and balancing selection seem to occur more frequently closer to the centromere of each chromosome. There is also an excess of hard sweeps on chromosome arm 2L.



Figure 4: Plot of the selected regions we infer for African *Drosophila melanogaster*.

Each of the 4 subplots is a chromosome arm (2L, 2R, 3L and 3R). We restricted the plot to selected sites with a probability of at least 0.95.

Upon examining the genes in the top regions classified to be under selection, we find several notable results. In the hard sweep category, a few of the top regions harbor many genes involved in chromatin assembly or disassembly. Further, there are two regions each containing exactly one gene, where the gene has a known function. These are *Fic domain-containing protein* (detection of light and visual behavior), and *charybde* (negative regulation of growth).

In the soft sweep category, we find many genes related to transcription, and several related to pheromone detection (chemosensory proteins). We also find four regions each containing exactly one gene, with that gene having a known function. These are *gooseberry* (segment polarity determination), *steppke* (positive regulation of growth), *Krüppel* (proper body formation), and *Accessory gland protein 36DE* (sperm storage,

[62]). Fly embryos with a mutant *Krüppel* (German for "cripple") gene have a "gap" phenotype with the middle body segments missing [63]. We also identify three interesting genes in regions with other genes: *seminal fluid proteins 36F* and *38D* (see [64]), and *deadlock*, which is involved in transposable element silencing, which preserves genome integrity [65].

Finally in the balancing selection category, one interesting gene we find is *nervana 3*, which is involved in the sensory perception of sound. We also find balancing selection regions containing the genes *cycle* (circadian rhythms) and *Dihydropterin deaminase* (eye pigment), although there are other genes within these putatively selected regions.

Most informative statistics

One of the advantages of deep learning is that it outputs an optimal weight on each edge connecting two nodes of the network, starting with the input statistics and ending with the output variables. By investigating this network, we are able to determine which statistics are the "best" or most helpful for estimating our parameters of interest.

To perform permutation testing, we randomly permuted the values of each statistic across all the test datasets and then measured the resulting decrease in accuracy. For each output (3 population sizes and selection category), we then retained the 25 statistics that caused the largest accuracy decreases. The results are shown in the 4-way Venn diagram in Fig 5, which highlights statistics common to each subset of outputs. For each statistic name in the diagram, *close*, *mid*, and *far* represent the genomic subregion where the statistic was calculated. The numbers after each colon refer to the position of the statistic within its distribution or order; for the folded site frequency spectrum (SFS) statistics, it is the number of minor alleles. Several statistics are informative for all the parameters, including the raw number of segregating sites ("S") in each subregion and the number of singletons ("SFS, far: 1"). There is generally more overlap between the statistics for selection and N_1 or N_2, than with the statistics for selection and N_3, possibly because selection occurs on a more recent timescale. Tajima's D is also most important for selection (which it was designed to detect) and N_1. Identity-by-state (IBS) statistics seem to play an increasingly more important role as the population sizes become more ancient. Perhaps unsurprisingly, LD statistics are most important for selection. Interestingly, the SFS is not

as informative as one might have anticipated. However, the number of singletons is very informative, especially for selection and N_1, which could have been foreseen given that recent events shape the terminal branches of a genealogy, and thus the singletons. The distances between segregating sites ("BET" statistics) do not generally seem very helpful. Neither does H2, the frequency of the second most common haplotype. It is comparing H1 to H2 (via the H12 statistic) that is helpful.

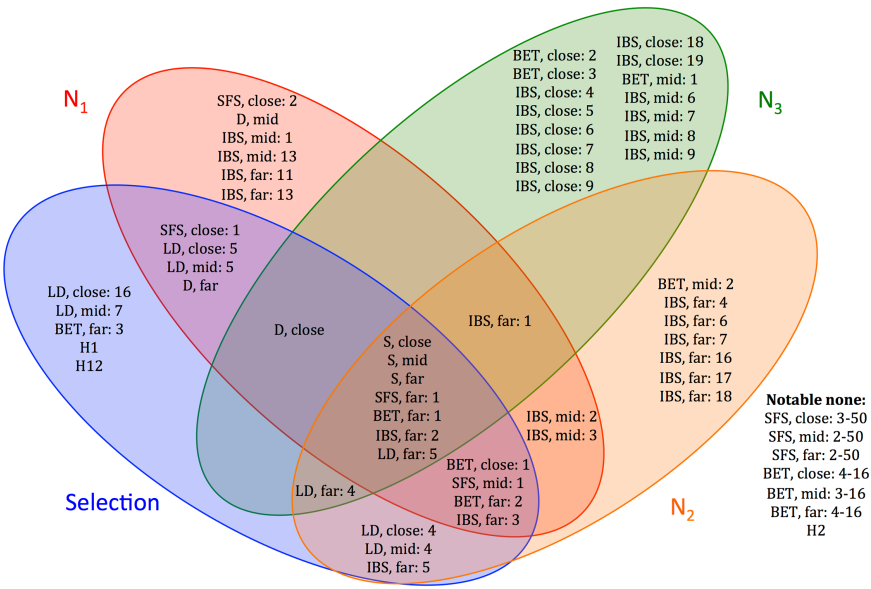


Figure 5: A Venn diagram of most informative statistics for each output variable (N_1, N_2, N_3, and selection).

For each variable, the top 25 statistics were chosen using permutation testing. The Venn diagram captures statistics common to each subset of output variables, with notable less informative statistics shown in the lower right. Close, mid, and far represent the genomic region where the statistic was calculated. The numbers after each colon refer to the position of the statistic within its distribution or order. For the SFS statistics, it is number of minor alleles. For each region, there are 50 SFS statistics, 16 BET statistics (distribution between segregating sites), 30 IBS statistics, and 16 LD statistics.

The results from the perturbation method share some similarities and have some differences from the Venn diagram in Fig 5. An interesting

set of four statistics is informative for all response variables, including very long IBS tracts close to the selected site ("IBS, close: 30") and the H12 statistic. Additionally, "LD, mid: 4" represents low LD between sites close to the selected site and sites mid-range from the selected site (likewise for LD "LD, far: 4"). Many low LD pairs could signal a lack of selection, and vice versa. IBS statistics are generally not as helpful for selection, but extremely informative for the population sizes, especially N_2 and N_3, as also observed in the permutation method. Bottlenecks can have a significant impact on IBS tract distributions, so it makes sense that N_2 (the bottleneck size) is the most reliant on IBS statistics. The number of segregating sites S is generally quite informative, especially for selection.

It is interesting that the list of notable less-informative statistics is very similar for both permutation and perturbation methods. Overall, roughly the same set of informative statistics were identified by each method, but the precise position of each statistic on the Venn diagram shifted.

Comparison with ABCtoolbox

Although ABC is not well suited for our scenario of interest and deep learning is a complementary method, we wanted to find a scenario where we could compare the performance of these two methods. To this effect, we restricted the analysis to estimating (continuous) demographic parameters only. We used the popular ABCtoolbox [54], using the same training and testing datasets as for deep learning. For ABC, the training data represents the data simulated under the prior distributions (uniform in our case), and each test dataset was compared with the training data separately. We retained 5% of the training datasets, and used half of these retained datasets for posterior density estimation. Overall, we used 75% of the datasets for training and 25% for testing.

We tested two scenarios, one with the full set of summary statistics (345 total), and the other with a reduced set of summary statistics (100 total). For the reduced set of summary statistics, we chose statistics which seemed to be informative: the number of segregating sites, Tajima's D, the first 15 entries of site frequency spectrum, H1, and the distribution of distances between segregating sites. It is interesting that the informative statistics identified above do not match this set of "intuitively" chosen

statistics. The results are shown in Table 8, which suggest that deep learning produces more accurate estimates of the recent population size (N_1) than does ABCtoolbox, whereas they have comparable accuracies for more ancient sizes.

Table 8: A comparison between ABCtoolbox and deep learning for demography only

Dataset	Method	N_1 error	N_2 error	N_3 error
Full summary statistics	ABCtoolbox	0.062	0.043	0.218
	Deep learning	0.044	0.028	0.221
Filtered summary statistics	ABCtoolbox	0.161	0.035	0.311
	Deep learning	0.065	0.055	0.319

doi:10.1371/journal.pcbi.1004845.t008

Out of 1000 demographies (160,000 datasets total), 75% were used for training and 25% for testing. In this scenario, deep learning generally outperforms ABCtoolbox, as measured by the relative error: $|N_{\text{est}} - N_{\text{true}}|/N_{\text{true}}$. There is generally more improvement using deep learning when the number of statistics is larger.

Regularization of the network weights

In deep learning, one hyper-parameter that should be investigated closely is the regularization parameter λ in the cost function, also called the weight decay parameter. (See Methods for details). If λ is set to be too high, large weights will be penalized too much, and interesting features of the data cannot be learned well. But if λ is set too low, the weights tend to display runaway behavior. Due to this balance, a validation procedure is typically used to find the right λ. In our case, the additional runtime of simulating more data and performing more training would be too computationally expensive.

Runtime

In terms of runtime, the majority is spent simulating the data. During training, most of the runtime is spent fine-tuning the deep network, which requires computing the cost function and derivatives many times. To speed up this computation, our deep learning implementation is parallelized across datasets, since each dataset adds to the cost function independently. This significantly improves the training time for deep learning, which can be run in a few days on our simulated dataset

(400,000 regions) with a modest number of hidden nodes/layers. Once the training is completed, an arbitrary number of datasets can be tested more or less instantaneously. In contrast, each of the "training" datasets for ABC must be examined for *each*test dataset. This takes several weeks for a dataset of this size (which is why we tested ABC on a subset of the data), although it could be parallelized across the test datasets. We include a runtime comparison on the reduced dataset in Table 9.

Table 9: Approximate runtime results

Task	ABCtoolbox	Deep Learning
Simulating data	370 hrs (10 ~ 15 cores)	370 hrs (10 ~ 15 cores)
Computing summary statistics	1800 hrs (1 core)	1800 hrs (1 core)
Demography only (1000 × 160 datasets)		
Training and testing (filtered statistics)	114 hrs (1 core)	3.75 hrs (20 cores)
Training and testing (unfiltered)	336 hrs (1 core)	11 hrs (20 cores)
Demography & selection (2500 × 160 datasets)		
Training	N/A	74 hrs (20 cores)
Testing	N/A	3 min (1 core)

doi:10.1371/journal.pcbi.1004845.t009

Simulating the data (which is the same for both methods) and computing the summary statistics is the most time consuming part of the process, although it is highly parallelized. ABCtoolbox could also be parallelized, but we did not implement that here. For the "filtered statistics" row, the 345 statistics have been downsampled to 100 for all the datasets.

DISCUSSION

In this paper, we have sought to demonstrate the potential of deep learning as a new inference framework for population genomic analysis. This investigation is still in its infancy, and there are many directions of future work. One important advantage of deep learning is that it provides a way to distinguish informative summary statistics from uninformative ones. Here we have presented two methods for extracting informative statistics given a trained deep network. Other methods are possible, and it is an open question which one would produce the most accurate results. It would be interesting to down-sample statistics using a variety of methods, then compare the results. Overall, learning more about how various summary statistics relate to parameters would be useful for

population genetics going forward.

The prospect of using deep learning to classify regions as neutral or selected is very appealing for subsequent demographic inference. There are other machine learning methods that perform such classification, but they are generally limited to two classes (selected or neutral). One exception is a study in humans [66], which classifies genomic regions as neutral or under positive, negative, or balancing selection. Although their approach does not jointly infer selection and demography, it would be interesting to see their method used on *Drosophila*.

We infer many hard sweeps in African *Drosophila*, which is perhaps expected given their large effective population size. However, when restricting our analysis to selected regions with high confidence, the numbers of hard sweeps and soft sweeps are comparable. It would be interesting to analyze our results in the context of a simulation study by Schrider *et al.* [67], which found that regions classified as soft sweeps are often truly the "shoulders" of hard sweeps. This possibility is worth investigating, as the signatures of soft sweeps and soft shoulders are extremely similar. In our simulations, both hard and soft sweeps are relatively recent, which makes them easier to detect. It would be useful to incorporate datasets with a wider range of selection onset times, which could capture regions of the real data that are currently being classified as neutral. Right now we have 4 different selection classes which are all forms of positive selection. To incorporate background or purifying selection, a 5th class of (weakly) deleterious mutations could be added.

One aspect of the simulations that warrants future study is the effect of the ratio of recombination to mutation rate, ρ/θ. From running PSMC on 20 haplotypes from Zambia, we inferred $\rho/\theta = 1$, whereas other studies have found $\rho/\theta = 6.8$ [59] and $\rho/\theta = 10$ [58] for a population from Zimbabwe. Both these works also show significant variation in this ratio across different *Drosophila* populations. Andolfatto and Przeworski [60] also find significant variation in this ratio across the genome. As noted by Haddrill *et al.* [59], recent bottlenecks can make this ratio appear lower than it actually is, which is possibly the case for our dataset. As shown in the Supporting Information, misspecification of this ratio can lead to poor selection classification results. In general, assuming a lower recombination rate than reality leads to conservative results, since sites are assumed to be less independent [58–60]. Our takeaway from this

analysis is that ρ/θ should be accurately estimated using a more robust method than PSMC or incorporated as a model parameter, with datasets simulated under a variety of ratios. It would also be interesting to investigate variation in the recombination rate across the genome, which could be incorporated into future simulations.

Deep learning can make efficient use of even a limited number of simulated datasets. In this vein, it would be interesting to use an approach such as ABC MCMC [24] to simulate data, and then use deep learning on these simulated datasets. Alternatively, deep learning could be used to select informative statistics for a subsequent method such as ABC [49]. Blum and François [28] used a single-layer neural network to perform (non-linear) regression to obtain a posterior mean, which they used to adjust the parameters of the posterior sample for a more accurate posterior distribution. It is unclear whether deep networks (where the theory is much more complicated) could be used in the same way, although there is the concept of a residual which could be used for parameter adjustment in an rejection-based framework. Such hybrid approaches could be a fruitful area of further exploration, potentially extending the generality and usefulness of ABC.

We would also like to apply deep learning to a wider variety of scenarios in populations genetics. Population structure and splits would be examples, although these scenarios would most likely require very different sets of summary statistics.

On the computer science side, deep learning has almost exclusively been used for classification, not continuous parameter inference. It would be interesting to see the type of continuous parameter inference presented here used in other fields and applications.

Finally, machine learning methods have been criticized for their "black-box" nature. In some sense they throw away a lot of the coalescent modeling that we know to be realistic, although this is included somewhat in the expert summary statistics of the data. It would be advantageous to somehow combine the strengths of coalescent theory and the strengths of machine learning to create a robust method for population genetic inference.

METHODS

Deep learning details

In this section we provide the theory behind training deep networks. The notation in this section follows that of [68]. Let $\boldsymbol{x}^{(i)}$ be the vector of summary statistics for dataset i, and $\boldsymbol{y}^{(i)}$ the vector of response variables that dataset i was simulated under. If we have m such datasets, then together $\{(\boldsymbol{x}^{(1)}, \boldsymbol{y}^{(1)}), \ldots, (\boldsymbol{x}^{(m)}, \boldsymbol{y}^{(m)})\}$ form the training data that will be used to learn the function from summary statistics to response variables. Deep neural networks are a way to express this type of complex, non-linear function. The first layer of the network is the input data, the next layers are the "hidden layers" of the network, and the final layer represents the network's prediction of the response variable.

Cost function for a deep network.

An example deep network is shown in Fig 6. The collection of weights between layer l and layer l+1 is denoted $\boldsymbol{W}^{(\ell)} = (w_{jk}^{(\ell)})$, where $w_{jk}^{(\ell)}$ is the weight associated with the connection between node j in layer l and node k in layer l+1. The biases for layer l is denoted $\boldsymbol{b}^{(l)}$. The total number of layers (including the input and output layers) is denoted L, and the number of hidden nodes in layer l is denoted u_l. The main goal is to learn the weights that best describe the function between the inputs and the outputs.

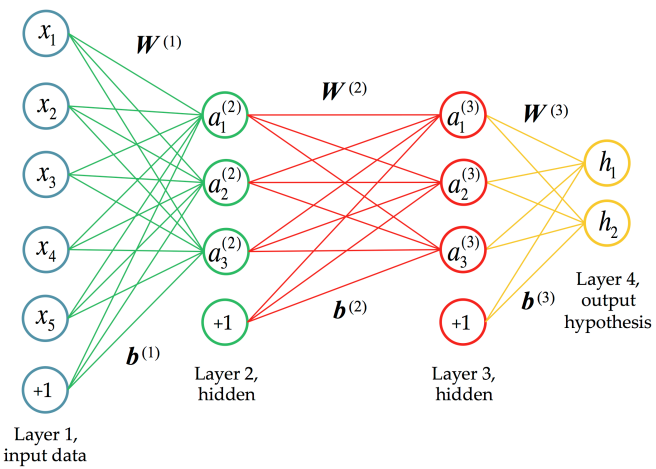


Figure 6: An example of a deep neural network with two hidden layers.

The first layer is the input data (each dataset has 5 statistics), and the last layer predicts the 2 response variables. The last node in each input layer (+1) represents the bias term. Here the number of layers $L = 4$, and the number of nodes (computational units) in each layer is $u_1 = 5$, $u_2 = 3$, $u_3 = 3$, and $u_4 = 2$ (these counts exclude the biases).

To learn this function, we first describe how the values of the hidden nodes are computed, given a trial weight vector. The value of hidden node j in layer $l \geq 2$ is denoted $a_j^{(\ell)}$, and is defined by

$$a_j^{(\ell)} = f\left(z_j^{(\ell)}\right) \quad \text{and} \quad z_j^{(\ell)} = \boldsymbol{W}_j^{(\ell-1)} \cdot a^{(\ell-1)} + b_j^{(\ell-1)}, \tag{1}$$

where $\boldsymbol{W}_j^{(\ell-1)}$ is the j^{th} *column* of the weight matrix $\boldsymbol{W}^{(\ell-1)}$ (i.e., all the weights going into node j of layer $l-1$), $\boldsymbol{a}^{(\ell-1)} = (a_k^{(\ell-1)})$ is a vector of the values of all the nodes in layer $l-1$ (with $a^{(l)} = x$, the input vector), and

$$f(z) := \frac{1}{1 + \exp(-z)} \tag{2}$$

is the *activation function*. Here we use a logistic function, but other functions can be used. Another common activation function is the hyperbolic tangent function.

Hence, given the input data and a set of weights, we can *feed forward* to learn the values of all hidden nodes, and a prediction of the response variables. These predictions are usually denoted by $\boldsymbol{h}_{\boldsymbol{W},\boldsymbol{b}}(\boldsymbol{x}^{(i)})$ for our *hypothesis* for dataset i, based on all the weights $\boldsymbol{W} = (\boldsymbol{W}^{(1)}, \ldots, \boldsymbol{W}^{(L-1)})$ and biases $\boldsymbol{b} = (\boldsymbol{b}^{(1)}, \ldots, \boldsymbol{b}^{(L-1)})$. We will discuss different ways to compute the hypothesis function later on. To find the best weights, we define a loss function based on the square of the l_2-norm between this hypothesis and the true response variables. This loss function is given by

$$J(\boldsymbol{W}, \boldsymbol{b}) = \frac{1}{m} \sum_{i=1}^{m} \frac{1}{2} \left\| \boldsymbol{h}_{\boldsymbol{W},\boldsymbol{b}}(\boldsymbol{x}^{(i)}) - \boldsymbol{y}^{(i)} \right\|^2. \tag{3}$$

The goal of deep learning is to find the weights ($\boldsymbol{W}$ and $\boldsymbol{b}$) that minimize this loss function. To efficiently find these optimal weights, we can use *backpropagation* to find the gradient. The intuition behind this approach is that once we have found the hypothesis, we then want to see how much each of the weights contributed to any differences between the hypothesis and the truth. Therefore we start at the last hidden layer,

see how much each of those weights contributed, then work our way backwards, using the gradient of the previous layer to compute the gradient of the next layer. For this procedure we need to compute the partial derivatives of the cost function with respect to each weight.

Consider a single training example $\boldsymbol{x}$ with associated output $\boldsymbol{y}$. The cost for this dataset is a term in Eq (3) and is denoted by

$$J(\boldsymbol{W}, \boldsymbol{b}, \boldsymbol{x}, \boldsymbol{y}) := \frac{1}{2} \|\boldsymbol{h}_{\boldsymbol{W},\boldsymbol{b}}(\boldsymbol{x}) - \boldsymbol{y}\|^2.$$

Then, define

$$\delta_j^{(\ell)} := \frac{\partial J(\boldsymbol{W}, \boldsymbol{b}, \boldsymbol{x}, \boldsymbol{y})}{\partial z_j^{(\ell)}},$$

where $z_j^{(\ell)}$, defined in Eq (1), is the input to the activation function for node j in layer l. We first consider $\delta_j^{(L)}$ for the final layer. Noting that the j^{th} entry of $\boldsymbol{h}_{\boldsymbol{W},\boldsymbol{b}}(\boldsymbol{x})$ is $f(z_j^{(L)})$, we get

$$\delta_j^{(L)} = [f(z_j^{(L)}) - y_j] f'\left(z_j^{(L)}\right).$$

Based on this initialization, we can recursively compute all the δ variables:

$$\delta_j^{(\ell)} = \left[\sum_{k=1}^{u_{\ell+1}} w_{jk}^{(\ell)} \delta_k^{(\ell+1)}\right] f'\left(z_j^{(\ell)}\right).$$

Now we can use the δ variables to recursively compute the partial derivatives for one dataset:

$$\begin{aligned} \frac{\partial J(\boldsymbol{W}, \boldsymbol{b}, \boldsymbol{x}, \boldsymbol{y})}{\partial w_{jk}^{(\ell)}} &= a_j^{(\ell)} \delta_k^{(\ell+1)}, \\ \frac{\partial J(\boldsymbol{W}, \boldsymbol{b}, \boldsymbol{x}, \boldsymbol{y})}{\partial b_k^{(\ell)}} &= \delta_k^{(\ell+1)}. \end{aligned}$$

Finally, putting all the datasets together we get

$$\frac{\partial J(\boldsymbol{W}, \boldsymbol{b})}{\partial w_{jk}^{(\ell)}} = \sum_{i=1}^{m} \frac{\partial J(\boldsymbol{W}, \boldsymbol{b}, \boldsymbol{x}^{(i)}, \boldsymbol{y}^{(i)})}{\partial w_{jk}^{(\ell)}} \quad \text{and} \quad \frac{\partial J(\boldsymbol{W}, \boldsymbol{b})}{\partial b_k^{(\ell)}} = \sum_{i=1}^{m} \frac{\partial J(\boldsymbol{W}, \boldsymbol{b}, \boldsymbol{x}^{(i)}, \boldsymbol{y}^{(i)})}{\partial b_k^{(\ell)}}.$$

Since we can compute the derivatives using this backpropagation algorithm, we can use the LBFGS optimization routine (as implemented in [69]) to find the weights that minimize the cost function.

Unsupervised pretraining using autoencoders.

It is possible to train a deep network by attempting to minimize the cost function described above directly, but in practice, this proved difficult due to the high-dimensionality and non-convexity of the optimization problem. Initializing the weights randomly before training resulted in poor local minima. Hinton and Salakhutdinov [37] sought to initialize the weights in a more informed way, using an unsupervised pretraining routine. Unsupervised training ignores the output (often called the "labels") and attempts to learn as much as possible about the structure of the data on its own. PCA is an example of unsupervised learning. In Hinton and Salakhutdinov, the unsupervised pretraining step uses an *autoencoder* to try to learn the best function from the data to itself, after it has gone through a dimensionality reduction step (this can be thought of as trying to compress the data, then reconstruct it with minimal loss). Autoencoding provides a way to initialize the weights of a deep network that will ideally be close to optimal for the supervised learning step as well.

Training an autoencoder is an optimization procedure in itself. As before, let $\boldsymbol{W}^{(1)}$ be the vector of weights connecting the input $\boldsymbol{x}$ to the hidden layer $\boldsymbol{a}$, and $\boldsymbol{x}^{(2)}$ be the vector of weights connecting $\boldsymbol{a}$ to the output layer $\hat{\boldsymbol{x}}$, which in this case should be as close as possible to the original input data. We again typically use the logistic function shown in Eq (2) as our activation function *f*, so we can compute the output using:

$$a_j = f(\boldsymbol{W}_j^{(1)} \cdot \boldsymbol{x} + b_j^{(1)}) \quad \text{and} \quad \hat{x}_k = f(\boldsymbol{W}_k^{(2)} \cdot \boldsymbol{a} + b_k^{(2)}).$$

If a linear activation function is used instead of a logistic function, the hidden layer becomes the principle components of the data. This makes dimensionality reduction with an autoencoder similar in spirit to PCA, which has been used frequently in genetic analysis (see [70] for an example). However, the non-linear nature of an autoencoder has been shown to reconstruct complex data more accurately than PCA. Using backpropagation as we did before, we can minimize the following autoencoder cost function using all *m* input datasets:

$$A(\boldsymbol{W}, \boldsymbol{b}) = \frac{1}{m} \sum_{i=1}^{m} \frac{1}{2} \| \hat{\boldsymbol{x}}^{(i)} - \boldsymbol{x}^{(i)} \|^2.$$

The resulting weights $\boldsymbol{W}^{*(1)}$ and $\boldsymbol{b}^{*(1)}$ will then be used to initialize the weights between the first and second layers of our deep network. The weights $\boldsymbol{W}^{*(2)}>$ and $\boldsymbol{b}^{*(2)}$ are discarded. To initialize the rest of the weights, we can repeat the autoencoder procedure, but this time we will use the hidden layer $\boldsymbol{a}^* = (a_j^*)$, where $a_j^* = f(\boldsymbol{W}_j^{*(1)} \cdot \boldsymbol{x} + b_j^{*(1)})$, as our input data, and feed it through the next hidden layer. In this way we can use "stacked" autoencoders to initialize all the weights of the deep network. Finally, the supervised training procedure described in the previous section can be used to *fine-tune* the weights to obtain the best function from the inputs to the response variables.

When the number of hidden nodes is large, we would like to constrain an autoencoder such that only a fraction of the hidden nodes are "firing" at any given time. This corresponds to the idea that only a subset of the neurons in our brains are firing at once, depending on the input stimulus. To create a similar phenomenon for an autoencoder, we create a *sparsity* constraint that ensures the activation of most of the nodes is close to 0, and the activation of a small fraction, ρ, of nodes is close to 1. Let $\hat{p}_j$ be the average activation of the hidden node j:

$$\hat{\rho}_j = \frac{1}{m}\sum_{i=1}^{m} a_j(\boldsymbol{x}^{(i)}),$$

where $a_j(\boldsymbol{x}^{(i)})$ is the value of the j^{th} hidden node when activated with dataset $\boldsymbol{x}^{(i)}$. To ensure sparsity, we would like $\hat{p}_j$ to be close to ρ, our desired fraction of active nodes. This can be accomplished by minimizing the KL divergence:

$$\sum_{j=1}^{u_2} \mathrm{KL}(\rho \| \hat{\rho}_j) = \sum_{j=1}^{u_2} \left[\rho \log \frac{\rho}{\hat{\rho}_j} + (1-\rho) \log \frac{1-\rho}{1-\hat{\rho}_j} \right],$$

where u_2 is the number of nodes in the hidden layer of the autoencoder. We multiply this term by a sparsity weight β.

In addition, a regularization term is included, which prevents the magnitude of the weights from becoming too large. To accomplish this, we add a penalty to the cost function that is the sum of the squares of all weights (excluding the biases), weighted by a well-chosen constant λ, which is often called the *weight decay parameter*. Including both sparsity and regularization, our final autoencoder cost becomes:

$$A_\lambda(\boldsymbol{W},\boldsymbol{b}) = \frac{1}{m}\sum_{i=1}^{m}\frac{1}{2}\|\hat{\boldsymbol{x}}^{(i)} - \boldsymbol{x}^{(i)}\|^2 + \beta\sum_{j=1}^{u_2}\mathrm{KL}(\rho\|\hat{\rho}_j) + \frac{\lambda}{2}\sum_{\ell=1}^{2}\sum_{j=1}^{u_\ell}\sum_{k=1}^{u_{\ell+1}}\left[\left(w_{jk}^{(\ell)}\right)\right]^2$$

We also regularize the weights on the last layer during fine-tuning, so our deep learning cost function becomes:

$$J_\lambda(\boldsymbol{W},\boldsymbol{b}) = \frac{1}{m}\sum_{i=1}^{m}\frac{1}{2}\|\boldsymbol{h}_{\boldsymbol{W},\boldsymbol{b}}(\boldsymbol{x}^{(i)}) - \boldsymbol{y}^{(i)}\|^2 + \frac{\lambda}{2}\sum_{j=1}^{u_{L-1}}\sum_{k=1}^{u_L}\left[w_{jk}^{(L-1)}\right]^2$$

The final layer: Continuous variable vs. classification.

In population genetics, often we want to estimate continuous response variables. To compute our hypothesis for a response variable, based on a set of weights, we could use a logistic activation function, Eq (2), as we did for the other layers. However, the logistic function is more suitable for binary classification. Instead, we use a linear activation function, so in the case of a single response variable, our hypothesis for dataset *i* becomes

$$\boldsymbol{h}_{\boldsymbol{W},\boldsymbol{b}}^{\text{linear}}(\boldsymbol{x}^{(i)}) = [\boldsymbol{a}^{(L-1)}(\boldsymbol{x}^{(i)})]^T\boldsymbol{W}^{(L-1)} + \boldsymbol{b}^{(L-1)}.$$

In other words, it is the dot product of the activations of the final hidden layer and the weights that connect the final hidden layer to the response variables.

For such classification results, if we had two classes, we could use logistic regression to find the probability a dataset was assigned to each class. With more than two classes, we can extend this concept and use *softmax regression* to assign a probability to each class. If we have *K* classes labeled 1, …, *K*, we can define our hypothesis as follows

$$\boldsymbol{h}_{\boldsymbol{W},\boldsymbol{b}}^{\text{softmax}}(\boldsymbol{x}^{(i)}) = \begin{bmatrix} p(y^{(i)} = 2 \mid \boldsymbol{x}^{(i)}; \boldsymbol{W}, \boldsymbol{b}) \\ p(y^{(i)} = 2 \mid \boldsymbol{x}^{(i)}; \boldsymbol{W}, \boldsymbol{b}) \\ \vdots \\ p(y^{(i)} = K \mid \boldsymbol{x}^{(i)}; \boldsymbol{W}, \boldsymbol{b}) \end{bmatrix}$$

$$= \frac{1}{Z}\begin{bmatrix} \exp\{\boldsymbol{W}_1^{(L-1)} \cdot \boldsymbol{a}^{(L-1)}(\boldsymbol{x}^{(i)}) + b_1^{(L-1)}\} \\ \exp\{\boldsymbol{W}_2^{(L-1)} \cdot \boldsymbol{a}^{(L-1)}(\boldsymbol{x}^{(i)}) + b_2^{(L-1)}\} \\ \vdots \\ \exp\{\boldsymbol{W}_K^{(L-1)} \cdot \boldsymbol{a}^{(L-1)}(\boldsymbol{x}^{(i)}) + b_K^{(L-1)}\} \end{bmatrix},$$

where Z is the sum of all the entries, so that our probabilities sum to 1. Using this formulation, we can define our classification cost function:

$$J_\lambda^{\text{softmax}}(\boldsymbol{W}, \boldsymbol{b}) = -\frac{1}{m}\sum_{i=1}^{m}\sum_{j=1}^{K} \mathbb{1}\{y^{(i)} = j\} \log p(y^{(i)} = j|\boldsymbol{x}^{(i)}; \boldsymbol{W}, \boldsymbol{b}) + \frac{\lambda}{2}\sum_{s=1}^{u_{L-1}}\sum_{t=1}^{u_L}\left[\boldsymbol{W}_{st}^{(L-1)}\right]^2$$

Intuitively, we can think about this cost function as making the log probability of the correct class as close to 0 as possible.

Transforming input data into summary statistics

For many deep learning applications, the raw data can be used directly (the pixels of an image, for example). Unfortunately, we currently cannot input raw genomic data into a deep learning method. Similarly to ABC, we need to transform the data into summary statistics that are potentially informative about the parameters of interest. Unlike ABC, however, deep learning should not be negatively affected by correlated or uninformative summary statistics. Thus we sought to include a large number of potentially informative summary statistics of the data. To account for the impact of selection, we divided each 100 kb region into three smaller regions: 1) close to the selected site (40–60 kb), 2) mid-range from the selected site (20–40 kb and 60–80 kb), and 3) far from the selected site (0–20 kb and 80–100 kb). These regions are based off of the simulation scenario in Peter *et al.* [23], and shown more explicitly in Fig 7. Within each of these three regions, we calculated the following statistics, except where noted.

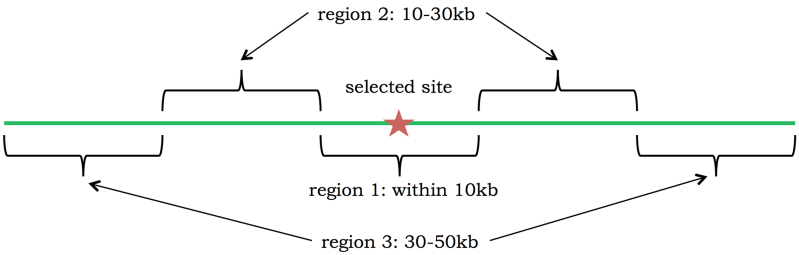


Figure 7: Regions used for computing the statistics, which are based off of Peter *et al.* [23].

Note that the selected site was chosen randomly within region 1.

For all the statistics described below, n is the haploid sample size. In the case of simulated data, $n = 100$. For the real data, we had 197 samples; within each 100 kb region we sorted the samples by missing data, then retained the 100 most complete samples, except where noted.

- Number of segregating sites within each smaller region, S. Since all of the statistics must be in [0, 1], we normalized by $S_{\max} = 5000$ (any $S > S_{\max}$ was truncated): 3 statistics.
- Tajima's D statistic [71], computed as follows

$$D = \frac{\pi - S/a_1}{\sqrt{\widehat{\mathrm{Var}}(\pi - S/a_1)}},$$

where ϖ is the average number of pairwise differences between two samples, and $a_1 = \sum_{i=1}^{n-1} 1/i$. We normalized by $D_{\min} = -3.0$ and $D_{\max} = 3.0$, again truncating the rare statistic outside this range: 3 statistics.

- Folded site frequency spectrum (SFS): η_i is the number of segregating sites where the minor allele occurs i times out of n samples, for $i = 1, 2, \ldots, \lfloor n/2 \rfloor$. For the real data, for each segregating site, enough samples were included to obtain 100 with non-missing data. If that was not possible for a given site, the site was not included. To normalize the SFS, we divided each η_i by the sum of the entries, which gives us the probability of observing i minor alleles, given the site was segregating: $50 \cdot 3 = 150$ statistics.

- Length distribution between segregating sites: let B_k be the number of bases between the k and k+1 segregating sites. To compute the distribution of these lengths, we define J bins and count the number of lengths that fall into each bin:

$$d_j^{\text{BET}} = \frac{|k \in \{1, 2, \cdots, S-1\} : B_k \in \text{bin } j|}{S}$$

We choose $J = 16$ equally spaced bins, the first starting at 0 and the last starting at 300: $16 \cdot 3 = 48$ statistics.

- Identity-by-state (IBS) tract length distribution: for each pair of samples, and IBS tract is a contiguous genomic region where the samples are identical at every base (delimited by bases where they differ). For all pairs, let L be the set of IBS tract lengths. In a similar fashion to the length distribution statistics, we define M bins, and count the number of IBS tract lengths that fall into each bin:

$$d_m^{\text{IBS}} = \frac{|\ell \in L : \ell \in \text{bin } m|}{|L|}$$

We choose $M = 30$ equally spaced bins, the first starting at 0 and the last starting at 5000: $30 \cdot 3 = 90$ statistics.

- Linkage disequilibrium (LD) distribution: LD is a measure of the correlation between two segregating sites. For example, if there was no recombination between two sites, their alleles would be highly correlated and LD would be high in magnitude. If there was infinite recombination between two sites, they would be independent and have LD close to 0. For two loci, let A, a be the alleles for the first site, and B, b be the alleles for the second site. Let p_A be the frequency of allele A, p_B be the frequency of allele B, and p_{AB}be the frequency of haplotype AB. Then the linkage disequilibrium is computed by

$$D_{AB} = p_{AB} - p_A p_B.$$

Here we compute the LD between pairs of sites, where one site is in the "selected" region, and the other is in each of the three regions (including the selected region). Then we create an LD distribution similar

to the IBS distribution above, using 16 bins, with the first bin ending at −0.05, and the last bin starting at 0.2: $16 \cdot 3 = 48$ statistics.

- H1, H12, and H2 statistics, as described in Garud *et al.* [72]. These statistics help to distinguish between hard and soft sweeps, and are calculated in the selected (middle) region only: 3 statistics.

This gives us a total of 345 statistics.

Deep learning for demography and selection

Deep learning model

To modify our deep learning method to accommodate this type of inference problem, during training we have an outer-loop that changes the demography as necessary, and an inner loop that accounts for differences in selection for each region. During testing, we compare three different methods for predicting the effective population sizes: (1) *Average stat*: we average the statistics for each region within the dataset (corresponding to the same demography), then run this average through the trained network. (2) *Final*: we run the statistics for each region through the trained network separately, then average the results. (3) *Neutral regions*: we use the same method as (2), but instead of using all the regions, we only use the ones we predict as being neutral. We also include a range between the 2.5th and 97.5th quantiles. To predict the selection class, we obtain a probability distribution over the classes for each region, then select the class with the highest probability.

One final complication is that we estimate continuous response variables for the population sizes, but consider selection to be a discrete response variable. This involves a linear activation function for the population sizes and a softmax classifier for selection. A diagram of our deep learning method is shown in Fig 8.

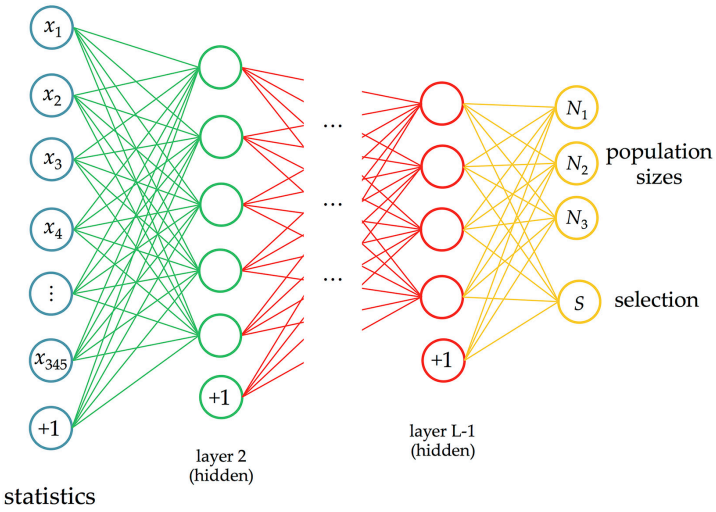


Figure 8: Our deep learning framework for effective population size changes and selection.

Analysis of informative statistics.

One advantage of deep learning is that the weights of the optimal network give us an interpretable link between the summary statistics and the output variables. However, from these weights, it is not immediately obvious which statistics are the most informative or "best" for a given output variable. If we only wanted to use only a small subset of the statistics, which ones would give us the best results? To answer this question, we use both a permutation testing approach and a perturbation method.

Software availability

An open-source software package (called evoNet) that implements deep learning algorithms for population genetic inference is available at https://sourceforge.net/projects/evonet.

ACKNOWLEDGMENTS

We thank Andrew Kern and Daniel Schrider for helpful discussion on classifying hard and soft sweeps, as well as Yuh Chwen (Grace) Lee for many insightful comments on our *Drosophila melanogaster* selection

results. Also, we are very grateful to John Pool and Kevin Thornton for helpful comments on our manuscript.

AUTHOR CONTRIBUTIONS

Conceived and designed the experiments: SS YSS. Performed the experiments: SS. Analyzed the data: SS YSS. Contributed reagents/materials/analysis tools: SS YSS. Wrote the paper: SS YSS. Developed the software used in analysis: SS.

REFERENCES

1. Jones N. Computer science: The learning machines. Nature. 2014;505(7482):146. pmid:24402264
2. Li J, Li H, Jakobsson M, Li S, Sjödin P, Lascoux M. Joint analysis of demography and selection in population genetics: where do we stand and where could we go? Molecular Ecology. 2012;21(1):28–44.
3. Duchen P, Živković D, Hutter S, Stephan W, Laurent S. Demographic inference reveals African and European admixture in the North American *Drosophila melanogaster*population. Genetics. 2013;193(1):291–301. pmid:23150605
4. González J, Macpherson JM, Messer PW, Petrov DA. Inferring the Strength of Selection in *Drosophila* under Complex Demographic Models. Molecular Biology and Evolution. 2009;26(3):513–526. pmid:19033258
5. Sella G, Petrov DA, Przeworski M, Andolfatto P. Pervasive natural selection in the *Drosophila* genome? PLoS Genetics. 2009;5(6):e1000495. pmid:19503600
6. Lack JB, Cardeno CM, Crepeau MW, Taylor W, Corbett-Detig RB, Stevens KA, et al. The Drosophila Genome Nexus: a population genomic resource of 623 *Drosophila melanogaster* genomes, including 197 from a single ancestral range population. Genetics. 2015;199(4):1229–1241. pmid:25631317
7. Pavlidis P, Jensen JD, Stephan W. Searching for footprints of positive selection in whole-genome SNP data from nonequilibrium populations. Genetics. 2010;185:907–922. pmid:20407129

8. Pavlidis P, Živković D, Stamatakis A, Alachiotis N. SweeD: likelihood-based detection of selective sweeps in thousands of genomes. Molecular Biology and Evolution. 2013;30(9):2224–2224. pmid:23777627
9. Ronen R, Udpa N, Halperin E, Bafna V. Learning natural selection from the site frequency spectrum. Genetics. 2013;195:181–193. pmid:23770700
10. Lin K, Li H, Schlötterer C, Futschik A. Distinguishing positive selection from neutral evolution: boosting the performance of summary statistics. Genetics. 2011;187:229–244. pmid:21041556
11. Lin K, Futschik A, Li H. A fast estimate for the population recombination rate based on regression. Genetics. 2013;194:473–484. pmid:23589457
12. Pybus M, Luisi P, Dall'Olio G, Uzkundun M, Laayouni H, Bertranpetit J, et al. Hierarchical Boosting: a machine-learning framework to detect and classify hard selective sweeps in human populations. Bioinformatics. 2015;In press. pmid:26315912
13. Li H, Durbin R. Inference of human population history from individual whole-genome sequences. Nature. 2011;475:493–496. pmid:21753753
14. Sheehan S, Harris K, Song YS. Estimating variable effective population sizes from multiple genomes: A sequentially Markov conditional sampling distribution approach. Genetics. 2013;194(3):647–662. pmid:23608192
15. Steinrücken M, Kamm JA, Song YS. Inference of complex population histories using whole-genome sequences from multiple populations; 2015. BioRxiv preprint, http://dx.doi.org/10.1101/026591.
16. Schiffels S, Durbin R. Inferring human population size and separation history from multiple genome sequences. Nature Genetics. 2014;46:919–925. pmid:24952747
17. Galtier N, Depaulis F, Barton NH. Detecting bottlenecks and selective sweeps from DNA sequence polymorphism. Genetics. 2000;155:981–987. pmid:10835415
18. Gossmann TI, Woolfit M, Eyre-Walker A. Quantifying the variation in the effective population size within a genome. Genetics. 2011;189:1389–1402. pmid:21954163

19. Pritchard JK, Seielstad MT, Perez-Lezaun A, Feldman MW. Population growth of human Y chromosomes: a study of Y chromosome microsatellites. Molecular Biology and Evolution. 1999;16(12):1791–1798. pmid:10605120
20. Beaumont MA, Zhang W, Balding DJ. Approximate Bayesian Computation in Population Genetics. Genetics. 2002;162(4):2025–2035. pmid:12524368
21. Becquet C, Przeworski M. A new approach to estimate parameters of speciation models with application to apes. Genome Res. 2007;17(10):1505–19. pmid:17712021
22. Jensen JD, Thornton KR, Andolfatto P. An approximate Bayesian estimator suggests strong, recurrent selective sweeps in *Drosophila*. PLoS Genetics. 2008;4(9):e1000198. pmid:18802463
23. Peter BM, Huerta-Sanchez E, Nielsen R. Distinguishing between selective sweeps from standing variation and from a *de novo* mutation. PLoS Genetics. 2012;8(10):e1003011. pmid:23071458
24. Marjoram P, Molitor J, Plagnol V, Tavare S. Markov chain Monte Carlo without likelihoods. PNAS. 2003;100(26):15324–8. pmid:14663152
25. Sisson SA, Fan Y, Tanaka MM. Sequential Monte carlo without likelihoods. Proceedings of the National Academy of Sciences. 2007;104(6):1760–1765.
26. Joyce P, Marjoram P. Approximately sufficient statistics and Bayesian computation. Statistical Applications in Genetics and Molecular Biology. 2008;7(1):Article26. pmid:18764775
27. Beaumont MA, Cornuet JM, Marin JM, Robert CP. Adaptive approximate Bayesian computation. Biometrika. 2009;96(4):983–990.
28. Blum MGB, François O. Non-linear regression models for Approximate Bayesian Computation. Statistics and Computing. 2010;20(1):63–73.
29. Nunes MA, Balding DJ. On optimal selection of summary statistics for approximate Bayesian computation. Statistical Applications in Genetics and Molecular Biology. 2010;9(1):34.
30. Didelot X, Everitt RG, Johansen AM, Lawson DJ, et al. Likelihood-

free estimation of model evidence. Bayesian Analysis. 2011;6(1):49–76.

31. Fearnhead P, Prangle D. Constructing summary statistics for approximate Bayesian computation: semi-automatic approximate Bayesian computation. Journal of the Royal Statistical Society: Series B (Statistical Methodology). 2012;74(3):419–474.
32. Beaumont MA. Approximate Bayesian computation in evolution and ecology. Annual Review of Ecology, Evolution, and Systematics. 2010;41:379–406.
33. Bazin E, Dawson KJ, Beaumont MA. Likelihood-Free Inference of Population Structure and Local Adaptation in a Bayesian Hierarchical Model. Genetics. 2010;185(2):587–602. pmid:20382835
34. Hopfield JJ. Neural networks and physical systems with emergent collective computational abilities. PNAS. 1982;79(8):2554–2558. pmid:6953413
35. Cybenko G. Approximations by superpositions of sigmoidal functions. Mathematics of Control, Signals, and Systems. 1989;2(4):303–314.
36. Hornik K. Approximation Capabilities of Multilayer Feedforward Networks. Neural Networks. 1991;4(2):251–257.
37. Hinton GE, Salakhutdinov RR. Reducing the dimensionality of data with neural networks. Science. 2006;313(5786):504–507. pmid:16873662
38. Rumelhart DE, Hinton GE, Williams RJ. Learning representations by back-propagating errors. Nature. 1986;323(9):533–536.
39. Mohamed Ar, Sainath TN, Dahl G, Ramabhadran B, Hinton GE, Picheny MA. Deep belief networks using discriminative features for phone recognition. In: Acoustics, Speech and Signal Processing (ICASSP), 2011 IEEE International Conference on; 2011. p. 5060–5063.
40. Graves A, rahman Mohamed A, Hinton G. Speech recognition with deep recurrent neural networks. In: IEEE International Conference on Acoustic Speech and Signal Processing (ICASSP); 2013. p. 6645–6649.
41. Krizhevsky A, Sutskever I, Hinton GE. Imagenet classification

with deep convolutional neural networks. In: Advances in Neural Information Processing Systems (NIPS); 2012. p. 1097–1105.

42. Hinton GE, Srivastava N, Krizhevsky A, Sutskever I, Salakhutdinov RR. Improving neural networks by preventing co-adaptation of feature detectors; 2012. ArXiv preprint, http://arxiv.org/abs/1207.0580.
43. Dahl GE, Sainath TN, Hinton GE. Improving deep neural networks for LVCSR using rectified linear units and dropout. In: IEEE International Conference on Acoustic Speech and Signal Processing (ICASSP); 2013. p. 8609–8613.
44. Kim JS, Greene MJ, Zlateski A, Lee K, Richardson M, Turaga SC, et al. Space-time wiring specificity supports direction selectivity in the retina. Nature. 2014;509(7500):331–336. pmid:24805243
45. Qi Y, Oja M, Weston J, Noble WS. A unified multitask architecture for predicting local protein properties. PLoS One. 2012;7(3):e32235. pmid:22461885
46. Leung MKK, Xiong HY, Lee LJ, Frey BJ. Deep learning of the tissue-regulated splicing code. Bioinformatics. 2014;30(12):i121–i129. pmid:24931975
47. Xiong HY, Alipanahi B, Lee LJ, Bretschneider H, Merico D, Ryan , et al. The human splicing code reveals new insights into the genetic determinants of disease. Science. 2015;347 (6218). pmid:25525159
48. Alipanahi B, Delong A, Weirauch MT, Frey BJ. Predicting the sequence specificities of DNA- and RNA-binding proteins by deep learning. Nature Biotechnology. 2015;33:831–838. pmid:26213851
49. Jiang B, Wu T, Zheng C, Wong WH. Learning Summary Statistic for Approximate Bayesian Computation via Deep Neural Network; 2015. ArXiv preprint: http://arxiv.org/abs/1510.02175.
50. Tavaré S, Balding DJ, Griffiths JRC, Donnelly P. Inferring Coalescence Times From DNA Sequence Data. Genetics. 1997;145:505–518. pmid:9071603
51. Aeschbacher S, Beaumont MA, Futschik A. A novel approach for choosing summary statistics in approximate Bayesian computation. Genetics. 2012;192(3):1027–1047. pmid:22960215

52. Nakagome S, Fukumizu K, Mano S. Kernel approximate Bayesian computation in population genetic inferences. Statistical Applications in Genetics and Molecular Biology. 2013;12(6):1–12.

53. Blum MGB, Nunes MA, Prangle D, Sisson SA. A comparative review of dimension reduction methods in approximate Bayesian computation. Statistical Science. 2013;28(2):189–208.

54. Wegmann D, Leuenberger C, Neuenschwander S, Excoffier L. ABCtoolbox: a versatile toolkit for approximate Bayesian computations. BMC Bioinformatics. 2010;11:116. pmid:20202215

55. Elyashiv E, Sattath S, Hu TT, Strustovsky A, McVicker G, Andolfatto P, et al. A genomic map of the effects of linked selection in *Drosophila*; 2014. ArXiv preprint, http://arxiv.org/abs/1408.5461.

56. Ewing G, Hermisson J. MSMS: A Coalescent Simulation Program Including Recombination, Demographic Structure, and Selection at a Single Locus. Bioinformatics. 2010;26:2064–2065. pmid:20591904

57. Haag-Liautard C, Dorris M, Maside X, Macaskill S, Halligan DL, Charlesworth B, et al. Direct estimation of per nucleotide and genomic deleterious mutation rates in *Drosophila*. Nature. 2007;445:82–85. pmid:17203060

58. Thornton K, Andolfatto P. Approximate Bayesian inference reveals evidence for a recent, severe bottleneck in a Netherlands population of *Drosophila melanogaster*. Genetics. 2006;172(3):1607–1619. pmid:16299396

59. Haddrill PR, Thornton KR, Charlesworth B, Andolfatto P. Multilocus patterns of nucleotide variability and the demographic and selection history of *Drosophila melanogaster* populations. Genome Research. 2005;15:790–799. pmid:15930491

60. Andolfatto P, Przeworski M. A Genome-Wide Departure From the Standard Neutral Model in Natural Populations of *Drosophila*. Genetics. 2000;156:257–268. pmid:10978290

61. Glaser-Schmitt A, Catalán A, Parsch J. Adaptive divergence of a transcriptional enhancer between populations of *Drosophila melanogaster*. Philos Trans R Soc Lond B Biol Sci. 2013;368(1632):20130024. pmid:24218636

62. Mueller JL, Ram KR, McGraw LA, Qazi MCB, Siggia ED, Clark AG, et al. Cross-Species Comparison of *Drosophila* Male Accessory Gland Protein Genes. Genetics. 2005;171(1):131–143. pmid:15944345

63. Liu PZ, Kaufman TC. *Kruppel* is a gap gene in the intermediate germband insect *Oncopeltus fasciatus* and is required for development of both blastoderm and germband-derived segments. Development. 2004;131:4567–4579. pmid:15342481

64. Rubinstein CD, Wolfner MF. Reproductive hacking: A male seminal protein acts through intact reproductive pathways in female *Drosophila*. Fly. 2014;8(2):80–85. pmid:25483253

65. Mohn F, Sienski G, Handler D, Brennecke J. The Rhino-Deadlock-Cutoff Complex Licenses Noncanonical Transcription of Dual-Strand piRNA Clusters in *Drosophila*. Cell. 2014;157(6):1364–1379. pmid:24906153

66. Zhou H, Hu S, Matveev R, Yu Q, Li J, Khaitovich P, et al. A Chronological Atlas of Natural Selection in the Human Genome during the Past Half-million Years; 2015. BioRxiv preprint, http://dx.doi.org/10.1101/018929.

67. Schrider DR, Mendes FK, Hahn MW, Kern AD. Soft Shoulders Ahead: Spurious Signatures of Soft and Partial Selective Sweeps Result from Linked Hard Sweeps. Genetics. 2015;200(1):267–284. pmid:25716978

68. Ng A, Ngiam J, Foo CY, Mai Y, Suen C. Unsupervised Feature Learning and Deep Learning Tutorial; 2013. http://ufldl.stanford.edu/wiki/index.php/UFLDL_Tutorial.

69. Dodier R. LBFGS optimization routine, Java translation; 1999. http://riso.sourceforge.net/.

70. Novembre J, Johnson T, Bryc K, Kutalik Z, Boyko AR, Auton A, et al. Genes mirror geography within Europe. Nature. 2008;456:98–101. pmid:18758442

71. Tajima F. Statistical method for testing the neutral mutation hypothesis by DNA polymorphism. Genetics. 1989;123(3):585–595. pmid:2513255

72. Garud NR, Messer PW, Buzbas EO, Petrov DA. Recent selective sweeps in North American *Drosophila melanogaster* show signatures of soft sweeps. PLoS Genetics. 2015;11(2):e1005004. pmid:25706129

CHAPTER 3

Deep Learning Automates the Quantitative Analysis of Individual Cells in Live-Cell Imaging Experiments

David A. Van Valen[1] , Takamasa Kudo[2] , Keara M. Lane[1] , Derek N. Macklin[1] , Nicolas T. Quach[1] , Mialy M. DeFelice[1] , Inbal Maayan[1] , Yu Tanouchi[1] , E uan A. Ashley[3,4], Markus W. Covert[1,2]

[1] Department of Bioengineering, Stanford University, Stanford, California, United States of America

[2] Department of Chemical and Systems Biology, Stanford University, Stanford, California, United States of America

[3] Department of Genetics, Stanford University, Stanford, California, United States of America

[4] Department of Cardiovascular Medicine, Stanford University, Stanford, California, United States of America

ABSTRACT

Live-cell imaging has opened an exciting window into the role cellular heterogeneity plays in dynamic, living systems. A major critical challenge for this class of experiments is the problem of image segmentation, or determining which parts of a microscope image correspond to which individual cells. Current approaches require many hours of manual curation and depend on approaches that are difficult

to share between labs. They are also unable to robustly segment the cytoplasms of mammalian cells. Here, we show that deep convolutional neural networks, a supervised machine learning method, can solve this challenge for multiple cell types across the domains of life. We demonstrate that this approach can robustly segment fluorescent images of cell nuclei as well as phase images of the cytoplasms of individual bacterial and mammalian cells from phase contrast images without the need for a fluorescent cytoplasmic marker. These networks also enable the simultaneous segmentation and identification of different mammalian cell types grown in co-culture. A quantitative comparison with prior methods demonstrates that convolutional neural networks have improved accuracy and lead to a significant reduction in curation time. We relay our experience in designing and optimizing deep convolutional neural networks for this task and outline several design rules that we found led to robust performance. We conclude that deep convolutional neural networks are an accurate method that require less curation time, are generalizable to a multiplicity of cell types, from bacteria to mammalian cells, and expand live-cell imaging capabilities to include multi-cell type systems.

INTRODUCTION

Live-cell imaging, in which living cells are imaged over a period of time using phase contrast and/or fluorescence microscopy, is a powerful method for interrogating living systems. This class of experiments has shed light on numerous biological problems, which include transcriptional regulation in both bacteria and eukaryotes as well as information transmission in mammalian signaling networks [1–8]. One common insight these single-cell measurements have been able to provide is the role cellular heterogeneity plays in these systems, as they have the ability to capture the differences between cells and observe how these differences evolve in time. This aspect of living systems is often missed in population level analyses and is best captured by live-cell imaging [9].

Central to the analysis for these experiments is the problem of image segmentation—the assignment of which pixels in an image belong to each individual cell [9]. Without this mapping, it is impossible to extract statistics on cellular fluorescence or geometry with single-cell resolution. The available solutions to date draw upon a standard set of

techniques—filtering, morphological operations, thresholding, and the watershed transform—from computer vision. Typically, a combination of these tools are tailored to each individual experiment [4, 10–13]. For example, CellProfiler and Oufti are tools to perform cell segmentation and tracking (primarily using thresholding, watershed transform, and Voronoi algorithms). Supervised machine learning methods have also seen significant success in this space [14–16]. Two notable examples are Ilastik and Microscopy Image Browser, two freely available programs that use supervised machine learning (edge and texture filters with random forest classification) to perform segmentation [12, 17–19].

Our lab's experience in performing all aspects live-cell imaging experiments—from data generation to analysis—have lead us to identify three main challenges in this space. These are curation time, segmentation accuracy, and solution sharing. The unfortunate reality is that the output of most programs built around the standard set of techniques described above requires significant amounts of manual curation. The specific time cost is rarely, if ever, reported, but work by our lab on NF-κ B signaling in mammalian cells, which required segmenting images of fluorescent nuclei, required on the order of 100+ hours of manual curation per manuscript [4, 5]. Similarly, recently-published work requiring segmentation of growing micro-colonies of *E. coli*required ~40 hours [20]. Much of this burden can be traced to inaccurate segmentation algorithms and the time required to separate accurately segmented cells from inaccurately segmented ones. The need for human curation is a significant drawback to these methods; not only are far fewer experiments performed than could be, but many types of experiments are never performed because the analysis is seen as prohibitive (co-culture, for example—see [5]).

The image analysis techniques mentioned above are also confounded by commonly-desired tasks, such as robust segmentation of mammalian cell cytoplasms, or bacterial cells in close proximity. Segmentation methods exist for the mammalian cytoplasm, but they typically require either imaging a cytoplasmic fluorescent protein (which removes a fluorescence channel) or imaging multiple focal planes (which increases acquisition time) [21–26]. Neither of these consequences are desirable. As a result, the cytoplasmic segmentation problem is frequently circumvented by sampling pixels in close proximity to the nucleus and using them as a proxy for the cytoplasm [27–29]. More progress has been

made in segmenting closely packed bacterial cells [17]; however, a robust method to identify the cytoplasm of mammalian cells or bacterial micro-colonies with single-cell resolution directly from phase microscopy images has remained elusive [17, 26, 30, 31].

Another challenge concerns generality, or the ability of existing solutions or software in one lab to be applied to the problems of another lab. Because different groups use highly-tuned combinations of these standard techniques to solve the image segmentation problem for specific experiments, there is a barrier to sharing work and ideas in this space. CellProfiler, Oufti, and Ilastik represent notable exceptions, and have empowered a number of experiments, including in labs which were otherwise new to computational image analysis [12, 17, 18]. However, the overall lack of sharable segmentation solutions means the cost of entering this field requires a significant—and often unanticipated—computational investment, beyond the obvious costs associated with the microscopy itself.

Recent advances in supervised machine learning, namely deep convolutional neural networks (referred to here as conv-nets) have shown remarkable performance for the task of image classification—that is, assigning descriptive labels to images [32, 33]. By "learning" on a set of manually annotated data, conv-nets can be used to classify new images with remarkable accuracy [33]. Prior work has shown that in addition to functioning as image classifiers, conv-nets can also perform semantic segmentation—the assignment of class labels to each individual pixel of an image rather than to the whole image itself—in a computationally efficient manner [34–36]. Despite their successful application to numerous real-world problems, conv-nets have only recently been applied to analyzing biological data. Recent work has demonstrated that they have significant potential to address the needs of live cell imaging experiments. Work by Ronnenberg et. al. won the 2015 ISBI cell tracking competition, and was the first to demonstrate that conv-nets can segment images of U373 (with almost perfect accuracy) and HeLa cells from DIC microscopy images [37]. Another recent paper addressed the problem of segmenting fluorescent images of yeast cells [37–40]. Much like Krizhevsky's initial application of conv-nets to the ImageNet image classification challenge in 2012, this work strongly suggests that conv-nets are a natural technology for live-cell imaging

experiments [33]. While these recent developments have the potential to be revolutionary, our experience both as experimentalists and as users of deep learning demonstrated that there was still work to be done before conv-nets can supplant prior image segmentation methods in this space. In this work, we build on these exciting advances to demonstrate that conv-nets address all of the image analysis problems described above for a large variety of cell types. Our aim was to leverage our lab's experience in live-cell imaging and deep learning to develop a robust tool that could be applied to a variety of live-cell imaging experiments.

Our paper is organized as follows. We first review how image segmentation can be thought of as image classification as well as the mathematical structure of conv-nets. We show that conv-nets can accurately segment the cytoplasms of bacterial cells and mammalian cell nuclei from fluorescent microscopy images. We show that integrating phase microscopy images with fluorescent microscopy images of a nuclear marker enables the accurate segmentation of the cytoplasm of mammalian cells. We show that by integrating conv-nets into an image analysis pipeline, we can quantify the growth of thousands of bacterial cells and track individual mammalian nuclei with almost no manual correction. We also show that incorporation of cytoplasmic segmentation masks provides a more accurate quantification of fluorescent protein localization kinase translocation reporters (KTRs) [6]. A quantitative comparison demonstrates that conv-nets are superior to other methods, both in terms of accuracy and curation time. We show how this approach can be used to both segment and classify different mammalian cell types in a co-culture using just the phase image and a nuclear marker. We highlight particular features of our work—image normalization, segmentation refinement with active contours, and receptive field size—which were critical for conv-nets to perform robustly on live-cell imaging data using relatively small training data sets. We also explore how much recent deep learning advances—namely dropout, batch normalization, and multi-resolution networks—impact segmentation performance [37, 41, 42]. All of the software and training data described here, as well as a Docker container, is available at https://simtk.org/projects/deepcell.

Image Segmentation as Image Classification

Image segmentation of single cells in microscopy images can be

converted into an image classification problem [38]. Consider a manually annotated image where each pixel has been identified as either a cell boundary, cellular interior, or background (non-cell) pixel, as depicted in Fig 1a. By sampling a small region around each pixel and assigning the resulting image that pixel's respective class, we can construct a training data set that contains representative images of each class. After the manually annotated image is reconstructed in this way, the image segmentation task is effectively reduced to finding a classifier that can distinguish between the three classes in the training data set and is robust enough to classify new images. Should such a classifier exist, any new microscope image could be segmented by decomposing it into small overlapping images, applying the classifier to each image, and then reassembling the classification prediction into a new image. This image can then be subjected to standard computer vision techniques to produce a segmentation mask—an image where the pixels for each cell's interior have been assigned a unique integer label.

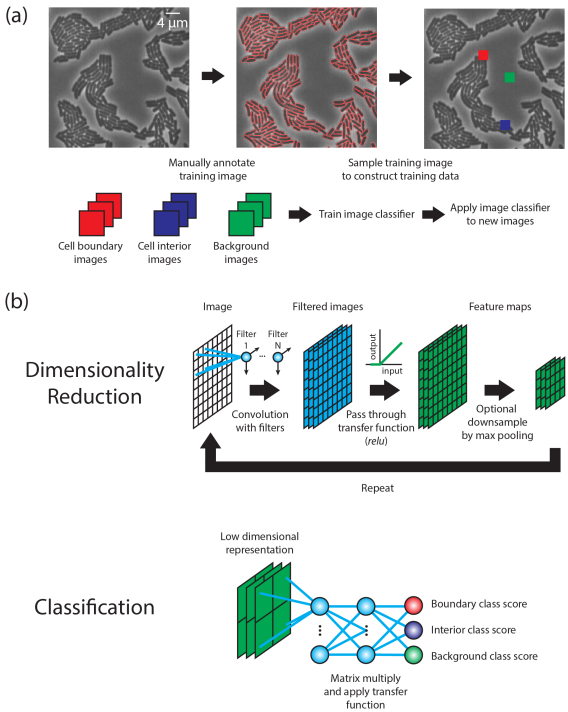


Figure 1: Performing image segmentation with deep convolutional neural networks.

(a) Image segmentation can be recast as an image classification task that is amenable to a supervised machine learning approach. A manually annotated image is converted into a training dataset by sampling regions around boundary, interior, and background pixels. These sample images are then used to train an image classifier that can then be applied to new images. (b) The mathematical structure of a conv-net. A conv-net can be broken down into two components. The first component is dimensionality reduction through the iterative application of three operations—convolutions, a transfer function, and down sampling. The second component is a classifier that uses the representation and outputs scores for each class.

Conv-nets can function as exactly such a classifier for data acquired from live-cell imaging experiments because they have both substantial representational power to encode the nonlinear relationship between images and classes, yet are general enough to provide robust predictions for new images [32, 33].

Here, we briefly review the mathematical structure of conv-nets and how they are trained to perform classification tasks [32, 38]. With respect to their mathematical structure, conv-nets can be thought of as having two components; this is schematized in Fig 1b. The first component constructs a low dimensional representation of an image using the iterative application of three operations, as depicted in Fig 1b. If we denote the input image as I, then the first operation is to convolve the image with a set of filters, denoted here as $\{w_1,\ldots, w_n\}$, to produce the filtered images $\{I * w_1,\ldots, I * w_n\}$. Each filter w_i has the same shape—a square with a size of 3–5 pixels—in the conv-nets considered here. The filters can be thought of as local feature extractors as their output only depends on pixels in close spatial proximity. The weights of these filters are variables—their values are changed during a training procedure to minimize the classification error on a training data set.

Once the image is filtered, the second operation is to apply a transfer function f to produce a set of feature maps $\{f(I * w_1),\ldots, f(I * w_n)\}$. The transfer function serves to enable the classifier to construct non-linear decision boundaries. In this work, we use the rectified linear unit, or relu, as our transfer function which is defined as $relu(x) = \max(0, x)$ [43]. The third operation is an optional down-sampling using max-pooling, which creates feature maps at a coarser spatial scale. The max-pooling operation

replaces an $m \times m$ window with the largest pixel value within that window. The window is then slid across the image with a particular step size to produce the down sampled image. In this work, we only consider max-pooling with a 2 ×2 window using a step size of 2. The output of these dimensionality reduction steps are a collection of down sampled (if max pooling was applied) feature maps $\{pool(f(I * w_1)),\ldots, pool(f(I * w_n))\}$. These feature maps can be viewed as a single multi-channel image, which then serves as the input to another round of the same three operations—convolution with a set of filters, application of a transfer function, and optional down-sampling by max pooling. By iterating these three operations for several rounds (4–6 times in this work), we construct a low dimensional representation of the input image.

The second component of a conv-net, classification, uses a fully connected neural network that takes the low dimensional representation described above an input and assigns class labels. The mathematical structure is similar to a linear classifier, which collapses the low dimensional representation of the image into a single vector **r** and then multiplies it by a matrix W. The dimensions of W are (number of classes X number of elements in r), which by design produces the appropriate number of class scores. The output is a vector of class scores, with the highest score being called as the winner.

The difference between the linear classifier and a fully connected neural network is that the latter uses iterative applications of matrix multiplication and a transfer function to construct a nonlinear mapping between the low dimensional representation and the set of class labels. Put concretely, the first step is to collapse the low-dimensional representation into a single vector **r**and then multiply with a matrix W_1. The product is then sent through the transfer function f to yield the output $f(W_1\boldsymbol{r})$. The repeated application of these two operations produce the output $W_q f(W_{q-1} f(\ldots f(W_1\boldsymbol{r})))$, where the number of rows of the last matrix W_q is equal to the number of classes. This output is a vector of class scores which serves as the classification prediction. For the networks considered in this work, we take $q = 3$ or 4.

With the mathematical structure of a conv-net defined, we now to turn to training. As with other supervised machine learning methods, conv-nets have to learn from manually annotated datasets to perform the classification task, such as the classifying of boundary, interior, and

background images mentioned earlier. We refer to this dataset as our "training data" because it is used to train the conv-net as a classifier. A small portion (10%) of this dataset is set aside as "validation data" and is used to benchmark the training procedure. In the case of conv-nets, training means tuning the weights of the filters w_i and matrices W_j to minimize the classification error. This is done by first constructing a cost function that provides a reward for predicting the correct class and a penalty for predicting the wrong class for a set of images. In this work, we use soft-max cost function, which is defined as

$$Cost = -\sum_{all\ images} \log\left(\frac{e^{correct\ class\ score}}{\sum_{all\ classes} e^{class\ score}}\right) + \lambda \sum_{w \in \{all\ filter\ weights\}} w^2.$$

The first term can be thought of as the negative log likelihood of choosing the correct class while the second is a regularization term included to reduce overfitting. With the cost function defined, batch stochastic gradient descent (or a variant like RMSprop) was used to tune the filter weights (after a random initialization) to minimize the cost function [44, 45]. Batch stochastic gradient descent works by computing the derivative of the cost function with respect to each weight for a batch of images. The cost function is minimized by adjusting the weights according to the formula

$$w = w - learning\ rate\ \times \frac{\partial Cost}{\partial w}.$$

The gradient of the cost with respect to the weights is computed analytically using the backpropagation algorithm [46]. We implemented training and execution of conv-nets in Python using the Keras, Theano, Numpy, Scipy, and Scikit-image Python packages[47–51]. Additional details about training are provided in the supplemental information.

RESULTS

Conv-nets accurately segment bacterial and mammalian cells

We trained conv-nets to perform 3 different segmentation tasks—bacterial cytoplasms (from phase microscopy images), mammalian cell

nuclei (from fluorescent microscopy images of cells expressing a nuclear marker), and mammalian cell cytoplasms (from phase microscopy images and images of a fluorescent nuclear marker). For the mammalian cell cytoplasm segmentation task, we opted to allow the conv-net to see both the phase image and an image of a nuclear marker. We did this for three reasons. First, most live-cell imaging experiments of mammalian cells collect images of a nuclear marker for cell tracking purposes, so we expect that data to typically be present. Second, we reasoned that because the nuclei and the cytoplasm overlap spatially, the nuclear channel might help when the phase image is ambiguous. Third, because the nuclei are typically separated spatially, we reasoned that the nuclear channel could be used to separate cells that are in close spatial proximity—either inherently by the conv-net or through a post processing step.

Once trained, a new image can be run through the conv-net to produce a prediction of which pixels are background, cellular boundary, or cellular interior. As opposed to processing a new image patch-by-patch, d-regularly sparse kernels can be used to process images in a fully convolutional, computationally efficient fashion [35, 36]. These predictions are images with pixel values ranging from 0 to 1 to reflect how certain the conv-net is of its pixel level classification. For live-cell imaging experiments, this prediction must be turned into a binary mask—an image with a 1 at every cellular interior pixel and a 0 everywhere else. We've found that this necessitated an additional downstream processing step. For bacterial cells and mammalian nuclei, thresholding was sufficient to produce accurate segmentation masks. For the mammalian cytoplasm, we first used a conv-net to segment the nuclei, and then used the nuclear prediction to seed a refinement of the cellular interior prediction with active contours [52].

Sample segmentation results are shown in Fig 2. To date we have trained and tested conv-nets to segment images of the following cell types/lines—*E. coli*, NIH-3T3, MCF10A, HeLa-S3, RAW 264.7, and bone marrow derived macrophages (BMDMs). To benchmark the segmentation results, we computed standard (Jaccard and Dice) indices of segmentation accuracy [53] (Table 1). For each cell, given a reference segmentation R and computed segmentation S, the Jaccard index is

defined as $JI = \frac{|R \cap S|}{|R \cup S|}$ and the Dice index is defined as $DI = \frac{2|R \cap S|}{|R|+|S|}$. The conv-net based approach has an accuracy that is either on par with or superior to the current methods developed to segment phase and fluorescent images of cellular cytoplasms and nuclei. The results in Table 1 also suggest two significant advantages of conv-nets over other approaches. First, the approach is general. Although the specifics of the architecture may need to be fine-tuned for each cell type, they have the capacity to segment images of multiple different cell lines. Second, because of their improved accuracy, less time is required to manually curate the segmentation results.

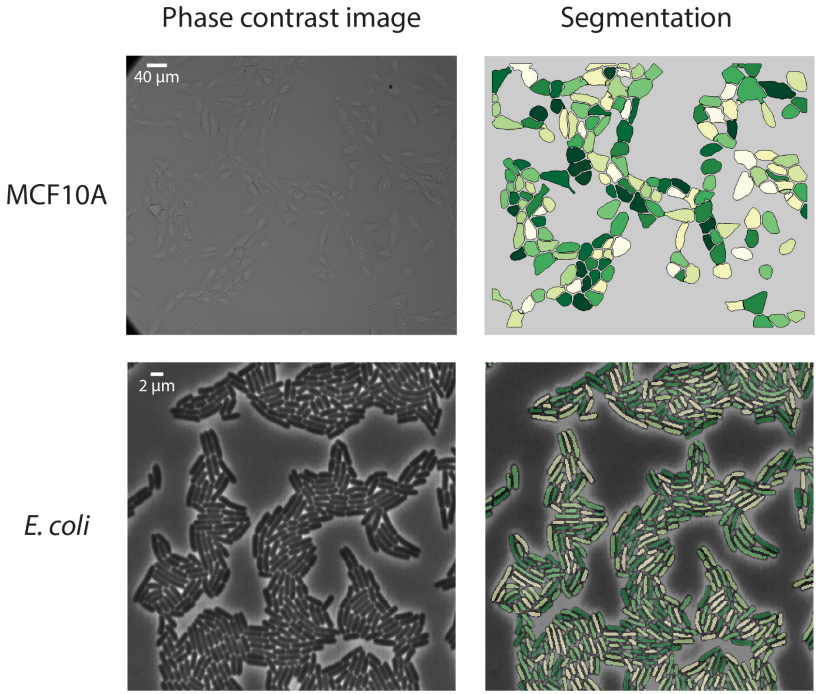


Figure 2: Sample images from live-cell experiments that were segmented using conv-nets.

Images of bacterial and mammalian cells were segmented using trained conv-nets and additional downstream processing. Thresholding for bacterial cells and an active contour based approach for mammalian cells were used to convert the conv-net prediction into a segmentation mask.

Table 1: Comparison of segmentation performance among different segmentation algorithms and datasets

Cell type(s)	Method	Imaging type	JI	DI	Dense	Processing time per image	Curation time per image	Citation
Bacteria								
Coliform bacteria	Edge finding and thresholding	Phase	~0.94		Yes	5 s	~10–30 s	[15,14]
Coliform bacteria	Conv-nets	Phase	~0.95		Yes	120 s	1–2 s	This work
Mammalian nuclei								
	Thresholding and level sets	Fluorescence	0.88		Yes	2–3 s	~ 5–10 s	[45]
	Conv-nets and thresholding	Fluorescence	0.89	0.94	Yes	10 s	~0–1 s	This work
Mammalian cytoplasm								
N1E115	Voronoi diagrams	Fluorescence	0.75		No	40 s	NA	[46]
Neural stem cells	Level sets	Bright field	0.79		No	NA	NA	[47]
NIH-3T3	Edge finding, thresholding, and morphological operations	Phase		0.76–0.92	No	NA	NA	[21]
HeLa-S3	Ilastik (texture/edge filters and random forest classifier)	Phase	0.56*	0.71*	Yes	1s	NA	[18], This work
HeLa-S3	Conv-nets	DIC	0.77		Yes	1s	NA	[37]
U373	Conv-nets	DIC	0.93		No	1s	NA	[37]
MCF10A	Conv-nets and active contours	Phase	0.77	0.86	Yes	75 s	~ 2–10 s	This work
NIH-3T3	Conv-nets and active contours	Phase	0.77	0.86	Yes	75 s	~ 2–10 s	This work
HeLa-S3	Conv-nets and active contours	Phase	0.71**-0.84	0.83**-0.92	Yes	75 s	~ 2–10 s	This work

*: See the supplemental information and S17 Fig for more details.

**: For fully confluent cells. See the supplemental information and S16 Fig for more details.

doi:10.1371/journal.pcbi.1005177.t001

The Jaccard (JI) and Dice (DI) indices were computed on a validation data set as previously described [53]. The Jaccard index for *E. coli* was estimated from the segmentation error rate (0.5% in this work and 2.6% in [17]) assuming each segmentation error was due to the incorrect joining of 2 bacteria. NA—Not available.

As a second validation test, we examined time lapse images of *E. coli* and HeLa cells to determine whether segmentation results were significantly different between images known to be closely related in time. This test also had the advantage of being particularly relevant to the most pressing application of our method. A linear assignment problem-based approach was used to track cells from frame to frame [54]. To estimate how many *E. coli* cells showed high variance in segmentation over time, we counted how many cells had a 30% or larger area increase from frame to frame—such a large increase typically means that two bacteria have been incorrectly segmented as a single cell. We found that 69 of the 11909 cells (across all frames) analyzed in Fig 3 met this criterion, which leads to an error rate of ~0.6%. This number is superior to the best results from standard approaches by four-fold (2.6%) [17]. For HeLa cells, we performed a similar analysis, but in this case comparing the segmentation contour for each cell to the actual phase image over time. We visually

inspected a segmented movie, and found that 21 of 145 cells tracked over time exhibited significant changes in segmentation. This leads to an error rate of 14%, which is sufficient to analyze most cells within a field of view, and represents a significant step forward in comparison to prior methods. We conclude that conv-nets are comparably accurate for a variety of cell types with a significantly reduced curation requirement.

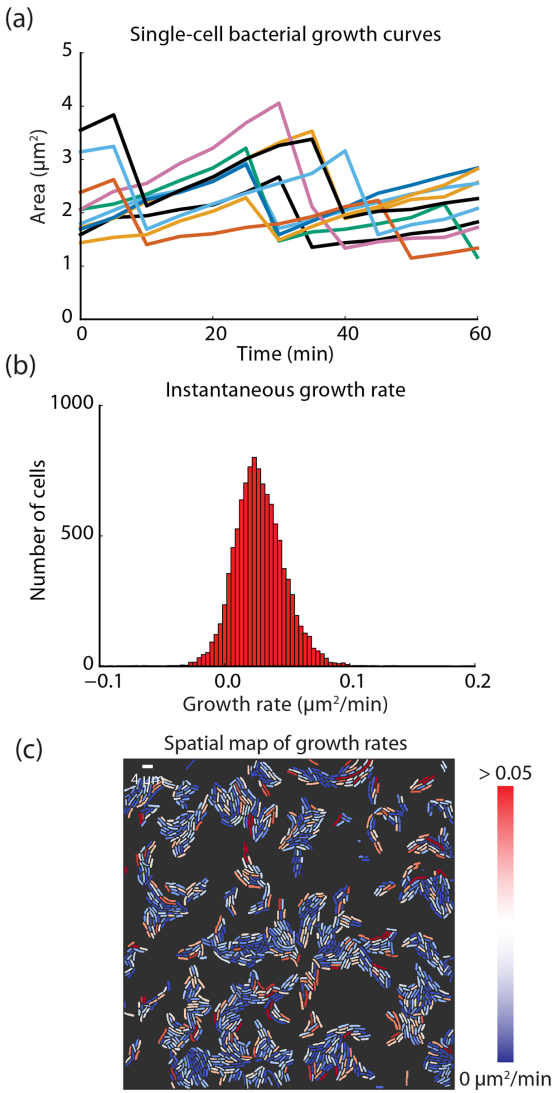


Figure 3: Extracting dynamic measurements of live-cell imaging experiments using conv-nets.

(a) Single-cell growth curves for *E. coli*. Because conv-nets allow for the robust segmentation of bacterial cells, we can construct single-cell growth curves from movies of growing bacterial micro-colonies. A linear assignment problem based method was used for lineage construction. (b) By computing the change in area from frame to frame for each cell, we can construct a histogram of the instantaneous growth rate. (c) Using the instantaneous growth rate and the segmentation masks, we can construct a spatial map of growth rates with single-cell resolution. Such a map allows rapid identification of slowly dividing cells (such as metabolically inactive cells).

Conv-nets automate the quantification of live-cell imaging experiments

Having established and validated the use of conv-nets for segmentation of live-cell images, we decided to apply them to pressing needs in our field: first, constructing single-cell growth curves for bacteria; and second, quantifying localization-based kinase reporters in mammalian cells. The analysis of single-cell growth curves has recently produced a number of insights into the mechanism of bacterial growth, including the identification of the adder model for bacterial growth and the observation of metabolic co-dependence within biofilms [20, 55–58]. The images for single-cell growth curves are typically generated by growing bacteria in a patterned microfluidic device, although agar pads and other microfluidic devices have also been used [59–61]. The generation of these growth curves has typically been difficult because of the close proximity of the growing cells, although recent efforts using standard approaches have shown some success [11, 17]. By using our conv-net based approach and a linear assignment problem method for lineage construction, we can construct single-cell growth curves for *E. coli*–even when they are growing in a micro-colony—and use them to compute the instantaneous growth rate for each cell. These are shown in Fig 3a, 3b. Because our approach can robustly produce accurate segmentation masks, it can be used to automate the analysis of thousands of bacterial cells regardless of their proximity. Using the instantaneous growth rate, we can create a spatial heat map of the micro-colony that is colored by the growth rate with single-cell resolution. This is shown in Fig 3c. We note that all of the panels in Fig 3 were generated without manual correction.

A second application of our method is the quantification of localization-based kinase reporters. These reporters are a recent advance that enable the monitoring of kinase activities inside individual mammalian cells [6]. These reporters consist of a kinase recruitment domain tethered to a nuclear localization signal, nuclear export signal, and a fluorescent protein. Activation of the associated kinase leads to phosphorylation of the reporter and a nucleocytoplasmic translocation event. Because the reporter is fused to a fluorescent protein, this event results in a change in fluorescence localization. Due to the complexity of segmenting mammalian cytoplasms, we had originally taken a 5-pixel wide ring around each nucleus as a proxy for the entire cytoplasm, as has been done in other studies [27–29]. With an improved ability to sample the cytoplasm pixels, we reasoned that our conv-net based approach would lead to more accurate quantification of KTRs. To test this, we acquired movies of HeLa-S3 cells expressing the JNK-KTR after they were stimulated with TNF-α and segmented them using our conv-net approach. For this analysis, we segmented both the nuclei and the cytoplasm using conv-nets. A montage of cells from this data set is shown in Fig 4a.

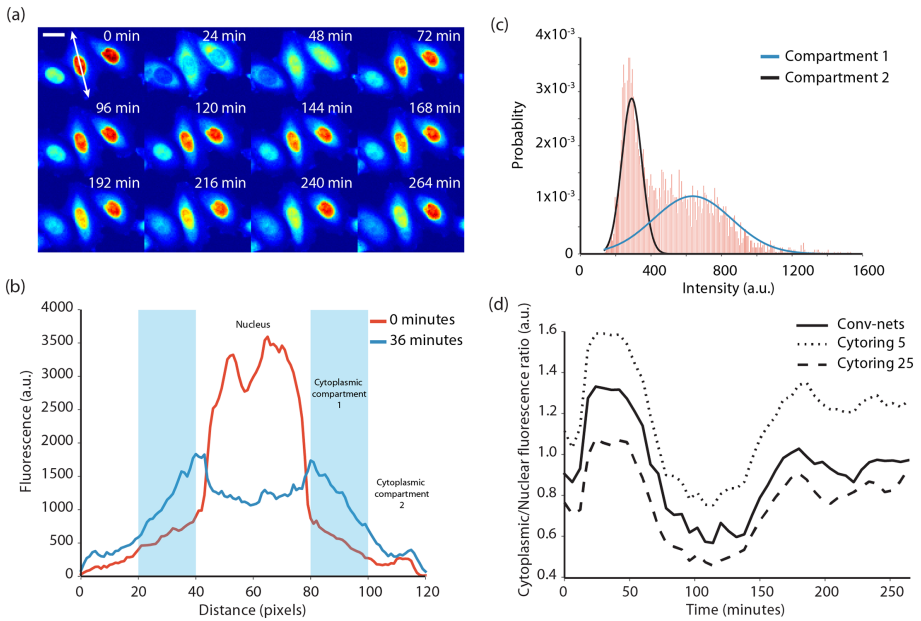


Figure 4: Analysis of JNK-KTR dynamics in single cells.

(a) A montage of HeLa-S3 cells expressing a JNK-KTR after stimulation

with TNF-α. The scale bar is 20 μm. (b) A line profile of the fluorescence of the cell highlighted in (a) which demonstrates that there is considerable spatial heterogeneity of the fluorescence in the cytoplasm. We model the cytoplasm as having two compartments, only one of which receives fluorescence from the nucleus during translocation. (c) A fit of a two component Gaussian mixture model to the cytoplasmic fluorescence of a HeLa-S3 cell. This method allows us to accurately estimate the fluorescence inside the cytoplasmic compartment that communicates with the nucleus. (d) Dynamics of the JNK-KTR after stimulation with TNF-α after segmentation with our conv-net based approach and quantification with the two component Gaussian mixture model. Plotted in comparison are dynamics obtained from using cytorings with radii of 5 pixels and 25 pixels.

To identify the most robust way to quantify the nuclear-to-cytoplasm ratio, we first visually inspected our movies. This inspection revealed that the cytoplasmic fluorescence is not uniform—rather the fluorescence is highest close to the nucleus and is much lower at the periphery (Fig 4a and 4b). Furthermore, as evidenced by Fig 4a and 4b, the fluorescence transfer from the nucleus is spatially restricted when it enters the cytoplasm. The HeLa-S3 cytoplasm can therefore be thought of as having two compartments, only one of which receives fluorescence from the nucleus. To see how this experimental reality influences the quantitation of the data, we varied the radius of the cytoring to see how its size influences the extracted dynamics (Fig 4d). We found that as we increased the radius, the qualitative shape of each curve was intact but there were important quantitative differences. These differences are important when using models to infer the concentration of active kinase inside a cell. The smallest cytoring almost exclusively samples the compartment in communication with the nucleus but also has the fewest number of pixels. The larger cytorings sample this compartment more thoroughly but also incorporate pixels that are not in the correct compartment. We therefore expect the cytoring to give qualitatively correct but quantitatively inaccurate results regardless of the radius. Our conv-net based approach improves on this method because it better samples the cytoplasmic pixels. To determine the average fluorescence of the compartment that is in communication with the nucleus, we fit the pixel intensity histograms of the cytoplasm of each cell to a 2-component

Gaussian mixture model as shown in Fig 4c. The resulting dynamics for this approach and comparison to the cytoring method are shown in Fig 4d. We feel this method is superior because it properly incorporates information from the entire cytoplasm whereas the cytoring forces one to make a decision between accuracy and sampling sparsity. We note that this method for quantifying KTRs may not be necessary for all cell types, as NIH-3T3 cells did not appear to have a two-compartment cytoplasm [6].

Conv-nets enable the identification of cell type in mammalian cell co-cultures

The ability to quantify fluorescence microscopy data is a powerful application of deep learning that will lead to more accurate quantification of live-cell imaging experiments. However, conv-nets also have the ability to perform semantic segmentation—that is, to both segment individual cells and also predict their cell type [35, 62]. We explored how well conv-nets perform semantic segmentation in this context by training a conv-net on a data set that contained images of both NIH-3T3 and MCF10A cells, which exhibit distinct morphologies under phase contrast. We modified the conv-net architecture by increasing the number of classes from 3 to 4 so that the conv-net could recognize the difference between the interiors of the two cell types. To generate a prediction for each cell, the trained conv-net was first applied to an image of a co-culture. We then used the sum of the NIH-3T3 and MCF10A normalized score images to generate segmentation masks as described above. We then computed the cellular classification score for each class (NIH-3T3 or MCF10A), which is given by

$$\textit{cellular classification score} = \frac{\sum_{i \in \{\textit{cell pixels}\}} \textit{class score for pixel } i}{\sum_{\textit{classes}} \sum_{i \in \{\textit{cell pixels}\}} \textit{class score for pixel } i}.$$

The results for semantic-segmentation are shown in Fig 5a and 5b. When compared with the ground truth (provided by distinct nuclear markers for the two cell types), our trained conv-net has an accuracy of 95% when it comes to a cellular level prediction of cell type. One image containing 92 NIH-3T3 cells and 182 MCF10A cells was classified to

determine the accuracy of cell type classification. Similar results were obtained for additional images.

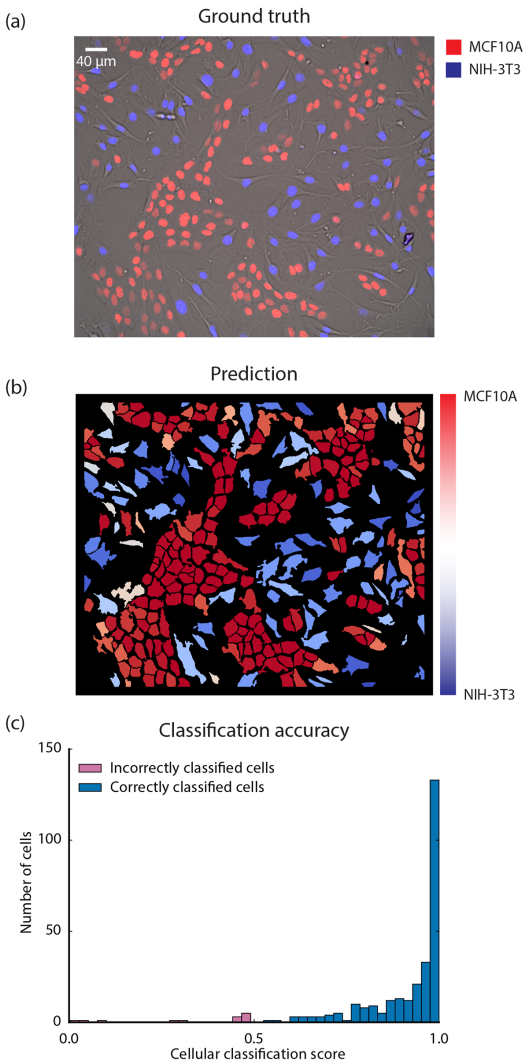


Figure 5: Semantic segmentation of a co-culture with MCF10A and NIH-3T3 cells.

A conv-net was trained to both segment and recognize NIH-3T3 and MCF10A cells. The training data was created from separate images of NIH-3T3 and MCF10A cells with each image having a nuclear marker (Hoechst 33342) as a separate channel. (a) A ground truth image of a

co-culture containing NIH-3T3 and MCF10A cells. The NIH-3T3 cells express a mCerulean nuclear marker (blue) while the MCF10A cells express an iRFP nuclear marker (red). Hoechst 33342 (image not shown) was also used to generate an image of a nuclear marker. (b) Simultaneous image segmentation and cell-type classification of the image in (a) using a trained conv-net. (c) Classification accuracy for the trained conv-net's cellular level cell type prediction. The cellular classification score of the correct cell type for each cell is plotted as a histogram. The size of the cellular classification score is strongly associated with making a correct prediction. A classification accuracy of 86% was achieved for NIH-3T3 cells and 100% for MCF10A cells.

We were curious whether stronger cellular classification scores were less prone to classification errors. To explore this, we constructed a histogram of the cellular classification scores of the correct class for each cell (Fig 5c). We found that classification errors are clustered around 0.5, with only 3 cells with errors having a value less than 0.25. This observation leads us to conclude that the closer a cellular classification score is to 1, the more likely that the prediction is correct.

DISCUSSION

Advantages of conv-nets over standard computer vision techniques

Image segmentation is the core task required to convert the data from live-cell imaging experiments into a quantitative, dynamic description of living systems with single-cell resolution. Here, we have shown that deep learning can perform this task with accuracy sufficient for live-cell experiments. The approach to single-cell segmentation we have outlined has four main advantages. First, incorporating conv-nets into an image segmentation pipeline significantly reduces the time required for manual supervision and correction. This improvement comes about from the improved accuracy of the segmentation masks. In the two case studies presented here, we were able to obtain our end result with no manual curation. We note that this ability depends on data quality and likely will not hold for all experiments. To get a sense of the actual time savings, we estimated the curation time required for our previous work and searched

the literature to obtain estimates of how long it takes to analyze a typical experiment [4, 5, 9]. We found that, including parameter tuning, for nuclear segmentation we typically spend 6–12 hours analyzing a typical data set (96 positions x 45 frames) using a watershed based segmentation approach. Analyzing the segmentation results for bacterial cell division to produce the results in [15] required approximately 40 hours. Similarly, manually segmenting the cytoplasms of 3T3 cells in ~200 images required over 200 hours [26]. Even in the cases where the computational cost is significant, conv-nets invariably lead to significant savings in human time. Moreover, human time and computation time are not equivalent—a 20 hour task may take a human nearly a week (given scheduling demands and ergonomic requirements) while a computer can complete it within a day (or less if parallelization is used).

The second advantage is the low time and hardware requirements for training and testing new conv-nets. We have found that for the training data we generate, it typically takes 2–4 hours to generate training data (human time) and 5–10 hours (computer time) to train the conv-net architectures considered here. Both of these represent a fixed initial cost which apply to all subsequent experiments for a given cell type or types. In addition, because a capable GPU is the key piece of hardware required for this work, it is relatively inexpensive (~500–1000 US dollars) to provide existing workstations with the capacity to train and execute conv-nets.

Third is the ability to perform semantic segmentation to identify different cell types without sacrificing an additional fluorescence channel. This ability enables the analysis of live-cell imaging experiments involving co-cultures, monitoring differentiation, or possibly even tissues.

Fourth, and most importantly, is the generality of the approach. Using our framework, segmenting new cell types involves simply creating a new training data set and training a new conv-net. To date we have been able to segment images from 5 different mammalian cell lines with no changes in the conv-net architecture. While the architecture may need refinement, deep learning will likely be applied to even more cell types and enable the analysis of a variety of live-cell imaging experiments.

Design rules for new conv-nets

Moving forward, we foresee two possible approaches. In one approach, similar to the ImageNet challenge, one large network would be trained to segment and classify the majority of cell lines that are in use today [33, 35]. In another approach, which we consider to be more likely, a collection of conv-nets would be designed for each specific segmentation task required for a live-cell imaging experiment. This approach we term laboratory scale deep learning. By making the trained conv-nets and the requisite training data publically available, laboratories performing this type of work would be able to build on each other's technical advances. In pursuit of this aim, here we outline some of the design rules we've extracted from our successes (and failures) in training conv-nets.

First, we found that image normalization was critically important for robustness, segmentation performance, and processing speed. We tried three different normalization schemes—normalizing images by the median pixel value, by the maximum pixel value, or not at all. We found that normalizing by the median pixel value was best with respect to robust performance on images acquired with different illumination intensities. In the case of semantic segmentation, we found that without proper normalization, the conv-nets learned intensity differences between the MCF10A and NIH-3T3 data sets instead of differences in cell morphology. This led to poor performance when the conv-nets were applied to new images, as they classified all the cells as the same cell-type based on the images illumination. Furthermore, the normalization choice impacts whether or not a fully convolutional implementation of conv-nets can be used when segmenting new images. Normalizing patches by their standard deviation, for instance, or performing PCA based whitening would require new images to be processed in a patch-by-patch manner. We note that in our experience, performance was optimal when the illumination intensity was reasonably similar to or higher than the illumination used to acquire the training data.

Second is the importance of data augmentation. One of our key results is that only a few hundred cells worth of training data is required to produce a well performing conv-net. This is in contrast to the thousands of images contained in the ImageNet and PASCAL databases. Smaller training datasets were sufficient for out purposes because we made maximal use of the data available by using data augmentation

through image rotation and reflection and by enabling the conv-net to "see" the nuclear channel—information that is collected in every live-cell imaging experiment done in our lab. Curiously, we found that using image shearing for data augmentation did not significantly improve the segmentation performance. While additional data augmentation techniques may improve performance, it is clear that the large numbers of images typically used to train conv-nets are not required for this purpose. Nevertheless, we anticipate that the incorporation of additional training data will lead to more accurate and robust conv-nets in the future.

Third, is the importance of hyper-parameters. The most important hyper-parameter in our experience was the conv-net's receptive field size, or the size of the image patches sampled from the training image. We found that this was a key hyper-parameter that had to be tuned to obtain a well-performing conv-net. Our experience leads us to believe that the optimal size is roughly on the order of the diameter of a cell. Receptive fields that are too small led to conv-nets that classified the background as edges while receptive fields that were too large took longer to train and are more prone to over-fitting. While we obtained good results given our choice of hyper-parameters (for instance the regularization strength), we note that optimization of the number of layers in a conv-net, the number of filters in each layer, the filter sizes for each layer, or the conv-net architecture may also lead to further improvements [35, 37, 63].

Fourth, we found that segmentation refinement was necessary to produce segmentation masks that could be applied to real data. While conv-nets produce remarkably accurate pixel-wise classification predictions, we found we still needed to refine these predictions to produce binary segmentation masks. In the end, the method we found that worked the best was using the conv-net predictions to guide active contours. One advantage of this novel approach was that we could seed this process using the segmentation masks of the cell nuclei, in essence using the cell nuclei to separate the masks of spatially proximate cells. While it is possible that additional training data and hyper-parameter optimization may make this refinement step unnecessary, this is likely an area for further improvement. For instance, conditional random fields have shown success in refining conv-net predictions for semantic segmentation and may prove to be of some use here [62]. Incorporating segmentation and tracking together in a unified framework may also provide opportunities

for performance improvement [17].

Fifth, we explored how much dropout, batch normalization, and multi-resolution fully connected layers, improve the segmentation performance of conv-nets. Dropout is a regularization technique in which filters are randomly turned off during training [41]. By training in this fashion, the conv-net learns to not depend too heavily on any one filter. Batch normalization is a solution to the covariate shift problem that works by using per batch statistics to convert the output of each conv-net layer into one that has zero mean and unit variance [42]. Batch normalization also functions as a regularization method. Multi-resolution fully connected layers are an advance geared towards improving semantic segmentation performance [34, 35, 37]. By allowing hidden layers to see multiple spatial scales, these networks can boost their segmentation performance. We added these features to conv-nets while keeping most of their features fixed, trained them on our HeLa-S3 training data, and then evaluated their performance (in conjunction with active contours) on a validation data set. We found that performance for most of these variations were similar. Curiously, conv-nets with dropout and multi-resolution fully connected layers had worse performance than an unmodified conv-net. Batch normalization in conjunction with multi-resolution layers produced the best performance, but these conv-nets were prone to overfitting (as judged by the difference between the training and validation classification error). The networks presented here use batch-normalization, which we opted for because of their regularization ability, their reduced training time, and a desire to avoid overfitting.

Conclusions

Deep learning has been a transformative technology for computer vision, and based on our experiences outlined in this paper, we expect that it will be a powerful tool for analyzing images from live-cell imaging experiments. We offer several thoughts in conclusion. First, we expect that laboratory scale deep learning (as described above) will the development model moving forward. Second, we expect that the availability of open source deep learning APIs, such as Keras (as used in this work), Theano, Torch, Caffe, and TensorFlow, in addition to the myriad of tutorials available online, will help make this technology more accessible to the causal user. Finally, we expect that in addition to being able to better

quantify fluorescence levels, there are other classes of live-cell imaging experiments where the localization of fluorescence must be quantified that will benefit from the approach outlined here [27, 64–66]. Given our experience detailed in this work, we expect deep learning to have a transformative impact on the analysis of live-cell imaging experiments.

MATERIALS AND METHODS

Constructing training datasets

To construct manually annotated datasets for *E. coli* strain MG1655 and 5 different mammalian cell lines (NIH-3T3, MCF10A, HeLa-S3, RAW 264.7, and bone marrow derived macrophages), we first acquired images to construct training datasets. For bacterial cells we acquired phase images of the bacterial micro-colonies while for mammalian cells, we acquired both fluorescent images of labeled nuclei and phase images of the cytoplasm. We then used ImageJ to annotate each training dataset and classify each pixel in a training image as either boundary, cellular interior, or background. Details of this annotation procedure are contained in the supplemental information. Empirically, we found that annotating one microscope image containing a total of ~300 bacteria, ~500 nuclei, or ~ 100 mammalian cells was sufficient to make a training dataset that led to a well performing conv-net; the annotation typically required ~2–4 hours. A small patch (31x31 pixels for bacteria, and 61x61 pixels for mammalian cytoplasm and nuclei) was sampled around each annotated pixel to create a collection of representative images for each class. We found that this sampling was best done at run time to preserve GPU memory. This collection of images was then randomly down-sampled so that each class had equal representation in the collection.

One concern regarding the performance of trained conv-nets we had was their robustness to fluctuations in illumination and microscope cameras. Sharable solutions to this segmentation problem require some amount of robustness to these changes, as different labs use different microscopes and cameras to acquire data. Even within labs, image acquisition parameters may not be identical between experiments. To account for this, we normalized our training data sets prior to using them

to train a conv-net. Each training image was first divided by the median pixel value. The local average of this normalized image was computed by applying an averaging filter (the same size as that of the neural network's effective receptive field size—in our case 31x31 for bacterial cells and 61x61 for mammalian nuclei and cytoplasms) and then subtracted from the normalized image to create the final image that was sampled to train a conv-net. After sampling our image, we then performed data augmentation by randomly rotating each patch by either 0, 90, 180, or 270 degrees and randomly reflecting them on the fly prior to their use our training dataset. Our reasoning was that class labels should be invariant to these transformations.

After normalization and data augmentation, each training dataset consisted of ~200,000–400,000 patches. The mammalian cytoplasm training datasets contained two channels—a phase image and an image in which cell nuclei were labelled with Hoechst 33342 (NIH-3T3 and MCF10A) or H2B-iRFP (HeLa-S3).

Segmenting new images

To construct a segmentation mask for a new image, the trained conv-net is applied to the entire image to produce pixel level classification predictions. This is done by using a fully convolutional implementation of conv-nets that can be directly applied to the entire image [35, 36]. We also used model parallelism (averaging the results of 5 trained networks) to improve our segmentation accuracy.

While the classification predictions produced by conv-nets were remarkably accurate, we found that they required further refinement to produce binary segmentation masks. As opposed to using the raw class scores for each pixel as the input for refinement, we instead use the softmax normalized score

$$\frac{e^{class\ score}}{\sum_{all\ classes} e^{class\ score}},$$

which we loosely interpret as the probability deduced by the conv-net that the pixel in question belongs to that particular class. For bacterial cells, we found that thresholding of the softmax normalized score for the cell interior class with a value of 0.6 was sufficient to produce accurate

masks. For nuclei, we found that thresholding with a value of 0.5–0.75 in a similar fashion to bacteria worked well in most cases, with adaptive thresholding of the softmax normalized cell interior score being used for images with very dim nuclei. For mammalian cells, the conv-net prediction in combination with the nuclear marker was used to guide an active contour algorithm to further refine the segmentation[52]. Training error and segmentation results for *E. coli*, MCF10A, NIH-3T3, and HeLa-S3 cells are shown in Fig 2.

ACKNOWLEDGMENTS

We are deeply grateful to a number of people for advice, assistance with experiments, and critical commentary on the manuscript, including Zev Bryant, Joe Levine, Fei Fei Li, Andrej Karpathy, Henrietta Lacks, Amanda Miguel, K.C. Huang, Robert Brewster, Rob Phillips, John Phillips, Jan Liphardt, Andy Spakowitz, and members of the Covert lab.

AUTHOR CONTRIBUTIONS

- **Conceptualization:** DAVV MWC.
- **Data curation:** DAVV KML NTQ MMD IM.
- **Formal analysis:** DAVV.
- **Funding acquisition:** DAVV TK KML DNM NTQ MMD IM YT EAA MWC.
- **Investigation:** DAVV TK KML DNM NTQ MMD IM YT EAA MWC.
- **Methodology:** DAVV.
- **Project administration:** DAVV MWC.
- **Resources:** DAVV TK KML DNM NTQ MMD IM YT.
- **Software:** DAVV TK DNM NTQ.
- **Supervision:** EAA MWC.
- **Validation:** DAVV MMD KML IM YT NTQ.
- **Visualization:** DAVV.
- **Writing – original draft:** DAVV MWC.
- **Writing – review & editing:** DAVV TK KML DNM NTQ MMD IM YT EAA MWC.

REFERENCES

1. Rosenfeld N, Young JW, Alon U, Swain PS, Elowitz MB. Gene regulation at the single-cell level. Science. 2005;307(5717):1962–5. pmid:15790856
2. Weinert FM, Brewster RC, Rydenfelt M, Phillips R, Kegel WK. Scaling of gene expression with transcription-factor fugacity. Physical review letters. 2014;113(25):258101. pmid:25554908
3. Selimkhanov J, Taylor B, Yao J, Pilko A, Albeck J, Hoffmann A, et al. Accurate information transmission through dynamic biochemical signaling networks. Science. 2014;346(6215):1370–3. pmid:25504722
4. Tay S, Hughey JJ, Lee TK, Lipniacki T, Quake SR, Covert MW. Single-cell NF-[kgr] B dynamics reveal digital activation and analogue information processing. Nature. 2010;466(7303):267–71. pmid:20581820
5. Lee TK, Denny EM, Sanghvi JC, Gaston JE, Maynard ND, Hughey JJ, et al. A noisy paracrine signal determines the cellular NF-κB response to lipopolysaccharide. Science signaling. 2009;2(93):ra65. pmid:19843957
6. Regot S, Hughey JJ, Bajar BT, Carrasco S, Covert MW. High-sensitivity measurements of multiple kinase activities in live single cells. Cell. 2014;157(7):1724–34. pmid:24949979
7. Cheong R, Rhee A, Wang CJ, Nemenman I, Levchenko A. Information transduction capacity of noisy biochemical signaling networks. Science. 2011;334(6054):354–8. pmid:21921160
8. Hughey JJ, Lee TK, Covert MW. Computational modeling of mammalian signaling networks. Wiley Interdisciplinary Reviews: Systems Biology and Medicine. 2010;2(2):194–209. pmid:20836022
9. Lee TK, Covert MW. High-throughput, single-cell NF-κB dynamics. Current opinion in genetics & development. 2010;20(6):677–83.
10. Sliusarenko O, Heinritz J, Emonet T, Jacobs-Wagner C. High-throughput, subpixel precision analysis of bacterial morphogenesis and intracellular spatio-temporal dynamics. Molecular microbiology. 2011;80(3):612–27. pmid:21414037
11. Young JW, Locke JC, Altinok A, Rosenfeld N, Bacarian T, Swain

PS, et al. Measuring single-cell gene expression dynamics in bacteria using fluorescence time-lapse microscopy. Nature protocols. 2012;7(1):80–8.

12. Carpenter AE, Jones TR, Lamprecht MR, Clarke C, Kang IH, Friman O, et al. CellProfiler: image analysis software for identifying and quantifying cell phenotypes. Genome biology. 2006;7(10):R100. pmid:17076895
13. Malpica N, Ortiz de Solorzano C, Vaquero JJ, Santos A, Vallcorba I, Garcia-Sagredo JM, et al. Applying watershed algorithms to the segmentation of clustered nuclei. 1997.
14. Martin DR, Fowlkes CC, Malik J. Learning to detect natural image boundaries using local brightness, color, and texture cues. IEEE transactions on pattern analysis and machine intelligence. 2004;26(5):530–49. pmid:15460277
15. Mao KZ, Zhao P, Tan P-H. Supervised learning-based cell image segmentation for p53 immunohistochemistry. IEEE Transactions on Biomedical Engineering. 2006;53(6):1153–63. pmid:16761842
16. Bertelli L, Yu T, Vu D, Gokturk B, editors. Kernelized structural SVM learning for supervised object segmentation. Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference on; 2011: IEEE.
17. Paintdakhi A, Parry B, Campos M, Irnov I, Elf J, Surovtsev I, et al. Oufti: an integrated software package for high-accuracy, high-throughput quantitative microscopy analysis. Molecular microbiology. 2016.
18. Sommer C, Straehle C, Koethe U, Hamprecht FA, editors. ilastik: Interactive learning and segmentation toolkit. Biomedical Imaging: From Nano to Macro, 2011 IEEE International Symposium on; 2011: IEEE.
19. Belevich I, Joensuu M, Kumar D, Vihinen H, Jokitalo E. Microscopy Image Browser: A Platform for Segmentation and Analysis of Multidimensional Datasets. PLoS Biol. 2016;14(1):e1002340. pmid:26727152
20. Tanouchi Y, Pai A, Park H, Huang S, Stamatov R, Buchler NE, et al. A noisy linear map underlies oscillations in cell size and

gene expression in bacteria. Nature. 2015;523(7560):357–60. pmid:26040722

21. Ali R, Gooding M, Christlieb M, Brady M, editors. Phase-based segmentation of cells from brightfield microscopy. Biomedical Imaging: From Nano to Macro, 2007 ISBI 2007 4th IEEE International Symposium on; 2007: IEEE.
22. Lindblad J, Wahlby C, Bengtsson E, Zaltsman A. Image analysis for automatic segmentation of cytoplasms and classification of Rac1 activation. Cytometry Part A: the journal of the International Society for Analytical Cytology. 2004;57(1):22–33. Epub 2003/12/31. pmid:14699602.
23. Wang Q, Niemi J, Tan CM, You L, West M. Image segmentation and dynamic lineage analysis in single-cell fluorescence microscopy. Cytometry Part A. 2010;77(1):101–10.
24. Mualla F, Scholl S, Sommerfeldt B, Maier A, Hornegger J. Automatic cell detection in bright-field microscope images using SIFT, random forests, and hierarchical clustering. Medical Imaging, IEEE Transactions on. 2013;32(12):2274–86.
25. Gudla PR, Nandy K, Collins J, Meaburn K, Misteli T, Lockett S. A high-throughput system for segmenting nuclei using multiscale techniques. Cytometry Part A. 2008;73(5):451–66.
26. Chalfoun J, Kociolek M, Dima A, Halter M, Cardone A, Peskin A, et al. Segmenting time-lapse phase contrast images of adjacent NIH 3T3 cells. Journal of microscopy. 2013;249(1):41–52. pmid:23126432
27. Rees P, Wills JW, Brown MR, Tonkin J, Holton MD, Hondow N, et al. Nanoparticle vesicle encoding for imaging and tracking cell populations. Nature methods. 2014;11(11):1177–81. pmid:25218182
28. Chong YT, Cox MJ, Andrews B. Proteome-wide screens in Saccharomyces cerevisiae using the yeast GFP collection. Advances in Systems Biology: Springer; 2012. p. 169–78.
29. Shedden K, Li Q, Liu F, Chang YT, Rosania GR. Machine vision-assisted analysis of structure-localization relationships in a combinatorial library of prospective bioimaging probes. Cytometry Part A. 2009;75(6):482–93.
30. Chalfoun J, Majurski M, Peskin A, Breen C, Bajcsy P, Brady M.

Empirical gradient threshold technique for automated segmentation across image modalities and cell lines. Journal of microscopy. 2015;260(1):86–99. pmid:26046924

31. Chen L, Zhang J, Chen S, Lin Y, Yao C, Zhang J. Hierarchical Mergence Approach to Cell Detection in Phase Contrast Microscopy Images. Computational and Mathematical Methods in Medicine. 2014;2014:10. pmid:24987452
32. LeCun Y, Bengio Y, Hinton G. Deep learning. Nature. 2015;521(7553):436–44. pmid:26017442
33. Krizhevsky A, Sutskever I, Hinton G. Advances in Neural Information Processing Systems. Cambridge, MA: MIT Press; 2012.
34. Sermanet P, Eigen D, Zhang X, Mathieu M, Fergus R, LeCun Y. Overfeat: Integrated recognition, localization and detection using convolutional networks. arXiv preprint arXiv:13126229. 2013.
35. Long J, Shelhamer E, Darrell T, editors. Fully convolutional networks for semantic segmentation. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition; 2015.
36. Li H, Zhao R, Wang X. Highly efficient forward and backward propagation of convolutional neural networks for pixelwise classification. arXiv preprint arXiv:14124526. 2014.
37. Ronneberger O, Fischer P, Brox T. U-Net: Convolutional Networks for Biomedical Image Segmentation. Medical Image Computing and Computer-Assisted Intervention–MICCAI 2015: Springer; 2015. p. 234–41.
38. Ciresan D, Giusti A, Gambardella LM, Schmidhuber J, editors. Deep neural networks segment neuronal membranes in electron microscopy images. Advances in neural information processing systems; 2012.
39. Cireşan DC, Giusti A, Gambardella LM, Schmidhuber J. Mitosis detection in breast cancer histology images with deep neural networks. Medical Image Computing and Computer-Assisted Intervention–MICCAI 2013: Springer; 2013. p. 411–8.
40. Kraus OZ, Ba LJ, Frey B. Classifying and Segmenting Microscopy Images Using Convolutional Multiple Instance Learning. arXiv preprint arXiv:151105286. 2015.

41. Srivastava N, Hinton GE, Krizhevsky A, Sutskever I, Salakhutdinov R. Dropout: a simple way to prevent neural networks from overfitting. Journal of Machine Learning Research. 2014;15(1):1929–58.
42. Ioffe S, Szegedy C. Batch normalization: Accelerating deep network training by reducing internal covariate shift. arXiv preprint arXiv:150203167. 2015.
43. Nair V, Hinton GE, editors. Rectified linear units improve restricted boltzmann machines. Proceedings of the 27th International Conference on Machine Learning (ICML-10); 2010.
44. Tieleman T, Hinton G. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. COURSERA: Neural Networks for Machine Learning. 2012;4:2.
45. He K, Zhang X, Ren S, Sun J, editors. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. Proceedings of the IEEE International Conference on Computer Vision; 2015.
46. Rumelhart DE, Hinton GE, Williams RJ. Learning representations by back-propagating errors. Nature. 1986;323(6088):533–6.
47. Bergstra J, Breuleux O, Bastien F, Lamblin P, Pascanu R, Desjardins G, et al., editors. Theano: a CPU and GPU math expression compiler. Proceedings of the Python for scientific computing conference (SciPy); 2010: Austin, TX.
48. Van Der Walt S, Schönberger JL, Nunez-Iglesias J, Boulogne F, Warner JD, Yager N, et al. scikit-image: image processing in Python. PeerJ. 2014;2:e453. pmid:25024921
49. Chollet F. Keras. GitHub repository: GitHub; 2015.
50. Jones E, Oliphant T, Peterson P. SciPy: Open source scientific tools for Python, 2001. http://www.scipy.org. 2015.
51. Van Der Walt S, Colbert SC, Varoquaux G. The NumPy array: a structure for efficient numerical computation. Computing in Science & Engineering. 2011;13(2):22–30.
52. Marquez-Neila P, Baumela L, Alvarez L. A morphological approach to curvature-based evolution of curves and surfaces. Pattern Analysis and Machine Intelligence, IEEE Transactions on. 2014;36(1):2–17.
53. Bajcsy P, Cardone A, Chalfoun J, Halter M, Juba D, Kociolek M, et

al. Survey statistics of automated segmentations applied to optical imaging of mammalian cells. BMC bioinformatics. 2015;16(1):1.

54. Jaqaman K, Loerke D, Mettlen M, Kuwata H, Grinstein S, Schmid SL, et al. Robust single-particle tracking in live-cell time-lapse sequences. Nature methods. 2008;5(8):695–702. pmid:18641657
55. Taheri-Araghi S, Bradde S, Sauls JT, Hill NS, Levin PA, Paulsson J, et al. Cell-size control and homeostasis in bacteria. Current Biology. 2015;25(3):385–91. pmid:25544609
56. Wang P, Robert L, Pelletier J, Dang WL, Taddei F, Wright A, et al. Robust growth of Escherichia coli. Current biology. 2010;20(12):1099–103. pmid:20537537
57. Campos M, Surovtsev IV, Kato S, Paintdakhi A, Beltran B, Ebmeier SE, et al. A constant size extension drives bacterial cell size homeostasis. Cell. 2014;159(6):1433–46. pmid:25480302
58. Liu J, Prindle A, Humphries J, Gabalda-Sagarra M, Asally M, Dong-yeon DL, et al. Metabolic co-dependence gives rise to collective oscillations within biofilms. Nature. 2015;523(7562):550–4. pmid:26200335
59. Nghe P, Boulineau S, Gude S, Recouvreux P, van Zon JS, Tans SJ. Microfabricated polyacrylamide devices for the controlled culture of growing cells and developing organisms. PloS one. 2013;8(9):e75537. pmid:24086559
60. Boulineau S, Tostevin F, Kiviet DJ, ten Wolde PR, Nghe P, Tans SJ. Single-cell dynamics reveals sustained growth during diauxic shifts. PLoS One. 2013;8(4):e61686. pmid:23637881
61. Kiviet DJ, Nghe P, Walker N, Boulineau S, Sunderlikova V, Tans SJ. Stochasticity of metabolism and growth at the single-cell level. Nature. 2014;514(7522):376–9. pmid:25186725
62. Chen L- C, Papandreou G, Kokkinos I, Murphy K, Yuille AL. Semantic image segmentation with deep convolutional nets and fully connected crfs. arXiv preprint arXiv:14127062. 2014.
63. Szegedy C, Liu W, Jia Y, Sermanet P, Reed S, Anguelov D, et al., editors. Going deeper with convolutions. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition; 2015.
64. Helaine S, Cheverton AM, Watson KG, Faure LM, Matthews

SA, Holden DW. Internalization of Salmonella by macrophages induces formation of nonreplicating persisters. Science. 2014;343(6167):204–8. pmid:24408438

65. Raj A, Peskin CS, Tranchina D, Vargas DY, Tyagi S. Stochastic mRNA Synthesis in Mammalian Cells. PLoS Biol. 2006;4(10):e309. pmid:17048983

66. Nielsen HJ, Youngren B, Hansen FG, Austin S. Dynamics of Escherichia coli chromosome segregation during multifork replication. Journal of bacteriology. 2007;189(23):8660–6. pmid:17905986

CHAPTER 4

Accurate De Novo Prediction of Protein Contact Map by Ultra-Deep Learning Model

Sheng Wang, Siqi Sun, Zhen Li, Renyu Zhang, Jinbo Xu

Toyota Technological Institute at Chicago, Chicago, Illinois, United States of America

ABSTRACT

Motivation

Protein contacts contain key information for the understanding of protein structure and function and thus, contact prediction from sequence is an important problem. Recently exciting progress has been made on this problem, but the predicted contacts for proteins without many sequence homologs is still of low quality and not very useful for de novo structure prediction.

Method

This paper presents a new deep learning method that predicts contacts by integrating both evolutionary coupling (EC) and sequence conservation

information through an ultra-deep neural network formed by two deep residual neural networks. The first residual network conducts a series of 1-dimensional convolutional transformation of sequential features; the second residual network conducts a series of 2-dimensional convolutional transformation of pairwise information including output of the first residual network, EC information and pairwise potential. By using very deep residual networks, we can accurately model contact occurrence patterns and complex sequence-structure relationship and thus, obtain higher-quality contact prediction regardless of how many sequence homologs are available for proteins in question.

Results

Our method greatly outperforms existing methods and leads to much more accurate contact-assisted folding. Tested on 105 CASP11 targets, 76 past CAMEO hard targets, and 398 membrane proteins, the average top L long-range prediction accuracy obtained by our method, one representative EC method CCMpred and the CASP11 winner MetaPSICOV is 0.47, 0.21 and 0.30, respectively; the average top L/10 long-range accuracy of our method, CCMpred and MetaPSICOV is 0.77, 0.47 and 0.59, respectively. Ab initio folding using our predicted contacts as restraints but without any force fields can yield correct folds (i.e., TMscore>0.6) for 203 of the 579 test proteins, while that using MetaPSICOV- and CCMpred-predicted contacts can do so for only 79 and 62 of them, respectively. Our contact-assisted models also have much better quality than template-based models especially for membrane proteins. The 3D models built from our contact prediction have TMscore>0.5 for 208 of the 398 membrane proteins, while those from homology modeling have TMscore>0.5 for only 10 of them. Further, even if trained mostly by soluble proteins, our deep learning method works very well on membrane proteins. In the recent blind CAMEO benchmark, our fully-automated web server implementing this method successfully folded 6 targets with a new fold and only 0.3L-2.3L effective sequence homologs, including one β protein of 182 residues, one α+β protein of 125 residues, one α protein of 140 residues, one α protein of 217 residues, one α/β of 260 residues and one α protein of 462 residues. Our method also achieved the highest F1 score on free-modeling targets in the latest CASP (Critical Assessment of Structure Prediction), although it was not fully implemented back then.

INTRODUCTION

De novo protein structure prediction from sequence alone is one of most challenging problems in computational biology. Recent progress has indicated that some correctly-predicted long-range contacts may allow accurate topology-level structure modeling [1] and that direct evolutionary coupling analysis (DCA) of multiple sequence alignment (MSA) may reveal some long-range native contacts for proteins and protein-protein interactions with a large number of sequence homologs [2, 3]. Therefore, contact prediction and contact-assisted protein folding has recently gained much attention in the community. However, for many proteins especially those without many sequence homologs, the predicted contacts by the state-of-the-art predictors such as CCMpred [4], PSICOV [5], Evfold [6], plmDCA[7], Gremlin[8], MetaPSICOV [9] and CoinDCA [10] are still of low quality and insufficient for accurate contact-assisted protein folding [11, 12]. This motivates us to develop a better contact prediction method, especially for proteins without a large number of sequence homologs. In this paper we define that two residues form a contact if they are spatially proximal in the native structure, i.e., the Euclidean distance of their C_β atoms less than 8Å [13].

Existing contact prediction methods roughly belong to two categories: evolutionary coupling analysis (ECA) and supervised machine learning. ECA predicts contacts by identifying co-evolved residues in a protein, such as EVfold [6], PSICOV [5], CCMpred [4], Gremlin [8], plmDCA and others [14–16]. However, DCA usually needs a large number of sequence homologs to be effective [10, 17]. Supervised machine learning predicts contacts from a variety of information, e.g., SVMSEQ [18], CMAPpro [13], PconsC2 [17], MetaPSICOV [9], PhyCMAP [19] and CoinDCA-NN [10]. Meanwhile, PconsC2 uses a 5-layer supervised learning architecture [17]; CoinDCA-NN and MetaPSICOV employ a 2-layer neural network [9]. CMAPpro uses a neural network with more layers, but its performance saturates at about 10 layers. Some supervised methods such as MetaPSICOV and CoinDCA-NN outperform ECA on proteins without many sequence homologs, but their performance is still limited by their shallow architectures.

To further improve supervised learning methods for contact prediction, we borrow ideas from very recent breakthrough in computer vision. In particular, we have greatly improved contact prediction by

developing a brand-new deep learning model called residual neural network [20] for contact prediction. Deep learning is a powerful machine learning technique that has revolutionized image classification [21, 22] and speech recognition [23]. In 2015, ultra-deep residual neural networks [24] demonstrated superior performance in several computer vision challenges (similar to CASP) such as image classification and object recognition [25]. If we treat a protein contact map as an image, then protein contact prediction is kind of similar to (but not exactly same as) pixel-level image labeling, so some techniques effective for image labeling may also work for contact prediction. However, there are some important differences between image labeling and contact prediction. First, in computer vision community, image-level labeling (i.e., classification of a single image) has been extensively studied, but there are much fewer studies on pixel-level image labeling (i.e., classification of an individual pixel). Second, in many image classification scenarios, image size is resized to a fixed value, but we cannot resize a contact map since we need to do prediction for every residue pair (equivalent to an image pixel). Third, contact prediction has much more complex input features (including both sequential and pairwise features) than image labeling. Fourth, the ratio of contacts in a protein is very small (<2%). That is, the number of positive and negative labels in contact prediction is extremely unbalanced.

In this paper we present a very deep residual neural network for contact prediction. Such a network can capture very complex sequence-contact relationship and high-order contact correlation. We train this deep neural network using a subset of proteins with solved structures and then test it on public data including the CASP [26, 27] and CAMEO [28] targets as well as many membrane proteins. Our experimental results show that our method yields much better accuracy than existing methods and also result in much more accurate contact-assisted folding. The deep learning method described here will also be useful for the prediction of protein-protein and protein-RNA interfacial contacts.

RESULTS

Deep learning model for contact prediction

Fig 1 illustrates our deep neural network model for contact prediction [29]. Different from previous supervised learning approaches[9, 13] for contact prediction that employ only a small number of hidden layers (i.e., a shallow architecture), our deep neural network employs dozens of hidden layers. By using a very deep architecture, our model can automatically learn the complex relationship between sequence information and contacts and also model the interdependency among contacts and thus, improve contact prediction [17]. Our model consists of two major modules, each being a residual neural network. The first module conducts a series of 1-dimensional (1D) convolutional transformations of sequential features (sequence profile, predicted secondary structure and solvent accessibility). The output of this 1D convolutional network is converted to a 2-dimensional (2D) matrix by outer concatenation (an operation similar to outer product) and then fed into the 2nd module together with pairwise features (i.e., co-evolution information, pairwise contact and distance potential). The 2nd module is a 2D residual network that conducts a series of 2D convolutional transformations of its input. Finally, the output of the 2D convolutional network is fed into a logistic regression, which predicts the probability of any two residues form a contact. In addition, each convolutional layer is also preceded by a simple nonlinear transformation called rectified linear unit [30]. Mathematically, the output of 1D residual network is just a 2D matrix with dimension L×m where m is the number of new features (or hidden neurons) generated by the last convolutional layer of the network. Biologically, this 1D residual network learns the sequential context of a residue. By stacking multiple convolution layers, the network can learn information in a very large sequential context. The output of a 2D convolutional layer has dimension L×L×n where n is the number of new features (or hidden neurons) generated by this layer for one residue pair. The 2D residual network mainly learns contact occurrence patterns or high-order residue correlation (i.e., 2D context of a residue pair). The number of hidden neurons may vary at each layer.

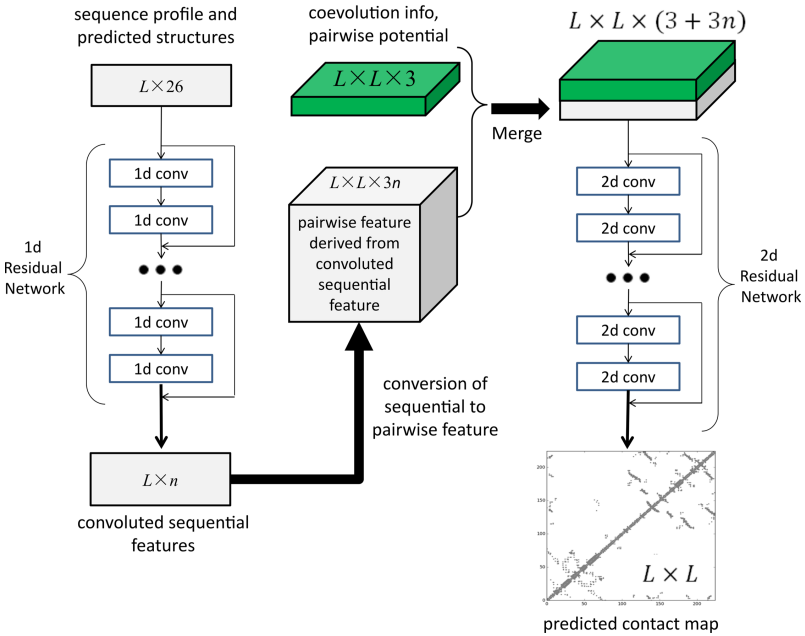


Figure 1: Illustration of our deep learning model for contact prediction where L is the sequence length of one protein under prediction.

Our test data includes the 150 Pfam families described in [5], 105 CASP11 test proteins [31], 398 membrane proteins and 76 CAMEO hard targets released from 10/17/2015 to 04/09/2016. The tested methods include PSICOV [5], Evfold [6], CCMpred [4], plmDCA[7], Gremlin[8], and MetaPSICOV [9]. The former 5 methods employs pure DCA while MetaPSICOV [9] is a supervised learning method that performed the best in CASP11 [31]. All the programs are run with parameters set according to their respective papers. We cannot evaluate PconsC2 [17] since we failed to obtain any results from its web server. PconsC2 did not outperform MetaPSICOV in CASP11 [31], so it may suffice to just compare our method with MetaPSICOV.

Overall performance

We evaluate the accuracy of the top L/k ($k = 10, 5, 2, 1$) predicted contacts where L is protein sequence length [10]. We define that a contact is short-, medium- and long-range when the sequence distance of the two

residues in a contact falls into [6, 11], [12, 23], and ≥24, respectively. The prediction accuracy is defined as the percentage of native contacts among the top *L/k* predicted contacts. When there are no *L/k* native (short- or medium-range) contacts, we replace the denominator by *L/k* in calculating accuracy. This may make the short- and medium-range accuracy look small although it is easier to predict short- and medium-range contacts than long-range ones.

As shown in Tables 1–4, our method outperforms all tested DCA methods and MetaPSICOV by a very large margin on the 4 test sets regardless of how many top predicted contacts are evaluated and no matter whether the contacts are short-, medium- or long-range. These results also show that two supervised learning methods greatly outperform the pure DCA methods and that the three pseudo-likelihood DCA methods plmDCA, Gremlin and CCMpred perform similarly, but outperform PSICOV (Gaussian model) and Evfold (maximum-entropy method). The advantage of our method is the smallest on the 150 Pfam families because many of them have a pretty large number of sequence homologs. In terms of top L long-range contact accuracy on the CASP11 set, our method exceeds CCMpred and MetaPSICOV by 0.32 and 0.20, respectively. On the 76 CAMEO hard targets, our method exceeds CCMpred and MetaPSICOV by 0.27 and 0.17, respectively. On the 398 membrane protein set, our method exceeds CCMpred and MetaPSICOV by 0.26 and 0.17, respectively. Our method uses a subset of protein features used by MetaPSICOV, but performs much better than MetaPSICOV due to our deep architecture and that we predict contacts of a protein simultaneously. Since the Pfam set is relatively easy, we will not analyze it any more in the following sections.

Table 1: Contact prediction accuracy on the 150 Pfam families

Method	Short				Medium				Long			
	L/10	L/5	L/2	L	L/10	L/5	L/2	L	L/10	L/5	L/2	L
EVfold	0.50	0.40	0.26	0.17	0.64	0.52	0.34	0.22	0.74	0.68	0.53	0.39
PSICOV	0.58	0.43	0.26	0.17	0.65	0.51	0.32	0.20	0.77	0.70	0.52	0.37
CCMpred	0.65	0.50	0.29	0.19	0.73	0.60	0.37	0.23	0.82	0.76	0.62	0.45
plmDCA	0.66	0.50	0.29	0.19	0.72	0.60	0.36	0.22	0.81	0.76	0.61	0.44
Gremlin	0.66	0.51	0.30	0.19	0.74	0.60	0.37	0.23	0.82	0.76	0.63	0.46
MetaPSICOV	0.82	0.70	0.45	0.27	0.83	0.73	0.52	0.33	0.92	0.87	0.74	0.58
Our method	0.93	0.81	0.51	0.30	0.93	0.86	0.62	0.38	0.98	0.96	0.89	0.74

doi:10.1371/journal.pcbi.1005324.t001

Table 2: Contact prediction accuracy on 105 CASP11 test proteins

Method	Short				Medium				Long			
	L/10	L/5	L/2	L	L/10	L/5	L/2	L	L/10	L/5	L/2	L
EVfold	0.25	0.21	0.15	0.12	0.33	0.27	0.19	0.13	0.37	0.33	0.25	0.19
PSICOV	0.29	0.23	0.15	0.12	0.34	0.27	0.18	0.13	0.38	0.33	0.25	0.19
CCMpred	0.35	0.28	0.17	0.12	0.40	0.32	0.21	0.14	0.43	0.39	0.31	0.23
plmDCA	0.32	0.26	0.17	0.12	0.39	0.31	0.21	0.14	0.42	0.38	0.30	0.23
Gremlin	0.35	0.27	0.17	0.12	0.40	0.31	0.21	0.14	0.44	0.40	0.31	0.23
MetaPSICOV	0.69	0.58	0.39	0.25	0.69	0.59	0.42	0.28	0.60	0.54	0.45	0.35
Our method	0.82	0.70	0.46	0.28	0.85	0.76	0.55	0.35	0.81	0.77	0.68	0.55

doi:10.1371/journal.pcbi.1005324.t002

Table 3: Contact prediction accuracy on 76 past CAMEO hard targets

Method	Short				Medium				Long			
	L/10	L/5	L/2	L	L/10	L/5	L/2	L	L/10	L/5	L/2	L
EVfold	0.17	0.13	0.11	0.09	0.23	0.19	0.13	0.10	0.25	0.22	0.17	0.13
PSICOV	0.20	0.15	0.11	0.08	0.24	0.19	0.13	0.09	0.25	0.23	0.18	0.13
CCMpred	0.22	0.16	0.11	0.09	0.27	0.22	0.14	0.10	0.30	0.26	0.20	0.15
plmDCA	0.23	0.18	0.12	0.09	0.27	0.22	0.14	0.10	030	0.26	0.20	0.15
Gremlin	0.21	0.17	0.11	0.08	0.27	0.22	0.14	0.10	0.31	0.26	0.20	0.15
MetaPSICOV	0.56	0.47	0.31	0.20	0.53	0.45	0.32	0.22	0.47	0.42	0.33	0.25
Our method	0.67	0.57	0.37	0.23	0.69	0.61	0.42	0.28	0.69	0.65	0.55	0.42

doi:10.1371/journal.pcbi.1005324.t003

Table 4: Contact prediction accuracy on 398 membrane proteins

Method	Short				Medium				Long			
	L/10	L/5	L/2	L	L/10	L/5	L/2	L	L/10	L/5	L/2	L
EVfold	0.16	0.13	0.09	0.07	0.28	0.22	0.13	0.09	0.44	0.37	0.26	0.18
PSICOV	0.22	0.16	0.10	0.07	0.29	0.21	0.13	0.09	0.42	0.34	0.23	0.16
CCMpred	0.27	0.19	0.11	0.08	0.36	0.26	0.15	0.10	0.52	0.45	0.31	0.21
plmDCA	0.26	0.18	0.11	0.08	0.35	0.25	0.14	0.09	0.51	0.42	0.29	0.20
Gremlin	0.27	0.19	0.11	0.07	0.37	0.26	0.15	0.10	0.52	0.45	0.32	0.21
MetaPSICOV	0.45	0.35	0.22	0.14	0.49	0.40	0.27	0.18	0.61	0.55	0.42	0.30
Our method	0.60	0.46	0.27	0.16	0.66	0.53	0.33	0.22	0.78	0.73	0.62	0.47

doi:10.1371/journal.pcbi.1005324.t004

Prediction accuracy with respect to the number of sequence homologs

To examine the performance of our method with respect to the amount of homologous information available for a protein under prediction, we measure the effective number of sequence homologs in multiple sequence alignment (MSA) by *Meff* [19], which can be roughly interpreted as the number of non-redundant sequence homologs when 70% sequence identity is used as cutoff to remove redundancy (see Method for its formula). A protein with a smaller *Meff* has less homologous information. We divide all the test proteins into 10 bins according to *ln(Meff)* and then calculate the average accuracy of proteins in each bin. We merge the first

3 bins for the membrane protein set since they have a small number of proteins.

Fig 2 shows that the top L/5 contact prediction accuracy increases with respect to *Meff*, i.e., the number of effective sequence homologs, and that our method outperforms both MetaPSICOV and CCMpred regardless of *Meff*. Our long-range prediction accuracy is even better when $ln(Meff) \leq 7$ (equivalently $Meff < 1100$), i.e., when the protein under prediction does not have a very large number of non-redundant sequence homologs. Our method has a large advantage over the other methods even when *Meff* is very big (>8000). This indicates that our method indeed benefits from some extra information such as inter-contact correlation or high-order residue correlation, which is orthogonal to pairwise co-evolution information.

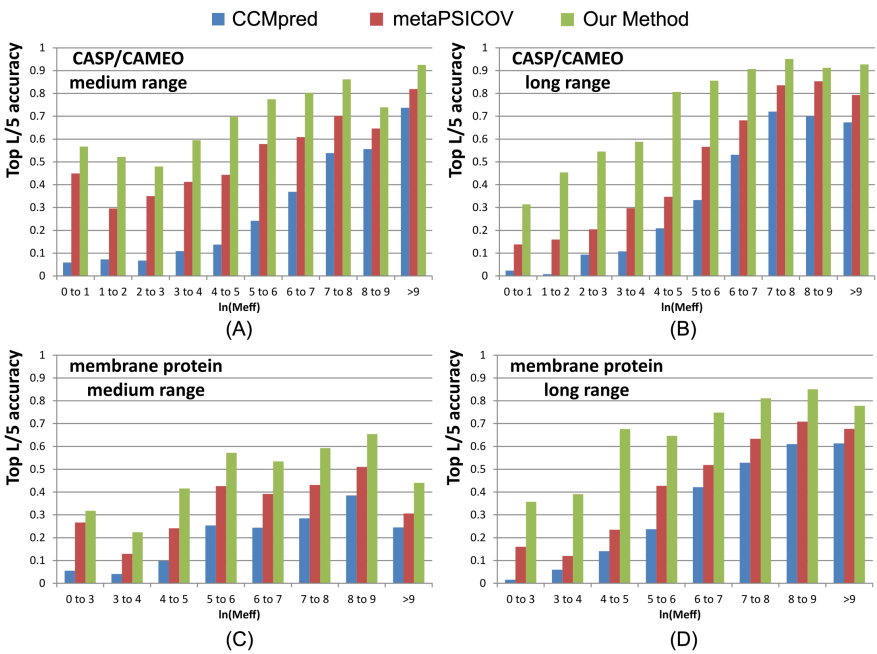


Figure 2: Top L/5 accuracy of our method (green), CCMpred (blue) and MetaPSICOV (red) with respect to the amount of homologous information measured by *ln(Meff)*.

The accuracy on the union of the 105 CASP and 76 CAMEO targets is displayed in (A) medium-range and (B) long-range. The accuracy on

the membrane protein set is displayed in (C) medium-range and (D) long-range.

Contact-assisted protein folding

One of the important goals of contact prediction is to perform contact-assisted protein folding [11]. To test if our contact prediction can lead to better 3D structure modeling than the others, we build structure models for all the test proteins using the top predicted contacts as restraints of ab initio folding. For each test protein, we feed the top predicted contacts as restraints into the CNS suite [32] to generate 3D models. We measure the quality of a 3D model by a superposition-dependent score TMscore [33], which ranges from 0 to 1, with 0 indicating the worst and 1 the best, respectively. According to Xu and Zhang [34], a model with TMscore>0.5 (TMscore>0.6) is likely (highly likely) to have a correct fold. We also measure the quality of a 3D model by a superposition-independent score lDDT, which ranges from 0 to 100, with 0 indicating the worst and 100 the best, respectively.

Fig 3 shows that our predicted contacts can generate much better 3D models than CCMpred and MetaPSICOV. On average, our 3D models are better than MetaPSICOV and CCMpred by ~0.12 TMscore unit and ~0.15 unit, respectively. When the top 1 models are evaluated, the average TMscore obtained by CCMpred, MetaPSICOV, and our method is 0.333, 0.377, and 0.518, respectively on the CASP dataset. The average lDDT of CCMpred, MetaPSICOV and our method is 31.7, 34.1 and 41.8, respectively. On the 76 CAMEO targets, the average TMsore of CCMpred, MetaPSICOV and our method is 0.256, 0.305 and 0.407, respectively. The average lDDT of CCMpred, MetaPSICOV and our method is 31.8, 35.4 and 40.2, respectively. On the membrane protein set, the average TMscore of CCMpred, MetaPSICOV and our method is 0.354, 0.387, and 0.493, respectively. The average lDDT of CCMpred, MetaPSICOV and our method is 38.1, 40.5 and 47.8, respectively. Same trend is observed when the best of top 5 models are evaluated. On the CASP set, the average TMscore of the models generated by CCMpred, MetaPSICOV, and our method is 0.352, 0.399, and 0.543, respectively. The average lDDT of CCMpred, MetaPSICOV and our method is 32.3, 34.9 and 42.4, respectively. On the 76 CAMEO proteins, the average TMscore of CCMpred, MetaPSICOV, and our method is 0.271, 0.334, and

0.431, respectively. The average lDDT of CCMpred, MetaPSICOV and our method is 32.4, 36.1 and 40.9, respectively. On the membrane protein set, the average TMscore of CCMpred, MetaPSICOV, and our method is 0.385, 0.417, and 0.516, respectively. The average lDDT of CCMpred, MetaPSICOV and our method is 38.9, 41.2 and 48.5, respectively. In particular, when the best of top 5 models are considered, our predicted contacts can result in correct folds (i.e., TMscore>0.6) for 203 of the 579 test proteins, while MetaPSICOV- and CCMpred-predicted contacts can do so for only 79 and 62 of them, respectively.

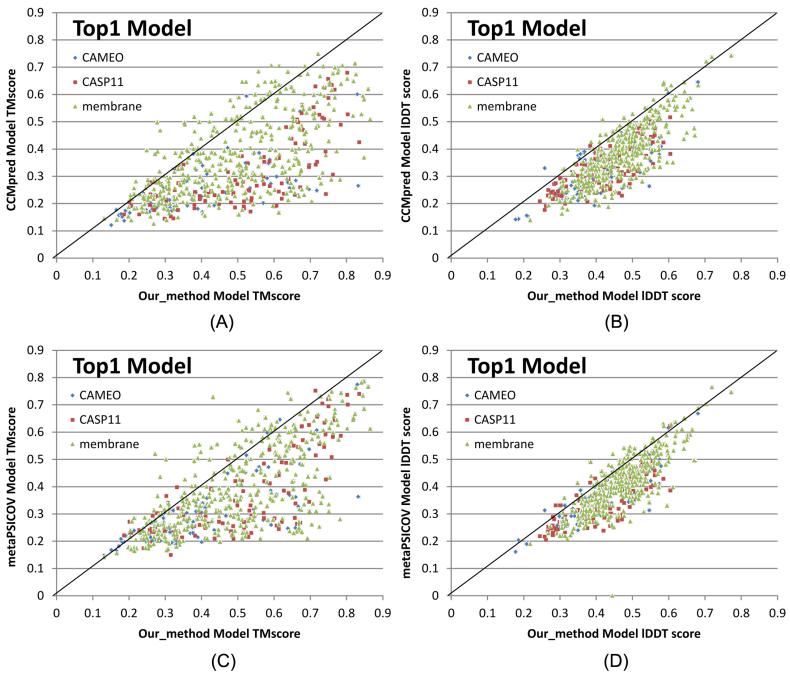


Figure 3: Quality comparison of top 1 contact-assisted models generated by our method, CCMpred and MetaPSICOV on the 105 CASP11 targets (red square), 76 CAMEO targets (blue diamond) and 398 membrane protein targets (green triangle), respectively.

(A) and **(B)**: comparison between our method (X-axis) and CCMpred (Y-axis) in terms of TMscore and lDDT, respectively. **(C)** and **(D)**: comparison between our method (X-axis) and MetaPSICOV (Y-axis) in terms of TMscore and lDDT, respectively. lDDT is scaled to between 0 and 1.

Our method also generates much better contact-assisted models for the test proteins without many non-redundant sequence homologs. When the 219 of 579 test proteins with *Meff*≤500 are evaluated, the average TMscore of the top 1 models generated by our predicted contacts for the CASP11, CAMEO and membrane sets is 0.426, 0.365, and 0.397, respectively. By contrast, the average TMscore of the top 1 models generated by CCMpred-predicted contacts for the CASP11, CAMEO and membrane sets is 0.236, 0.214, and 0.241, respectively. The average TMscore of the top 1 models generated by MetaPSICOV-predicted contacts for the CASP11, CAMEO and membrane sets is 0.292, 0.272, and 0.274, respectively.

Contact-assisted models vs. template-based models

To compare the quality of our contact-assisted models and template-based models (TBMs), we built TBMs for all the test proteins using our training proteins as candidate templates. To generate TBMs for a test protein, we first run HHblits (with the UniProt20_2016 library) to generate an HMM file for the test protein, then run HHsearch with this HMM file to search for the best templates among the 6767 training proteins, and finally run MODELLER to build a TBM from each of the top 5 templates. Fig 4 shows the head-to-head comparison between our top 1 contact-assisted models and the top 1 TBMs on these three test sets in terms of both TMscore and lDDT. The average lDDT of our top 1 contact-assisted models is 45.7 while that of top 1 TBMs is only 20.7. When only the first models are evaluated, our contact-assisted models for the 76 CAMEO test proteins have an average TMscore 0.407 while the TBMs have an average TMscore 0.317. On the 105 CASP11 test proteins, the average TMscore of our contact-assisted models is 0.518 while that of the TBMs is only 0.393. On the 398 membrane proteins, the average TMscore of our contact-assisted models is 0.493 while that of the TBMs is only 0.149. Same trend is observed when top 5 models are compared. The average lDDT of our top 5 contact-assisted models is 46.4 while that of top 5 TBMs is only 24.0. On the 76 CAMEO test proteins, the average TMscore of our contact-assisted models is 0.431 while that of the TBMs is only 0.366. On the 105 CASP11 test proteins, the average TMscore of our contact-assisted models is 0.543 while that of the TBMs is only 0.441. On the 398 membrane proteins, the average

TMscore of our contact-assisted models is 0.516 while that of the TBMs is only 0.187. The low quality of TBMs further confirms that there is little redundancy between our training and test proteins (especially membrane proteins). This also indicates that our deep model does not predict contacts by simply copying from training proteins. That is, our method can predict contacts for a protein with a new fold.

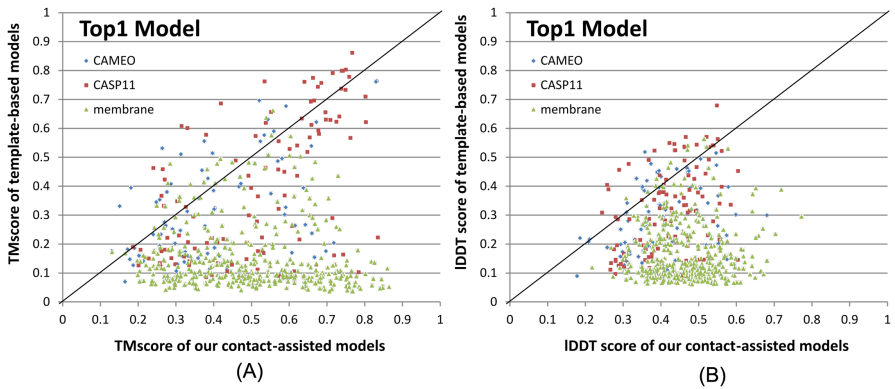


Figure 4. Comparison between our contact-assisted models of the three test sets and their template-based models in terms of (A) TMscore and (B) lDDT score. The top 1 models are evaluated.

Further, when the best of top 5 models are considered for all the methods, our contact-assisted models have TMscore>0.5 for 24 of the 76 CAMEO targets while TBMs have TMscore>0.5 for only 18 of them. Our contact-assisted models have TMscore >0.5 for 67 of the 105 CASP11 targets while TBMs have TMscore>0.5 for only 44 of them. Our contact-assisted models have TMscore>0.5 for 208 of the 398 membrane proteins while TBMs have TMscore >0.5 for only 10 of them. Our contact-assisted models for membrane proteins are much better than their TBMs because there is little similarity between the 6767 training proteins and the 398 test membrane proteins. When the 219 test proteins with ≤500 non-redundant sequence homologs are evaluated, the average TMscore of the TBMs is 0.254 while that of our contact-assisted models is 0.421. Among these 219 proteins, our contact-assisted models have TMscore>0.5 for 72 of them while TBMs have TMscore>0.5 for only 17 of them.

The above results imply that 1) when a query protein has no close templates, our contact-assisted modeling may work better than template-based modeling; 2) contact-assisted modeling shall be particularly useful for membrane proteins; and 3) our deep learning model does not predict contacts by simply copying contacts from the training proteins since our predicted contacts may result in much better 3D models than homology modeling.

Blind test in CAMEO

We have implemented our algorithm as a fully-automated contact prediction web server (http://raptorx.uchicago.edu/ContactMap/) and in September 2016 started to blindly test it through the weekly live benchmark CAMEO (http://www.cameo3d.org/). CAMEO is operated by the Schwede group, with whom we have never collaborated. CAMEO can be interpreted as a fully-automated CASP, but has a smaller number (~30) of participating servers since many CASP-participating servers are not fully automated and thus, cannot handle the large number of test targets used by CAMEO. Nevertheless, the CAMEO participants include some well-known servers such as Robetta[35], Phyre[36], RaptorX[37], Swiss-Model[38] and HHpred[39]. Meanwhile Robetta employs both ab initio folding and template-based modeling while the latter four employ mainly template-based modeling. Every weekend CAMEO sends test sequences to participating servers for prediction and then evaluates 3D models collected from servers. The test proteins used by CAMEO have no publicly available native structures until CAMEO finishes collecting models from participating servers.

From 9/3/2016 to 10/31/2016, CAMEO in total released 41 hard targets. Although classified as hard by CAMEO, some of them may have distantly-related templates. Table 5 lists the contact prediction accuracy of our server in the blind CAMEO test as compared to the other methods. Again, our method outperforms the others by a very large margin no matter how many contacts are evaluated. The CAMEO evaluation of our contact-assisted 3D models is available at the CAMEO web site. You will need to register CAMEO in order to see all the detailed results of our contact server (ID: server60). Although our server currently build 3D models using only top predicted contacts without any force fields and fragment assembly procedures, our server predicts 3D models with

TMscore>0.5 for 28 of the 41 targets and TMscore>0.6 for 16 of them. The average TMscore of the best of top 5 models built from the contacts predicted by our server, CCMpred and MetaPSICOV is 0.535, 0.316 and 0.392, respectively. See Fig 5 for the detailed comparison of the 3D models generated by our server, CCMpred and MetaPSICOV. Our server has also successfully folded 4 targets with a new fold plus one released in November 2016 (5flgB). See Table 6 for a summary of our prediction results of these targets and the below subsections for a detailed analysis. Among these targets, 5f5pH is particularly interesting since it has a sequence homolog in PDB but adopting a different conformation. That is, a template-based technique cannot obtain a good prediction for this target.

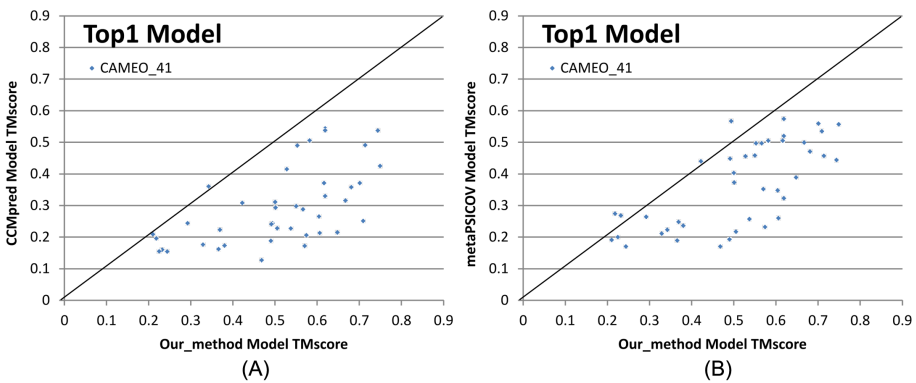


Figure 5: Quality comparison (measured by TMscore) of contact-assisted models generated by our server, CCMpred and MetaPSICOV on the 41 CAMEO hard targets.

(A) our server (X-axis) vs. CCMpred and (B) our server (X-axis) vs. MetaPSICOV.

Table 5: Contact prediction accuracy on 41 recent CAMEO hard targets

Method	Short				Medium				Long			
	L/10	L/5	L/2	L	L/10	L/5	L/2	L	L/10	L/5	L/2	L
EVfold	0.20	0.15	0.11	0.08	0.25	0.19	0.12	0.09	0.33	0.29	0.21	0.15
PSICOV	0.21	0.16	0.11	0.08	0.26	0.20	0.11	0.08	0.33	0.30	0.21	0.15
plmDCA	0.26	0.19	0.12	0.09	0.28	0.23	0.13	0.09	0.38	0.33	0.24	0.17
Gremlin	0.25	0.18	0.12	0.08	0.29	0.22	0.13	0.09	0.37	0.34	0.25	0.17
CCMpred	0.24	0.18	0.12	0.08	0.29	0.22	0.13	0.09	0.37	0.34	0.24	0.17
MetaPSICOV	0.53	0.43	0.27	0.17	0.51	0.42	0.28	0.19	0.60	0.54	0.40	0.30
Our server	0.67	0.52	0.32	0.20	0.68	0.58	0.38	0.24	0.82	0.75	0.62	0.46

doi:10.1371/journal.pcbi.1005324.t005

Table 6: A summary of our blind prediction results on 5 CAMEO hard targets with a new fold

Target	CAMEO ID	Type	Len	Meff	Method	RMSD(Å)	TMscore
2nc8A	2016-09-10_00000002_1	β	182	250	Our server	6.5	0.61
					Best of the others	12.18	0.47
5dcjA	2016-09-17_00000018_1	α+β	125	180	Our server	7.9	0.52
					Best of the others	10.0	0.53
5djeB	2016-09-24_00000052_1	α	140	330	Our server	5.81	0.65
					Best of the others	14.98	0.34
5f5pH	2016-10-15_00000047_1	α	217	65	Our server	4.21	0.71
					Best of the others	>40.0	0.48
5flgB	2016-11-12_00000046_1	α/β	260	113	Our server	7.12	0.61
					Best of the others	16.9	0.35

doi:10.1371/journal.pcbi.1005324.t006

Among these 41 hard targets, there are five multi-domain proteins: 5idoA, 5hmqF, 5b86B, 5b2gG and 5cylH. Table 7 shows that the average contact prediction accuracy of our method on these 5 multi-domain proteins is much better than the others. For multi-domain proteins, we use a superposition-independent score lDDT instead of TMscore to measure the quality of a 3D model. As shown in Table 8, the 3D models built by our server from predicted contacts have much better lDDT score than those built from CCMpred and MetaPSICOV.

Table 7: The average contact prediction accuracy of our method and the others on 5 multi-domain proteins among the 41 CAMEO hard targets

Method	Short				Medium				Long			
	L/10	L/5	L/2	L	L/10	L/5	L/2	L	L/10	L/5	L/2	L
EVfold	0.17	0.13	0.09	0.07	0.18	0.12	0.08	0.06	0.54	0.40	0.26	0.18
PSICOV	0.27	0.18	0.10	0.07	0.26	0.17	0.11	0.07	0.62	0.49	0.31	0.20
plmDCA	0.29	0.23	0.11	0.07	0.32	0.22	0.11	0.08	0.66	0.51	0.34	0.22
Gremlin	0.30	0.24	0.12	0.08	0.32	0.22	0.12	0.07	0.67	0.52	0.36	0.23
CCMpred	0.30	0.23	0.12	0.08	0.32	0.22	0.12	0.08	0.66	0.51	0.35	0.23
MetaPSICOV	0.52	0.37	0.21	0.14	0.32	0.26	0.16	0.11	0.72	0.58	0.41	0.26
Our method	0.74	0.58	0.33	0.19	0.68	0.55	0.33	0.20	0.96	0.91	0.76	0.57

doi:10.1371/journal.pcbi.1005324.t007

Table 8: The lDDT score of the 3D models built for the 5 multi-domain proteins using predicted contacts

Targets	Length	CCMpred	MetaPSICOV	Our
5idoA	512	23.67	24.24	36.83
5hmqF	637	24.84	25.91	33.16
5b86B	600	29.88	32.85	42.58
5b2gG	364	28.52	30.47	47.91
5cylH	370	22.21	23.37	30.62

doi:10.1371/journal.pcbi.1005324.t008

Study of CAMEO target 2nc8A (CAMEO ID: 2016-09-10_00000002_1, PDB ID:2nc8)

On September 10, 2016, CAMEO released two hard test targets for structure prediction. Our contact server successfully folded the hardest one (PDB ID: 2nc8), a mainly β protein of 182 residues. Table 9 shows that our server produced a much better contact prediction than CCMpred and MetaPSICOV. CCMpred has very low accuracy since HHblits detected only ~250 non-redundant sequence homologs for this protein, i.e., its Meff = 250. Fig 6 shows the predicted contact maps and their overlap with the native. MetaPSICOV fails to predict many long-range contacts while CCMpred introduces too many false positives.

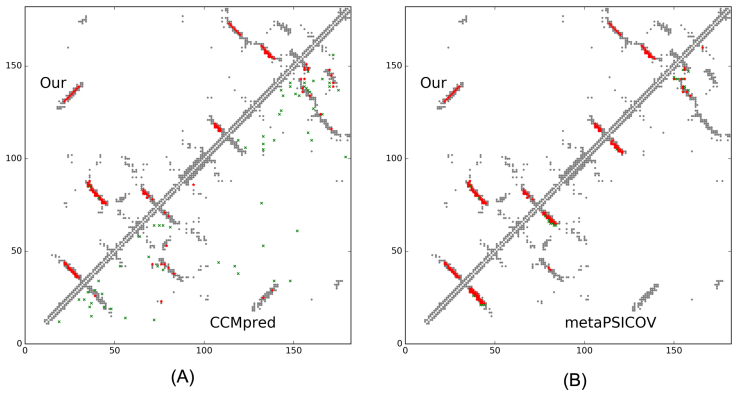


Figure 6: Overlap between top L/2 predicted contacts (in red or green) and the native contact map (in grey) for CAMEO target 2nc8A.

Red (green) dots indicate correct (incorrect) prediction. (A) The comparison between our prediction (in upper-left triangle) and CCMpred (in lower-right triangle). (B) The comparison between our prediction (in upper-left triangle) and MetaPSICOV (in lower-right triangle).

Table 9: The long- and medium-range contact prediction accuracy of our method, MetaPSICOV and CCMpred on the CAMEO target 2nc8A

	Long-range accuracy				Medium-range accuracy			
	L	L/2	L/5	L/10	L	L/2	L/5	L/10
Our method	0.764	0.923	0.972	1.0	0.450	0.769	0.972	1.0
MetaPSICOV	0.258	0.374	0.556	0.667	0.390	0.626	0.806	0.944
CCMpred	0.165	0.231	0.389	0.333	0.148	0.187	0.167	0.222

doi:10.1371/journal.pcbi.1005324.t009

The 3D model submitted by our contact server has TMscore 0.570 (As of September 16, 2016, our server submits only one 3D model for each test protein) and the best of our top 5 models has TMscore 0.612 and RMSD 6.5Å. Fig 7 shows that the beta strands of our predicted model (red) matches well with the native (blue). To examine the superimposition of our model with its native structure from various angles, please see http://raptorx.uchicago.edu/DeepAlign/75097011/. By contrast, the best of top 5 models built by CNS from CCMpred- and MetaPSICOV-predicted contacts have TMscore 0.206 and 0.307, respectively, and RMSD 15.8Å and 14.2Å, respectively. The best TMscore obtained by the other CAMEO-participating servers is only 0.47 (Fig 8). Three top-notch servers HHpred, RaptorX and Robetta only submitted models with TMscore≤0.30. According to Xu and Zhang [34], a 3D model with TMscore<0.5 is unlikely to have a correct fold while a model with TMscore≥0.6 surely has a correct fold. That is, our contact server predicted a correct fold for this test protein while the others failed to.

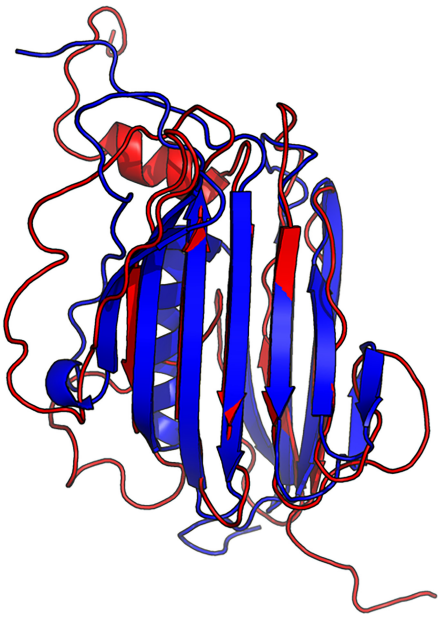


Figure 7: Superimposition between our predicted model (red) and its native structure (blue) for the CAMEO test protein (PDB ID 2nc8 and chain A).

Server Name	Predictions	From	To	Cov. (%)	GDT_HA	RMSD	MaxSub	TMScore
Server 60	Model 1	1	182	100	26.46	7.69	0.37	0.57
IntFOLD4-TSb	Model 1	1	182	100	19.88	12.18	0.28	0.47
Server 56	Model 1	1	182	100	19.88	12.18	0.28	0.47
IntFOLD4-TSb	Model 2	1	182	100	19.44	19.77	0.26	0.41
Server 56	Model 2	1	182	100	19.44	19.77	0.26	0.41
Server 45	Model 1	1	182	100	19.01	21.83	0.23	0.36
Server 57	Model 1	1	182	100	12.43	13.54	0.17	0.36
Server 57	Model 2	1	182	100	12.43	13.54	0.17	0.36
Server 57	Model 3	1	182	100	12.43	13.54	0.17	0.36
Server 57	Model 4	1	182	100	12.43	13.54	0.17	0.36
Server 57	Model 5	1	182	100	11.70	16.85	0.16	0.34
IntFOLD4-TSb	Model 5	1	182	100	11.55	16.53	0.15	0.33
Robetta	Model 3	10	182	95	13.74	15.59	0.16	0.33
HHpredB	Model 1	1	182	100	12.87	20.72	0.17	0.30
RaptorX	Model 1	1	182	100	12.57	17.55	0.16	0.30

Figure 8: The list of top models submitted by CAMEO servers for 2nc8A and their quality scores.

The rightmost column displays the TMscore of submitted models. Server60 is our contact web server.

This test protein represents almost a novel fold. Our in-house structural homolog search tool DeepSearch[40] cannot identify structurally very similar proteins in PDB70 (created right before September 10, 2016) for this target. PDB70 is a set of representative structures in PDB, in which any two share less than 70% sequence identity. DeepSearch returned two weakly similar proteins 4kx7A and 4g2aA, which have TMscore 0.521 and 0.535 with the native structure of the target, respectively, and TMscore 0.465 and 0.466 with our best model, respectively. This is consistent with the fact that none of the template-based servers in CAMEO submitted a model with TMscore>0.5. We cannot find structurally similar proteins in PDB70 for our best model either; the best TMscore between PDB70 and our best model is only 0.480. That is, the models predicted by our method are not simply copied from the solved structures in PDB, and our method can indeed fold a relatively large β protein with a novel fold.

Study of CAMEO target 5dcjA (CAMEO ID: 2016-09-17_00000018_1, PDB ID:5dcj).

This target was released by CAMEO on September 17, 2016. It is an α+β sandwich protein of 125 residues. The four beta sheets of this protein are wrapped by one and three alpha helixes at two sides. Table 10 shows that our server produced a much better contact prediction than CCMpred and MetaPSICOV. Specifically, the contact map predicted by our method has L/2 long-range accuracy 0.645 while that by CCMpred and MetaPSICOV has L/2 accuracy only 0.05 and 0.194, respectively. CCMpred has very low accuracy since HHblits can only find ~180 non-redundant sequence homologs for this protein, i.e., its Meff = 180. Fig 9 shows the predicted contact maps and their overlap with the native. Both CCMpred and MetaPSICOV failed to predict some long-range contacts.

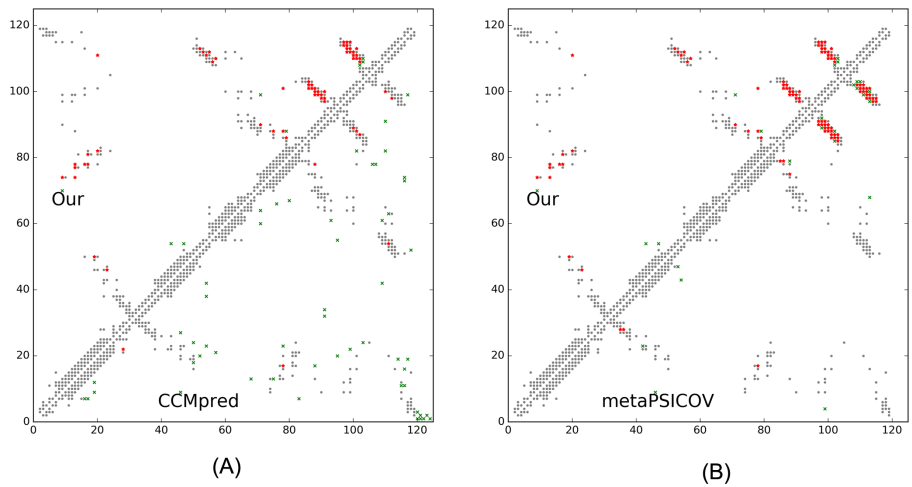


Figure 9: Overlap between top L/2 predicted contacts (in red or green) and the native contact map (in grey) for CAMEO target 5dcjA.

Red (green) dots indicate correct (incorrect) prediction. (A) The comparison between our prediction (in upper-left triangle) and CCMpred (in lower-right triangle). (B) The comparison between our prediction (in upper-left triangle) and MetaPSICOV (in lower-right triangle).

Table 10: The long- and medium-range contact prediction accuracy of our method, MetaPSICOV and CCMpred on the CAMEO target 5dcjA

	Long range				Medium range			
	L	L/2	L/5	L/10	L	L/2	L/5	L/10
Our method	0.456	0.645	0.88	0.833	0.36	0.645	0.92	1.0
metaPSICOV	0.144	0.194	0.32	0.25	0.344	0.532	0.8	1.0
CCMpred	0.05	0.05	0.08	0.08	0.1	0.129	0.12	0.25

doi:10.1371/journal.pcbi.1005324.t010

The first 3D model submitted by our contact server has TMscore 0.50 and the best of our 5 models has TMscore 0.52 and RMSD 7.9Å. The best of top 5 models built by CNS from CCMpred- and MetaPSICOV-predicted contacts have TMscore 0.243 and 0.361, respectively. Fig 10(A) shows that all the beta strands and the three surrounding alpha helices of our predicted model (in red) matches well with the native structure (blue), while the models from CCMpred (Fig 10(B)) and MetaPSICOV (Fig 10(C)) do not have a correct fold. To examine the superimposition of our model with its native structure from various angles, please see http://raptorx.uchicago.edu/DeepAlign/92913404/.

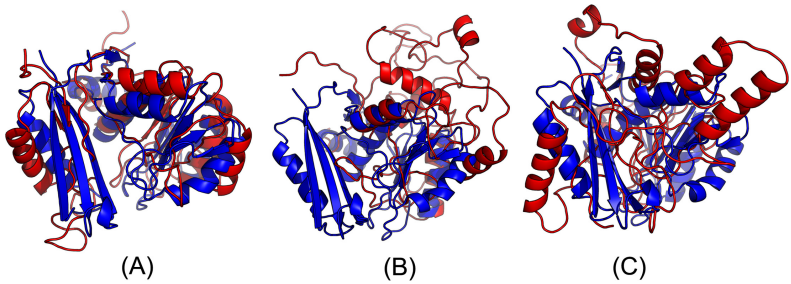


Figure 10: Superimposition between the predicted models (red) and the native structure (blue) for the CAMEO test protein (PDB ID 5dcj and chain A).

The models are built by CNS from the contacts predicted by **(A)** our method, **(B)**CCMpred, and **(C)** MetaPSICOV.

In terms of TMscore, our models have comparable quality to Robetta, but better than the other servers (Fig 11). In terms of RMSD and lDDT-Cα score, our models are better than all the others. In particular, our method produced better models than the popular homology modeling server HHpredB and our own template-based modeling server RaptorX, which submitted models with TMscore≤0.45.

Server Name	Predictions	From	To	Cov. (%)	GDT_HA	RMSD	MaxSub	TMScore
Robetta	Model 3	1	125	100	31.78	10.77	0.42	0.53
Robetta	Model 5	1	125	100	33.69	12.81	0.43	0.52
Server 60	Model 3	1	125	100	30.09	7.89	0.40	0.52
Server 60	Model 4	1	125	100	29.45	8.05	0.37	0.52
Server 60	Model 5	1	125	100	28.60	9.24	0.37	0.52
Server 60	Model 2	1	125	100	28.81	9.03	0.37	0.51
Robetta	Model 1	1	125	100	29.66	10.39	0.41	0.50
Server 60	Model 1	1	125	100	27.97	8.93	0.35	0.50
IntFOLD4-TSb	Model 1	1	125	100	28.39	10.06	0.38	0.48
Robetta	Model 2	1	125	100	27.97	11.59	0.36	0.48
Server 56	Model 1	1	125	100	28.39	10.06	0.38	0.48
Robetta	Model 4	1	125	100	29.24	10.73	0.37	0.47
IntFOLD4-TSb	Model 3	1	125	100	25.42	10.21	0.35	0.46
Princeton_TEMPLATE	Model 2	1	125	100	25.00	10.30	0.34	0.46
Princeton_TEMPLATE	Model 4	1	125	100	23.73	10.55	0.33	0.46

Figure 11: The list of top models submitted by CAMEO-participating servers for 5dcjA and their quality scores.

The rightmost column displays the TMscore of submitted models. Server60 is our contact web server.

This target represents a novel fold. Searching through PDB70 created right before September 17, 2016 by our in-house structural homolog search tool DeepSearch cannot identify structurally similar proteins for this target. The most structurally similar proteins are 3lr5A and 5ereA, which have TMscore 0.431 and 0.45 with the target, respectively. This is consistent with the fact that none of the template-based servers in CAMEO can predict a good model for this target. By contrast, our contact-assisted model has TMscore 0.52, which is higher than all the template-based models.

Study of CAMEO target 5djeB (CAMEO ID: 2016-09-24_00000052_1, PDB ID: 5dje).

This target was released on September 24, 2016. It is an alpha protein of 140 residues with a novel fold. Table 11 shows that our server produced a much better contact prediction than CCMpred and MetaPSICOV. Specifically, the contact map predicted by our method has L/5 and

L/10 long-range accuracy 50.0% and 71.4%, respectively, while that by CCMpred and MetaPSICOV has L/5 and L/10 accuracy less than 30%. HHblits can only find ~330 non-redundant sequence homologs for this target, i.e., its Meff = 330. Fig 12 shows the predicted contact maps and their overlap with the native. Both CCMpred and metaPSICOV failed to predict some long-range contacts.

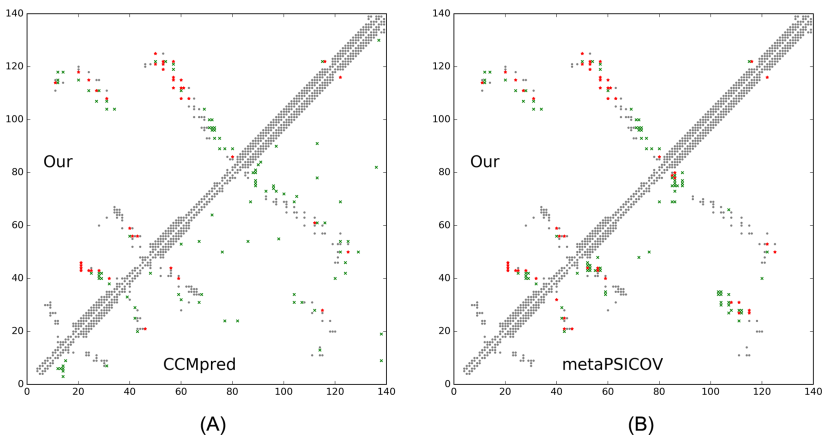


Figure 12: Overlap between top L/2 predicted contacts (in red and green) and the native contact map (in grey) for CAMEO target 5djeB.

Red (green) dots indicate correct (incorrect) prediction. (A) The comparison between our prediction (in upper-left triangle) and CCMpred (in lower-right triangle). (B)The comparison between our prediction (in upper-left triangle) and MetaPSICOV (in lower-right triangle).

Table 11: The long- and medium-range contact prediction accuracy of our method, MetaPSICOV and CCMpred on the CAMEO target 5djeB

	Long range accuracy				Medium range accuracy			
	L	L/2	L/5	L/10	L	L/2	L/5	L/10
Our method	0.300	0.357	0.500	0.714	0.186	0.229	0.357	0.357
metaPSICOV	0.193	0.200	0.286	0.286	0.100	0.143	0.214	0.286
CCMpred	0.079	0.114	0.107	0.214	0.036	0.029	0.071	0.143

doi:10.1371/journal.pcbi.1005324.t011

The first 3D model submitted by our contact server has TMscore 0.65, while the best of our 5 models has TMscore 0.65 and RMSD 5.6Å. By contrast, the best of top 5 models built by CNS from CCMpred- and MetaPSICOV-predicted contacts have TMscore 0.404 and 0.427, respectively. Fig 13(A) shows that all the four alpha helices of our predicted model (in red) matches well with the native structure (blue), while the

models from CCMpred (Fig 13(B)) and MetaPSICOV (Fig 13(C)) fail to predict the 3rd long helix correctly. To examine the superimposition of our model with its native structure from various angles, please see http://raptorx.uchicago.edu/DeepAlign/26652330/. Further, all other CAMEO registered servers, including the top-notch servers such as HHpred, RaptorX, SPARKS-X, and RBO Aleph (template-based and ab initio folding) only submitted models with TMscore≤0.35, i.e., failed to predict a correct fold (Fig 14).

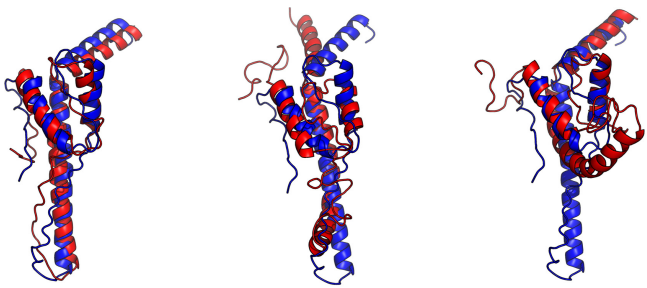


Figure 13: Superimposition between the predicted models (red) and the native structure (blue) for the CAMEO test protein (PDB ID 5dje and chain B).

The models are built by CNS from the contacts predicted by **(A)** our method, **(B)**CCMpred, and **(C)** MetaPSICOV.

Server Name	Predictions	From	To	Cov. (%)	GDT_HA	RMSD	MaxSub	TMScore
Server 60	Model 1	1	140	100	36.03	5.81	0.51	0.65
Server 60	Model 2	1	140	100	36.21	5.57	0.51	0.65
Server 60	Model 4	1	140	100	36.03	5.78	0.50	0.62
Server 60	Model 3	1	140	100	33.82	5.71	0.46	0.61
Server 60	Model 5	1	140	100	30.15	6.56	0.36	0.56
RBO Aleph	Model 5	1	140	100	22.06	17.28	0.26	0.34
RaptorX	Model 1	1	140	100	18.93	18.88	0.22	0.34
IntFOLD4-TSb	Model 3	1	140	100	22.98	18.17	0.26	0.33
RBO Aleph	Model 4	1	140	100	21.14	17.87	0.26	0.33
SPARKS-X	Model 3	1	140	100	18.75	21.95	0.22	0.32
RBO Aleph	Model 1	1	140	100	18.57	14.98	0.19	0.31
Server 45	Model 1	1	140	100	19.12	20.97	0.21	0.31
Server 55	Model 2	1	140	100	25.92	47.47	0.28	0.31
SPARKS-X	Model 2	1	140	100	15.99	20.33	0.17	0.28
SPARKS-X	Model 1	1	140	100	16.54	24.40	0.19	0.27

Figure 14: The list of top models submitted by CAMEO-participating servers for 5djeB and their quality scores.

The rightmost column displays the TMscore of submitted models. Server60 is our contact web server.

This target represents a novel fold. Searching through PDB70 created right before September 24, 2016 by our in-house structural homolog search tool DeepSearch cannot identify structurally similar proteins for this test protein. The most structurally similar proteins are 1u7lA and 4x5uA, which have TMscore 0.439 and 0.442 with the test protein, respectively. This is consistent with the fact that none of the template-based CAMEO-participating servers predicted a good model for this target. By contrast, our contact-assisted model has TMscore 0.65, much better than all the template-based models.

Study of CAMEO target 5f5pH (CAMEO ID: 2016-10-15_00000047_1, PDB ID: 5f5p).

On October 15, 2016, our contact web server successfully folded a very hard and also interesting CAMEO target (PDB ID: 5f5pH, CAMEO ID: 2016-10-15_00000047_1). This target is an alpha protein of 217 residues with four helices. Table 12 shows that our server produced a much better long-range contact prediction than CCMpred and MetaPSICOV. Specifically, our contact prediction has L/5 and L/10 long-range accuracy 76.7% and 95.2%, respectively, while MetaPSICOV has L/5 and L/10 accuracy less than 40%. This target has only ~65 non-redundant sequence homologs, i.e., its Meff = 65. The three methods have low L/k (k = 1, 2) medium-range accuracy because there are fewer than L/k native medium-range contacts while we use L/k as the denominator in calculating accuracy. As shown in Fig 15, CCMpred predicts too many false positives while MetaPSICOV predicts very few correct long-range contacts.

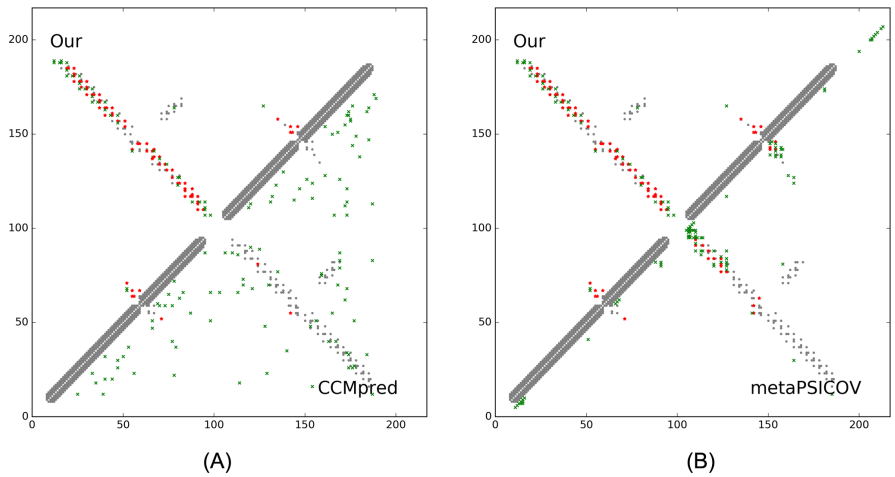


Figure 15: Overlap between top L/2 predicted contacts (in red and green) and the native contact map (in grey) for CAMEO target 5f5pH.

Red (green) dots indicate correct (incorrect) prediction. (A) The comparison between our prediction (in upper-left triangle) and CCMpred (in lower-right triangle). (B) The comparison between our prediction (in upper-left triangle) and MetaPSICOV (in lower-right triangle).

Table 12: The long- and medium-range contact prediction accuracy of our method, MetaPSICOV and CCMpred on the CAMEO target 5f5pH

	Long-range accuracy				Medium-range accuracy			
	L	L/2	L/5	L/10	L	L/2	L/5	L/10
Our server	0.382	0.602	0.767	0.952	0.041	0.083	0.209	0.381
metaPSICOV	0.161	0.250	0.326	0.476	0.041	0.083	0.163	0.190
CCMpred	0.032	0.037	0.047	0.048	0.009	0.019	0.023	0.032

doi:10.1371/journal.pcbi.1005324.t012

Our submitted 3D model has TMscore 0.71 and RMSD 4.21Å. By contrast, the best of top 5 models built by CNS from CCMpred- and MetaPSICOV-predicted contacts have TMscore 0.280 and 0.472, respectively. Fig 16(A) shows that our predicted model (in red) match well with the native structure (blue), while the model from CCMpred (Fig 16(B)) is completely wrong and the model from MetaPSICOV (Fig 16(C)) fails to place the 1st and 4th helices correctly. Please see http://raptorx.uchicago.edu/DeepAlign/14544627/ for the animated superimposition of our model with its native structure. As shown in the ranking list (Fig 17), all the other CAMEO-participating servers, including Robetta, HHpred,

RaptorX, SPARKS-X, and RBO Aleph (template-based and ab initio folding) only submitted models with TMscore≤0.48 and RMSD>43.82Å. Our contact server is the only one that predicted a correct fold for this target.

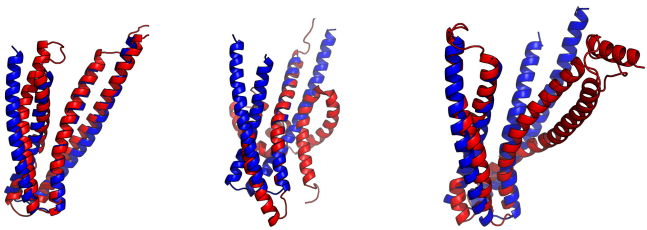


Figure 16: Superimposition between the predicted models (red) and the native structure (blue) for the CAMEO target 5f5pH.

The models are built by CNS from the contacts predicted by **(A)** our method, **(B)**CCMpred, and **(C)** MetaPSICOV.

Server Name	Predictions	From	To	Cov. (%)	GDT_HA	RMSD	MaxSub	TMScore
Server 60	Model 1	1	217	100	34.80	4.21	0.51	0.71
Server 60	Model 2	1	217	100	28.80	7.39	0.38	0.62
Server 60	Model 4	1	217	100	29.09	5.25	0.40	0.62
Server 60	Model 5	1	217	100	27.92	7.64	0.38	0.57
Server 60	Model 3	1	217	100	24.85	6.88	0.33	0.56
SWISS-MODEL	Model 1	10	189	82	36.55	50.09	0.43	0.48
Server 46	Model 1	10	189	82	36.99	50.09	0.44	0.48
Server 46	Model 2	10	189	82	36.70	50.10	0.43	0.48
Server 46	Model 3	10	189	82	36.55	50.10	0.43	0.48
Server 48	Model 1	10	189	82	36.99	50.09	0.44	0.48
Server 48	Model 2	10	189	82	36.70	50.10	0.43	0.48
Server 48	Model 3	10	189	82	36.55	50.10	0.43	0.48
Server 54	Model 1	10	189	82	36.55	50.09	0.43	0.48
Server 55	Model 1	1	217	100	36.40	49.95	0.43	0.48
Phyre2	Model 1	12	189	82	36.11	50.40	0.43	0.47

Figure 17: The list of top models submitted by CAMEO-participating servers for 5f5pH and their quality scores.

The rightmost column displays the TMscore of submitted models. Server60 is our contact web server.

To make sure our best model is not simply copied from the database of solved structures, we search our best model against PDB70 created right before October 15, 2016 using our in-house structural homolog search tool DeepSearch, which yields two weakly similar proteins 2yfaA and 4k1pA. They have TMscore 0.536 and 0.511 with our best model, respectively. This implies that our model is not simply copied from a solved structure in PDB.

We ran BLAST on this target against PDB70 and surprisingly, found one protein 3thfA with E-value 3E-16 and sequence identity 35%. In fact, 3thfA and 5f5pH are two SD2 proteins from Drosophila and Human[41], respectively. Although homologous, they adopt different conformations and oligomerizations. In particular, 3thfA is a dimer and each monomer adopts a fold consisting of three segmented anti-parallel coiled-coil[42], whereas 5f5pH is a monomer that consists of two segmented antiparallel coiled-coils[41]. Superimposing the Human SD2 monomer onto the Drosophila SD2 dimer shows that the former structure was located directly in between the two structurally identical halves of the latter structure (see Fig 18(A)). That is, if our method predicts the contacts of 5f5pH by simply copying from 3thfA, it would produce a wrong 3D model. By contrast, all the other CAMEO-participating servers produced a wrong prediction for this target by using 3thfA as the template.

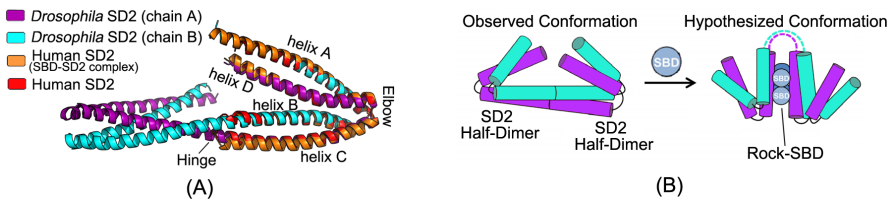


Figure 18.

(A) Structure superimposition of Drosophila SD2 and Human SD2. (B) Conformation change of Drosophila SD2 in binding with Rock-SBD.

Since SD2 protein may have conformational change when docking with Rock SBD protein, we check if the Drosophila SD2 monomer would change to a similar fold as the Human SD2 monomer or not. According to[41], the Human SD2 adopts a similar fold no matter whether it docks with the Rock SBD or not. According to [42], although the Drosophila

SD2 dimer may have conformational change in the presence of Rock, the change only occurs in the hinge regions, but not at the adjacent identical halves. That is, even conformational change happens, the Drosophila SD2 monomer would not resemble the Human SD2 monomer (Fig 18(B)).

Study of CAMEO target 5flgB (CAMEO ID: 2016-11-12_00000046_1, PDB ID: 5flgB).

This target was released by CAMEO on November 12, 2016 and not included in the abovementioned 41 CAMEO hard targets. This target is a unique α/β protein with 260 residues. Table 13 shows that our server produced a much better (long-range) contact prediction than CCMpred and MetaPSICOV. In particular, our predicted contact map has L, L/2, L/5 and L/10 long-range accuracy 71.1%, 86.1%, 96.1% and 100.0%, respectively, while CCMpred- and MetaPSICOV-predicted contacts have long-range accuracy less than 35% since there are only ~113 effective sequence homologs for this protein, i.e., its Meff = 113. Fig 19 shows that both CCMpred and MetaPSICOV generated many false positive contact predictions and failed to predict long-range contacts.

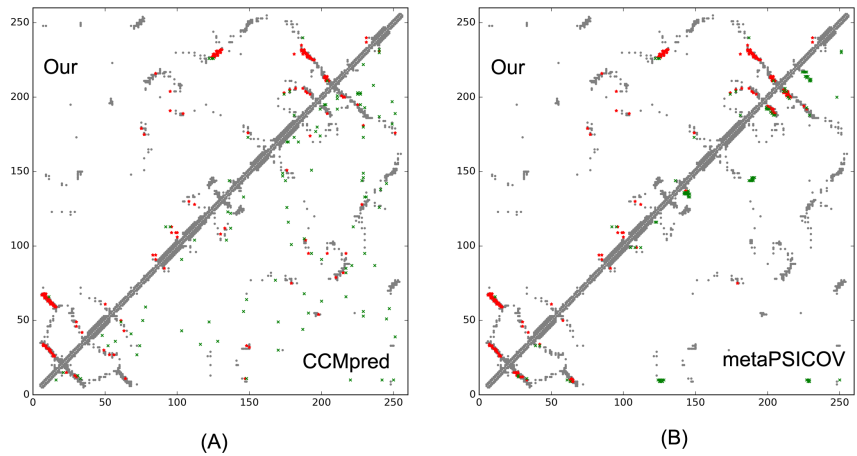


Figure 19: Overlap between predicted contacts (in red and green) and the native (in grey) for CAMEO target 5flgB.

Red (green) dots indicate correct (incorrect) prediction. Top L/2 predicted contacts by each method are shown. The left picture shows the comparison between our prediction (in upper-left triangle) and CCMpred (in lower-right triangle) and the right picture shows the comparison

between our prediction (in upper-left triangle) and MetaPSICOV (in lower-right triangle).

Table 13: The long- and medium-range contact prediction accuracy of our method, MetaPSICOV and CCMpred on the CAMEO target 5flgB

	Long-range accuracy				Medium-range accuracy			
	L	L/2	L/5	L/10	L	L/2	L/5	L/10
Our server	0.711	0.861	0.961	1.00	0.331	0.500	0.750	0.808
MetaPSICOV	0.208	0.262	0.269	0.288	0.242	0.285	0.442	0.615
CCMpred	0.165	0.184	0.308	0.346	0.150	0.215	0.346	0.385

doi:10.1371/journal.pcbi.1005324.t013

The 3D model submitted by our contact server has TMscore 0.61 and RMSD 7.12Å. The best of top 5 models built by CNS from CCMpred- and MetaPSICOV-predicted contacts have TMscore 0.240 and 0.267, respectively. Fig 20 shows that our method correctly modeled the overall fold, while CCMpred and MetaPSICOV failed. To examine the superimposition of our model with its native structure from various angles, please see http://raptorx.uchicago.edu/DeepAlign/12043612/. Furthermore, all the other CAMEO-participating servers, including the top-notch servers Robetta, HHpred, RaptorX, SPARKS-X, and RBO Aleph (template-based and ab initio folding), only submitted models with TMscore≤0.25 and RMSD>16.90Å (Fig 21). A 3D model with TMscore less than 0.25 does not have the correct fold while a model with TMscore≥0.6 very likely has a correct fold. That is, our contact server predicted a correct fold for this target while the others failed to.

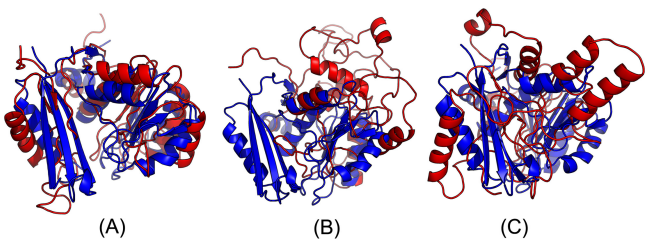


Figure 20: Superimposition between the predicted models (red) and the native structure (blue) for the CAMEO test protein 5flgB.

The models are built by CNS from the contacts predicted by **(A)** our method, **(B)**CCMpred, and **(C)** MetaPSICOV.

Server Name	Predictions	From	To	Cov. (%)	GDT_HA	RMSD	MaxSub	TMScore
Server 60	Model 1	1	260	100	21.53	7.12	0.30	0.61
Server 60	Model 3	1	260	100	19.15	6.83	0.27	0.60
Server 60	Model 2	1	260	100	17.16	8.07	0.23	0.56
Server 60	Model 4	1	260	100	15.48	6.85	0.21	0.56
Server 60	Model 5	1	260	100	17.26	9.19	0.24	0.56
Server 19	Model 5	1	260	100	11.81	19.72	0.14	0.35
IntFOLD4-TSb	Model 1	1	260	100	7.14	22.73	0.07	0.24
Server 56	Model 1	1	260	100	7.14	22.73	0.07	0.24
Server 19	Model 1	1	260	100	9.23	24.62	0.11	0.23
Server 19	Model 2	1	260	100	8.13	24.96	0.09	0.23
Princeton_TEMPLATE	Model 1	1	260	100	7.24	26.50	0.07	0.22
Princeton_TEMPLATE	Model 2	1	260	100	6.94	26.81	0.07	0.22
Princeton_TEMPLATE	Model 3	1	260	100	7.14	26.89	0.08	0.22
Princeton_TEMPLATE	Model 4	1	260	100	7.24	26.52	0.08	0.22
Princeton_TEMPLATE	Model 5	1	260	100	7.14	27.28	0.08	0.22

Figure 21: The list of top models submitted by CAMEO-participating servers for 5flgB and their quality scores.

The rightmost column displays the model TMscore. Server60 is our contact web server.

This test protein has a novel fold. Searching through PDB70 created right before November 12, 2016 by our in-house structural homolog search tool DeepSearch cannot identify any similar structures. The most structurally similar proteins returned by DeepSearch are 2fb5A and 5dwmA, which have TMscore 0.367 and 0.355 with the native structure of this target, respectively. This is consistent with the fact that all the other CAMEO-participating servers failed to predict a correct fold for this target.

In addition, on December 3, 2016 CAMEO released a Leucine rich repeat protein (PDB chain: 5f0pA) of 462 residues with a new fold and Meff = 212 (see http://www.cameo3d.org/sp/targets/1-week/target/2016-12-03/5F0P_A/). Our server submitted a model with TMscore 0.70 and RMSD 6.89Å while the best model submitted by the others has TMscore only 0.42. Here we do not describe this target in detail to avoid elongating our paper further.

CONCLUSION AND DISCUSSION

This paper has presented a new deep (supervised) learning method that can greatly improve protein contact prediction. Our method distinguishes itself from previous supervised learning methods in that we employ a concatenation of two deep residual neural networks to model sequence-contact relationship, one for modeling of sequential features (i.e., sequence profile, predicted secondary structure and solvent accessibility) and the other for modeling of pairwise features (e.g., coevolution information). Ultra-deep residual network is the latest breakthrough in computer vision and has demonstrated the best performance in the computer vision challenge tasks (similar to CASP) in 2015. Our method is unique in that we predict all contacts of a protein simultaneously, which allows us to easily model high-order residue correlation. By contrast, existing supervised learning methods predict if two residues form a contact or not independent of the other residue pairs. Our (blind) test results show that our method dramatically improves contact prediction, exceeding currently the best methods (e.g., CCMpred, Evfold, PSICOV and MetaPSICOV) by a very large margin. Even without using any force fields and fragment assembly, ab initio folding using our predicted contacts as restraints can yield 3D structural models of correct fold for many more test proteins. Further, our experimental results also show that our contact-assisted models are much better than template-based models built from the training proteins of our deep model. We expect that our contact prediction methods can help reveal much more biological insights for those protein families without solved structures and close structural homologs.

Our method outperforms ECA due to a couple of reasons. First, ECA predicts contacts using information only in a single protein family, while our method learns sequence-structure relationship from thousands of protein families. Second, ECA considers only pairwise residue correlation, while our deep network architecture can capture high-order residue correlation (or contact occurrence patterns) very well. Our method uses a subset of protein features used by MetaPSICOV, but outperforms MetaPSICOV mainly because we explicitly model contact patterns (or high-order correlation), which is enabled by predicting contacts of a single protein simultaneously. MetaPSICOV employs a 2-stage approach. The 1st stage predicts if there is a contact between a

pair of residues independent of the other residues. The 2nd stage considers the correlation between one residue pair and its neighboring pairs, but not in a very good way. In particular, the prediction errors in the 1st stage of MetaPSICOV cannot be corrected by the 2nd stage since two stages are trained separately. By contrast, we train all 2D convolution layers simultaneously (each layer is equivalent to one stage) so that later stages can correct prediction errors in early stages. In addition, a deep network can model much higher-order correlation and thus, capture information in a much larger context.

Our deep model does not predict contact maps by simply recognizing them from PDB, as evidenced by our experimental settings and results. First, we employ a strict criterion to remove redundancy so that there are no training proteins with sequence identity >25% or BLAST E-value <0.1 with any test proteins. Second, our contact-assisted models also have better quality than homology models, so it is unlikely that our predicted contact maps are simply copied from the training proteins. Third, our deep model trained by only soluble proteins works very well on membrane proteins. By contrast, the homology models built from soluble proteins for the membrane proteins have very low quality. Their average TMscore is no more than 0.17, which is the expected TMscore of any two randomly-chosen proteins. Finally, the blind CAMEO test indicates that our method successfully folded several targets with a new fold.

Our contact prediction method also performed the best in CASP12 in terms of the F1 score calculated on top L/2 long- and medium-range contacts of 38 free-modeling targets, although back then (May-July 2016) our method was not fully implemented. F1 score is a well-established and robust metric in evaluating the performance of a prediction method. Our method outperformed the 2nd best server iFold_1 by about 7.6% in terms of the total F1 score and the 3rd best server (i.e., an improved MetaPSICOV) by about 10.0%. Our advantage is even bigger when only top L/5 long- and medium-range contacts are evaluated. iFold_1 also used a deep neural network while the new MetaPSICOV used a deeper and wider network and more input features than the old version. This CASP result further confirms that deep learning can indeed improve protein contact prediction.

We have studied the impact of different input features. First of all, the co-evolution strength produced by CCMpred is very important. Without it, the top L/10 long-range prediction accuracy may drop by 0.15 for soluble proteins and more for membrane proteins. The larger performance degradation for membrane proteins is mainly because information learned from sequential features of soluble proteins is not very useful for membrane proteins. The depth of our deep model is as important as CCMpred, as evidenced by the fact that our deep method has much better accuracy than MetaPSICOV although we use a subset of protein features used by MetaPSICOV. Our test shows that a deep model with 9 and 30 layers have top L/10 accuracy ~0.1 and ~0.03 worse than a 60-layer model, respectively. This suggests that it is very important to model contact occurrence patterns (i.e., high-order residue correlation) by a deep architecture. The pairwise contact potential and mutual information may impact the accuracy by 0.02–0.03. The secondary structure and solvent accessibility may impact the accuracy by 0.01–0.02.

An interesting finding is that although our training set contains only ~100 membrane proteins, our model works well for membrane proteins, much better than CCMpred and MetaPSICOV. Even without using any membrane proteins in our training set, our deep models have almost the same accuracy on membrane proteins as those trained with membrane proteins. This implies that the sequence-structure relationship learned by our model from non-membrane proteins can generalize well to membrane protein contact prediction. This may be because that both soluble and membrane proteins share similar contact occurrence patterns in their contact maps and our deep method improves over previous methods by making a good use of contact occurrence patterns. We are going to study if we can further improve contact prediction accuracy of membrane proteins by including many more membrane proteins in the training set.

We may further improve contact prediction accuracy by enlarging the training set. First, the latest PDB25 has more than 10,000 proteins, which can provide many more training proteins than what we are using now. Second, when removing redundancy between training and test proteins, we may relax the BLAST E-value cutoff to 0.001 or simply drop it. This will improve the top L/k (k = 1, 2, 5, 10) contact prediction accuracy by 1–3% and accordingly the quality of the resultant 3D models by 0.01–0.02 in terms of TMscore. We may also improve the 3D model

quality by combining our predicted contacts with energy function and fragment assembly. For example, we may feed our predicted contacts to Rosetta to build 3D models. Compared to CNS, Rosetta makes use of energy function and more local structural restraints through fragment assembly and thus, shall result in much better 3D models. Finally, instead of predicting contacts, our deep learning model actually can predict inter-residue distance distribution (i.e., distance matrix), which provides finer-grained information than contact maps and thus, shall benefit 3D structure modeling more than predicted contacts.

Our model achieves pretty good performance when using around 60–70 convolutional layers. A natural question to ask is can we further improve prediction accuracy by using many more convolutional layers? In computer vision, it has been shown that a 1001-layer residual neural network can yield better accuracy for image-level classification than a 100-layer network (but no result on pixel-level labeling is reported). Currently we cannot apply more than 100 layers to our model due to insufficient memory of a GPU card (12G). We plan to overcome the memory limitation by extending our training algorithm to run on multiple GPU cards. Then we will train a model with hundreds of layers to see if we can further improve prediction accuracy or not.

METHOD

Deep learning model details

Residual network blocks.

Our network consists of two residual neural networks, each in turn consisting of some residual blocks concatenated together. Fig 22 shows an example of a residual block consisting of 2 convolution layers and 2 activation layers. In this figure, X_l and X_{l+1} are the input and output of the block, respectively. The activation layer conducts a simple nonlinear transformation of its input without using any parameters. Here we use the ReLU activation function [30] for such a transformation. Let $f(X_l)$ denote the result of X_l going through the two activation layers and the two convolution layers. Then, X_{l+1} is equal to $X_l + f(X_l)$. That is, X_{l+1} is a combination of X_l and its nonlinear transformation. Since $f(X_l)$ is equal

to the difference between X_{l+1} and X_l, *f* is called residual function and this network called residual network. In the first residual network, $X_l$and X_{l+1} represent sequential features and have dimension $L \times n_l$ and $L \times n_{l+1}$, respectively, where L is protein sequence length and n_l (n_{l+1}) can be interpreted as the number of features or hidden neurons at each position (i.e., residue). In the 2nd residual network, X_l and X_{l+1}represent pairwise features and have dimension $L \times L \times n_l$ and $L \times L \times n_{l+1}$, respectively, where n_l (n_{l+1}) can be interpreted as the number of features or hidden neurons at one position (i.e., residue pair). Typically, we enforce $n_l \leq n_{l+1}$ since one position at a higher level is supposed to carry more information. When $n_l < n_{l+1}$, in calculating $X_l + f(X_l)$ we shall pad zeros to X_l so that it has the same dimension as X_{l+1}. To speed up training, we also add a batch normalization layer [43] before each activation layer, which normalizes its input to have mean 0 and standard deviation 1. The filter size (i.e., window size) used by a 1D convolution layer is 17 while that used by a 2D convolution layer is 3×3 or 5×5. By stacking many residual blocks together, even if at each convolution layer we use a small window size, our network can model very long-range interdependency between input features and contacts as well as the long-range interdependency between two different residue pairs. We fix the depth (i.e., the number of convolution layers) of the 1D residual network to 6, but vary the depth of the 2D residual network. Our experimental results show that with ~60 hidden neurons at each position and ~60 convolution layers for the 2nd residual network, our model can yield pretty good performance. Note that it has been shown that for image classification a convolutional neural network with a smaller window size but many more layers usually outperforms a network with a larger window size but fewer layers. Further, a 2D convolutional neural network with a smaller window size also has a smaller number of parameters than a network with a larger window size. See https://github.com/KaimingHe/deep-residual-networks for some existing implementations of 2D residual neural network. However, they assume an input of fixed dimension, while our network needs to take variable-length proteins as input.

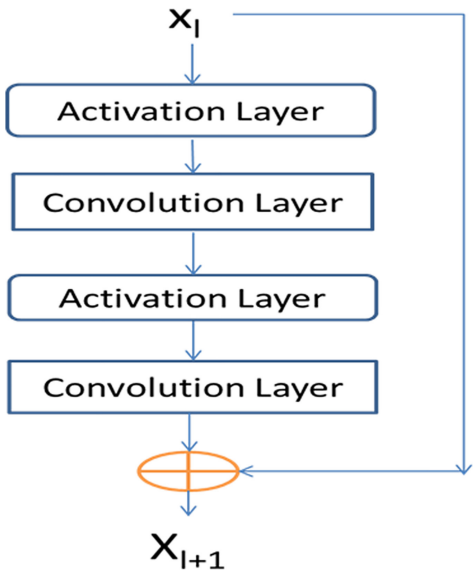


Figure 22: A building block of our residual network with X_l **and** X_{l+1} **being input and output, respectively.**

Each block consists of two convolution layers and two activation layers.

Our deep learning method for contact prediction is unique in at least two aspects. First, our model employs two multi-layer residual neural networks, which have not been applied to contact prediction before. Residual neural networks can pass both linear and nonlinear information from end to end (i.e., from the initial input to the final output). Second, we do contact prediction on the whole contact map by treating it as an individual image. In contrast, previous supervised learning methods separate the prediction of one residue pair from the others. By predicting contacts of a protein simultaneously, we can easily model long-range contact correlation and high-order residue correlation and long-range correlation between a contact and input features.

Convolutional operation.

Existing deep learning development toolkits such as Theano (http://deeplearning.net/software/theano/) and Tensorflow (https://www.tensorflow.org/) have provided an API (application programming interface) for convolutional operation so that we do not need to implement

it by ourselves. See http://deeplearning.net/tutorial/lenet.html and https://www.nervanasys.com/convolutional-neural-networks/ for a good tutorial of convolutional network. Please also see [44] for a detailed account of 1D convolutional network with application to protein sequence labeling. Roughly, a 1D convolution operation is de facto matrix-vector multiplication and 2D convolution can be interpreted similarly. Let X and Y (with dimensions L×m and L×n, respectively) be the input and output of a 1D convolutional layer, respectively. Let the window size be 2w+1 and s = (2w+1)m. The convolutional operator that transforms X to Y can be represented as a 2D matrix with dimension n×s, denoted as C. C is protein length-independent and each convolutional layer may have a different C. Let X_i be a submatrix of X centered at residue i ($1 \leq i \leq L$) with dimension (2w+1)×m, and Y_i be the i-th row of Y. We may calculate Y_i by first flattening X_i to a vector of length s and then multiplying C and the flattened X_i.

Conversion of sequential features to pairwise features.

We convert the output of the first module of our model (i.e., the 1-d residual neural network) to a 2D representation using an operation similar to outer product. Simply speaking, let $v = \{v_1, v_2, \ldots, v_i, \ldots, v_L\}$ be the final output of the first module where L is protein sequence length and v_i is a feature vector storing the output information for residue *i*. For a pair of residues *i* and *j*, we concatenate v_i, $v_{(i+j)/2}$ and v_j to a single vector and use it as one input feature of this residue pair. The input features for this pair also include mutual information, the EC information calculated by CCMpred and pairwise contact potential [45, 46].

Loss function.

We use maximum-likelihood method to train model parameters. That is, we maximize the occurrence probability of the native contacts (and non-contacts) of the training proteins. Therefore, the loss function is defined as the negative log-likelihood averaged over all the residue pairs of the training proteins. Since the ratio of contacts among all the residue pairs is very small, to make the training algorithm converge fast, we assign a larger weight to the residue pairs forming a contact. The weight is assigned such that the total weight assigned to contacts is approximately 1/8 of the number of non-contacts in the training set.

Regularization and optimization.

To prevent overfitting, we employ L_2-norm regularization to reduce the parameter space. That is, we want to find a set of parameters with a small L_2 norm to minimize the loss function, so the final objective function to be minimized is the sum of loss function and the L_2 norm of the model parameters (multiplied by a regularization factor). We use a stochastic gradient descent algorithm to minimize the objective function. It takes 20–30 epochs (each epoch scans through all the training proteins exactly once) to obtain a very good solution. The whole algorithm is implemented by Theano [47] and mainly runs on GPU.

Training and dealing with proteins of different lengths.

Our network can take as input variable-length proteins. We train our deep network in a minibatch mode, which is routinely used in deep learning. That is, at each training iteration, we use a minibatch of proteins to calculate gradient and update the model parameters. A minibatch may have one or several proteins. We sort all training proteins by length and group proteins of similar lengths into minibatches. Considering that most proteins have length up to 600 residues, proteins in a minibatch often have the same length. In the case that they do not, we add zero padding to shorter proteins. Our convolutional operation is protein-length independent, so two different minibatches are allowed to have different protein lengths. We have tested minibatches with only a single protein or with several proteins. Both work well. However, it is much easier to implement minibatches with only a single protein.

Since our network can take as input variable-length lengths, we do not need to cut a long protein into segments in predicting contact maps. Instead we predict all contacts of a protein chain simultaneously. There is no need to use zero padding when only a single protein is predicted in a batch. Zero padding is needed only when several proteins of different lengths are predicted in a batch.

Training and test data

Our test data includes the 150 Pfam families [5], 105 CASP11 test proteins, 76 hard CAMEO test proteins released in 2015 and 398 membrane proteins. All test membrane proteins have length no more

than 400 residues and any two membrane proteins share less than 40% sequence identity. For the CASP test proteins, we use the official domain definitions, but we do not parse a CAMEO or membrane protein into domains.

Our training set is a subset of PDB25 created in February 2015, in which any two proteins share less than 25% sequence identity. We exclude a protein from the training set if it satisfies one of the following conditions: (i) sequence length smaller than 26 or larger than 700, (ii) resolution worse than 2.5Å, (iii) has domains made up of multiple protein chains, (iv) no DSSP information, and (v) there is inconsistency between its PDB, DSSP and ASTRAL sequences [48]. To remove redundancy with the test sets, we exclude any training proteins sharing >25% sequence identity or having BLAST E-value <0.1 with any test proteins. In total there are 6767 proteins in our training set, from which we have trained 7 different models. For each model, we randomly sampled ~6000 proteins from the training set to train the model and used the remaining proteins to validate the model and determine the hyper-parameters (i.e., regularization factor). The final model is the average of these 7 models.

Protein features

We use similar but fewer protein features as MetaPSICOV. In particular, the input features include protein sequence profile (i.e., position-specific scoring matrix), predicted 3-state secondary structure and 3-state solvent accessibility, direct co-evolutionary information generated by CCMpred, mutual information and pairwise potential [45, 46]. To derive these features, we need to generate MSA (multiple sequence alignment). For a training protein, we run PSI-BLAST (with E-value 0.001 and 3 iterations) to search the NR (non-redundant) protein sequence database dated in October 2012 to find its sequence homologs, and then build its MSA and sequence profile and predict other features (i.e., secondary structure and solvent accessibility). Sequence profile is represented as a 2D matrix with dimension L×20 where L is the protein length. Predicted secondary structure is represented as a 2D matrix with dimension L××3 (each entry is a predicted score or probability), so is the predicted solvent accessibility. Concatenating them together, we have a 2D matrix with dimension L×26, which is the input of our 1D residual network.

For a test protein, we generate four different MSAs by running HHblits [39] with 3 iterations and E-value set to 0.001 and 1, respectively, to search through the uniprot20 HMM library released in November 2015 and February 2016. From each individual MSA, we derive one sequence profile and employ our in-house tool RaptorX-Property [49] to predict the secondary structure and solvent accessibility accordingly. That is, for each test protein we generate 4 sets of input features and accordingly 4 different contact predictions. Then we average these 4 predictions to obtain the final contact prediction. This averaged contact prediction is about 1–2% better than that predicted from a single set of features. Although currently there are quite a few packages that can generate direct evolutionary coupling information, we only employ CCMpred to do so because it runs fast on GPU [4].

Programs to compare and evaluation metrics

We compare our method with PSICOV [5], Evfold [6], CCMpred [4], plmDCA, Gremlin, and MetaPSICOV [9]. The first 5 methods conduct pure DCA while MetaPSICOV employs supervised learning. MetaPSICOV [9] performed the best in CASP11 [31]. CCMpred, plmDCA, Gremlin perform similarly, but better than PSICOV and Evfold. All the programs are run with parameters set according to their respective papers. We evaluate the accuracy of the top L/k (k= 10, 5, 2, 1) predicted contacts where L is protein sequence length. The prediction accuracy is defined as the percentage of native contacts among the top L/k predicted contacts. We also divide contacts into three groups according to the sequence distance of two residues in a contact. That is, a contact is short-, medium- and long-range when its sequence distance falls into [6, 11], [12, 23], and ≥24, respectively.

Calculation of meff

Meff measures the amount of homologous information in an MSA (multiple sequence alignment). It can be interpreted as the number of non-redundant sequence homologs in an MSA when 70% sequence identity is used as cutoff. To calculate Meff, we first calculate the sequence identity between any two proteins in the MSA. Let a binary variable S_{ij} denote the similarity between two protein sequences i and j. S_{ij} is equal to 1 if and only if the sequence identity between i and j is at least 70%. For a protein

i, we calculate the sum of S_{ij} over all the proteins (including itself) in the MSA and denote it as S_i. Finally, we calculate Meff as the sum of $1/S_i$ over all the protein sequences in this MSA.

3D model construction by contact-assisted folding

We use a similar approach as described in [11] to build the 3D models of a test protein by feeding predicted contacts and secondary structure to the Crystallography & NMR System (CNS) suite [32]. We predict secondary structure using our in-house tool RaptorX-Property [49] and then convert it to distance, angle and h-bond restraints using a script in the Confold package [11]. For each test protein, we choose top 2L predicted contacts (L is sequence length) no matter whether they are short-, medium- or long-range and then convert them to distance restraints. That is, a pair of residues predicted to form a contact is assumed to have distance between 3.5Å and 8.0 Å. In current implementation, we do not use any force fields to help with folding. We generate twenty 3D structure models using CNS and select top 5 models by the NOE score yielded by CNS[32]. The NOE score mainly reflects the degree of violation of the model against the input constraints (i.e., predicted secondary structure and contacts). The lower the NOE score, the more likely the model has a higher quality. When CCMpred- and MetaPSICOV-predicted contacts are used to build 3D models, we also use the secondary structure predicted by RaptorX-Property to warrant a fair comparison.

Template-based modeling (TBM) of the test proteins

To generate template-based models (TBMs) for a test protein, we first run HHblits (with the UniProt20_2016 library) to generate an HMM file for the test protein, then run HHsearch with this HMM file to search for the best templates among the 6767 training proteins of our deep learning model, and finally run MODELLER to build a TBM from each of the top 5 templates.

AUTHOR CONTRIBUTIONS

- **Conceptualization:** JX.
- **Data curation:** SW.
- **Formal analysis:** SW RZ JX.

- **Funding acquisition:** JX.
- **Investigation:** JX SW SS.
- **Methodology:** JX.
- **Project administration:** JX.
- **Resources:** JX.
- **Software:** JX SS SW ZL.
- **Supervision:** JX.
- **Validation:** JX SW SS RZ.
- **Visualization:** SW.
- **Writing – original draft:** JX SW.
- **Writing – review & editing:** JX.

REFERENCES

1. Kim DE, DiMaio F, Yu-Ruei Wang R, Song Y, Baker D. One contact for every twelve residues allows robust and accurate topology-level protein structure modeling. Proteins: Structure, Function, and Bioinformatics. 2014;82(S2):208–18.
2. de Juan D, Pazos F, Valencia A. Emerging methods in protein co-evolution. Nature Reviews Genetics. 2013;14(4):249–61. pmid:23458856
3. Weigt M, White RA, Szurmant H, Hoch JA, Hwa T. Identification of direct residue contacts in protein-protein interaction by message passing. P Natl Acad Sci USA. 2009;106(1):67–72.
4. Seemayer S, Gruber M, Söding J. CCMpred—fast and precise prediction of protein residue–residue contacts from correlated mutations. Bioinformatics. 2014;30(21):3128–30. pmid:25064567
5. Jones DT, Buchan DW, Cozzetto D, Pontil M. PSICOV: precise structural contact prediction using sparse inverse covariance estimation on large multiple sequence alignments. Bioinformatics. 2012;28(2):184–90. pmid:22101153
6. Marks DS, Colwell LJ, Sheridan R, Hopf TA, Pagnani A, Zecchina R, et al. Protein 3D structure computed from evolutionary sequence variation. PloS one. 2011;6(12):e28766. pmid:22163331
7. Ekeberg M, Hartonen T, Aurell E. Fast pseudolikelihood

maximization for direct-coupling analysis of protein structure from many homologous amino-acid sequences. J Comput Phys. 2014;276:341–56.

8. Kamisetty H, Ovchinnikov S, Baker D. Assessing the utility of coevolution-based residue–residue contact predictions in a sequence-and structure-rich era. Proceedings of the National Academy of Sciences. 2013;110(39):15674–9.
9. Jones DT, Singh T, Kosciolek T, Tetchner S. MetaPSICOV: combining coevolution methods for accurate prediction of contacts and long range hydrogen bonding in proteins. Bioinformatics. 2015;31(7):999–1006. pmid:25431331
10. Ma J, Wang S, Wang Z, Xu J. Protein contact prediction by integrating joint evolutionary coupling analysis and supervised learning. Bioinformatics. 2015:btv472.
11. Adhikari B, Bhattacharya D, Cao R, Cheng J. CONFOLD: residue-residue contact-guided ab initio protein folding. Proteins: Structure, Function, and Bioinformatics. 2015;83(8):1436–49.
12. Wang S, Li W, Zhang R, Liu S, Xu J. CoinFold: a web server for protein contact prediction and contact-assisted protein folding. Nucleic acids research. 2016:gkw307.
13. Di Lena P, Nagata K, Baldi P. Deep architectures for protein contact map prediction. Bioinformatics. 2012;28(19):2449–57. pmid:22847931
14. Ekeberg M, Lövkvist C, Lan Y, Weigt M, Aurell E. Improved contact prediction in proteins: using pseudolikelihoods to infer Potts models. Phys Rev E. 2013;87(1):012707.
15. Göbel U, Sander C, Schneider R, Valencia A. Correlated mutations and residue contacts in proteins. Proteins: Structure, Function, and Bioinformatics. 1994;18(4):309–17.
16. Morcos F, Pagnani A, Lunt B, Bertolino A, Marks DS, Sander C, et al. Direct-coupling analysis of residue coevolution captures native contacts across many protein families. Proceedings of the National Academy of Sciences. 2011;108(49):E1293–E301.
17. Skwark MJ, Raimondi D, Michel M, Elofsson A. Improved contact predictions using the recognition of protein like contact patterns. PLoS Comput Biol. 2014;10(11):e1003889. pmid:25375897

18. Wu S, Zhang Y. A comprehensive assessment of sequence-based and template-based methods for protein contact prediction. Bioinformatics. 2008;24(7):924–31. pmid:18296462
19. Wang Z, Xu J. Predicting protein contact map using evolutionary and physical constraints by integer programming. Bioinformatics. 2013;29(13):i266–i73. pmid:23812992
20. He K, Zhang X, Ren S, Sun J. Deep residual learning for image recognition. arXiv preprint arXiv:151203385. 2015.
21. Krizhevsky A, Hinton G. Convolutional deep belief networks on cifar-10. Unpublished manuscript. 2010;40.
22. Srivastava RK, Greff K, Schmidhuber J, editors. Training very deep networks. Advances in neural information processing systems; 2015.
23. Hinton G, Deng L, Yu D, Dahl GE, Mohamed A-r, Jaitly N, et al. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. IEEE Signal Processing Magazine. 2012;29(6):82–97.
24. LeCun Y, Bengio Y, Hinton G. Deep learning. Nature. 2015;521(7553):436–44. pmid:26017442
25. Szegedy C, Liu W, Jia Y, Sermanet P, Reed S, Anguelov D, et al., editors. Going deeper with convolutions. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition; 2015.
26. Moult J, Fidelis K, Kryshtafovych A, Schwede T, Tramontano A. Critical assessment of methods of protein structure prediction (CASP)—round x. Proteins: Structure, Function, and Bioinformatics. 2014;82(S2):1–6.
27. Moult J, Fidelis K, Kryshtafovych A, Schwede T, Tramontano A. Critical assessment of methods of protein structure prediction: Progress and new directions in round XI. Proteins: Structure, Function, and Bioinformatics. 2016.
28. Haas J, Roth S, Arnold K, Kiefer F, Schmidt T, Bordoli L, et al. The Protein Model Portal—a comprehensive resource for protein structure and model information. Database. 2013;2013:bat031.
29. Pinheiro PH, Collobert R, editors. Recurrent Convolutional Neural Networks for Scene Labeling. ICML; 2014.

30. Nair V, Hinton GE, editors. Rectified linear units improve restricted boltzmann machines. Proceedings of the 27th International Conference on Machine Learning (ICML-10); 2010.
31. Monastyrskyy B, D'Andrea D, Fidelis K, Tramontano A, Kryshtafovych A. New encouraging developments in contact prediction: Assessment of the CASP11 results. Proteins: Structure, Function, and Bioinformatics. 2015.
32. Briinger AT, Adams PD, Clore GM, DeLano WL, Gros P, Grosse-Kunstleve RW, et al. Crystallography & NMR system: A new software suite for macromolecular structure determination. Acta Crystallogr D Biol Crystallogr. 1998;54(5):905–21.
33. Zhang Y, Skolnick J. Scoring function for automated assessment of protein structure template quality. Proteins: Structure, Function, and Bioinformatics. 2004;57(4):702–10.
34. Xu JR, Zhang Y. How significant is a protein structure similarity with TM-score = 0.5? Bioinformatics. 2010;26(7):889–95. pmid:20164152
35. Kim DE, Chivian D, Baker D. Protein structure prediction and analysis using the Robetta server. Nucleic Acids Research. 2004;32:W526–W31. pmid:15215442
36. Kelley LA, Mezulis S, Yates CM, Wass MN, Sternberg MJE. The Phyre2 web portal for protein modeling, prediction and analysis. Nature protocols. 2015;10(6):845–58. pmid:25950237
37. Källberg M, Wang H, Wang S, Peng J, Wang Z, Lu H, et al. Template-based protein structure modeling using the RaptorX web server. Nature protocols. 2012;7(8):1511–22. pmid:22814390
38. Biasini M, Bienert S, Waterhouse A, Arnold K, Studer G, Schmidt T, et al. SWISS-MODEL: modelling protein tertiary and quaternary structure using evolutionary information. Nucleic Acids Research. 2014;42(W1):W252–W8.
39. Remmert M, Biegert A, Hauser A, Söding J. HHblits: lightning-fast iterative protein sequence searching by HMM-HMM alignment. Nature methods. 2012;9(2):173–5.
40. Wang S, Ma J, Peng J, Xu J. Protein structure alignment beyond spatial proximity. Sci Rep. 2013;3:1448. Epub 2013/03/15. PubMed Central PMCID: PMC3596798. pmid:23486213

41. Zalewski JK, Mo JH, Heber S, Heroux A, Gardner RG, Hildebrand JD, et al. Structure of the Shroom-Rho kinase complex reveals a binding interface with monomeric Shroom that regulates cell morphology and stimulates kinase activity. J Biol Chem. 2016. Epub 2016/10/21.
42. Mohan S, Rizaldy R, Das D, Bauer RJ, Heroux A, Trakselis MA, et al. Structure of Shroom domain 2 reveals a three-segmented coiled-coil required for dimerization, Rock binding, and apical constriction. Mol Biol Cell. 2012;23(11):2131–42. Epub 2012/04/12. PubMed Central PMCID: PMC3364177. pmid:22493320
43. Ioffe S, Szegedy C, editors. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. Proceedings of The 32nd International Conference on Machine Learning; 2015.
44. Wang S, Peng J, Ma J, Xu J. Protein secondary structure prediction using deep convolutional neural fields. Sci Rep. 2016;6.
45. Miyazawa S, Jernigan RL. Estimation of effective interresidue contact energies from protein crystal structures: quasi-chemical approximation. Macromolecules. 1985;18(3):534–52.
46. Betancourt MR, Thirumalai D. Pair potentials for protein folding: choice of reference states and sensitivity of predicted native states to variations in the interaction schemes. Protein Science. 1999;8(02):361–9.
47. Bergstra J, Breuleux O, Bastien F, Lamblin P, Pascanu R, Desjardins G, et al., editors. Theano: A CPU and GPU math compiler in Python. Proc 9th Python in Science Conf; 2010.
48. Drozdetskiy A, Cole C, Procter J, Barton GJ. JPred4: a protein secondary structure prediction server. Nucleic acids research. 2015:gkv332.
49. Wang S, Li W, Liu S, Xu J. RaptorX-Property: a web server for protein structure property prediction. Nucleic acids research. 2016:gkw306.

CHAPTER 5

Predicting Virtual World User Population Fluctuations with Deep Learning

Young Bin Kim[1], Nuri Park[2], Qimeng Zhang[1], Jun Gi Kim[3], Shin Jin Kang[3], Chang Hun Kim[2]

[1] Interdisciplinary Program in Visual Information Processing, Korea University, Seoul, Korea

[2] Department of Computer and Radio Communications Engineering, Korea University, Seoul, Korea

[3] School of Games, Hongik University, Seoul, Korea

ABSTRACT

This paper proposes a system for predicting increases in virtual world user actions. The virtual world user population is a very important aspect of these worlds; however, methods for predicting fluctuations in these populations have not been well documented. Therefore, we attempt to predict changes in virtual world user populations with deep learning, using easily accessible online data, including formal datasets from Google Trends, Wikipedia, and online communities, as well as informal datasets collected from online forums. We use the proposed system to analyze the user population of EVE Online, one of the largest virtual worlds.

INTRODUCTION

Evolution in network technology and computing power has enabled people to interact with one another over the Internet. In the same vein, a growing number of users interact with one another and engage in economic, educational, and artistic activities in large virtual worlds (e.g., Second Life). Additionally, massively multiplayer online games (MMOGs)—e.g., World of Warcraft and EVE Online—have attracted an increasing number of active users who build communities and participate in a range of interactions [1–6].

Research on virtual worlds or online societies has been extensive. Easily accessible data from Pardus, an MMOG, enables researchers to investigate social theories in large virtual populations [7–12]. In particular, research on the structure and dynamic evolution of socioeconomic networks in virtual worlds has demonstrated significant results using diverse approaches [13–18]. Moreover, data from prior research has been analyzed from different perspectives [19–24]. In addition, research on forecasting the value of virtual currencies used in economic activity among virtual world users is currently underway [25, 26].

Previous studies focused on analyzing virtual world users based on social theories; research into user dynamics in virtual worlds is difficult to find. By contrast, extensive research on predicting real-world population fluctuations has been conducted [27–33]. Still, most researchers have adopted long-term perspectives regarding predictions of real-world population dynamics. These methods could be applied to virtual environments and combined with a range of internal components to find meaning in long-term population trends. In this study, we focus on the characteristics of virtual worlds such as MMOGs, whose rapid changes correlate with increasing or decreasing user populations [34–36].

The present paper focuses on EVE Online, one of the largest MMOG virtual worlds, which has attracted approximately 0.5 million subscribers since it was released in May 2003 [37]. We predict the daily fluctuations in the number of users of EVE Online through deep learning [38, 39], based on Google Trends [40], Wikipedia usage [41, 42], online forum usage [43] and sentiment data from online forum postings. The proposed method achieved a certain level of success in predicting daily user population fluctuations by drawing on easily available data closely associated with virtual worlds.

METHODS

System overview

For the proposed system, we collected data associated with MMOG EVE Online and tagged each posting and reply in its online forum with a positive or negative sentiment value. Based on this data and deep learning [38], we created a model for predicting fluctuations in the number of users (Fig 1).

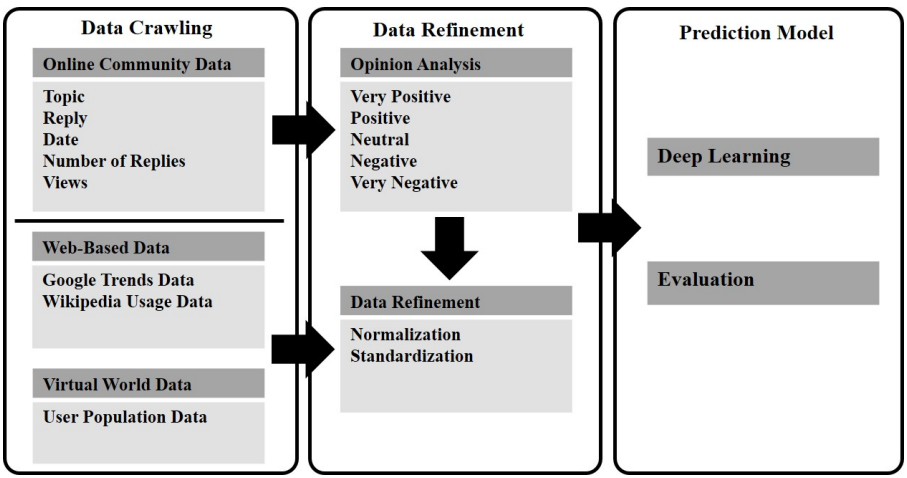


Figure 1: System overview.

Data collection

To generate prediction models, we selected data considered to be associated with an increase or decrease in the number of EVE Online users. The data selected was easily accessible online and was gathered from three sources: Google Trends, Wikipedia, and EVE Online forums [40–43].

Google Trends measures search interest in keywords searched on Google over a given period on a scale from 1 to 100. Google Trends data are widely used to analyze relevant phenomena in a variety of fields [44–49]. We gathered Google Trends data about the keyword "EVE Online" for this paper.

Second, Wikipedia usage data [41, 42] was gathered, showing the number of page views for certain keywords on a given day. These data are also widely used to analyze phenomena on the Internet [46, 50, 51]. Again, we gathered the Wikipedia page data relevant to "EVE Online".

We also gathered data from EVE Online forums [43], which are used by community members to upload postings and exchange opinions regarding topics of common interest [25, 52–54]. Therefore, online forums provide good sources for monitoring the daily responses of many users to certain MMOGs. Communities or forums are widely used in MMOGs for information exchange [52]. According to one study, EVE Online linked its forums with economic activity among users [25], which we found to be relevant when predicting fluctuations in the current number of virtual world users. Comments and relevant replies posted by users on general discussion boards in EVE Online forums were crawled, along with post time, comment and reply post times, the number of replies to each comment, and the number of views. Replies quoting previous comments and replies were crawled to exclude overlapping sentences.

Each HTML page was crawled using Python regex to parse HTML tags and extract the number of topics, the number of replies, the dates on which the topics and replies were posted, and the URL of each topic from the general discussion boards. Based on the URLs of extracted topics, content and replies to them were also extracted. These data were saved in.json format, which was in turn converted to other formats (e.g. csv and xlsx) for different purposes. The.json files from the EVE Online forums crawled can be viewed in the supporting information. One researcher carried out this data collection on a single PC for approximately 72 hours.

We collected data over a period extending from September 12, 2011 to April 29, 2016 (see Table 1) in a manner that complies with the terms and conditions stipulated by each service. Moreover, the collected data do not include any personal information.

Table 1: Summary of crawled data

Source	Boundary	Data Volume
EVE Online Community	Sep. 12, 2011–Apr. 29, 2016	1,156,608 User Replies
		37,418 User Threads
Google Trends (EVE Online)	Sep. 12, 2011–Apr. 29, 2016	1,692 Google Trends Values (1 value per day)
Wikipedia Usage (EVE Online)	Sep. 12, 2011–Apr. 29, 2016	1,692 Wikipedia Usage Values (1 value per day)

doi:10.1371/journal.pone.0167153.t001

Tagging user comment data and correlation analysis

We tagged the collected user comment data with positive or negative sentiment values. Previous studies have mostly focused on classifying user comments in certain fields. Emoticons, neologisms, and ungrammatical expressions are frequently seen on Internet forums. C.J. Hutto and Eric Gilbert [55] proposed an algorithm (VADER) to complement such informal expressions and suggested a method of analyzing social media text using a rule-based model. The online forums of interest here are comparable to social media text. Thus, we utilized the VADER algorithm to tag user comment data from the forums.

VADER normalizes negative and positive sentiment on a scale of -1 to 1. A comment was tagged as very negative, negative, positive or very positive, for $-1 \leq x < -0.6$, $-0.6 \leq x < -0.2$, $0.2 \leq x < 0.6$, and $0.6 \leq x \leq 1.0$, respectively (where x is a number). Each posting and reply was tagged (see the opinion analysis example in Table 2).

Table 2: EVE Online Forum Opinion Analysis Example

Opinion Criteria	Example topic sentences
Very Positive	“Pretty happy with Crucible” / “Happy New Year from CCP Games!” / “The days of "Erotica 1" are gone and I AM GLAD” / “Great job CCP!”
Positive	“Nice launcher” / “Better Missile graphics are good” / “CCP Thank you for new Capital Rats”
Neutral	“Guide to Fleet Commanding” / “Drone ships completely useless especially for new players.” / “A New Player Guide” / “Worth coming back to eve?”
Negative	“Why does Something Awful forums looks so old and terrible?” / “Back to play this terrible game terribly” / “Server crashed?” / “Awful download speed since launcher is released.”
Very Negative	“Those WTF moments” / “AFK Cloaking in System is a Terrible Mechanic” / “Verification Failure—Still Happening” / “Eve Installer not working”

doi:10.1371/journal.pone.0167153.t002

We verified the correlation between tagged sentiment values and the increase or decrease in user population. Here, the Pearson correlation coefficient [56] was used to determine the correlation between data sources and the increase or decrease in user population.

As shown in Eq (1), the results of opinion analysis based on the topics and replies (VADER-based tagged values) and population fluctuations were transformed each into z-scores for standardization against those of the previous 20 days. On a given date t (t = 20 in the paper), the z-score of expected value $\mathbb{E}$, denoted by $Z_{\mathbb{E}}$, was defined as:

$$Z_{\mathbb{E}_t} = \frac{\mathbb{E} - \bar{x}(\mathbb{E})}{\sigma(\mathbb{E})} \quad (1)$$

where $\bar{x}(\mathbb{E})$ and $\sigma(\mathbb{E})$ represent the mean and standard deviation of each item for every date with a time granularity of 1 day. Fig 2 shows an example of test results comparing the fluctuations in population and opinion analysis z-scores.

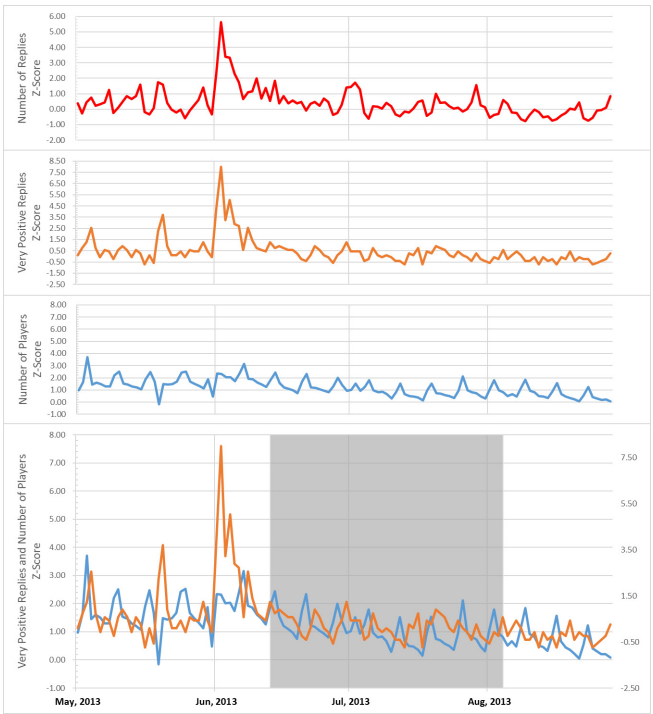


Figure 2: Z-scores of fluctuations in population and results of opinion analysis.

Some opinions show a trend similar to that of fluctuations in the population.

Table 3 shows the Pearson correlation coefficients between the z-scores for the opinion analysis results and the z-scores for population fluctuation.

Table 3: Pearson Correlation Coefficient Result

Opinion	Pearson Correlation Coefficient between results of opinion analysis and the population
Very Negative Topic	0.2547
Negative Topic	0.2422
Neutral Topic	0.2136
Positive Topic	0.2784
Very Positive Topic	**0.3037**
Very Negative Reply	**0.3178**
Negative Reply	**0.3667**
Neutral Reply	**0.3047**
Positive Reply	**0.3380**
Very Positive Reply	**0.3298**

doi:10.1371/journal.pone.0167153.t003

Overall, a positive linear relationship was found. Notably, there was a clear correlation between user replies and very positive topics.

Prediction Modeling

We used the data collected and the tagged opinion data for the model with the intent to predict fluctuations in user population based on deep learning. Deep learning is widely used for solving a range of problems [38, 39, 57–62]. Data sources have increased quantitatively and qualitatively in proportion to the history of virtual worlds. Still, little research on applying deep learning to virtual worlds for problem solving has been conducted. We developed a setting for applying deep learning based on collected data from a 4.5-year period.

First, we refined the data for the learning models. Specifically, the data gathered (as explained above) was standardized against its applicability to learning. We used long-term data and feature vectors by standardizing

the data against the previous 20 days to lessen the impact of the significant changes in the range of data values over longer time periods. An example of applicable input data is shown in Table 4.

Table 4: Example of a deep learning dataset

Data Class	Date	Opinion Data										Formal Data				
		Very positiveTopic	PositiveTopic	NeutralTopic	NegativeTopic	Very negativeTopic	Very positiveReply	PositiveReply	NeutralReply	NegativeReply	Very negativeReply	Number of Topics	Sum of Replies	Sum of Views	Google Trends Value	Wikipedia Page Views
Crawled Raw Data	Apr 02, 2016	*A*	*B*	*C*	*D*	*E*	*F*	*G*	*H*	*I*	*J*	*V*	*W*	*X*	*Y*	*Z*
Input Learning Data	Apr 02, 2016	Z_A	Z_B	Z_C	Z_D	Z_E	Z_F	Z_G	Z_H	Z_I	Z_J	Z_V	Z_W	Z_X	Z_Y	Z_Z

doi:10.1371/journal.pone.0167153.t004

The z-score $Z_{\mathbb{E}_t}$ for data from the previous 20 days was used as the values A–J, which indicate the value of the sum of forum opinion on a given date. V–Z denote formal data values (number of topics, sum of replies, sum of views, Google Trends value, and Wikipedia page views) on a given date.

We built deep learning models to perform predictions based on the input data. We accumulated multiple hidden layers to learn the deep structure of the data. Here, we configured 1, 2, 3, 5, and 7 hidden layers; of these, we selected the one that returned the best prediction results. We allocated 1,500 neurons to the single hidden layer; when using 2 hidden layers, we allocated 1,024 neurons to each; when using 3 hidden layers, we allocated 1,024, 1,024, and 512 neurons; when using the 5 hidden layers, we allocated 2,048, 1,024, 1,024, 512 and 512 neurons to them; and finally, when using 7 hidden layers, we allocated 2,048, 1,024, 1,024, 1,024, 512, 512 and 512 neurons to the layers.

For the input layer, based on the input data in Table 4, we represented 15 input data as continuing vectors and allocated a different number of neurons to a different number of cumulative days used for learning (i.e., 45, 75, 105, 135, and 180 neurons were allocated to 3, 5, 7, 9, and 12 cumulative days, respectively). For the output layer, 2 neurons were allocated to represent the probabilities of population increase or decrease with the softmax function.

We implemented the model using the Google Tensorflow library [38] and accelerated the deep learning model with GPU operation (nVIDIA CUDA). The gap in the prediction values between the optimal model and other model configurations is discussed in the following section.

EXPERIMENTAL RESULTS

We performed prediction modeling by means of deep learning based on the collected and refined data, and predicted fluctuations in the EVE Online user population. We used 90% of the data from the period between September 12, 2011 and April 29, 2016 for learning and 10% for validation. The accuracy rate, F-measure, and Matthews correlation coefficient (MCC) were used to evaluate the performance of the proposed models.

Table 5 and Fig 3 show the prediction results. The highest accuracy (87.57%) resulted from the 2-layer neural network model that used data from the previous 7 days for learning. Table 5outlines the prediction results by layer configuration and learning data. If the number of hidden layers and days used were less than 2 and 7, respectively, learning was insufficient and prediction accuracy decreased slightly. Conversely, overfitting could occur (with the prediction accuracy failing to significantly improve) if these numbers exceeded 2 and 7, respectively.

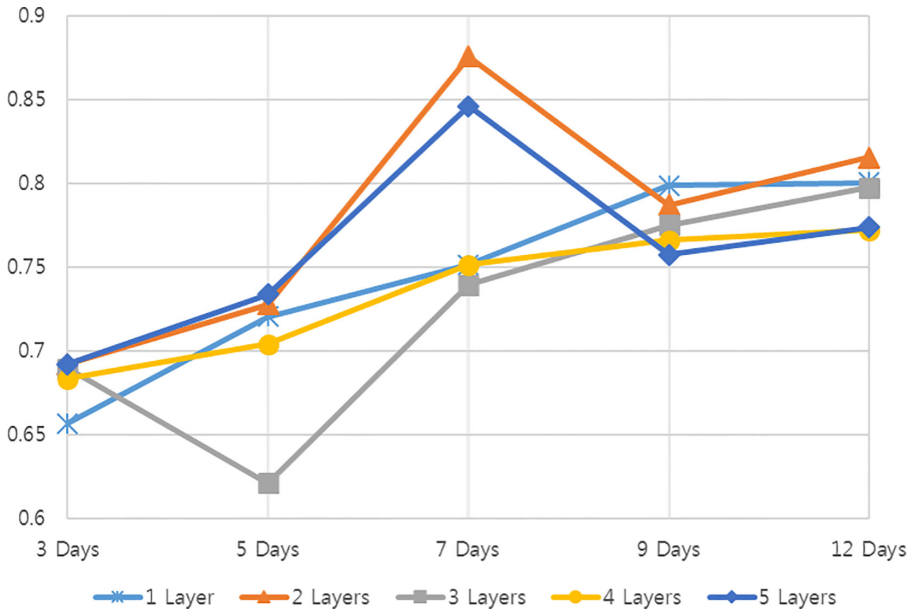


Figure 3: Experimental results.

Values by data and layers used.

Table 5: Experimental results of predicted fluctuations in the EVE Online user population

Data Set		Accuracy (%)	F1-Score	MCC
Hidden Layers	**Learning Days**			
1 Hidden Layer	3 Days	65.68%	0.6577	0.3098
	5 Days	72.04%	0.7199	0.4342
	7 Days	75.15%	0.753	0.5145
	9 Days	79.88%	0.8006	0.6042
	12 Days	80.06%	0.8027	0.6027
2 Hidden Layers	3 Days	69.23%	0.6933	0.3815
	5 Days	72.78%	0.7271	0.4495
	7 Days	**87.57%**	**0.8763**	**0.7517**
	9 Days	78.70%	0.788	0.5802
	12 Days	81.55%	0.8147	0.6127
3 Hidden Layers	3 Days	68.94%	0.6895	0.3756
	5 Days	62.13%	0.6213	0.2283
	7 Days	73.96%	0.74	0.4822
	9 Days	77.52%	0.7763	0.5438
	12 Days	79.76%	0.7988	0.5981
5 Hidden Layers	3 Days	68.34%	0.6844	0.3572
	5 Days	70.41%	0.7040	0.4078
	7 Days	75.15%	0.7521	0.4987
	9 Days	76.63%	0.768	0.5344
	12 Days	77.23%	0.7732	0.536
7 Hidden Layers	3 Days	69.23%	0.6935	0.408
	5 Days	73.37%	0.7337	0.4675
	7 Days	84.62%	0.8468	0.6897
	9 Days	75.74%	0.7578	0.5079
	12 Days	77.38%	0.7751	0.5386

doi:10.1371/journal.pone.0167153.t005

DISCUSSION AND CONCLUSION

This paper proposed a new method for predicting fluctuations in the number of users in a massive virtual world, EVE Online, with a deep-learning prediction model based on data from a variety of sources. The proposed method successfully predicted fluctuations in the number of EVE Online users based on the easily accessible data relevant to the virtual world. User comments in online forums were found to affect user actions in the virtual world.

The proposed method could be applicable to diverse fields, e.g., verifying newly added content in the creation and management of virtual worlds and solving network problems by forecasting the number of users. The proposed method could also be applied to previous findings,

e.g., space and NPC management in virtual worlds [63, 64] and to the optimization of virtual currency systems [25]. In addition, it could be used to apply social science theories to virtual worlds so as to understand the large, diverse user base.

Due to the paucity of previous findings in this field, our proposal has some limitations that need to be rectified in future work. First, the data need to be enriched further for better results. Diverse types of data would increase prediction accuracy. This research was limited to using data gathered for the prediction model only by means of deep learning; more diversified data would be applicable to feature selection as well. The VADER algorithm, optimized for social media analysis, performed well in this study. However, additional analysis of sarcastic or ironic language would improve our results. Moreover, Word2Vec [65] could also potentially improve these results. Sentiment analysis using the Word2Vec-based Doc2Vec [66] focuses on user reviews, making it difficult to apply directly to the analysis of online community users. Still, it might be applicable to and improve the analysis of user comments in online communities. As all data used for learning were formal data, data refinement methods should also be improved. Furthermore, other sizable virtual worlds, e.g., World of Warcraft and Second Life, could be used for prediction.

Virtual worlds have been growing in size and diversity. Considerable research on them has already been conducted. Still, the size of the virtual world user population has not been elucidated. With ongoing improvement for wider application, the proposed method of applying data from different sources to virtual world user population dynamics will contribute to enhancing the understanding of virtual worlds and their user bases.

AUTHOR CONTRIBUTIONS

- **Conceptualization:** YBK SJK.
- **Data curation:** YBK JGK.
- **Formal analysis:** YBK NP QZ.
- **Investigation:** YBK NP JGK.
- **Methodology:** YBK NP.

- **Project administration:** YBK CHK.
- **Resources:** YBK NP QZ.
- **Software:** YBK NP QZ JGK.
- **Supervision:** CHK.
- **Validation:** YBK NP QZ.
- **Visualization:** YBK.
- **Writing – original draft:** YBK NP.
- **Writing – review & editing:** YBK CHK.

REFERENCES

1. De Lucia A, Francese R, Passero I, Tortora G. Development and evaluation of a virtual campus on Second Life: The case of SecondDMI. Comput Educ. 2009;52(1):220–33.
2. Jarmon L, Traphagan T, Mayrath M, Trivedi A. Virtual world teaching, experiential learning, and assessment: An interdisciplinary communication course in Second Life. Comput Educ. 2009;53(1):169–82.
3. Ratan RA, Chung JE, Shen CH, Williams D, Poole MS. Schmoozing and Smiting: Trust, Social Institutions, and Communication Patterns in an MMOG. J Comput-Mediat Comm. 2010;16(1).
4. Kong JSL, Kwok RCW, Fang YL. The effects of peer intrinsic and extrinsic motivation on MMOG game-based collaborative learning. Inform Manage-Amster. 2012;49(1):1–9.
5. Waddell JC, Peng W. Does it matter with whom you slay? The effects of competition, cooperation and relationship type among video game players. Comput Hum Behav. 2014;38:331–8.
6. Pena J, Blackburn K. The Priming Effects of Virtual Environments on Interpersonal Perceptions and Behaviors. J Commun. 2013;63(4):703–20.
7. Mryglod O, Fuchs B, Szell M, Holovatch Y, Thurner S. Interevent time distributions of human multi-level activity in a virtual world. Physica A. 2015;419:681–90.
8. Szell M, Lambiotte R, Thurner S. Multirelational organization of large-scale social networks in an online world. P Natl Acad Sci

USA. 2010;107(31):13636–41.

9. Szell M, Sinatra R, Petri G, Thurner S, Latora V. Understanding mobility in a social petri dish. Sci Rep-Uk. 2012;2.
10. Szell M, Thurner S. Measuring social dynamics in a massive multiplayer online game. Soc Networks. 2010;32(4):313–29.
11. Szell M, Thurner S. Social Dynamics in a Large-Scale Online Game. Adv Complex Syst. 2012;15(6).
12. Thurner S, Szell M, Sinatra R. Emergence of Good Conduct, Scaling and Zipf Laws in Human Behavioral Sequences in an Online World. Plos One. 2012;7(1).
13. Bainbridge WS. The scientific research potential of virtual worlds. Science. 2007;317(5837):472–6. pmid:17656715
14. Messinger PR, Strolulia E, Lyons K, Bone M, Niu RH, Smirnov K, et al. Virtual worlds—past, present, and future: New directions in social computing. Decis Support Syst. 2009;47(3):204–28.
15. Xie W-J, Li M-X, Jiang Z-Q, Tan Q-Z, Podobnik B, Zhou W-X, et al. Division of labor, skill complementarity, and heterophily in socioeconomic networks. arXiv preprint arXiv:150303746. 2015.
16. Fuchs B, Sornette D, Thurner S. Fractal multi-level organisation of human groups in a virtual world. arXiv preprint arXiv:14033228. 2014.
17. Jiang Z-Q, Zhou W-X, Tan Q-Z. Online-offline activities and game-playing behaviors of avatars in a massive multiplayer online role-playing game. EPL (Europhysics Letters). 2009;88(4):48007.
18. Xie W-J, Li M-X, Jiang Z-Q, Tan Q-Z, Podobnik B, Zhou W-X, et al. Skill complementarity enhances heterophily in collaboration networks. Sci Rep-Uk. 2016;6.
19. Corominas-Murtra B, Fuchs B, Thurner S. Detection of the Elite Structure in a Virtual Multiplex Social System by Means of a Generalised K-Core. Plos One. 2014;9(12).
20. Fuchs B, Thurner S. Behavioral and Network Origins of Wealth Inequality: Insights from a Virtual World. Plos One. 2014;9(8).
21. Kang SJ, Kim YB, Park T, Kim CH. Automatic player behavior analysis system using trajectory data in a massive multiplayer online

game. Multimed Tools Appl. 2013;66(3):383–404.

22. Klimek P, Thurner S. Triadic closure dynamics drives scaling laws in social multiplex networks. New J Phys. 2013;15.
23. Szell M, Thurner S. How women organize social networks different from men. Sci Rep-Uk. 2013;3.
24. Xie WJ, Li MX, Jiang ZQ, Zhou WX. Triadic motifs in the dependence networks of virtual societies. Sci Rep-Uk. 2014;4.
25. Kim YB, Lee SH, Kang SJ, Choi MJ, Lee J, Kim CH. Virtual World Currency Value Fluctuation Prediction System Based on User Sentiment Analysis. Plos One. 2015;10(8):e0132944. PubMed Central PMCID: PMCPMC4524693. pmid:26241496
26. Kim YB, Kim JG, Kim W, Im JH, Kim TH, Kang SJ, et al. Predicting Fluctuations in Cryptocurrency Transactions Based on User Comments and Replies. Plos One. 2016;11(8):e0161197. pmid:27533113
27. Brass W. Perspectives in Population Prediction—Illustrated by Statistics of England and Wales. J Roy Stat Soc a Sta. 1974;137:532–83.
28. Wisniowski A, Smith PWF, Bijak J, Raymer J, Forster JJ. Bayesian Population Forecasting: Extending the Lee-Carter Method. Demography. 2015;52(3):1035–59. pmid:25962866
29. Shang HL, Smith PWF, Bijak J, Wisniowski A. A multilevel functional data method for forecasting population, with an application to the United Kingdom. Int J Forecasting. 2016;32(3):629–49.
30. Ward EJ, Holmes EE, Thorson JT, Collen B. Complexity is costly: a meta-analysis of parametric and non-parametric methods for short-term population forecasting. Oikos. 2014;123(6):652–61.
31. Tayman J, Smith SK, Lin J. Precision, bias, and uncertainty for state population forecasts: an exploratory analysis of time series models. Popul Res Policy Rev. 2007;26(3):347–69.
32. Chi GQ. Can Knowledge Improve Population Forecasts at Subcounty Levels? Demography. 2009;46(2):405–27. pmid:21305400
33. Chi GQ, Voss PR. Small-area Population Forecasting: Borrowing Strength across Space and Time. Popul Space Place. 2011;17(5):505–20.

34. O'Donnell C. On the Back of a Flying Gryphon: Soaring Over/Through the Global Game Industry Nick Dyer-Witheford and Greig de Peuter, Games of Empire Bonnie A. Nardi, My Life as a Night Elf Priest William Sims Bainbridge, The Warcraft Civilization. Technol Cult. 2012;53(1):196–9.
35. Gursimsek RA. Being there together: Social interaction in virtual environments. Convergence-Us. 2012;18(1):112–4.
36. Daniel P, Chris G, editors. A measurement study of virtual populations in massively multiplayer online games. 6th ACM SIGCOMM workshop on Network and system support for games; 2007.
37. Bergstrom K, Carter M, Woodford D, Paul C. Constructing the ideal EVE online player. Proceedings of DiGRA 2013: DeFragging Game Studies. 2013.
38. Abadi M, Agarwal A, Barham P, Brevdo E, Chen Z, Citro C, et al. TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems. arXiv:1603.04467; 2016.
39. LeCun Y, Bengio Y, Hinton G. Deep learning. Nature. 2015;521(7553):436–44. pmid:26017442
40. Google Trends: Google; [cited 2016 Apr 29]. Available from: https://www.google.com/trends/.
41. Wikipedia article traffic statistics [cited 2016 Apr 29]. Available from: http://stats.grok.se/.
42. Wikipedia Traffic Grapher [cited 2016 Apr 29]. Available from: http://tools.wmflabs.org/traffic-grapher/Wikipedia-Traffic-Grapher/index.php.
43. EVE Online Forums: CCP; [cited 2016 Apr 29]. Available from: http://forums.eveonline.com.
44. Choi HY, Varian H. Predicting the Present with Google Trends. Econ Rec. 2012;88:2–9.
45. Preis T, Moat HS, Stanley HE. Quantifying Trading Behavior in Financial Markets Using Google Trends. Sci Rep-Uk. 2013;3.
46. Kristoufek L. BitCoin meets Google Trends and Wikipedia: Quantifying the relationship between phenomena of the Internet era. Sci Rep-Uk. 2013;3.

47. Kristoufek L. Can Google Trends search queries contribute to risk diversification? Sci Rep-Uk. 2013;3.
48. Kang M, Zhong HJ, He JF, Rutherford S, Yang F. Using Google Trends for Influenza Surveillance in South China. Plos One. 2013;8(1).
49. Choi H, Varian H. Predicting the present with Google Trends. Econ Rec. 2012;88(s1):2–9.
50. Moat HS, Curme C, Avakian A, Kenett DY, Stanley HE, Preis T. Quantifying Wikipedia Usage Patterns Before Stock Market Moves. Sci Rep-Uk. 2013;3.
51. Mestyan M, Yasseri T, Kertesz J. Early Prediction of Movie Box Office Success Based on Wikipedia Activity Big Data. Plos One. 2013;8(8).
52. Hau YS, Kim YG. Why would online gamers share their innovation-conducive knowledge in the online game user community? Integrating individual motivations and social capital perspectives. Comput Hum Behav. 2011;27(2):956–70.
53. Panzarasa P, Opsahl T, Carley KM. Patterns and Dynamics of Users' Behavior and Interaction: Network Analysis of an Online Community. J Am Soc Inf Sci Tec. 2009;60(5):911–32.
54. Sing CC, Khine MS. An analysis of interaction and participation patterns in Online community. Educ Technol Soc. 2006;9(1):250–61.
55. Hutto CJ, Gilbert E. Vader: A parsimonious rule-based model for sentiment analysis of social media text. Eighth International AAAI Conference on Weblogs and Social Media2014.
56. Benesty J, Chen J, Huang Y, Cohen I. Pearson correlation coefficient. Noise reduction in speech processing: Springer; 2009. p. 1–4.
57. Mnih V, Kavukcuoglu K, Silver D, Rusu AA, Veness J, Bellemare MG, et al. Human-level control through deep reinforcement learning. Nature. 2015;518(7540):529–33. pmid:25719670
58. Silver D, Huang A, Maddison CJ, Guez A, Sifre L, Van Den Driessche G, et al. Mastering the game of Go with deep neural networks and tree search. Nature. 2016;529(7587):484–9. pmid:26819042

59. CireşAn D, Meier U, Masci J, Schmidhuber J. Multi-column deep neural network for traffic sign classification. Neural Networks. 2012;32:333–8. pmid:22386783

60. Deng L, Hinton G, Kingsbury B, editors. New types of deep neural network learning for speech recognition and related applications: An overview. 2013 IEEE International Conference on Acoustics, Speech and Signal Processing; 2013: IEEE.

61. Yu D, Yao K, Su H, Li G, Seide F, editors. KL-divergence regularized deep neural network adaptation for improved large vocabulary speech recognition. 2013 IEEE International Conference on Acoustics, Speech and Signal Processing; 2013: IEEE.

62. Krizhevsky A, Sutskever I, Hinton GE, editors. Imagenet classification with deep convolutional neural networks. Advances in neural information processing systems; 2012.

63. Kang S-J, Kim Y, Kim C-H. Live path: adaptive agent navigation in the interactive virtual world. The Visual Computer. 2010;26(6–8):467–76.

64. Kang S-J, Kim YB, Park T, Kim C-H. Automatic player behavior analysis system using trajectory data in a massive multiplayer online game. Multimed Tools Appl. 2013;66(3):383–404.

65. Goldberg Y, Levy O. word2vec Explained: deriving Mikolov et al.'s negative-sampling word-embedding method. arXiv preprint arXiv:14023722. 2014.

66. Le QV, Mikolov T, editors. Distributed Representations of Sentences and Documents. ICML; 2014.

CHAPTER 6

Recognition of Prokaryotic and Eukaryotic Promoters Using Convolutional Deep Learning Neural Networks

Ramzan Kh. Umarov[1], Victor V. Solovyev[2]

[1] King Abdullah University of Science and Technology, Thuwal, Saudi Arabia

[2] Softberry Inc., Mount Kisco, United States of America

ABSTRACT

Accurate computational identification of promoters remains a challenge as these key DNA regulatory regions have variable structures composed of functional motifs that provide gene-specific initiation of transcription. In this paper we utilize Convolutional Neural Networks (CNN) to analyze sequence characteristics of prokaryotic and eukaryotic promoters and build their predictive models. We trained a similar CNN architecture on promoters of five distant organisms: human, mouse, plant (*Arabidopsis*), and two bacteria (*Escherichia coli* and *Bacillus subtilis*). We found that CNN trained on sigma70 subclass of *Escherichia coli* promoter gives an excellent classification of promoters and non-promoter sequences (Sn = 0.90, Sp = 0.96, CC = 0.84). The *Bacillus subtilis* promoters identification CNN model achieves Sn = 0.91, Sp = 0.95, and CC = 0.86.

For human, mouse and *Arabidopsis* promoters we employed CNNs for identification of two well-known promoter classes (TATA and non-TATA promoters). CNN models nicely recognize these complex functional regions. For human promoters Sn/Sp/CC accuracy of prediction reached 0.95/0.98/0,90 on TATA and 0.90/0.98/0.89 for non-TATA promoter sequences, respectively. For *Arabidopsis* we observed Sn/Sp/CC 0.95/0.97/0.91 (TATA) and 0.94/0.94/0.86 (non-TATA) promoters. Thus, the developed CNN models, implemented in CNNProm program, demonstrated the ability of deep learning approach to grasp complex promoter sequence characteristics and achieve significantly higher accuracy compared to the previously developed promoter prediction programs. We also propose random substitution procedure to discover positionally conserved promoter functional elements. As the suggested approach does not require knowledge of any specific promoter features, it can be easily extended to identify promoters and other complex functional regions in sequences of many other and especially newly sequenced genomes. The CNNProm program is available to run at web server http://www.softberry.com.

INTRODUCTION

Promoter is a key region that is involved in differential transcription regulation of protein-coding and RNA genes. Gene-specific architecture of promoter sequences makes it extremely difficult to devise the general strategy for their computational identification [1, 2]. Promoter 5'-flanking regions may contain many short (5–10 bases long) motifs that serve as recognition sites for proteins providing initiation of transcription as well as specific regulation of gene expression.

A minimal eukaryotic promoter region, called core promoter, is capable of initiating basal transcription and contains a transcription start site (TSS). About 30–50% of all known eukaryotic promoters contain a TATA-box at a position ~30 bp upstream from the transcription start site. Many highly expressed genes contain a strong TATA box in their core promoter. At the same time, large groups of genes including housekeeping genes, some oncogenes and growth factor genes possess TATA-less promoters. In these promoters Inr (the initiator region) or the recently found downstream promoter element (DPE), usually located

~25–30 bp downstream of TSS, may control the exact position of the transcription start [1, 2].

Bacterial promoters contain two short conserved sequence elements approximately -10 and -35 nucleotides upstream from the transcription start site. The -10 box is absolutely essential to start transcription in prokaryotes. The sequence of -35 box affects the transcription rate [3–6]. Those consensus sequences, while conserved on average, are not found intact in most promoters.

Accurate prediction of promoters is fundamental for interpreting gene expression patterns, and for constructing and understanding genetic regulatory networks. In the last decade, genomes of many organisms have been sequenced and their gene content was mainly computationally identified. Promoters and transcriptional start sites (TSS), however, are still left largely undetermined and the efficient software able to accurately predict promoters in newly sequenced genomes is not yet available in public domain.

There were many attempts to develop promoter prediction software as for bacterial as well as for eukaryotic genomes. Most of them implemented very diverse computational algorithms, which often account some specific sequence features discovered during experimental studies. Fickett and Hatzigeorgiou [7] presented one of the first reviews of eukaryotic promoter prediction programs. Among these were oligonucleotide content-based neural network and linear discriminant approaches.

It was shown that many general-purpose promoter prediction programs can typically recognize only ~50% of the promoters with false positive (FP) rate of ~1 per 700–1000 bp [7]. The study to make a critical assessment of the human promoter prediction field also demonstrated a pretty low level of sensitivity of 58% for the specificity of 92% and correlation coefficient (CC) ranged from 0.52 to 0.73 for evaluated promoter predictors [8]. Much better accuracy has been observed for methods of identification of plant promoters [9–14]. Their specificity level, however, does not exceed 90% that will generate significant number of false positives when the methods would be applied to analyze long genomic sequences. The top two performers TSSP_TCM [9] and Promobot [10] with Sn = 0.88-0.89 and Sp = 0.84-0.86 outperform NNPP

[11] (Sn/Sp:0.74/0.70), PromoterScan [12] (Sn/Sp:0.08/0.04), Promoter [13] (Sn/Sp:0.24/0.34), and Prom-Machine [14] (Sn/Sp:0.86/0.81).

While bacterial promoters have simpler structure than transcription initiation regions of higher organisms, their identification is also a challenging task. Using sequence alignment kernel and SVM classifier Gordon et al. [15] achieved Sn = 0.82 and Sp = 0.84 discriminating between σ^{70} promoter and non-promoter *E.coli* sequences. Similar accuracy was observed for popular bacterial promoter prediction program Bprom [16]. These programs clearly outperform the NNPP (trained on *E. coli* K12 sequences) [11] and SIDD [17] programs. For example, SIDD correctly predicted only 74.6% of actual promoters with a false positive rate of 18%. When NNPP correctly predicted 66.4% of the real promoters, its false positive rate was 22.4%.

Thousands bacteria and eukaryotic genomes are already sequenced and many more are on the way, while little transcriptional information is available for most of them. Moreover, new genomes may have promoter features that are different from those observed in model organisms. For example, recent studies have shown that TATA boxes and Initiators are not universal features of plant promoters, and that other motifs such as Y patches may play a major role in the transcription initiation in plants [10, 18, 19]. We face the situation that specific promoter characteristics that are often used in developing promoter predictors are poorly understood in many new genomes. This creates favorable circumstances for developing universally applicable algorithm of promoter prediction and in this paper we propose the use of convolutional neural networks, with an input consisting of only genomic sequence, as a rather general approach to solution of this problem.

Deep convolutional neural networks are capable of achieving record-breaking results in processing images, video, speech and audio on highly challenging datasets using purely supervised learning and they have recently won a large number of contests in pattern recognition and machine learning [20–23]. There are a few successful examples of applying them to biological problems. Deep learning–based algorithmic framework, DeepSEA, can predict chromatin effects of sequence alterations and prioritize functional SNPs by learning a regulatory sequence code from large-scale chromatin-profiling data [24]. Improved performance for this task was reported using DanQ [25], a hybrid framework that combines

convolutional and bi-directional long short-term memory recurrent neural networks [26]. Chen et al. applied deep learning method (abbreviated as D-GEX) to infer the expression of target genes from the expression of landmark genes [27]. Finally, DeepBind, a computational approach based on deep convolutional neural networks, can discover new DNA and RNA binding sites using a set of sequences and, for each sequence, an experimentally determined binding score [28].

In this paper we utilize Convolutional Neural Networks (CNN) to analyze sequence characteristics of prokaryotic and eukaryotic promoters and build their predictive models. The developed CNN models, implemented in CNNProm program, demonstrated the ability of deep learning to grasp complex promoter sequence characteristics and achieve significantly higher accuracy compared to previously developed promoter prediction programs.

MATERIALS AND METHODS

Training and testing data

In this study, in order to demonstrate universality of the suggested approach to promoter prediction problem we selected promoter sequences from very distant groups of organisms: two bacteria, human, mouse and a plant. The studied number of promoter and non-promoter sequences for each organism is shown in Table 1.

Table 1: Numbers, lengths and locations of promoter and non-promoter sequences for studied organisms

Organism	#promoter sequences	#non-promoter sequences	Length/Location
Escherichia coli s70	839	3000	81/-60 - +20
Bacillus subtilis	746	2000	81/-60 - +20
Human TATA	1426	8256	251/-200 - +50
Human non-TATA	19811	27731	251/-200 - +50
Mouse TATA	1255	3530	251/-200 - +50
Mouse non-TATA	16283	24822	251/-200 - +50
Arabidopsis TATA	1497	2879	251/-200 - +50
Arabidopsis non-TATA	5905	11459	251/-200 - +50

doi:10.1371/journal.pone.0171410.t001

Locations are given relative to the TSS (Transcription Start Site) position.

We used bacterial promoter and non-promoter sequences of length 81 nt (nucleotides). Bacterial non-promoter sequences were taken from the corresponding genome sequences: we randomly selected fragments of protein-coding genes and took their opposite (non-coding) chain sequences. *Escherichia coli* σ70 promoter sequences were extracted from manually curated RegulonDB [29]. *Bacillus subtilis* promoters were taken from a collection described in [30]. As for human, mouse and *Arabidopsis* non-promoter sequences (size 251 nt) we used random fragments of their genes located after first exons. Eukaryotic promoter sequences were extracted from the well-known EPD database [31].

We used 20% of each set sequences in our test sets. 70% of the remaining sequences were used as training and 10% as validation sets. Training sets provide data to generate parameters of CNNmodels, while validation sets are used to find the optimum number of learning epochs (cycles) that should be limited to avoid over-fitting.

Convolutional networks

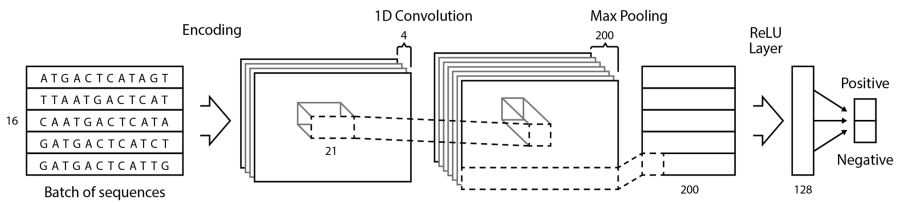
Convolutional layer is a core building block of convolutional networks [20–23]. A layer consists of filters, which are small matrices (W), for example $L \times L \times D$, where D is depth of input data and L is called filter length. These filters are convolved with an input, i.e. moved spatially across an input, and a dot product is calculated for each position: $W \times x + b$, where W is our filter, x is a small chunk of an input and b is bias. A local $L \times L$ area in our input is called a receptive field, and a distance of each step of a filter sliding across an input is called stride. Calculating a dot product at each position gives us an activation map for our filter. Next layer takes as an input activation maps from all filters. Activation map is in fact partially connected neurons, which share the same weight, i.e. weight corresponding to a filter. This weight sharing is an important property of convolutional networks. It dramatically reduces a number of required parameters compared to a fully connected layer.

Convolutional layer can be followed by another convolutional layer, in which case the depth of the input is the number of filters from a previous layer. Convolutional layers are eventually followed by a pooling layer. This is a simple layer that operates on each activation map, making it smaller and more manageable. The most common pooling technique is Max-Pooling, which chooses the largest of several values for further

representation. Convolutional layers augmented by Max-Pooling are prevalent in many modern Deep Learners [23]. They can be useful for working with biological sequences because convolution filters can capture information on functional sequence motifs.

CNN architecture for building promoter recognition models

There are many network architectures and the task is to choose a suitable one for a particular research problem. In **learnCNN.py** program we implemented CNN model using Keras—a minimalist, highly modular neural networks library, written in Python [32]. It uses Theano library [33, 34] as a backend and utilizes GPU [35] for fast neural network training. Adam optimizer [36] was used for training with categorical cross-entropy as a loss function. Our CNN architecture (Fig 1) in most cases consisted of just one convolutional layer with 200 filters having length 21. After convolutional layer, we have a standard Max-Pooling layer. The output from the Max-Pooling layer is fed into a standard fully connected ReLU layer with 128 neurons. Pooling size was usually 2. Finally, the ReLU layer is connected to output layer with sigmoid activation, where neurons correspond to promoter and non-promoter classes. The batch size used for training was 16.



Figurer 1: Basic CNN architecture that was used in building promoter models implemented in the learnCNN.py program (see text for description).

Input of the network consisted of nucleotide sequences where each nucleotide is encoded by a four dimensional vector A (1,0,0,0), T(0,1,0,0), G(0,0,1,0) and C(0,0,0,1). Output is a two dimensional vector: promoter (1, 0) and Non-promoter (0, 1) prediction. The training takes a few minutes on GTX 980 Ti GPU.

We intentionally used, in most cases, one layer CNN architecture, but sometimes to get a proper balance of accuracy between positives examples (promoters) and negative examples (non-promoter) two or three layers may be applied. A typical example of the model computation is shown in Fig 2.

```
%learncnn.py params_mou_tr.txt
Parameters File: params_mou_tr.txt
Positives: mm9_t.fa   1255
Negatives: mou_nprtc.fa   3530
Positives for training: 1003
Negatives for training: 2823
Positives for testing:  251
Negatives for testing:  706
Train on 3347 samples, validate on 479 samples
Epoch 1/5
3347/3347 [======] - 3s - loss: 0.2697 - acc: 0.8838 - val_loss: 0.1284 - val_acc: 0.956
Epoch 2/5
3347/3347 [======] - 2s - loss: 0.0801 - acc: 0.9689 - val_loss: 0.0557 - val_acc: 0.9770
Epoch 3/5
3347/3347 [======] - 2s - loss: 0.0341 - acc: 0.9883 - val_loss: 0.0537 - val_acc: 0.9749
Epoch 4/5
3347/3347 [======] - 2s - loss: 0.0092 - acc: 0.9982 - val_loss: 0.0406 - val_acc: 0.9854
Epoch 5/5
3347/3347 [======] - 2s - loss: 0.0032 - acc: 1.0000 - val_loss: 0.0983 - val_acc: 0.9645
TP: 243.0
TN: 686.0
FP: 20.0
FN: 8.0
Test accuracy: 0.970741901776
Sensitivity: 0.96812749004
Specificity: 0.971671388102
CC: 0.926011806827
```

Figure 2: An example of learning CNN models for mouse promoters.

The training, and validation accuracy is presented for each learning epoch. Finally, the performance on the test data is shown.

Performance measures

Several measures to estimate the accuracy of a recognition function were introduced in genomic research [37, 38]. Consider that we have S sites (positive examples) and N non-sites (negative examples). By applying a recognition function, we correctly identify TP sites (true positives) and TN non-sites (true negatives). At the same time FP (false positives) sites were wrongly classified as non-sites and FN (false negative) non-sites were wrongly classified as sites. Sensitivity (Sn) (true positive rate) measures a fraction of the true positive examples that are correctly predicted: $Sn = TP/(TP + FN)$. Specificity (Sp) (true negative rate) measures a fraction of the predicted sites that are correct amongst those predicted: $Sp = TN/(TN + FP)$. Accuracy $AC = (TP + TN)/(TN + TP + FN + FP)$ measures an average performance on positive and

negative datasets. However, this measures does not take into account the possible difference in sizes of site and non-sites sets. More correct single measure (correlation coefficient) takes the relation between correctly predictive positives and negatives as well as false positives and negatives into account [38]:

$$CC = \frac{TP \times TN - FP \times FN}{\sqrt{(TP + FP)(TP + FN)(TN + FP)(TN + FN)}} \quad (1)$$

RESULTS AND DISCUSSION

The accuracy of promoter identification by constructed CNN models

Using CNN architecture described above implemented in **learnCNN.py** program we analyzed the promoter and non-promoter sequences (Table 1). **LearnCNN.py** learns parameters of CNN model and outputs the accuracy of promoter prediction for the test set of sequences. It also writes computed CNN Model (PPCNNmodel) to a file, which can be used later in programs for promoter identification in a given sequence.

The accuracy information and some parameters of CNN architecture used for the particular datasets are show in Table 2.

Table 2: The accuracy and parameters of CNN models

Organism	Sn	Sp	CC	CNN architecture
Escherichia coli s70	0.90	0.96	0.84	100,7, 0 / 150, 21, 12
Bacillus subtilis	0.91	0.95	0.86	100,15, 2 / 250, 17, 2
Human TATA	0.95	0.98	0.90	200, 21, 4
Human non-TATA	0.90	0.98	0.89	300, 21, 231
Mouse TATA	0.97	0.97	0.93	200, 21, 6
Mouse non-TATA	0.88	0.94	0.83	100, 15, 2 / 250, 21, 2
Arabidopsis TATA	0.95	0.97	0.91	200, 21, 4
Arabidopsis non TATA	0.94	0.94	0.86	200, 21, 2

doi:10.1371/journal.pone.0171410.t002

Architecture as 200, 21, 4 describes one layer with 200 filters, filter length 21 and pooling size 4; '/' separates data for two layers.

We found that the computed CNN models demonstrated the ability of deep learning to grasp complex promoter sequence characteristics and achieve significantly higher accuracy compared to previously developed promoter prediction programs. For example, CNN trained on

sigma70 sub-class of *Escherichia coli* promoters provides an excellent classification of promoters and non-promoter sequences (Sn = 0.90, Sp = 0.96). For human, mouse and *Arabidopsis* promoters, we employed CNNs to identify two well-known promoter classes (TATA and non-TATA promoters). CNN models nicely recognize these complex functional regions. For human, Sn/Sp accuracy of prediction reached 0.95/0.98 for TATA and 0.90/0.98 for non-TATA promoter sequences. For mouse, Sn/Sp accuracy of prediction achieved 0.97/0.97 for TATA and 0.88/0.94 for non-TATA promoters. The same outstanding performance was observed on *Arabidopsis* data: Sp/Sn for TATA promoters 0.95/0.97 and for non-TATA promoters 0.94/0.94. This is a very significant improvement in prediction performance compared to previously evaluated human promoter predictors, where the sensitivity of 58% and specificity of 92% with correlation coefficient (CC) in the range of 0.52–0.73 were observed [8].

In this work, we studied sequences of promoter regions extracted from the EPDnew promoter database [31], which recently extended promoter collection beyond cases based on evidence from TSS mapping experiments on single genes. Currently, TSS positions provided by EPD are inferred from next-generation sequencing data and are automatically generated from multiple, carefully selected input datasets that include chromatin signatures in addition to mRNA 5'tags to improve location of promoters for weekly expressed genes. The authors of EPDnew database have demonstrated its higher quality over ENSEMBL-derived [39] human promoter set [31]. We also observed apparent better quality of a promoter identification program when using EPDnew data. For example, for CNN predictor computed on 1083 mouse TATA promoter regions extracted from DBTSS [40], we also reached a pretty good performance on a test set of 271 promoters: Sn = 0.94, Sp = 0.94 and CC = 0.86. However, CNN model trained using mouse TATA promoters regions from EPDnew demonstrated noticeably better results: Sn = 0.94, Sp = 0.98 and CC = 0.93 (Table 2).

We would like to point out an important benefit of the considered CNN models. While using only nucleotide sequences, they can outperform recognition functions built based on preselected significant features. For example, widely used Bprom [16] promoter prediction program utilizes a set of seven features (five relatively conserved sequence

motifs, represented by their weight matrices, the distance between −10 and −35 elements and the ratio of densities of octa-nucleotides overrepresented in known bacterial transcription factor binding sites relative to their occurrence in the coding regions). Computing these features for a set of 839 experimentally verified σ70 promoters from Regulon database [29] and 3000 non-promoter *E.coli* sequences and using LinearDiscriminantAnalysis and other discrimination approaches from scikit-learn Python library [41], we reached an average accuracy of 0.92 for classification of promoter and non-promoter sequences by applying cross-validation evaluation. The CNN model demonstrated a better recognition rate (Table 2) for the same data.

To apply our Promoter Prediction CNN (PPCNN) models to classifying sequences into promoters and non-promoters we designed **CNNprom.py** program. It takes the fasta format files as an input, together with the model parameters file, and outputs classification results for each sequence. If a sequence is classified as a promoter, the score assigned by network is provided in an output as well. To build the CNNProm program execution environment we installed Python, Theano and Keras libraries [32, 34] on the Softberry public Web server. The program can be run to analyze sequences of five studied organisms (two bacterial and three eukaryotic) at http://www.softberry.com/berry.phtml?topic=index&group=programs&subgroup=deeplearn. The developed PPCNN models have been recently applied in the programs for finding promoters in genomic sequences, which are also accessible at that server. The data sets with promoter and non-promoter sequences used to train and test CNN models can be downloaded at https://github.com/solovictor/CNNPromoterData.git.

Random substitution method to discover positionally conserved functional elements

By analyzing network behavior we can extract some information on significant elements of the input data. Promoter sequences usually contain binding sites of regulatory proteins. Some of them occupy various locations relative to TSS and can be found in direct or complementary DNA chain. However, there are a number of well-known functional sites (such as bacterial -10 –box or eukaryotic TATA-box) that occupy approximately the same position in each promoter sequence. To discover such sites we

suggest the following procedure. Take a window of length L (including positions from x_1 to x_2) and change the sequence within this window to a random sequence. Evaluate the accuracy of the site prediction after such change. Using sliding window moving from the beginning of a functional site sequence, we can build a performance profile that reflects an effect of a random sequence, inserted in each sequence position in place of an original sequence, on the accuracy of the site prediction. An example of such profile computed with window size 6 nt is shown in Fig 3.

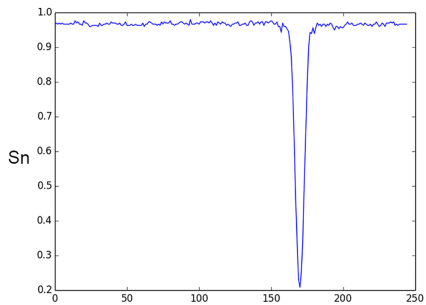


Figure 3: Effect of 6-nt sequence window substitution by random sequence on accuracy of classification of human TATA promoters.

X-axis is the window position, Y-axis is promoter identification sensitivity after a substitution.

We can see that substitution of the sequence located between -45 and -20 positions relative to TSS (located in position 201) of human promoters drastically decreases the prediction accuracy. These positions include the well-known functional motif called TATA-box. The sequence logo [42] demonstrating conserved sequences of that motif is shown in Fig 4.

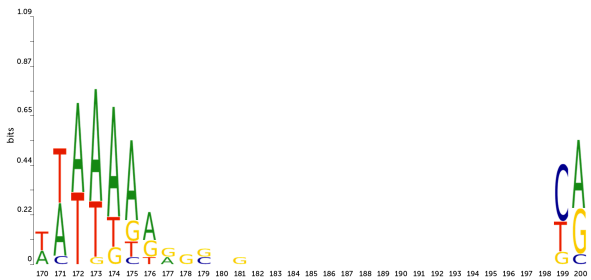


Figure 4: Sequence logo for human TATA promoter sequences in the TATA-box region and TSS region.

X-axis is a position in promoter sequence, Y-axis is informational content in bits.

Another interesting example was observed while applying the random substitution procedure to *Arabidopsis* non-TATA promoters, see Fig 5.

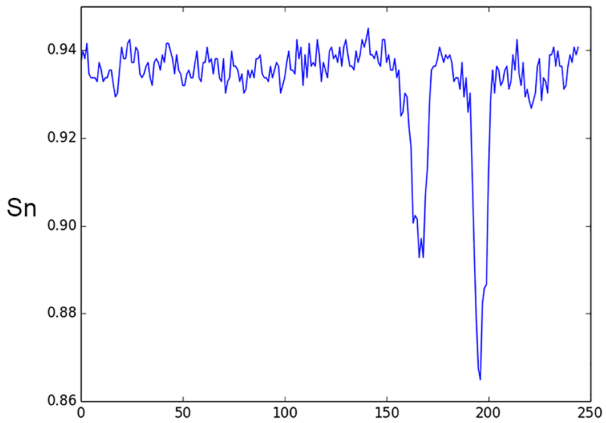


Figure 5: Effect of 6-nt sequence window substitution by random sequence on accuracy of classification of *Arabidopsis* **non-TATA promoters.**

X-axis is a position of a window, Y-axis is promoter identification sensitivity after a substitution.

Here we observe two positionally conserved and potentially functionally important elements (Fig 6). One is located approximately in positions -34–-28 and another in positions -2–0 relatively to TSS (located in position 201).

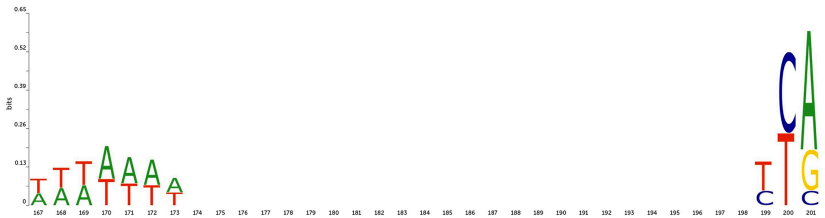


Figure 6: Sequence logo for *Arabidopsis* **non-TATA promoter sequences.**

X-axis is a position in promoter sequence, Y-axis is informational content in bits.

Thus, the suggested random substitution procedure can enable discovery of functionally important sites (sub-regions) that are still

often unknown. Due to relatively high accuracy of CNNprom promoter prediction it would be interesting to use it in known or predicted upstream gene regions in combination with gene-recognition software tools to improve gene identification accuracy, as well as to annotate promoter regions.

CONCLUSION

Present study demonstrates very good performance of CNN models in classifying promoter and non-promoter sequences. Accurate identification of promoters in long genome sequences, however, remains a major challenge, requiring not only accurate classifiers, but also appropriate selection of unique predictions among multiple overlapping high scoring genomic segments. In this task, it is also very important to account for multiple or alternative promoters for each transcription unit, possibly applying nonparametric methods recently described and tested on promoter regions of a model dicot plant *Arabidopsis thaliana* [43]. While we already incorporate developed CNN classifiers into a program of promoter identification in genome sequences, the approaches to resolve many difficult aspects of this task will be considered in our follow-up studies.

The suggested application of deep learning in promoter prediction and positional analysis of functional sites does not require knowledge of any specific promoter features. Since the convolution filters are able to automatically capture sequence motifs and other significant characteristics of biological/genomic sequences, this approach can be easily extended to identify promoters and other complex functional regions in sequences of many other genomes, making it very useful, especially considering that complete genomic sequence of thousand organisms will soon be available and how little transcriptional information is available for most of them.

AUTHOR CONTRIBUTIONS

- **Conceptualization:** VS RU.
- **Data curation:** VS RU.
- **Formal analysis:** RU VS.

- **Funding acquisition:** VS.
- **Investigation:** VS RU.
- **Methodology:** VS RU.
- **Project administration:** VS.
- **Resources:** VS RU.
- **Software:** RU VS.
- **Supervision:** VS.
- **Validation:** RU VS.
- **Visualization:** RU VS.
- **Writing – original draft:** VS RU.
- **Writing – review & editing:** VS RU.

REFERENCES

1. Sandelin A, Carninci P, Lenhard B, Ponjavic J, Hayashizaki Y, Hume D. Mammalian RNA polymerase II core promoters: insights from genome-wide studies. Nat Rev Genet. 2007;8(6):424–436. pmid:17486122
2. Solovyev V, Shahmuradov I, Salamov A. Identification of promoter regions and regulatory sites. Methods Mol Biol. 2010;674:57–83. pmid:20827586
3. Harley CB, Reynolds RP. Analysis of E.coli promoter sequences. Nucleic Acids Res. 1987;15:2343–2361. pmid:3550697
4. Lisser S, Margalit H. Compilation of e.coli mrna promoter sequences. Nucleic Acids Res. 1993;21:1507–1516. pmid:8479900
5. Jacques P, Rodrigue S, Gaudreau L, Goulet J, Brzezinski R. Detection of prokaryotic promoters from the genomic distribution of hexanucleotide pairs. BMC Bioinformatics. 2006;7:423. pmid:17014715
6. Meysman P, Collado-Vides J, Morett E, Viola R, Engelen Kea. Structural properties of prokaryotic promoter regions correlate with functional features. PLoS ONE. 2014;9. pmid:24516674
7. Fickett J, Hatzigeorgiou A. Eukaryotic Promoter Recognition. Genome Res. 1997;7:861–878. pmid:9314492
8. Bajic V, Brent M, Brown R, Frankish A, Harrow J, Ohler U, et

al. Performance assessment of promoter predictions on ENCODE regions in the EGASP experiment. Genome Biol. 2006;7:1–13. pmid:16925837

9. Shahmuradov I, Solovyev V, Gammerman A. Plant promoter prediction with confidence estimation. Nucleic Acids Research. 2005;33(3):1069–1076. pmid:15722481
10. Azad A, Shahid S, Noman N, Lee H. Prediction of plant promoters based on hexamers and random triplet pair analysis. Algorithms Mol Biol. 2011;6:19. pmid:21711543
11. Reese M. Application of a time-delay neural network to promoter annotation in the Drosophila melanogaster genome. Comput Chem. 2001;26(1):51–56. pmid:11765852
12. Prestridge D. Predicting Pol II promoter sequences using transcription factor binding sites. J Mol Biol. 1995;249(5):923–932. pmid:7791218
13. Knudsen S. Promoter2.0: for the recognition of PolII promoter sequences. Bioinformatics. 1999;15(5):356–361. pmid:10366655
14. Anwar F, Baker M, Jabid T, Hasan M, Shoyaib M, Khan H, et al. Pol II promoter prediction using characteristic 4-mer motifs: a machine learning approach. BMC Bioinformatics. 2008;9:414. pmid:18834544
15. Gordon L, Chervonenkis A, Gammerman A, Shahmuradov I, Solovyev V. Sequence alignment kernel for recognition of promoter regions. Bioinformatics. 2003;19(15):1964–1971. pmid:14555630
16. Solovyev V, Salamov A. Automatic Annotation of Microbial Genomes and Metagenomic Sequences. In: Metagenomics and its Applications in Agriculture. Nova Science Publishers, p: Biomedicine and Environmental Studies (Ed. Li R.W.); 2011. p. 61–78.
17. Wang H, Benham C. Promoter prediction and annotation of microbial genomes based on DNA sequence and structural responses to superhelical stress. BMC Bioinformatics. 2006;7:248. pmid:16677393
18. Yamamoto Y, Ichida H, Abe T, Suzuki Y, Sugano S, Obokata

J. Differentiation of core promoter architecture between plants and mammals revealed by LDSS analysis. Nucleic Acids Res. 2007;35(18):6219–6226. pmid:17855401

19. Civan P, Svec M. Genome-wide analysis of rice (Oryza sativa L. subsp. japonica) TATA box and Y Patch promoter elements. Genome. 2009;52(3):294–297. pmid:19234558
20. Krizhevsky A, Sutskever I, Hinton GE. Imagenet classification with deep convolutional neural networks. In Advances in neural information processing systems. 2012; p. 1097–1105.
21. Szegedy C, Liu W, Jia Y, Sermanet P, Reed S, Anguelov D, et al. Going Deeper With Convolutions. The IEEE Conference on Computer Vision and Pattern Recognition (CVPR). 2015; p. 1–9.
22. LeCun Y, Bengio Y, Hinton G. Deep learning. Nature. 2015;521:436–444. pmid:26017442
23. Schmidhuber J. Deep learning in neural networks: An overview. Neural Networks. 2015;61:85–117. pmid:25462637
24. Zhou J T O. Predicting effects of noncoding variants with deep learning-based sequence model. Nat Methods. 2015;12(10):931–934. pmid:26301843
25. Quang D, X X. DanQ:a hybrid convolutional and recurrent deep neural network for quantifying the function of DNA sequences. Nucleic Acids Res. 2016;. pmid:27084946
26. Graves A, Schmidhuber J. Framewise phoneme classification with bidirectional LSTM and other neural network architectures. Neural Net. 2005;18:602–610. pmid:16112549
27. Chen Y, Li Y, Narayan R, Subramanian A, Xie X. Gene expression inference with deep learning. Bioinformatics. 2016;2(12):1832–1839. pmid:26873929
28. Alipanahi B, Delong A, Weirauch M, Frey BJ. Predicting the sequence specificities of DNA- and RNA-binding proteins by deep learning. Nat Biotechnol. 2015;33:831–838. pmid:26213851
29. Gama-Castro S, Salgado H, Santos-Zavaleta A, et al LT D. RegulonDB version 9.0: high-level integration of gene regulation, coexpression, motif clustering and beyond. Nucleic Acids Res. 2016;44:133–143.

30. Ishii T, Yoshida Ki, Terai G, Fujita Y, Nakai K. DBTBS: a database of Bacillus subtilis promoters and transcription factors. Nucleic acids research. 2001;29(1):278–280. pmid:11125112

31. Dreos R, Ambrosini G, Périer R, Bucher P. EPD and EPDnew, high-quality promoter resources in the next-generation sequencing era. Nucleic Acids Research. 2013;41(Database issue)(D157–64). pmid:23193273

32. Chollet F. Keras: Deep Learning library for Theano and TensorFlow. GitHub; 2015. Available from: https://github.com/fchollet/keras.

33. Bastien F, Lamblin P, Pascanu R, Bergstra J, Goodfellow IJ, Bergeron A, et al. Theano: new features and speed improvements; 2012. Deep Learning and Unsupervised Feature Learning NIPS 2012 Workshop.

34. Theano Development Team. Theano: A Python framework for fast computation of mathematical expressions. arXiv e-prints. 2016;abs/1605.02688.

35. Nickolls J, Buck I, Garland M, Skadron K. Scalable Parallel Programming with CUDA. ACM Queue. 2008;6(2):40–53.

36. Kingma D, Ba J. Adam: A method for stochastic optimization. arXiv preprint arXiv:14126980. 2014;.

37. Burset M, Guigo R. Evaluation of gene structure prediction programs. Genomics. 1996;34(3):353–367. pmid:8786136

38. Matthews BW. Comparison of the predicted and observed secondary structure of T4 phage lysozyme. BiochemBiophysActa. 1975;405:442–451. pmid:1180967

39. Aken BL, Ayling S, Barrell D, Clarke L, Curwen V, Fairley S, et al. The Ensembl gene annotation system. Database. 2016;2016:baw093. pmid:27337980

40. Suzuki A, Wakaguri H, Yamashita R, Kawano S, Tsuchihara K, Sugano S, et al. DBTSS as an integrative platform for transcriptome, epigenome and genome sequence variation data. Nucleic acids research. 2015;43(D1):D87–D91. pmid:25378318

41. Pedregosa F, Varoquaux G, Gramfort A, Michel V, Thirion B, Grisel O, et al. Scikit-learn: Machine Learning in Python. Journal of Machine Learning Research. 2011;12:2825–2830.

42. Schneider TD, Stephens RM. Sequence logos: a new way to display consensus sequences. Nucleic acids research. 1990;18(20):6097–6100. pmid:2172928

43. Tatarinova T, Kryshchenko A, Triska M, Hassan M, Murphy D, Neely M, et al. NPEST: a nonparametric method and a database for Transcription Start Site prediction. Quantitative biology. 2013;1(4):261–271. pmid:25197613

CHAPTER 7

Intrusion Detection System Using Deep Neural Network for In-Vehicle Network Security

Min-Joo Kang, Je-Won Kang

The Department of Electronics Engineering, Ewha W. University, Seoul, Republic of Korea

ABSTRACT

A novel intrusion detection system (IDS) using a deep neural network (DNN) is proposed to enhance the security of in-vehicular network. The parameters building the DNN structure are trained with probability-based feature vectors that are extracted from the in-vehicular network packets. For a given packet, the DNN provides the probability of each class discriminating normal and attack packets, and, thus the sensor can identify any malicious attack to the vehicle. As compared to the traditional artificial neural network applied to the IDS, the proposed technique adopts recent advances in deep learning studies such as initializing the parameters through the unsupervised pre-training of deep belief networks (DBN), therefore improving the detection accuracy. It is demonstrated with experimental results that the proposed technique can provide a real-time response to the attack with a significantly improved detection ratio in controller area network (CAN) bus.

INTRODUCTION

Recently, a major advance in an automotive system has been made with integrating a number of computing devices called Electronics Control Unit (ECU). ECU is used for controlling and monitoring a subsystem of a vehicle for energy efficiency enhancement, and noise and vibration reduction. The ECU replaces conventional mechanical controlling parts [1]. More recently, automotive networking services such as Vehicle-to-Vehicle (V2V) and Vehicle-to-Infrastructure (V2I) require computing devices to perform intra-vehicular communication [2] and inter-vehicular communication [3, 4]. The vehicular communications can be applied to many practical traffic systems [5, 6]. Tang *et al*. propose to use the communications to understand driving behaviors such as each vehicle's speed and fuel consumptions [7, 8]. Jin *et al*. show the robust V2V communications depending on a traffic stream [9]. Kesting *et al*. developed a novel message passing scheme in the communication [10]. In [11–13] efficient fuel consumptions are considered with estimating the speeds of the connected cars or their distances. Cooperative platooning enabled by the wireless communications can also improve traffic flow [14]. In the Grand Cooperative Driving Challenge (GCDC) the best performing results show the recent innovations in the fields of realistic cooperative driving [15–17]. Accordingly the ability of the computing devices in a vehicle dramatically increases.

Different communication protocols are developed to support the communication [4]. Among the protocols, Controller Area Network (CAN) [18] as the de factor standard of in-vehicle network communication is such a simple communication protocol supporting to connect sensors and actuators with ECUs, and the adoption of CAN facilitates emerging automotive applications [19]. Quite often important information such as diagnostic, informative, and controlling data is delivered through a CAN bus to serve the automotive services such as self-driving and advanced driver assistance systems (ADAS). The information must be secured for the safety of a driver. However, the growth of networking capability is accompanied with significant security concerns, and unfortunately the in-vehicular network includes several security flaws [20–23]. ECUs can obtain any ECU-to-ECU broadcasting messages in the same bus, and they are unable to identify a sender [20]. It is shown in how faked packets

can confuse critical components securing driver's safety by malicious attacks such as a packet injection and data manipulation [21–23].

There have been several research works considering safety problems in inter and intra vehicular communications [24–29]. In particular, an intrusion detection sensor (IDS) gains much attention due to the efficiency and simplicity in detecting the attacks [24–27]. Hoppe *et al*. propose an intrusion detection method by using several representative attack patterns predefined in a database [25]. Larson *et al*. develop a specification-based approach, comparing the behavior of the current specification system to the designated patterns [26]. In [27], a sensor-based detection method recognize a malicious intrusion by using several sensors designed for the attack scenarios. Secured protocols in accordance with the conventional specifications are proposed in [28, 29].

The previous intrusion detection methods may be effective only for specific threat models that have been already considered in design stages [30, 31]. To cope with the problem machine learning based IDS techniques are employed, mainly, for conventional communication networks [32]. The idea is to capture underlying statistical features of data and use them to detect any malicious attack [33]. Intrusion detection methods using artificial neural network (ANN) [34, 35] and support vector machine [36] are developed for classifying attack types. The advanced machine learning algorithms are barely used for a vehicular network because the computing power of the conventional ECU is limited to process the complex process. However, the computing power of ECU has been notably increasing to process enormous real-time tasks in the most recent vehicular system [19].

In this paper, an intrusion detection system using the deep neural network (DNN) structure [37] is proposed to secure the in-vehicular network, *e.g*. CAN network. The proposed technique trains high-dimensional CAN packet data after the dimension reduction to figure out the underlying statistical properties of normal and attack packets, and, in defense, it extracts the corresponding features to identify the attack. DNN has been shown to be effective for classifying statistical patterns and mapping complex non-linear input-to-output relations in various research fields such as artificial intelligence, multimedia processing, security [37–40] as well as in intelligent vehicular systems

[41–44]. Our work is the first to employ the deep learning structure in the IDS of in-vehicular networks, which differs from earlier ANN-based intrusion detection methods [34, 35]. Specifically, we use unsupervised deep belief network (DBN) pre-training methods [45] to efficiently train the parameters initializing the deep neural network. The parameters are tuned later to achieve a better classification result with the supervised learning. Experimental results demonstrate that the proposed method yields a superior performance in terms of a classification error with little computation complexity in the decision.

RELATED WORK

CAN

CAN is designed for half-duplex and high-speed broadcast bus in-vehicular network, providing the communication rate up to 1Mbps [18]. The CAN protocol is widely used in automotive manufactures as the de factor standard. In the protocol, each ECU broadcasts a message to the network using a data packet. A sender ECU may include its unique ID number in the packet, and a receiver ECU retrieves the packet by identifying the ID of the sender. Thus CAN packet has no explicit destination field.

Fig 1 shows the syntax of the CAN data packet. The arbitration field includes an 11 bit ID field where each ID corresponds to a specific ECU. The arbitration field offers two functions: (1) prioritizing a message by the ID in the decreasing order and (2) enabling each ECU to filter an interesting message. The ID field is used for a collision avoidance algorithm in the bus, which is extended to 29 bits later. The data field contains maximum 8 bytes information to be transmitted in a message, for example, the value of the steering wheel angle and the on/off status of components in display panel. The control field contains the size of the data field. The cyclic redundancy check (CRC) field detects any error in the data packet. The acknowledgement field confirms the receipt of a valid CAN packet.

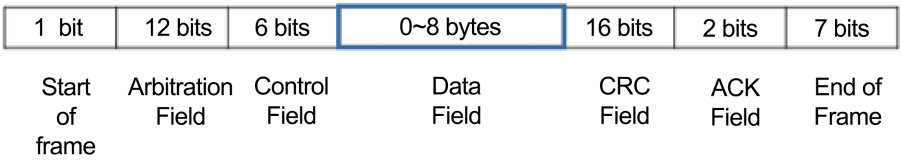


Figure 1: CAN packet syntax.

Intrusion Detection with Machine Learning

Intrusion detection techniques have been actively studied to help the conventional network resist malicious attacks. In literature quite a number of the intrusion detection techniques are developed based on machine learning techniques, based on the assumption that the patterns of the attack packets differ from those of the normal packets. In [34–36] artificial neural networks (ANN) and support vector machine (SVM) are applied to the intrusion detection, using a statistical modeling on a packet data. In [46] a frequency-based encoding method is used for a packet feature in ANN and SVM. The aforementioned works are based on supervised machine learning techniques, and, thus a number of labeled data sets are required in the training. As compared to the approach, Kayacik *et al*. employ an unsupervised machine learning technique such as a self-organized feature map (SOM) for network intrusion detection.

Fig 2 shows a common architecture of the IDS based on machine learning. The IDS includes various modules for gathering and analyzing a large amount of data packets. Typically, the monitoring module detects a type of an incoming packet after feature extraction. The profiling module contains the features trained off-line. If the monitoring module identifies a new attack type, the profiling module may update the database of the profiling module for upcoming packets.

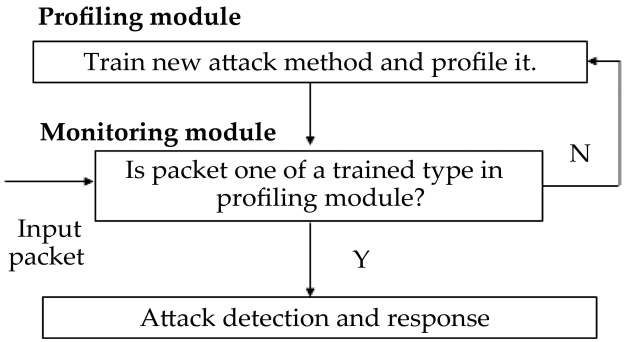


Figure 2: Architecture of IDS based on machine learning techniques.

Deep Learning for Classification

Deep learning refers to a machine learning technique using an architecture comprising a number of hierarchical layers of non-linear processing stages. The architecture can be categorized into two types, *i.e.*, a discriminative deep architecture and a generative deep architecture, depending on how the architectures are exploited [47]. The discriminative deep architecture provides abilities for pattern classification with the supervised learning as in the conventional feed-forward artificial neural networks (ANN). The deep structure, namely, deep neural network (DNN) can be augmented with multiple hidden layers from the ANN structure.

However, the augmented neural networks are inefficiently trained using the back-propagation learning with a gradient descent optimization due to the vanishing gradient problem [48]. In the back propagation, the gradient of the error surface is computed in each layer while the gradient exponentially decreases with the number of the layers, thus causing a extremely slow convergent speed. To prevent the problem, the generative deep architecture characterizing the correlation of the observed data and the associated classes is used for initializing parameters of the discriminative architecture [49], called the unsupervised pre-training scheme. In [49], the weight parameters interconnecting nodes in adjacent layers are efficiently trained using a top-down approach by considering the nodes as restricted Boltzmann Machines (RBM). After the pre-training, fine-tuning is performed using the gradient descent method with the supervised learning as in the conventional feed-forward ANN

[50]. The deep belief networks (DBN) [45] as a probabilistic generative model include several layers of stochastic hidden units on top of a single bottom layer of observed data to efficiently solve the vanishing gradient problem [49, 50]. The DBN structure is shown in Fig 3(a) where the top-two layers contain undirected connections, and the lower layers contain directed connections to the layers below. In this top-down manner, the weight vector w_n is generated to form the visible data vector v, and the set of w_n is used for initializing the parameters of the proposed classifiers later. The solution is used similarly for many practical applications [41, 43, 51] using the DBN learning structures, and, therefore adopted in the proposed technique to pretrain the parameter as well.

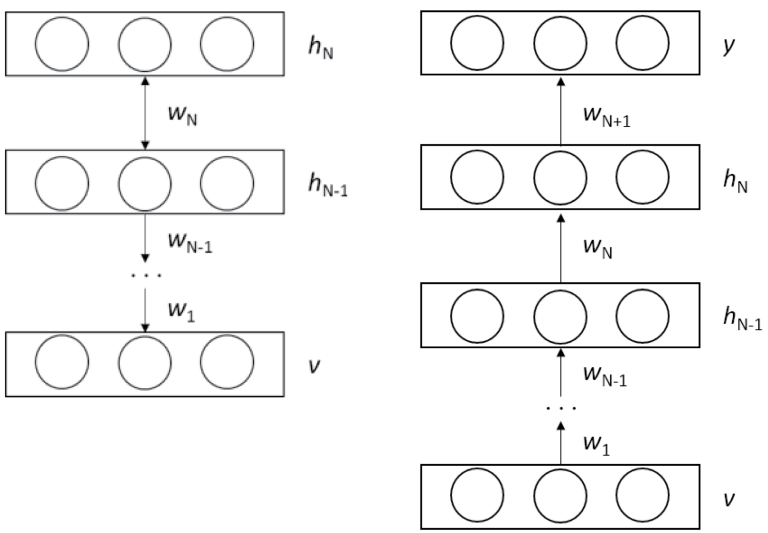


Figure 3: **(a) DBN structure with n hidden layers built with a top-down manner and (b) DNN structure involving the pre-trained** wight parameters in n **hidden layers built with a bottom-up manner.**

PROPOSED TECHNIQUE

Proposed Intrusion Detection System with Deep Neural Network Structure

The proposed intrusion detection system considers a general type of an attack scenario where malicious data packets are injected into an in-

vehicle CAN bus. In-vehicular networks are accessed from the mobile communication links [20] such as 3G, 4G, and WIFI or a self-diagnostic tool such as OBD paired with the driver's mobile device [29]. The proposed intrusion detection system monitors broadcasting CAN packets in the bus and determines an attack, as shown in Fig 4.

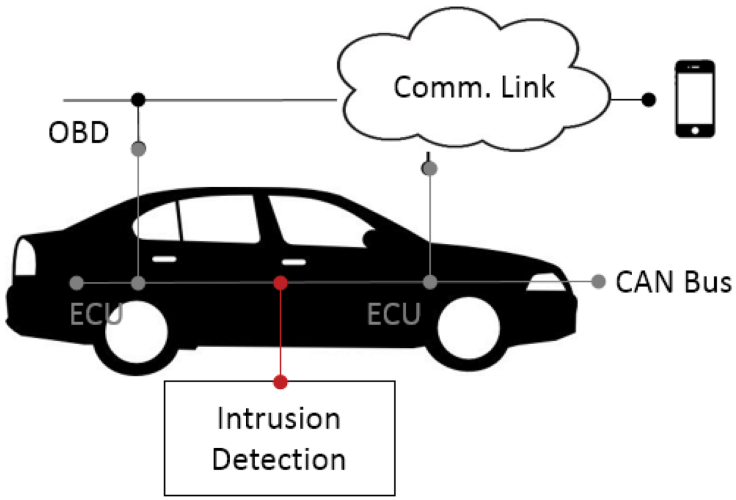


Figure 4: Attack scenario in the connected car.

Our IDS design consists of two main phases, *i.e.*, the training phase and the detection phase as in the conventional machine-learning based IDS, as shown in Fig 5. The training phase is performed off-line as the training is time-consuming. In the training phase a CAN packet is processed to extract a feature that represents a statistical behavior of the network. Each training CAN packet has its binary label, *i.e.*, either a normal packet or an attack packet in supervised learning. Thus the corresponding features are expected to represent the label information. We adopt the DNN structure to train the features, in which the weight parameters on the edges connecting the nodes are obtained. The detection phase is also shown in Fig 5. The same feature is extracted from an incoming packet through a CAN bus, and the DNN structure computes with the trained parameters to make the binary decision.

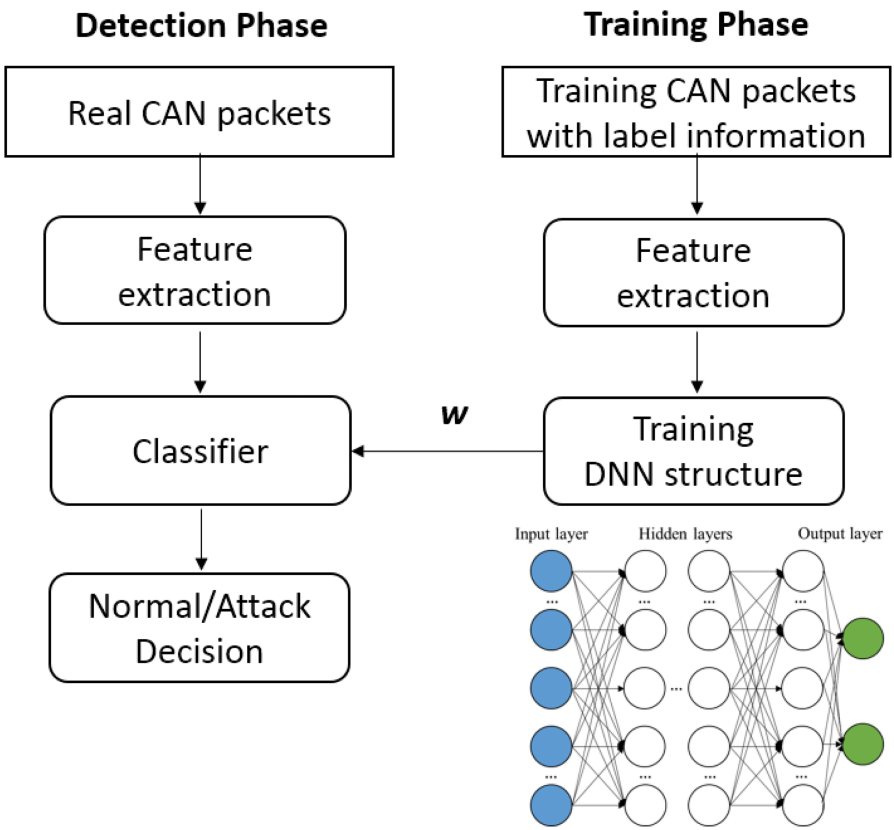


Figure 5: Overview of the proposed intrusion detection system.

The learning structure should be configured for the supervised learning as the DBN model in Fig 3(a) provides unsupervised learning mechanism. To this aim, the final classification layer including label information is added to the top layer of the DBN model to construct the discriminative deep learning structure. Fig 3(b) shows the modified structure into the deep feed-forward ANN structure where the structure is trained with the bottom-up supervised learning manner, owing to the label information y. It is highlighted that the weights w_i in the hidden nodes of the DBN structure are obtained from the unsupervised pre-training at first. However, the parameters are used only for initializing the weights, and, they are fine-tuned by using the gradient descent method in the deep feed-forward ANN structure later.

CAN Packet Feature

CAN feature is an abstract representation of a CAN packet. The feature is designed by considering computational efficiency. In other words, the feature is extracted directly from a bitstream of a CAN packet so that the decoding is not necessary during the extraction. The occurrences of bit-symbols in a data packet are taken into an account. In particular We choose the *DATA* field that includes 64 bit positions (= 8 Bytes) in the CAN syntax and investigate the probability distributions of the bit-symbols. Mathematically the data vector $\mathbf{p}_o \in \mathbb{R}^{64}$ is given as,

$$\mathbf{p}_o = \{P(b_0), P(b_0), \ldots, P(b_{63})\}, \quad (1)$$

where $P(b_i)$ is the probability of a bit-symbol "1" observed in the i-th bit position, and

$$\mathbf{p} = L(\mathbf{p}_o), \quad (2)$$

where the function $L : \mathbb{R}^{64} \rightarrow \mathbb{R}^{64}$ is the logistic function: if $P(b_i)$ is greater than a half, the probability is mapped to 1. Otherwise, it is mapped to 0.

All the bit positions in the *DATA* field may be used for generating the feature. However, the dimension can be reduced by considering specific semantics in the corresponding syntax element. The proposed technique regards mode information and value information according to the semantics. The mode information represents a command state of an ECU, for example, controlling wheels, and the value information represents the value of the mode, for example, the wheel angle or the speed, as shown in Fig 6. The mode information is constant in a short period, while the value information may change with some noises. In the proposed technique the value information is only used for the training phase. The usage of the mode information will be shown in the detection phase.

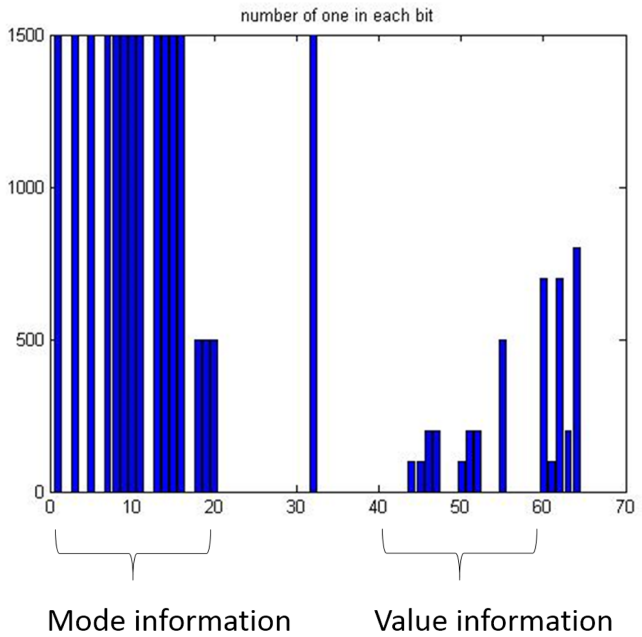


Figure 6: The occurrences of a bit-symbol "1" in the *DATA* **field of 8 Bytes, consisting of mode information and value information, at time** *t*.

Denote $\mathbf{p}_v$ is the data vector reduced from $\mathbf{p}$. Then the feature vector $\mathbf{v}$ at time instance n is generated as,

$$\mathbf{v}(n) = \mathbf{p}_v(n) \oplus \mathbf{p}_v(n-1), \tag{3}$$

where $\oplus$ is an exclusive-or operator applied to each position of bits in the vector.

Training the Deep Neural Network Structure

The learning mechanism of the proposed DNN structure to classify a normal packet and an attack packet is explained. Fig 7 shows an input layer, multiple hidden layers, and an output layer. The feature vector is inputted to the input nodes of the structure. Each node in Fig 7computes an output with an activation function using rectified linear unit (ReLU), and the linear combinations of the outputs are linked to the next hidden layers.

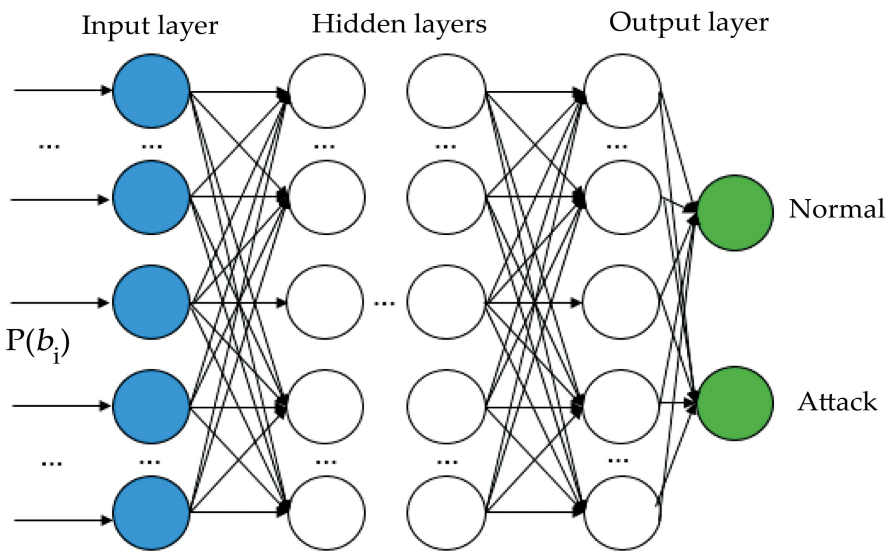


Figure 7: Deep neural network structure in the proposed technique.

Considering a supervised learning, there is a training set $\{(\mathbf{v}^1, y^1), (\mathbf{v}^2, y^2),\ldots, (\mathbf{v}^K, y^K)\}$ of Ksamples. The data vector $\mathbf{v}$ is the feature vector consisting of the probability of the bit-symbol "1", and y is the binary label information, assigned to each training sample. In the learning phase, the input feature $\mathbf{v}$ goes through the visible nodes at the bottom of the neural network structure, in which initial weights are given by the DBN learning. Then, the weight vectors are fine-tuned in sequel. For this, we minimize a cost function C given as the mean squared error function between the prediction value and the output:

$$C(\mathbf{w};\mathbf{v},y) = \frac{1}{2} \| h_w(\mathbf{v}) - y\|^2, \tag{4}$$

where $\mathbf{w}$ is the set of the weights in the network to be trained, y is the label, and $h_w(\mathbf{v})$ is a hypothesis function yielding an estimated output. The overall cost function for a batch training is defined as

$$C(\mathbf{w}) = \frac{1}{K}\sum_k C(\mathbf{w};\mathbf{v}^k,y^k) + \frac{\lambda}{2}\sum_n^N \sum_i^{M_l} \sum_j^{M_{l+1}} (w_{ji}^n)^2, \tag{5}$$

where N is the depth of the neural network, M_l is the number of the nodes in the l-th layer, and $w_{ji}^n \in \mathbf{w}$ is the weight of the edges between a node i in the layer $n-1$ and a node j in the layer n. We want to obtain the optimal parameter set $\mathbf{w}^*$ to achieve the minimization of the objective function as follows:

$$\mathbf{w}^* = \arg\min_{\mathbf{w}} C(\mathbf{w}), \tag{6}$$

which can be achieved by the back propagation algorithm. In the back propagation algorithm the weight vectors are updated from the top layer to the bottom layer by using the stochastic gradient method,

$$w_{ji}^n = w_{ji}^{n-1} + \zeta \frac{\partial}{\partial w_{ji}^{n-1}} C(\mathbf{w}), \tag{7}$$

where ζ is an adaptation parameter.

Attack Detection

The class of a testing CAN packet is predicted in the detection phase. The output is computed with the trained weight parameters and the feature set extracted from the testing CAN packet as in the training. The classifier provides the logistic value 0 or 1, telling if the sample is normal packet or the attack packet, respectively.

There can be a number of attack scenarios considered in an ECU, and the weight vectors can be trained fitted to each scenario. The mode information is used for identifying the scenario in the proposed method, so that the appropriate training set can be applied. For this, template matching is developed in the proposed method. The template comprising the mode information refers to the information along with the training samples used for the specific scenario. Fig 8shows an example of the template matching where the template is colored with yellow. As shown, if the template is matched between in the training sample and in the CAN packet to be tested, the detector uses the corresponding trained parameters obtained from the value information.

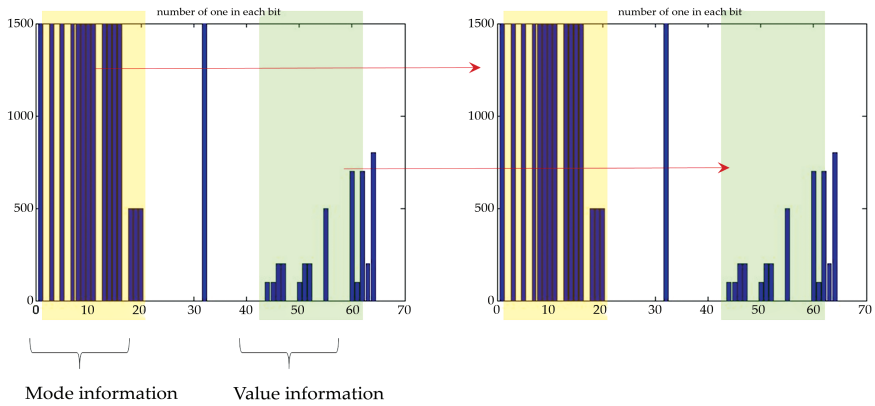


Figure 8: Template matching method to find the proper trained parameters.

EXPERIMENTAL RESULTS

Data Set

We simulate the in-vehicular network communicating with several ECUs and the packets in the CAN bus as shown in Fig 9. The packets are created by the packet generator named Open Car Test-bed and Network Experiments (OCTANE) [52] in the simulation, and they are sent to the CAN bus. Our IDS monitors the network packets. The number of the generated packets is about 200,000 in a simulation. To avoid the over-fitting problem, we assign 70% packets to the training data and 30% packets to the testing data. In the attack scenario some of the packets are injected and are manipulated to deceive the system. Note the attack packets are inserted with some time intervals, so that they are not burst in the in-vehicle network.

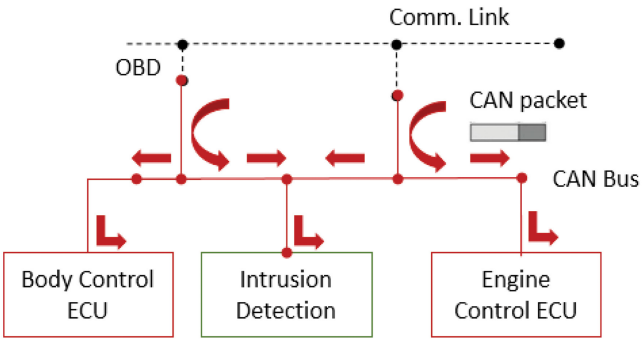


Figure 9: Simulation configuration.

Table 1 shows the CAN data packets including the identifiers (ID) and *DATA* fields to control ECUs, considered in the experiment. Because each ECU has a unique ID, the data packet can be generated for the corresponding ECU. The constant bit fields in the packet syntax are used for the mode information while the variables denoted by α_i β_i are used for the time-varying value information. Furthermore we add a Gaussian noise into the value information to give randomness.

Table 1: CAN packets used in the simulation

CAN ID	*DATA* field	Target ECU
10F	02 $\alpha_0\beta_0$ A0 B2 $\alpha_1\beta_1$ $\alpha_2\beta_2$ $\alpha_3\beta_3$ $\alpha_4\beta_4$	Engine
24F	44 $\alpha_0\beta_0$ 1B A5 $\alpha_1\beta_1$ $\alpha_2\beta_2$ $\alpha_3\beta_3$ $\alpha_4\beta_4$	Body control
400	00 $\alpha_0\beta_0$ EF 01 $\alpha_1\beta_1$ $\alpha_2\beta_2$ $\alpha_3\beta_3$ $\alpha_4\beta_4$	Display panel

doi:10.1371/journal.pone.0155781.t001

Performance Evaluation

We measure the false negative rate and the false positive rate to evaluate the classification performance. R_A and R_N refer to the detection ratios of an attacking packet and a normal packet, respectively, given as,

$$R_A(\%) = \frac{D_A}{T_A} \times 100, \tag{8}$$

And

$$R_N(\%) = \frac{D_N}{T_N} \times 100, \tag{9}$$

where T_A and T_N are the total number of the attack packets and normal packets, respectively, and D_A and D_N are the number of the detected attack packets and normal packets, respectively. False positive rate should be small because it is considered more important in the attack detection. To evaluate this, we show the Receiver Operating Characteristic (ROC). The curves can be obtained by plotting pairs of the false positive rate and the hit rate with a given detection threshold, so that it provides the means to measure the trade-off between the false positive error and the correct detection. It is noted that a ROC curve shows a better detection performance when the points are ploted more in the top-left corner. Fig 10 shows the ROC curve of the proposed technique as compared to those of the artificial neural network (ANN) and the support vector machine (SVM) in the experiments. The curves clearly show that the proposed technique outperforms the conventional works in the detection ratio. The detection ratio is more than 99% when the false positive error is less than 1-2%. We also show confusion matrices in Fig 11 to evaluate the quantitative detection performances. The performance of the proposed method provides a significantly high detection ratio. The false positive error is only about 1.6%, and the false negative error is about 2.8%. The total accuracy is about 97.8%

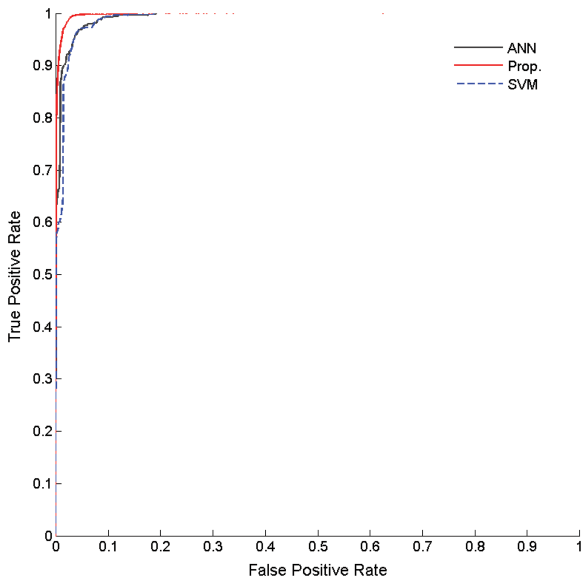


Figure 10: Intrusion detection performance evaluations with ROC curves.

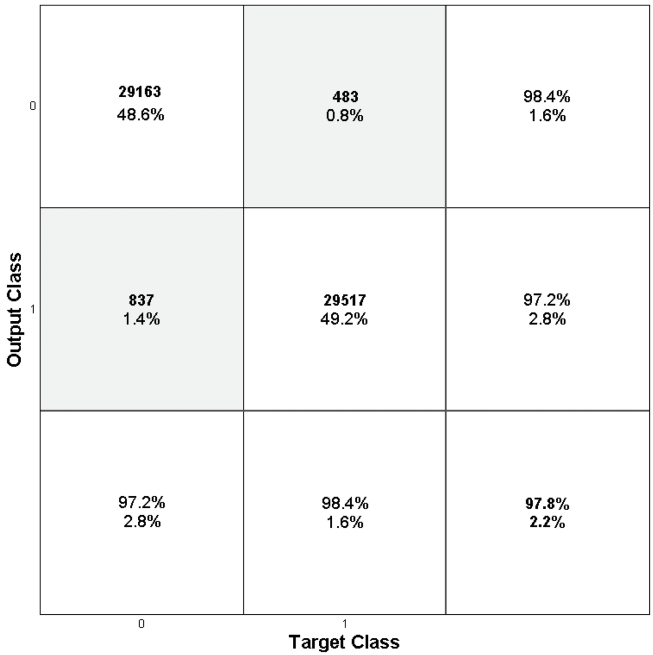


Figure 11: Confusion Matrix Results.

We compare the intrusion detection performances of two variations of the proposed deep learning structure using the DNN structure to that of the conventional feed-forward artificial neural network with respect to the number of the layers. As shown in Fig 12, the proposed technique yields more accurate and consist detection performance $(R_A + R_N)/2$ of the two scenarios than the feed-forward ANN. The ANN structure suffers from the vanishing gradient problem, causing the unstable performances with the number of the layers. For example, the lowest detection performances are observed when the number of the layer is 11. In contrast, the performance of the proposed method is significantly higher than the conventional ANN structure, but also the performance is monotonically increasing with the number of the layers. Next we show the detection performances of two variations of the proposed method. In Fig 12, DNN(ALL) presents the proposed method using a feature including all the bits (64 bits) in the *DATA* field. DNN(M+V) uses the feature including only the value information, *i.e.* α_i β_i in Table 1. As shown in Fig 12, DNN(M+V) is the best-performing method.

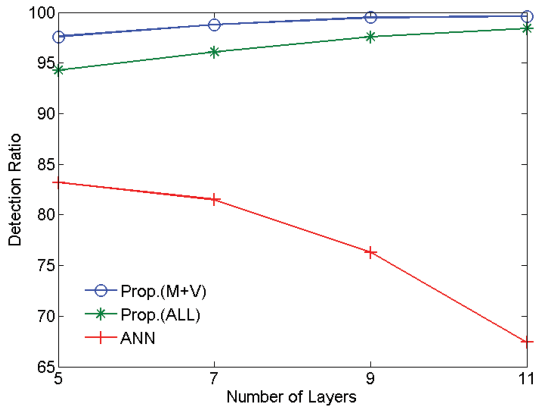


Figure 12: Intrusion detection performances with respect to the number of the layer.

We also show the time complexity in the detection, depending on a different number of hidden layers in Table 2. The training time represents the measurement time needed in training the DNN structure in a training phase, and the testing time represents the measurement time in examining each packet over the network. The time complexity in a training is about 4-11 seconds, and, thus the training should be done off-line. However, the time complexity in a testing time during the packet inspection is the only 8-9 μs for processing features per packet and 2-5 *ms* for classifying the packets, which can be applied to a real-time application.

Table 2: Time complexity in a different number of layers

Layers	Testing(s)	Testing	
		Feature extraction(μs)	Classification(*ms*)
5	4.15	8.4	2.05
7	6.32	8.5	2.29
9	9.58	8.7	3.17
11	10.81	8.7	3.78

doi:10.1371/journal.pone.0155781.t002

CONCLUSION

We proposed an efficient intrusion detection system (IDS) based on a deep neural network (DNN) for the security of in-vehicular network. We trained the parameters of DNN with probability-based feature vectors extracted from the in-vehicular network packets by using unsupervised pre-training method of deep belief networks, followed by the conventional

stochastic gradient descent method. The DNN provides the probability of each class to discriminate normal and hacking packets, and, thus the system can identify any malicious attack to the vehicle as a result. We also proposed a novel feature vector comprising the mode information and the value information extracted from the network packets, and they are efficiently used in the training and the testing. It was demonstrated with experimental results that the proposed technique could provide a real-time response to the attack with a significantly accurate detection ratio about 98% on average when the computational complexity with the number of the layers is modestly small.

ACKNOWLEDGMENTS

This research was supported by the Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education (NRF-2014R1A1A2056587).

AUTHOR CONTRIBUTIONS

Conceived and designed the experiments: MJK JWK. Performed the experiments: MJK JWK. Analyzed the data: MJK JWK. Contributed reagents/materials/analysis tools: MJK JWK. Wrote the paper: MJK JWK.

REFERENCES

1. Park T, Han C, Lee S. Development of the electronic control unit for the rack-actuating steer-by-wire using the hardware-in-the-loop simulation system. Mechatronics 2015; 15: 899–918.
2. Tuohy S, Glavin M, Hughes C, Jones E, Trivedi M, Kilmartin L. Intra-Vehicle Networks: a review. IEEE Trans. on Intelligent Transportation Systems 2015; 2: 534–545.
3. Biswas S, Tatchikou R, Dion F. Vehicle-to-vehicle wireless communication protocols for enhancing highway traffic safety. IEEE Signal Processing Magazine 2006; 44: 82–97.
4. Fan Y, Dao L, Crolla DA. Integrated vehicle dynamics control state-of-the art review. Vehicle Power and Propulsion Conference 2008.
5. Tsugawa S. Inter-vehicle communications and their applications to

intelligent vehicles: an overview. IEEE Intell. Veh. Symp. 2002.

6. Lenz H, Wagner CK, Sollacher R. Multi-anticipative car-following model, Eur. Phys. J.B 1999; 7.
7. Tang T, Shi W, Shang H, Wang Y, A new car-following model with consideration of inter-vehicle communication. Nonlinear Dynamics 2014; 76: 2017–2023.
8. Tang T, Shi W, Shang H, Wang Y. An extended car-following model with consideration of the reliability of inter-vehicle communication. Measurement 2014; 58: 286–293.
9. Jin WL, Recker WW. Instantaneous information propagation in a traffic stream through inter-vehicle communication. Transp. Res. B 2006; 3.
10. Kesting A, Treiber M, Helbing D. Connectivity Statistics of Store-and-forward Intervehicle Communication. IEEE Transactions on Intelligent Transportation System 2010.
11. Yu S, Shi Z. Fuel consumptions and exhaust emissions induced by the cooperative adaptive cruise control strategy. International Journal of Modern Physics B 2015; 29(14)
12. Yu S, Shi Z. The effects of vehicular gap changes with memory on traffic flow in cooperative adaptive cruise control strategy. Physica A 2015; 428(15): 206–223.
13. Yu S, Shi Z. Dynamics of connected cruise control systems considering velocity changes with memory feedback. Measurement 2015; 64: 34–48.
14. Yu S, Shi Z. An extended car-following model at signalized intersections. Physica A 2014; 407(1): 152–159.
15. Nunen E, Kwakkernaat R, Ploeg J, Netten B. Cooperative Competition for Future Mobility. IEEE Transactions on Intelligent Transportation System 2012; 13(3): 1018–1025.
16. Geiger A, Lauer M, Moosmann F, Ranft B, Rapp H, Stiller C, et al. Team Annie WAY's Entry to the 2011 Grand Cooperative Driving Challenge. IEEE Transactions on Intelligent Transportation System 2012; 13(3): 1018–1025.
17. Lidstrom K, Sjoberg K, Holmberg U, Andersson J, Bergh F, Bjade M, et al. A Modular CACC System Integration and Design. IEEE

Transactions on Intelligent Transportation System 2012; 13(3): 1008–1017

18. Farsi M, Ratcliff K, Barbosa M. An overview of Controller Area Network. Computing and Control Engineering Journal 1999; 10.
19. Johansson KH, Aurngren M, Nielsen L. Vehicle applications of controller area network. Handbook of Networked and Embedded Control Systems 2005.
20. Koscher K, Czeskis A, Roesner F, Patel S, Kohno T, Checkoway S, et al. Experimental security analysis of a modern automobile. IEEE Symposium on Security and Privacy, 2010.
21. Charlie M, Chris V. Adventures in Automotive Networks and Control Units. 2013.
22. Checkoway S, McCoy D, Kantor B, Anderson D, Shacham H, Savage S, et al. Comprehensive Experimental Analyses of Automotive Attack Surfaces. Proceedings of USENIX Security 2011.
23. Kleberger P, Olovsson T, Jonsson E. Security aspects of the in-vehicle network in the connected car. Intelligent Vehicles Symposium (IV) 2011.
24. Kemmerer RA and Vigna G. Intrusion detection: A brief history and overview. Computer 2002.
25. Hoppe T, Kiltz S, Dittmann J. Security Threats to Automotive CAN Networks—Practical Examples and Selected Short-Term Countermeasures. Proceedings of the 27th International Conference SAFECOMP 2008.
26. Larson E, Nilsson, Dennis K, Jonsson E. An approach to specification-based attack detection for in-vehicle networks. IEEE Intelligent Vehicles Symposium 2008.
27. Muter M, Groll A, Freiling FC. Structured approach to anomaly detection for in-vehicle networks. 6th International Conference on Information Assurance and Security 2010.
28. Patsakisa C, Delliosb K, Bourochea M. Towards a distributed secure in-vehicle communication architecture for modern vehicles. Computers and Security 2014.
29. Woo S, Jo HJ, Lee DH. A Practical Wireless Attack on the Connected Car and Security Protocol for In-Vehicle CAN. IEEE Trans. on

Intelligent Transportation Systems 2015.

30. Tyagi P, Dembla DA. Investigating the Security Threats in Vehicular ad hoc Networks (VANETs): Towards Security Engineering for Safer on-road Transportation. Advances in Computing, Communications and Informatics 2014.
31. Sun X, Yan B, Zhang X, Rong C. An Integrated Intrusion Detection Model of Cluster-Based Wireless Sensor Network. PLoS ONE 2015; 10(10)
32. Deepaa AJ, Kavitha V. A Comprehensive Survey on Approaches to Intrusion Detection System. Procedia Engineering 2012.
33. Tsaia C, Hsub Y, Linc C, Lin W. Intrusion detection by machine learning: A review. Expert Systems with Applications 2009.
34. Golovko V, Kochurko P. Intrusion Recognition Using Neural Networks. Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications 2005.
35. Zhang Z, Li J, Manikopoulos C, Jorgenson J, Ucles JP. HIDE: a Hierarchical Network Intrusion Detection System Using Statistical Preprocessing and Neural Network Classification, IEEE Workshop on Information Assurance and Security 2001.
36. Hu W, Liao Y, Vemuri V R. Robust Anomaly Detection Using Support Vector Machines. International Conference on Machine Learning 2003.
37. Hinton G, Deng L, Yu D, Dahl G, Mohamed A, Jaitly N, et al. Deep Neural Networks for Acoustic Modeling in Speech Recognition: The Shared Views of Four Research Groups,Communications Magazine 2012.
38. Bengio Y. Learning deep architectures for AI. Foundat. and Trends Mach. Learn. 2009.
39. Krizhevsky A, Sutskever I, Hinton GE. ImageNet Classification with Deep Convolutional Neural Networks, Advances in Neural Information Processing Systems 25 (NIPS 2012) 2012.
40. Wu J, Peng D, Li Z, Zhao L, Ling H. Network Intrusion Detection Based on a General Regression Neural Network Optimized by an Improved Artificial Immune Algorithm. PLoS ONE 2015; 10(3)

41. Lv Y, Duan Y, Kang W, Li Z, Wang F. Traffic Flow Prediction With Big Data: A Deep Learning Approach. IEEE Trans. on Intelligent Transportation Systems 2015.
42. Zhang J, Wang F, Wang K, Lin W, Xu X, Chen C. Data-driven intelligent transportation systems: A survey. IEEE Trans. on Intelligent Transportation Systems 2011.
43. Dan C, Ueli M, Jonathan M, Jürgen S, Multi-column deep neural network for traffic sign classification. Neural Networks 2012.
44. Ma X, Yu H, Wang Y, Wang Y. Large-Scale Transportation Network Congestion Evolution Prediction Using Deep Learning Theory. PLoS ONE 2015; 10(3).
45. Hinton GE, Osindero S, Teh Y, A fast learning algorithm for deep belief nets. Neural Computation 2006.
46. Chen WH, Hsu SH, Shen HP, Application of SVM and ANN for intrusion detection, Computers and Operations Research 2005.
47. Deng L, An Overview of Deep-Structured Learning for Information Processing. APSIPA 2011.
48. Bengio Y, Simard P, Frasconi P, Learning long-term dependencies with gradient descent is difficult. IEEE Trans. on Neural Networks 1994. pmid:18267787
49. Erhan D, Bengio Y, Courville A, Manzagol P, Vincent P, Bengio S. Why Does Unsupervised Pre-training Help Deep Learning. The Journal of Machine Learning Research 2010.
50. Bengio Y, Lamblin P, Popovici D, Larochelle H. Greedy layerwise training of deep networks. Advances in Neural Information Processing Systems 25 (NIPS 2012) 2012.
51. Zhou S, Chen Q, Wang X. Active Semi-Supervised Learning Method with Hybrid Deep Belief Networks. PLoS ONE 2014; 9(9)
52. Borazjani P, Everett C, McCoy D, OCTANE: An Extensible Open Source Car Security Testbed, Proceedings of the Embedded Security in Cars Conference 2014.

CHAPTER 8

Event Recognition Based on Deep Learning in Chinese Texts

Yajun Zhang, Zongtian Liu, Wen Zhou

National Natural Science Foundation of China (NSFC), Shanghai University, School of Computer Engineering and Science, Shanghai, China

ABSTRACT

Event recognition is the most fundamental and critical task in event-based natural language processing systems. Existing event recognition methods based on rules and shallow neural networks have certain limitations. For example, extracting features using methods based on rules is difficult; methods based on shallow neural networks converge too quickly to a local minimum, resulting in low recognition precision. To address these problems, we propose the Chinese emergency event recognition model based on deep learning (CEERM). Firstly, we use a word segmentation system to segment sentences. According to event elements labeled in the CEC 2.0 corpus, we classify words into five categories: trigger words,

participants, objects, time and location. Each word is vectorized according to the following six feature layers: part of speech, dependency grammar, length, location, distance between trigger word and core word and trigger word frequency. We obtain deep semantic features of words by training a feature vector set using a deep belief network (DBN), then analyze those features in order to identify trigger words by means of a back propagation neural network. Extensive testing shows that the CEERM achieves excellent recognition performance, with a maximum F-measure value of 85.17%. Moreover, we propose the dynamic-supervised DBN, which adds supervised fine-tuning to a restricted Boltzmann machine layer by monitoring its training performance. Test analysis reveals that the new DBN improves recognition performance and effectively controls the training time. Although the F-measure increases to 88.11%, the training time increases by only 25.35%.

INTRODUCTION

Natural-language organized texts express higher-level semantic information through events. Recognizing those events can help computers easily understand the exact meaning of texts and lay a solid foundation for realizing event-based natural language processing (NLP) systems [1, 2]. We use trigger words to label events in texts. Typically, a trigger word is a major indicator of an event occurring and contains the maximum amount of information in a sentence. For example, in the sentence "Wenchuan suffered an earthquake on July 20, 2008.", "earthquake" is a trigger word and therefore this event can be classified as an "earthquake" event. Its location element is "Wenchuan". Differing from other elements that may be absent in an event, the trigger word is absolutely necessary for any event. Therefore, the essence of event recognition is trigger word recognition.

Current, widely-used trigger-word recognition methods are based on rules or shallow neural networks. However, these methods have certain limitations. For example, for methods based on rules, extracting features is difficult. They also have a poor portability. Methods based on shallow neural networks can not realize complex function approximations and suffer from the vanishing gradient problem, resulting in low recognition precision. In recent years, deep learning (DL) within the machine

learning field [3] has shown that it can be successfully applied to reduce the data dimension by layer-wise training, abstract deep features of data and use those features to considerably improve classification precision. DL mimics the human brain's information processing mechanism [4–6] in an unsupervised manner to learn a deep nonlinear network structure, achieves complex function approximations, obtains the feature functions of high-dimensional data and ultimately improves the accuracy of classification or prediction. DL has achieved breakthroughs in image identification, audio processing and video recognition. There are also preliminary applications of DL in many NLP fields such as syntactic analysis, entity relation extraction and emotional analysis. Glorot et al. [7] analyzed current recommendations and reviews on the internet, the number of which is exponentially increasing. As labeling these data is difficult, the researchers proposed a novel unsupervised study model based on DL which learns how to extract meaningful information from recommendation and review data on the internet. Xi et al. [8] proposed a Chinese anaphora resolution model based on deep belief networks (DBN). Although the recognition ability of the proposed method is not better than that of traditional support vector machine (SVM) anaphora resolution algorithms, the study provides a meaningful application of DL to Chinese NLP. Comparing to DL, Weng proposed a Finite Automaton in Developmental Network (FA-in-DN) [9], which incrementally learns the FA but takes sensory images directly and produces motor images directly.

Although DL can be applied to these fields on a preliminary basis [10, 11], there are few studies which show that the method can successfully extract deep semantic information in the field of event recognition. In this study, we propose a Chinese emergency event recognition model (CEERM) based on deep learning. We use a DL mechanism to mine deep semantic features automatically and analyze the role that these features play in event recognition. Using a feature analysis method, we abstract six feature layers of words in texts and then vectorize every word to a feature vector set. Finally, we analyze the feature vector sets using the CEERM to obtain recognition results. Test analysis shows that recognition performance in our model is better than that of existing methods.

Our model has some attractive features:

- Based on a deep learning framework, it can approximate complex functions to obtain word semantics features effectively and contribute to classification.
- Two types of classifiers are used in our model: unsupervised and dynamical supervised. One can alternate the classifier based on the application, which enhances the scalability and flexibility of the model.
- It can overcome some of the shortcomings of exiting event recognition methods that are based on rules or shallow neural networks and can therefore improve recognition performance.
- The model conducts a meaningful exploration for the application of DL to the NLP field and provides a foundation for its wider application in NLP.

Related work

As a popular research field in NLP, event recognition technology has been widely applied to automatic summarization [12–15], automatic question answering [16–19] and information retrieval [20–23]. Existing event recognition methods are primarily divided into two types: those based on rules and those based on shallow neural networks. Methods based on rules commonly use predetermined rules to design templates for event recognition. For example, Yankova et al. [24] proposed a football event extraction system, and Lee et al. [25]developed a meteorological event extraction system based on ontology. Jiang [26]proposed a portable event pattern acquisition method which manually defines events, extracts tasks and automatically obtains a new event pattern from a raw corpus by building the mapping between event targets and corresponding event roles. Chen et al. [27] proposed an information extraction model based on event frames and built a catastrophic event frame which uses inheritance and inductive characteristics of the frames to simplify system implementation processes and summarize event information. Feng [28] proposed an emergency event information extraction method base on frames. It uses a predetermined event frame to extract news elements and examines new aspect information that was not pre-established in the frame as a complement to that same frame.

Methods based on shallow neural networks focus on constructing a classifier and the discovery, combination and selection of features. These methods consider event recognition as a classification problem and thus selecting appropriate features is critical for event classification. In 2002, Chieu and Ng introduced a maximum entropy classifier [29] and applied it for the purpose of recognizing events and their elements. For addressing the problem that existing event recognition methods did not consider context, Fu et al. proposed an event extraction method based on weight features of event elements in the event ontology [30]. In 2006, Ahn [31] proposed a method to recognize event classes and elements. In Ahn's method, the most important step is identifying the most suitable trigger word to describe an event, then, use a classifier to recognize the class of this event, and lastly use the event class information to construct classifiers for each event element class to recognize them. Wang et al. [32] proposed a verb-driven method to extract event semantic information (5W1H) from Chinese texts. They proved its reliability and feasibility through extensive experimental data. Yankova et al. [24] proposed an event extraction method that integrates machine learning and semantic role labeling based on the PropBank corpus. This method achieved satisfactory results when applied to English language texts. McCracken et al. [33] extracted multiple elements using a syntax tree pattern matching method. Kim et al. [34] used a method that combines lexical semantics and semantic roles to recognize and classify events automatically.

Although current event recognition methods are efficient, certain limitations exist. Methods based on rules lack portability and robustness, have difficulties extracting features and need update rules to maintain optimal performance for new field texts. This requires many scientists with specialized knowledge, experienced linguists and time. Methods based on shallow neural networks search for optimal solutions by using a gradient descent algorithm. However, the nonlinear mapping between input and output of the algorithm converts the network error function space into a nonlinear one consisting of multiple minimal points. The gradient search direction is the only direction that reduces network errors; therefore, it always converges to a local minimum, with worsening performance as the number of network layers increases [35], critically affecting recognition performance. Considering the aforementioned

problems, we propose the CEERM, which uses a DL model to obtain deep semantic features of event trigger words without manually designed rules and overcomes the local minimum problem. The model not only completes event recognition tasks, but also helps to recognize other event elements.

MATERIALS AND METHODS

Study Model

Restricted Boltzmann machine(RBM).

RBM is a generative stochastic neural network proposed by Hinton and Sejnowsk [36] and is illustrated in Fig 1. This network consists of some visible units (mapping to visible variables that are data samples) and some hidden units (mapping to hidden variables). Visible or hidden variables are both binary variables whose state belongs to 0,1. The entire network is a bipartite graph and an edge connection exists only between visible and hidden units. This means that no edge connection exists within visible or hidden units. This represents the major difference between RBM and a general Boltzmann machine.

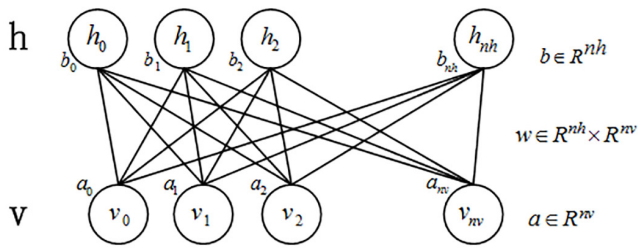


Figure 1: RBM structure.

The energy function of the network structure is defined by Eq (1).

$$E_\theta(v,h) = -\sum_{}^{n_v} a_i v_i - \sum_{}^{n_h} b_j h_j - \sum_{}^{n_v}\sum_{}^{n_h} h_j w_{j,i} v_i \quad (1)$$

Where h and v represent hidden and visible layers, w represents the weight value between visible and hidden layers, and a and b represent their offset values, respectively. According to the known node value of a visible layer, we can obtain that of a hidden layer using the joint probability distribution equation [36] given by

$$P(h_j = 1|v) = \frac{1}{1 + exp(-b_j - \sum_{i=1}^{n_v} w_{j,i} v_i)} \quad (2)$$

According to [36], RBM is a symmetric network. Therefore, we can obtain the node value of a visible layer from a known node value of a hidden layer using the equation as follows.

$$P(v_i = 1|h) = \frac{1}{1 + exp(-a_i - \sum_{j=1}^{n_h} w_{j,i} h_j)} \quad (3)$$

The goal of training the RBM is to obtain the parameters $\theta(w, a, b)$, which maximize the joint probability p(v,h). The usual approach used to solve this problem is the Markov chain Monte Carlo(MCMC) algorithm. This approach constantly updates the node values of the hidden and visual layers until the Markov chain approach tends to be stable; thus, the joint probability p (v,h) reaches the maximum value [37]. The derivatives of the maximum and initial joint probability distributions are then obtained. Finally, the weight value θ is updated using the following equation, where τ is the iteration number and η is the learning speed.

$$\theta^{(\tau+1)} = \theta^{(\tau)} + \eta \frac{\partial \log P(v,h)}{\partial \theta} \quad (4)$$

Input vector v^0 of the RBM is the vector of the visible layer when t = 0. Based on Eq (2), vector h^0 of the hidden layer is obtained by v^0 and v^1 is the vector of the visible layer when time t = 1, obtained through h^0 using Eq (3). In addition, v^∞ and h^∞ are the vectors of the visual and hidden layers, respectively, when time $t = \infty$. The slope of the joint probability distribution can be obtained by using the following equation.

$$\eta \frac{\partial \log P(v,h)}{\partial \theta} = < h_j^0 (v_i^0 - v_j^1) > + < v_i^1 (h_j^0 - h_j^0) > + \cdots = < h_j^0 v_i^0 > - < h_j^\infty v_i^\infty > \quad (5)$$

In Eq (5), h^0v^0 is the average value of point multiplication of the input feature vector and its corresponding feature vector in the hidden layer and $h^\infty v^\infty$ is the average value of the feature vector of the visible layer at the end of the Markov chain and its corresponding feature vector in the hidden layer. The Markov chain converges at $h^\infty v^\infty$. From Eq (5), the slope of the joint probability distribution is unrelated to the intermediate state and is concerned only with the initial and final states of the RMB network. According to Eq (4), the updated parameter θ can be obtained in order to implement self-training. In a traditional Markov chain approach, when solving for the maximum joint probability $P(h^\infty v^\infty)$ and initial joint distribution probability p(v,h), guaranteeing the convergence rate is difficult. Moreover, the step ∞ is also difficult to calculate. Therefore, Hinton [38] proposed using contrastive divergence (CD) guidelines, which can speed up calculation without compromising accuracy. The Kullback-Leibler(KL) distance is used to measure the "difference" of two probability distributions, expressed as $KL(P||P')$, shown in the following equation.

$$CD_n = KL(P_0||P_\infty) - KL(P_n||P_\infty) \quad (6)$$

CD_n can be understood as a position measurement of p_n between p_0 and p_∞. Constantly assigning p_n to p_0, we can obtain new values of p_0 and p_n [39]. Experiments demonstrate that CD_n tends to 0 after r times of calculating the slope correction parameter, and the accuracy using the method is similar to that of the Markov chain approach. Therefore, this study uses the RBM network training method based on CD guidelines [40].

Deep belief network based on dynamic supervised learning.

In the existing DBN network, the training way of each RBM layer is unsupervised; however, it is finally reverse supervised fine-tuned using the back propagation network (BP). The structure can improve training speed, but no supervision training exists between RBM layers. This can

cause errors bottom-up propagation in RBM layers, which ultimately affects recognition performance. Therefore, we propose DBN based on dynamic supervision. In this network, RBM training results are estimated. Determining the use of BP training with supervision depends on the results, as shown in Fig 2. The average error and its standard deviation are used as evaluation criteria. If one of these is greater than the given threshold value, then the BP algorithm is used to fine-tune the RBM network parameters in order to perform training with supervision. Therefore, our DBN can reduce the risk of error propagating in RBM layers and thus improve recognition performance. Our method does not require supervised training in all RBM layers. Therefore, it limits DBN network training time and improves recognition performance.

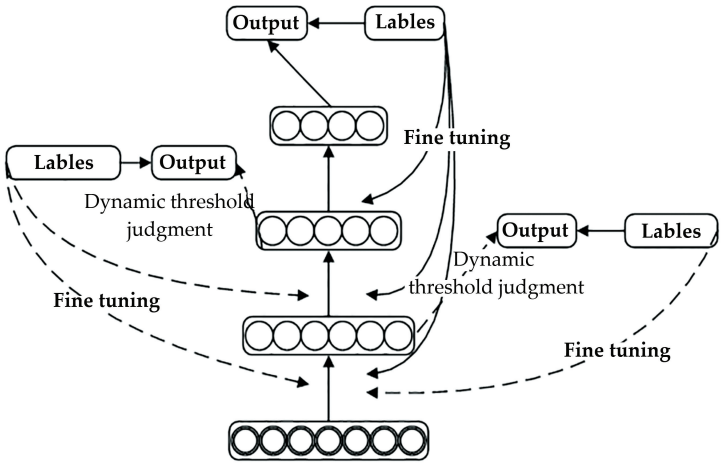


Figure 2: Structure of deep belief network based on dynamic supervision.

Two evaluation parameters of DBN are used based on dynamic supervision: the average error of one time RBM training, as seen in Eq (7), and the unbiased standard deviation of errors of one time RBM training, as shown in Eq (8). We define the average value of total error and the average value of total error standard deviation as thresholds by training unsupervised DBN several times.

$$\bar{e} = \frac{e_1 + e_2 + e_3 + \cdots + e_n}{n} = \frac{\sum_{i=1}^{n} e_i}{n} \tag{7}$$

$$S = \sqrt{\frac{\sum_{i=1}^{n} (e_i - \bar{e})^2}{n}} \quad (8)$$

Chinese emergency event recognition model based on deep learning(CEERM)

CEERM has four main modules: corpus selection, a pre-processing system, feature vector generation and a deep classifier, as shown in Fig 3. The main function of the pre-processing module is to segment sentences in the corpus and then classify every word according to its labeled class. The feature vector generation module generates feature vectors for words based on six feature layers: part of speech layers (POSL), dependency grammar layers (DPL), length layers (LENL), location layers (LOCL), distance between trigger word and core word layers (DISL), and trigger words frequency layers (TFWL). Feature vectors are binary patterns. The deep classifier generates a stable DBN network by training the corpus and recognizing trigger words in the test corpus. We use two types of classifiers: without supervision and with dynamic supervision.

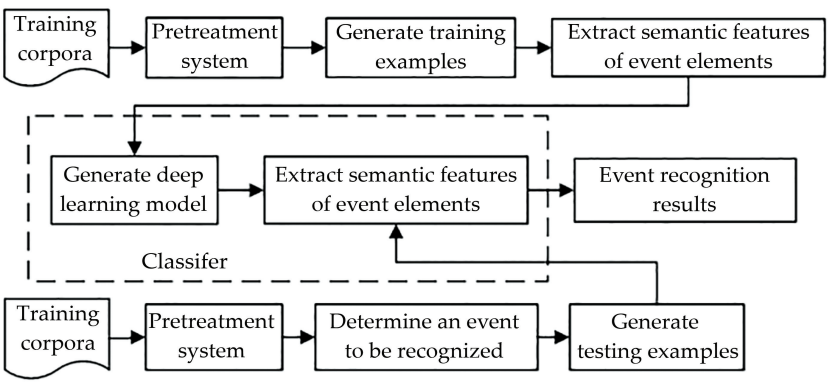


Figure 3: CEERM model frame.

Corpus selection.

Chinese Corpus CEC 2.0 (https://github.com/shijiebei2009/CEC-Corpus) is an event ontology corpus developed by the Semantic Intelligence Laboratory of Shanghai University. It has collected 333 newspaper texts

about earthquakes, fires, traffic accidents, terrorist attacks and food poisoning. Descriptive statistics for these texts are shown in Table 1. We labeled event trigger words, participants, objects, time, location and other elements using a semi-automatic method. The corpus is divided into two parts: training and test. The training corpus contains 271 newspaper articles of five kinds obtained from CEC 2.0 and includes 51 articles about earthquakes, 61 about fires, 70 about traffic accidents, 50 about food poisoning, 39 about terrorist attacks, with a total of 4690 trigger words. The test corpus contains 1301 trigger words in 62 corpora of all five article types including 12 articles about earthquakes, 14 about fires, 15 about traffic accidents, 11 about food poisoning and 10 about terrorist attacks.

Table 1: Event Statistics in CEC 2.0

Corpus Class	Section	Sentence	Event Sentence	Event	Ratio of Event Sentences(%)
Earthquake	63	496	463	1053	93.3
Fire	75	512	473	1216	92.4
Traffic accident	85	561	523	1790	93.2
Terrorist attacks	49	387	362	823	93.5
Food poisoning	61	439	409	1109	93.2
Total	333	2395	2230	5991	93.1

doi:10.1371/journal.pone.0160147.t001

LTP segmentation system.

The language technology platform (LTP) is a Chinese NLP system built by the Social Computing and Information Retrieval Research Center at the Harbin Institute of Technology. LTP has developed five Chinese NLP modules, including word segmentation, named entity recognition, morphology, sentence structure and semantics [41]. LTP has been widely recognized and praised, and the academic version of LTP has been shared with more than 500 research institutions free of charge.

Pre-processing system.

The main function of the pre-processing system is to process the XML format corpus files, obtain parts of speech (POS), a dependency grammar (DP) of words and finally to save the files in XML format. First, LTP segments the sentences into words. We then compare the words in this sentence with previously labeled trigger words. If the words are labeled trigger words, we classify them as trigger words. We classify other elements such as participants, objects, time and location using

the same method. Pre-processing results form the foundation of feature representation and vector generation, detailed in Fig 4.

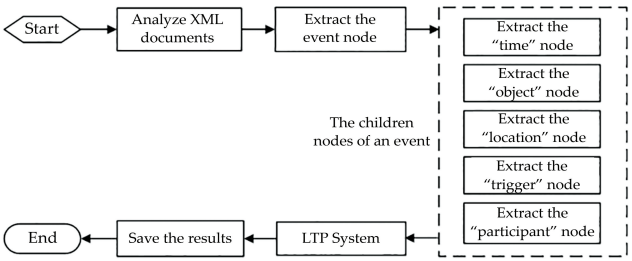


Figure 4: Flow of pre-processing system.

Deep classifier.

The deep classifier uses word feature vectors as inputs and treats word classifications as training and test comparison criteria. The manner of training is such that it involves the training of the small batch pattern. Two kinds of classifiers are employed. One is an unsupervised DBN which consists of a multiple-layer unsupervised RBM and a single-layer supervised BP network. The other is based on dynamic supervised DBN and determines whether a supervised adjustment RBM network is required by analyzing the RBM training results. It thus controls training time and improves recognition performance.

Feature abstracting.

Traditional event recognition methods based on features primarily use single learning mechanisms during the learning process. They do not consider the impact of feature layers on system learning; they only consider the role of the feature value. In fact, the human cognitive process has an abstract hierarchical nature. Recognizing unknown objects progresses from the abstract to the concrete. The higher the abstract level, the higher the recognition accuracy. Therefore, layering feature types and allowing them to perform their role adequately during the learning process is necessary.

According to the semantic features of trigger words in texts, we define six feature layers in order to transform the trigger word recognition into a feature classification problem. As DL can mine text features more

effectively than traditional methods, it can recognize events with greater accuracy.

POSL: Part of speech is the most basic grammatical feature of a word. In a sentence, grammar imposes great restrictions on POS. For instance, nouns or pronouns can act as a subject, but using an adverb as the subject is absolutely impermissible. Therefore, taking POS as a feature layer conforms with the expression characteristics of text semantics information and the basic cognition when people identify events [42]. In CEC 2.0, trigger words have a centralized distribution, 80% of trigger words are verbs and 14% are nouns. Therefore, using POSL as a trigger word's abstract feature layer can improve recognition accuracy.

DPL: According to dependency grammar, the predicate verb of a sentence is considered the center that dominates other sentence components. However, a predicate verb itself is not dominated by any other components and all dominated components subordinate to their dominator in one dependency relationship [43]. As dependency grammar exactly describes semantic role relationships between words, it has extremely high semantic performance. Containing the most information and having the clearest expression in a sentence, trigger words, in a certain sense, play the role as predicate verbs of dependency grammar. According to dependency grammar analysis of trigger words in CEC 2.0, 62% of their role in dependency grammar is head and 18% is verb-object. In most cases, the predicate verbs of dependency grammar accord with trigger words.

LENL: In any sentence, a close connection exists between a word's length (LEN) and its role [44]. For instance, the subject and object usually are longer than the predicate because the predicate more often expresses action. In addition, words of this kind are usually more concise and provide clearer expression. Length analysis of trigger words in CEC 2.0 shows that the length of 81% of trigger words is 2 and of 12% of words is 1. Due to its centralized distribution, this feature is crucial in recognizing trigger words effectively.

LOCL: The word index in a sentence denotes a word's position, which affects its semantic information. Word position in a sentence is a kind of specification that takes form gradually during the writing process. Every language places words having different roles in different positions based on the language's unique grammatical logic. Therefore, a word

index has a positive meaning for determining the role a word plays in a sentence [45]. Considering that sentence lengths are different, we use only 11 location indices (0 to 10) to annotate the abstract layer. Location (LOC) analysis in CEC 2.0 indicates that 16% of trigger words are in the first position of sentences and 12% are in the second position. The location feature contributes minimally to recognition accuracy because of its low distribution concentration degree.

DISL: Core words are the main components of sentences, which means they usually express the core content of the sentence. Trigger words express the most important content of a sentence [46]. Therefore, trigger words and core words have a high degree of similarity. The shorter the distance (DIS) between a regular word and a core word, the greater is the possibility that the former is a trigger word. In this study, a core word refers to a word whose dependency grammar role is head. Analyzing CEC 2.0, we find that the distance between trigger and core words in 12% of the cases is 0 and in 27% the distance is 1.

TWFL: Many trigger words reappear in CEC 2.0. Some trigger words appear more often than others. Trigger word frequency (TWF) is divided into eight grades: hp, a, b, c, d, e, f, and g, definitions of which are as given in Table 2.

Table 2: Grade rules of trigger word frequency

Identifier	Feature abstract layer	Grade	Define
101	TWFL	hp	Fr $\geq$ 1
102	TWFL	a	Fr> 1 and Fr$\leq$ 5
103	TWFL	b	Fr> 5 and Fr$\leq$ 10
104	TWFL	c	Fr> 10 and Fr$\leq$ 20
105	TWFL	d	Fr> 20 and Fr$\leq$ 30
106	TWFL	e	Fr> 30 and Fr$\leq$ 40
107	TWFL	f	Fr> 40 and Fr$\leq$ 50
108	TWFL	g	Fr> 50

Note: Fr denotes Frequency.

doi:10.1371/journal.pone.0160147.t002

Feature representation and samples generation.

Feature representation mainly uses a binary representation model. Feature vector generation of TWFL is accumulated and five other vector generations of abstract layers follow the same pattern. We describe all feature representations in detail below.

Feature representation of POSL: Twenty-nine dimensions of POS feature vectors represent 29 types of POS. If the POS of a candidate word corresponds with that of a certain dimension representation of a feature vector, then the feature vector value of the word at the dimension is 1, whereas that at the remaining 28 dimensions are 0.

Feature representation of DPL: On this feature layer, 14 vector dimensions exist, thus representing 14 kinds of DP.

Feature representation of LENL: On this layer, the vector dimension is 10, meaning the candidate word length ranges from 1 to10. If a word length exceeds 10, then every feature vector value of the candidate word is 0.

Feature representation of LOCL: The vector dimension is 11, denoting 11 index positions from 0 to 10. If an index position exceeds 10, then every feature vector value of the candidate word is 0.

Feature representation of DISL: The vector dimension is 11, representing DIS from 0 to 10. If it is greater than 10, then each feature vector value of the candidate word is 0.

Feature representation of TWFL: The dimension is 8, representing 8 grades of a trigger word. This layer's vector representation uses the accumulation approach. For example, the grade of the candidate word "earthquakes" is e. Therefore, the feature vector value is "1 1 1 1 1 1 0 0".

Based on the aforementioned feature representations, we give the following example: "The police arrested the suspect on the spot last night.". In this sentence, Types 1, 2, 3, and 4 represents the participant, time, trigger word and location, respectively. The results are shown in Table 3.

Table 3: Example of feature representation

Candidate word	Feature abstraction layer	Feature vector value	Class
last night	POSL	00000000000000000100000000000	2
	DPL+LENL	00000010000000 0100000000	
	LOCL+DISL+TWFL	10000000000 00010000000 00000000	
police	POSL	00000000000100000000000000000	1
	DPL+LENL	10000000000000 0100000000	
	LOCL+DISL+TWFL	01000000000 00100000000 00000000	
on the spot	POSL	00010000000000000000000000000	4
	DPL+LENL	00000010000000 0100000000	
	LOCL+DISL+TWFL	00100000000 01000000000 00000000	
arrested	POSL	00000000000000000000000010000	3
	DPL+LENL	00000000000001 0100000000	
	LOCL+DISL+TWFL	00010000000 10000000000 11000000	
suspect	POSL	00000000000100000000000000000	1
	DPL+LENL	01000000000000 0010000000	
	LOCL+DISL+TWFL	00001000000 01000000000 00000000	

doi:10.1371/journal.pone.0160147.t003

RESULTS

We used the CEC 2.0 Chinese corpus for experimental data and then extracted semantic information required for event recognition according to the feature representations described in the Feature abstracting section. Then, based on the process described in the Feature representation and samples generation section, training and test samples are generated. We rapidly trained the training samples using CEERM to generate a stable deep classifier, which could recognize the events in test samples. The values of deep-classifier-related parameters numepochs, batchsize, momentum, and alpha are set to 1, 100, 0, and 1, respectively. In our experiments, we used precision (P), recall rate (R), and F-measure to evaluate recognition performance. In addition, we used F incremental (FI), F incremental rate (FIR), F comparison incremental (FCI), and F comparison incremental rate (FCIR) to contrast the recognition performance of the two classifiers.

Threshold analysis of dynamic supervised classifier

The dynamic supervised classifier monitors the average and standard deviation of training errors in a single RBM training period. If either exceeds a threshold, the classifier uses the BP network to fine-tune the RBM layer to achieve the following: ensure error is controlled within a specific range, reduce the probability of error being propagated layer-wise and improve recognition performance. However, the choice of the threshold impacts recognition performance and training time. If the threshold is too low, fine-tuning during each RBM training period is possible, which greatly improves model training time. If the threshold is

too high, it may reduce the possibility of eliminating those errors in the lower layer in time and may thus propagate to the upper layer, resulting in low recognition performance. In this study, we use the average and standard deviation of errors as the threshold values after several rounds training the unsupervised DBN. In Table 4, we increase the amount of test data to obtain test results. In these tests, the number of RBM layers is four, all feature layers are added and RBM1 represents the error in the first RBM layer training. Analysis of the parameters in Table 4enables us to set the dynamic classifier threshold to an average of 4 and the standard deviation to 2.

Table 4: Threshold analysis of dynamic supervised classifier

Model+Abstract Layers	Number of vectors	RBM1(%)	RBM2(%)	RBM3(%)	RBM4(%)	Average(%)	Standard deviation(%)
DBN4+ POSL+DPL+ LENL+LOCL+ DISL+TWFL	2000	7.25	4.17	2.61	2.52	4.14	2.21
	4000	7.18	4.09	2.52	2.49	4.07	2.2
	6000	7.13	4.09	2.51	2.48	4.05	2.19
	8000	7.12	4.11	2.52	2.45	4.05	2.19
	10000	7.08	4.01	2.46	2.42	3.99	2.19
	12000	7.01	3.99	2.45	2.46	3.98	2.15
	14000	6.98	3.97	2.42	2.41	3.95	2.15
	16000	6.97	3.93	2.38	2.41	3.92	2.16
	18000	6.9	3.89	2.35	2.36	3.88	2.14
	20000	6.86	3.91	2.38	2.32	3.87	2.13
Average value	11000	7.048	4.016	2.46	2.432	3.99	2.17

doi:10.1371/journal.pone.0160147.t004

CEERM recognition performance of unsupervised classifier

We combine the features of POS, DP, LEN, LOC, DIS and TWF to generate training and testing samples, then use DBNi to complete two sets of experiments. Thereafter, we compare them. I represents the number of RBM layers contained in DBN with the values set to 1,2,3,4,5,6,7 and 8, respectively. In the first experiment, we test recognition performance with the same feature layers while the number of RBM layer increases. In the second experiment, we test the performance when introducing different semantic feature layers with the same RBM layers.

Table 5 shows the results of the first experiment. An increase in the number of RBM layers can improve recognition performance when the number is below five. The values of DBN4 performance indices R, P and F are 87.32%, 83.12% and 85.17%, which are highest in all results. Compared to the result of DBN1, FI and FIR increase by 5.73% and

7.21%. As the model can extract the main feature information from data using multiple-layer mapping units, its effect is greater than that of a single-layer structure. However, when the number of RBM layers is greater than four, its recognition performance degrades with each increase in the number of RBM layers. The values of DBN8 result indices R, P and F are only 82.59%, 80.12% and 81.34%. This means that the recognition performance is not directly proportional to the increase in the number of RBM layers. Therefore, in an actual application, we should choose a suitable depth of RBM based on extensive testing and analysis. In addition, system training time increases as the number of RBM layers increases, because the number of neural network training nodes increases and therefore the workload of combining and training increases dramatically. This establishes a higher demand for the processing ability of the system platform.

Table 5: Unsupervised CEERM recognition performance based on different numbers of RBM layers

Model+Abstract Layers	R	P	F	FI	FIR(%)
DBN1+POSL+DPL+LENL+LOCL+DISL+TWFL	80.4	78.5	79.44	0	0
DBN2+POSL+DPL+LENL+LOCL+DISL+TWFL	83.2	78.7	80.89	1.45	1.82
DBN3+POSL+DPL+LENL+LOCL+DISL+TWFL	84.7	82.8	83.74	2.85	3.53
DBN4+POSL+DPL+LENL+LOCL+DISL+TWFL	87.32	83.12	85.17	1.43	1.71
DBN5+POSL+DPL+LENL+LOCL+DISL+TWFL	86.13	83.21	84.64	−0.52	−0.61
DBN6+POSL+DPL+LENL+LOCL+DISL+TWFL	85.21	81.21	83.16	−1.48	−1.75
DBN7+POSL+DPL+LENL+LOCL+DISL+TWFL	83.13	81.32	82.22	−0.95	−1.14
DBN8+POSL+DPL+LENL+LOCL+DISL+TWFL	82.59	80.12	81.34	−0.88	−1.07

Note: F_n is the F measure when the number of RBM layers is n, FI is difference between F_n and F_{n-1}, FIR is equal to $\frac{F_n - F_{n-1}}{F_{n-1}}$.

doi:10.1371/journal.pone.0160147.t005

The second experiment tests the recognition performance of the CREEM model when introducing different feature layers. The results, shown in Table 6, reveal that introducing different feature layers can considerably improve overall performance. With an increase in the number of feature layers, all three performance indices improve. However, different feature layers contribute differently to the improvement in recognition performance. From Table 6, we see that DPL contributes most to performance, with FI and FIR increasing by 5.62% and 7.83%, because the DP can precisely describe semantic role relationships. Using relationship information can therefore add the performance when recognizing events. LENL also has a big contribution to the performance improvement, with FI and FIR increase by 3.16% and 4.08%. The reason for this improvement is that many verbs and verb noun phrases are

used as trigger words. These phrases are mainly composed of two or three single words so the model will easily identify such trigger words through length. LOCL contributes minimally, FI and FIR only increase by 0.52% and 0.65%, because event sentences don't have a consistent length. Furthermore, even if two event sentences have a approximate length, the location of their trigger words may not be consistent leading to a smaller overall contribution. DISL contributes little to recognition performance, with FI and FIR increase by 2.01% and 2.48%, because the written style is irregular for Chinese and the distance between a core word and a trigger word becomes more uncertain. If we use a corpus of another language with a clear writing style as our experimental data, the DISL may contribute more. Due to the test corpus and training corpus being disjoint, the TWFL only retains the frequency of trigger words in the training corpus. This reduces the contribution to the performance of the model.

Table 6: Unsupervised CEERM recognition performance based on different feature layers

Model+Abstract Layers	R	P	F	FI	FIR(%)
DBN4+POSL	78.13	66.32	71.74	0	0
DBN4+POSL+DPL	92.42	66.52	77.36	5.62	7.83
DBN4+POSL+DPL+LENL	83.87	77.42	80.52	3.16	4.08
DBN4+POSL+DPL+LENL+LOCL	83.87	78.39	81.04	0.52	0.65
DBN4+POSL+DPL+LENL+LOCL+DISL	80.01	86.32	83.05	2.01	2.48
DBN4+POSL+DPL+LENL+LOCL+DISL+TWFL	87.32	83.12	85.17	2.12	2.56

Note: F_c is the F measure when the number of feature layers is c, FI is difference between F_c and F_{c-1}, FIR is equal to $\frac{F_c - F_{c-1}}{F_{c-1}}$.

doi:10.1371/journal.pone.0160147.t006

CEERM recognition effect comparison of two classifiers

To compare our two classifiers, we test recognition performance as the number of DBN and feature layers increase. We also compare and analyze the stability of the classifiers with respect to recognition performance as the number of DBN layers increases to a limited degree. We then compare their training time.

A comparison of the unsupervised and dynamic supervised classifiers is shown in Table 7. The average increment of the three indices and the total absolute increment of the dynamic supervised classifier are greater than that of the unsupervised classifier when the numbers of RBM layers increases. The highest FCI and FCIR are 5.56% and 6.43%. As the

structure of dynamic supervised DBN can hinder the errors propagating layer by layer, recognition performance is increasing. At the same time, we also show that the FI and FIR of the two DBN decline when the numbers of RBM layers is more than four, but that the rate of decline is small for dynamic supervised DBN and therefore has a higher stability. We can conclude that adding dynamic supervised learning can improve recognition performance to a certain extent.

Table 7: Comparison of recognition effect between dynamic supervised and unsupervised CEERM based on different numbers of RBM layers

Model+Abstract Layers	R	P	F	FI	FIR(%)	FCI	FCIR(%)
DBN1+POSL+DPL+LENL+LOCL+DISL+TWFL	80.4	78.5	79.44	0	0	0	0
DBN2+POSL+DPL+LENL+LOCL+DISL+TWFL	89.5	83.6	86.45	7.01	8.83	5.56	6.43
DBN3+POSL+DPL+LENL+LOCL+DISL+TWFL	89.6	84.7	87.08	0.63	0.73	3.34	3.84
DBN4+POSL+DPL+LENL+LOCL+DISL+TWFL	90.32	86	88.11	1.03	1.18	2.94	3.34
DBN5+POSL+DPL+LENL+LOCL+DISL+TWFL	90.3	85	87.57	−0.54	−0.61	2.93	3.35
DBN6+POSL+DPL+LENL+LOCL+DISL+TWFL	89.7	84.8	87.18	−0.39	−0.44	4.02	4.61
DBN7+POSL+DPL+LENL+LOCL+DISL+TWFL	89.1	84.2	86.58	−0.6	−0.69	4.36	5.04
DBN8+POSL+DPL+LENL+LOCL+DISL+TWFL	90.1	82.4	86.08	−0.5	−0.58	4.74	5.51

Note: F_d is the F measure of the dynamic supervised classifier, F_u is the F measure of the unsupervised classifier, FCI is difference between F_d and F_u, FCIR is equal to $\frac{F_d - F_u}{F_u}$.

doi:10.1371/journal.pone.0160147.t007

Table 8 shows a comparison of the unsupervised and dynamic supervised classifiers. The experimental results show that an increasing number of feature layers can considerably improve overall performance for the two classifiers. DPL contributes the most whereas LOCL contributes minimally. Each layer's contribution is the same as that of the unsupervised classifier. The effects of feature layers in event recognition are very similar for the two classifiers. However, the dynamic supervised classifier has better performance than unsupervised, the highest FCI and FCIR are 4.89% and 6.83%, the minimal are 1.7% and 2.0%, and the average are 2.67% and 3.26%. In conclusion, the recognition performance of the dynamic classifier is better than that of the unsupervised classifier, which means the way of supervision is very important for the performance of deep learning, and we should treat it as a key factor when using deep learning in NLP applications.

Table 8: Comparison of recognition effect between dynamic supervised and unsupervised CEERM based on different feature layers

Model+Abstract Layers	R	P	F	FI	FIR(%)	FCI	FCIR(%)
DBN4+POSL	75.11	78.21	76.63	0	0	4.89	6.38
DBN4+POSL+DPL	79.13	80.35	79.74	3.11	4.05	2.38	2.98
DBN4+POSL+DPL+LENL	83.21	80.81	81.99	2.26	2.83	1.47	1.79
DBN4+POSL+DPL+LENL+LOCL	87.32	78.61	82.74	0.74	0.91	1.7	2.05
DBN4+POSL+DPL+LENL+LOCL+DISL	91.76	80.28	85.64	2.9	3.51	2.59	3.02
DBN4+POSL+DPL+LENL+LOCL+DISL+TWFL	90.32	86	88.11	2.47	2.88	2.94	3.34

Note: See the note to Table 7.

doi:10.1371/journal.pone.0160147.t008

Fig 5 illustrates our comparisons and analysis of the stability of the classifiers regarding recognition performance. We analyzed their stability by increasing the number of RBM layers in the same number of feature layers. Fig 5 shows that with an increase in RBM layers, errors at the bottom layer propagate upward to higher layers and the recognition performance of the unsupervised classifier declines gradually, the difference between the maximum and minimum of the three indexes R, P, F are 28.11%, 29.25% and 28.75%. In addition, the classifier has a large amplitude and poor stability. By contrast, the dynamic supervised classifier has high stability, the difference between the maximum and minimum of the three indexes R, P, F are 21.97%, 21.83% and 21.91%, which are all lower than the values of the unsupervised classifier.

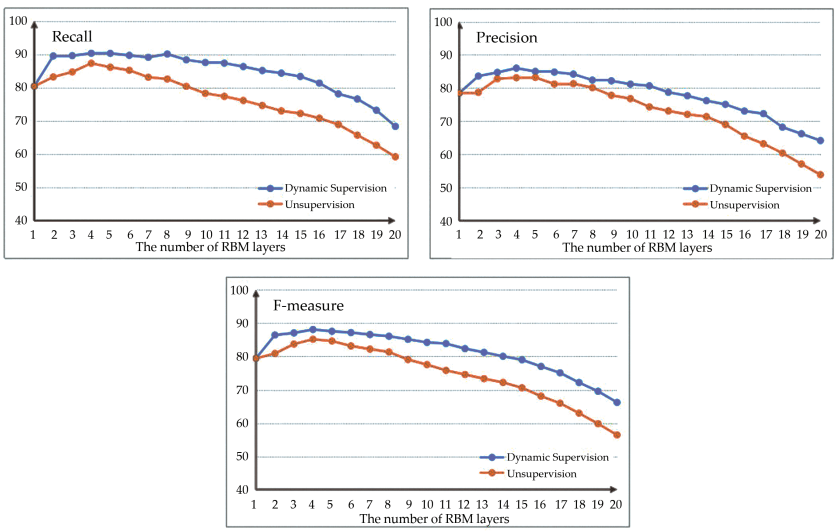


Figure 5: Comparison of the stability of the two classifiers.

Table 9 shows the training time of the two deep classifiers. The table shows that the training time of the dynamic supervised classifier is longer because adding fine-tuning increases the training time. However, the increase is less than 30%. Considering the improved performance, we can allow such a limited increase in time. The classifier without supervision has a shorter training time. We chose the classifier's way of supervision according to specific situations and platform performance in actual applications. If training efficiency is emphasized, the classifier without supervision is the better of the two classifiers. Dynamic supervision is chosen because of its better recognition performance.

Table 9: Comparison of the training time of the two classifiers

Test condition	Training time of one period	U(ms)	D(ms)	Increment Rate(%)
All feature layer are the same Increasing the RBM layers from 1 to 8	Average time	1290	1595	23.64
	Total time	9761	12235	25.35
RBM layer is fixed Increasing the feature layers from 1 to 6	Average time	1089	1317	20.94
	Total time	6135	7574	23.46

Note: U refers to the unsupervised classifier and D refers to the dynamic supervised classifier.

doi:10.1371/journal.pone.0160147.t009

We compared the recognition performance of CEERM based on rules and SVM. Table 10shows the experimental results. We can conclude from these results that the DL-based method has the best recognition performance. They also reveal that DL is extremely effective in text mining and suitable for a wider application in NLP.

Table 10: Recognition performance of CEERM, based on rules and SVM

Methods	Supervision way	R	P	F
CEERM	unsupervision	87.32	83.12	85.17
CEERM	dynamic supervision	90.32	86	88.11
Based on rules	--	80.3	78.6	79.44
Based on SVM	--	84.5	81.1	82.77

doi:10.1371/journal.pone.0160147.t010

Although the method proposed in this paper can overcome the limitations of the existing event recognition methods and offers a great improvement in the recognition performance and stability, the analysis of our experimental results reveals that the classification performance does not reach 100%. This is likely due to a number of different reasons;

- DBN is an effective multi-classification deep neural network and the recognition performance of DBN is generally better than that of shallow layer neural networks. However, regardless of whether the application is to image recognition or other areas,

the method has not been able to achieve a 100% classification result. This is why the method proposed in this paper also fails to achieve a perfect recognition result.

- In this paper, the testing and training corpus are disjoint, leading to the semantic features of some events in the test corpus not being learned by the deep classifiers, which can affect recognition performance. But if the testing and training corpus are joint, then the scalability of the model will be affected to a certain extent. Especially during application, it may be necessary to process a large number of non-labeled texts. This scenario requires that the testing and training corpus are disjoint.
- This method is mainly used for event recognition in Chinese texts. Different from English NLP, Chinese NLP must use word segmentation technology. In this paper, we apply the LTP of the Harbin Institute of Technology as a segmentation tool, which has a higher recall and precision rate, but has at the same time still failed to reach 100% recognition performance. The errors due to an imperfect application of LTP also influence the model recognition performance negatively, thereby reducing the overall model recognition performance.
- In order to improve event recognition performance, this paper proposes a variety of event recognition features. However these do not capture all features of events, there may be other potential features to be excavated, which may improve recognition performance of the system. In future research, we believe that we need to mine more features so as to achieve the best possible recognition results.

DISCUSSION

In this study, we examined event recognition problems combining semantic information. We transformed event recognition into semantic feature classification and proposed CEERM by applying DL to event recognition. Our method performed better than existing event recognition methods. Based on the DL framework, CEERM can obtain deep features of text data to improve event recognition performance. We

also designed two kinds of classifiers. One is the unsupervised classifier, which is efficient but shows poor recognition performance. The other is the dynamic supervised classifier, which not only improves recognition performance but also limits training time. Our study shows that the dynamic supervised classifier obtains the most comprehensive results.

We also tested the recognition performance of CEERM. Experimental results show that recognition performance can be improved when the number of RBM training layers increases to a certain extent. Simultaneously, the relationship between recognition performance and increasing number of RBM layers is nonlinear. When the RBM layers reach a certain number, the recognition performance of the two classifiers diminishes. Therefore, we must choose suitable RBM layers based on specific application requirements. Introducing feature layers to CEERM can help improve recognition performance. In fact, test results show that adding feature layers can improve system performance regardless of the number of RBM layers in CEERM.

In the future, the performance and scalability of CEERM can be improved. In addition, this model could be applied to other element recognition in event ontology. Although DL has achieved breakthroughs in audio and image recognition, it has not yet a considerable influence on NLP. Thus, the connections between words, sentences and paragraphs are relatively obscure. A limited RBM structure of an existing DL can not express the vector structure of texts precisely. Therefore, designing a DL model that is suitable for text analysis may become a trend in future research.

ACKNOWLEDGMENTS

This study is supported by the National Natural Science Foundation of China (NSFC), No.61305053, No.61273328, and No.71203135.

AUTHOR CONTRIBUTIONS

- **Conceived and designed the experiments:** YZ ZL.
- **Performed the experiments:** YZ.
- **Analyzed the data:** YZ.
- **Contributed reagents/materials/analysis tools:** ZL WZ.

- **Wrote the paper:** YZ WZ.

REFERENCES

1. Wen Z, Liu Z, Ying L, et al. Event-Based Knowledge Acquisition for Ontology Learning. IEEE, 2007:498–501.
2. Liu Z T, Huang M L, Zhou W, et al. Research on Event-oriented Ontology Model. Computer Science, 2009, 36(11): 189–192.
3. Hinton G E, Osindero S, Teh Y W. A fast learning algorithm for deep belief nets. Neural computation, 2006, 18(7): 1527–1554. pmid:16764513
4. Lee T S, Mumford D. Hierarchical Bayesian inference in the visual cortex. JOSA A, 2003, 20(7): 1434–1448. pmid:12868647
5. Serre T, Wolf L, Bileschi S, et al. Robust object recognition with cortex-like mechanisms. Pattern Analysis and Machine Intelligence, IEEE Transactions on, 2007, 29(3): 411–426.
6. Lee T S, Mumford D, Romero R, et al. The role of the primary visual cortex in higher level vision. Vision research, 1998, 38(15): 2429–2454. pmid:9798008
7. Glorot X, Bordes A, Bengio Y. Domain adaptation for large-scale sentiment classification: A deep learning approach Proceedings of the 28th International Conference on Machine Learning (ICML-11). 2011: 513–520.
8. Xuefeng X I, Zhou G. Pronoun Resolution Based on Deep Learning Acta Scientiarum Naturalium Universitatis Pekinensis, 2014, 50(1):100–110.
9. Weng J. Brain as an emergent finite automaton: A theory and three theorems[J]. International Journal of Intelligence Science, 2015, 5(02):112.
10. Erhan D, Bengio Y, Courville A, et al. Why Does Unsupervised Pre-training Help Deep Learning? Journal of Machine Learning Research, 2010, 11(3):625–660.
11. Hinton G E, Sejnowski T J. Learning and releaming in Boltzmann machines Parallel distributed processing: Explorations in the microstructure of cognition, 1986, 1: 282–317.
12. Shen M, Dong B, Xu L. An Improved Method for the Feature

Extraction of Chinese Text by Combining Rough Set Theory with Automatic Abstracting Technology Contemporary Research on E-business Technology and Strategy. Springer Berlin Heidelberg, 2012: 496–509.

13. He R, Qin B, Liu T. A novel approach to update summarization using evolutionary manifold-ranking and spectral clustering Expert Systems with Applications, 2012, 39(3):2375–2384.
14. Bombieri N, Fummi F, Pravadelli G. Automatic abstraction of RTL IPs into equivalent TLM descriptions Computers, IEEE Transactions on, 2011, 60(12): 1730–1743.
15. Pitler E, Louis A, Nenkova A. Automatic evaluation of linguistic quality in multi-document summarization. Proceedings of the 48th annual meeting of the Association for Computational Linguistics. Association for Computational Linguistics, 2010: 544–554.
16. Yen S J, Wu Y C, Yang J C, et al. A support vector machine-based context-ranking model for question answering. Information Sciences, 2013, 224: 77–87.
17. Cao Y, Liu F, Simpson P, et al. AskHERMES: An online question answering system for complex clinical questions. Journal of biomedical informatics, 2011, 44(2): 277–288. pmid:21256977
18. Yao Y, Tong H, Xie T, et al. Detecting high-quality posts in community question answering sites. Information Sciences, 2015, 302: 70–82.
19. Comas P R, Turmo J, Màrquez L. Sibyl. A Factoid Question-Answering System For Spoken Documents. Acm Transactions on Information Systems, 2012, 30(3):248–248.
20. Carpineto C, Romano G. A survey of automatic query expansion in information retrieval. ACM Computing Surveys (CSUR), 2012, 44(1): 1.
21. Gagie T, Navarro G, Puglisi S J. New algorithms on wavelet trees and applications to information retrieval. Theoretical Computer Science, 2012, 426: 25–41.
22. Paltoglou G, Salampasis M, Satratzemi M. Collection-integral source selection for uncooperative distributed information retrieval environments. Information Sciences, 2010, 180(14): 2763–2776.

23. Zhou D, Truran M, Brailsford T, et al. Translation techniques in cross-language information retrieval. ACM Computing Surveys (CSUR), 2012, 45(1): 1.
24. Yankova M, Boytcheva S. Focusing on scenario recognition in information extraction. Proceedings of the tenth conference on European chapter of the Association for Computational Linguistics-Volume 2. Association for Computational Linguistics, 2003: 41–48.
25. Lee C S, Chen Y J, Jian Z W. Ontology-based fuzzy event extraction agent for Chinese e-news summarization. Expert Systems with Applications, 2003, 25(3): 431–447.
26. Jiang J. An Event IE Pattern Acquisition Method. Computer Engineering, 2005, 31(15):96–98.
27. Liang H, Chen X, Wu P B. Information extraction system based on event frame. Journal of Chinese Information Processing, 2006, 20(2): 40–46.
28. Feng Li. Breaking events' information extraction based on event frame. Shanghai Jiao Tong University, 2008
29. Chieu H L, Ng H T. A maximum entropy approach to information extraction from semi-structured and free text. AAAI/IAAI, 2002, 2002: 786–791.
30. Fu J, Liu Z, Zhong Z, et al. Chinese event extraction based on feature weighting. Information Technology Journal, 2010, 9(1): 184–187.
31. Ahn D. The stages of event extraction. Proceedings of the Workshop on Annotating and Reasoning about Time and Events. Association for Computational Linguistics, 2006: 1–8.
32. Wang W, Zhao D, Zou L, et al. Extracting 5W1H event semantic elements from Chinese online news. Web-Age Information Management. Springer Berlin Heidelberg, 2010: 644–655.
33. McCracken N, Ozgencil N E, Symonenko S. Combining techniques for event extraction in summary reports. AAAI 2006 Workshop Event Extraction and Synthesis. 2006: 7–11.
34. Kim S, Jeong M, Lee G G. A local tree alignment approach to relation extraction of multiple arguments. Information Processing & Management, 2011, 47(4): 593–605.

35. Yu X H, Chen G A. Efficient backpropagation learning using optimal learning rate and momentum. Neural Networks, 1997, 10(3): 517–527.
36. Hinton G E, Salakhutdinov R R. Reducing the dimensionality of data with neural networks. Science, 2006, 313(5786):504–507. pmid:16873662
37. Hinton G E. Products of experts. Artificial Neural Networks, 1999. ICANN 99. Ninth International Conference on (Conf. Publ. No. 470). IET, 1999, 1: 1–6.
38. Hinton G E. Training products of experts by minimizing contrastive divergence. Neural computation, 2002, 14(8): 1771–1800. pmid:12180402
39. Carreira-Perpinan M A, Hinton G. On Contrastive Divergence Learning. AISTATS. 2005, 10: 33–40.
40. Hinton G E, Salakhutdinov R R. A better way to pretrain deep boltzmann machines. Advances in Neural Information Processing Systems. 2012: 2447–2455.
41. Che W, Li Z, Liu T. Ltp: A chinese language technology platform. Proceedings of the 23rd International Conference on Computational Linguistics: Demonstrations. Association for Computational Linguistics, 2010: 13–16.
42. Miwa M, Sætre R, Kim J D, et al. Event extraction with complex event classification using rich features. Journal of bioinformatics and computational biology, 2010, 8(01): 131–146. pmid:20183879
43. Nivre J. Dependency grammar and dependency parsing. MSI report, 2005, 5133(1959): 1–32.
44. Hongye Tan. Research on Chinese event extraction[D]. Harbin Institute of Technology, 2008.
45. Ding Xiao. Research on sentence level Chinese event extraction[D]. Harbin Institute of Technology, 2011.
46. Li PF, Zhou GD, Zhu QM. Semantics-Based joint model of Chinese event trigger extraction. Ruan Jian XueBao/Journal of Software, 2016,27(2):280–294 (inChinese).

CHAPTER 9

Imputation for Transcription Factor Binding Predictions Based on Deep Learning

Qian Qin, Jianxing Feng

Department of Bioinformatics, School of Life Sciences and Technology, Tongji University, Shanghai, China

ABSTRACT

Understanding the cell-specific binding patterns of transcription factors (TFs) is fundamental to studying gene regulatory networks in biological systems, for which ChIP-seq not only provides valuable data but is also considered as the gold standard. Despite tremendous efforts from the scientific community to conduct TF ChIP-seq experiments, the available data represent only a limited percentage of ChIP-seq experiments, considering all possible combinations of TFs and cell lines. In this study, we demonstrate a method for accurately predicting cell-specific TF binding for TF-cell line combinations based on only a small fraction (4%) of the combinations using available ChIP-seq data. The proposed

model, termed TFImpute, is based on a deep neural network with a multi-task learning setting to borrow information across transcription factors and cell lines. Compared with existing methods, TFImpute achieves comparable accuracy on TF-cell line combinations with ChIP-seq data; moreover, TFImpute achieves better accuracy on TF-cell line combinations without ChIP-seq data. This approach can predict cell line specific enhancer activities in K562 and HepG2 cell lines, as measured by massively parallel reporter assays, and predicts the impact of SNPs on TF binding.

INTRODUCTION

Transcription factors (TFs) play a central role in regulating gene expression. Although TF binding to chromatin is primarily dictated by the DNA sequence, the binding patterns can be cell-specific through cooperative interactions between many different TFs. Mutations in cis-regulatory elements can influence TF binding, with potentially deleterious effects [1–3]; therefore, it is important to assess the impact of cis-element mutations on cell-specific TF binding and gene expression. ChIP-seq has been the gold standard for evaluating cell-specific TF binding [4]. However, despite tremendous efforts from the scientific community to generate large-scale TF ChIP-seq data, such as those undertaken by the ENCODE consortium [5], most TFs have been profiled for only a limited number of cells and conditions. Therefore, computational modeling of TF ChIP-seq data to infer the underlying binding rules and to predict TF binding under un-profiled conditions could be a useful and economical approach.

The classical computational model employed to describe TF binding preferences is based on position weight matrices (PWMs) or motifs [6–11]. In addition to binding motifs, other sequence features, such as low-affinity binding sites, flanking DNA, and certain repeat sequence symmetries flanking specific targets, also influence TF binding affinity [12–14]. To better model TF-DNA binding specificity, many DNA-sequence-based models beyond PWM have been proposed [9,13,15,16]. In vivo TF binding is also affected by many other factors such as TF-TF interactions and sequence GC bias. Such information is naturally incorporated in TF ChIP-seq data yet are missing from TF motifs.

Therefore, computational models built on ChIP-seq data [17–19] have the potential to capture this information to achieve higher accuracy than those built solely on DNA motifs for in vivo TF binding prediction [20]; however, such studies are limited by the available ChIP-seq data. Last, but not least, the epigenetic landscapes, such as those profiled by histone modification ChIP-seq and DNase-seq, are closely related to TF binding. Computational methods considering motif occurrences in open chromatin regions [21–24] often achieve high accuracy. Such models and TF ChIP-seq data based models may compensate each other.

Deep learning has been actively studied in the machine learning community and has been successfully applied to diverse problems such as image processing, computer vision, speech recognition, and natural language processing over the past decade [25,26]. Recently, deep learning techniques were applied to regulatory genomics and showed exciting performance improvements compared to previous methods [19,27]. Intuitively, when models based on deep learning are trained on ChIP-seq datasets, they can automatically extract sequence features determining TF binding affinity. The extracted features contain not only traditional motifs but also sequence features from a wide window (1,000 bp) flanking the binding sites and TF-TF interactions. However, these models rely on training and cross validation with available ChIP-seq or DNase-seq data and thus are not suitable when such data for a specific cell condition are missing.

In this study, we developed a deep learning method, i.e., TFImpute, to predict cell-specific TF binding based on existing ChIP-seq data. TFImpute embeds cell lines and TFs into continuous vectors that serve as part of the model input. After being trained on available ChIP-seq data from 562 unique TF-cell line combinations (4% of all possible combinations) with 172 transcription factors and 91 cell lines [5], TFImpute predicted TF binding for the remaining 96% TF-cell combinations, for which ChIP-seq data were unavailable. Compared to the existing methods, TFImpute gives more accurate predictions for the binding of TFs on cell types without corresponding ChIP-seq data. Although the imputation of TF binding is considered more challenging than the imputation of epigenetic marks on cell lines without ChIP-seq data [28], our predictions based on

TFImpute are highly consistent with results based on massively parallel reporter assays and with evidence in the literature for the selected SNPs.

MODELS

Given a DNA sequence, a TF and a cell line, TFImpute (Fig 1) predicts whether the TF would bind to the DNA sequence in this cell line. For DNA sequences, the 1-hot encoded signal [19,27] is first passed through a convolution layer [29], which consists of a set of filters. Each of the filters detects whether a specific sequence pattern, i.e. motif, appears in the DNA sequence.

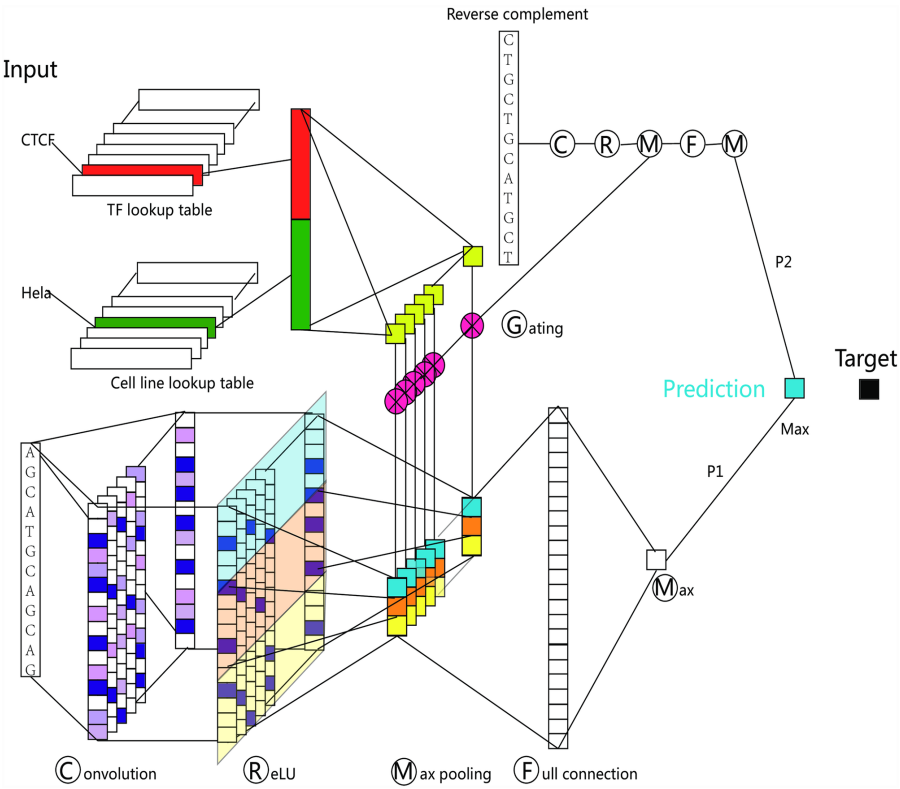


Figure 1: The TFImpute model.

Each input is a TF-cell-sequence triple. In the convolution layer, each filter (motif) corresponds to a column. Each filter scans the input sequence and produces one value at each stop. For each filter, the max-pooling layer

partitions the signal into three windows and takes the maximum value in each window to obtain three values. The same gate signal operates on the three values, and the gate signal is different for different filters. For each input, the reverse complement of the input sequence together with the TF and cell line is constructed and used as another input for the same network. Therefore, for each input, we obtained two values for forward and reverse strand of the sequence: P1 and P2. The maximum of P1 and P2 is taken as the final prediction. During training, the prediction was compared with the target, and the error was back-propagated to learn the parameters of the whole network.

For TFs and cell lines, a technique called embedding [30] is used to encode each TF or cell line as a continuous vector. For each input TF and cell line, their embedding is concatenated and used to calculate a set of gates. The gates are used to tune the signal processed by the convolution layer such that only signals detected by filters related to the current TF and cell line will pass through. For example, if the input DNA sequence contains a motif of the input TF but the TF is not expressed in the input cell line, the signal detected by the filter related to the motif can be tuned down by the gating mechanism.

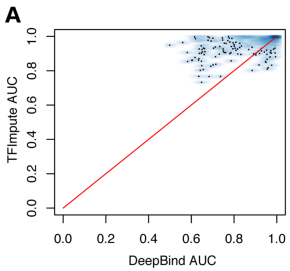
In the model, embedding for TFs and cell lines, the sequence pattern filters and all other network connections are parameters. To learn these parameters, TFImpute first gives a prediction based on its current parameters and the input, and then compares the prediction with the true label. The difference between the prediction and the truth, i.e. the error, is back-propagated through the whole network, and all the model parameters are updated using AdaDelta [31], a variant of the stochastic gradient descent algorithm (See the Materials and methods section for details).

RESULTS

Using random sequences as background

To validate the TFImpute model, we first compared it with DeepBind [19], one of the latest deep learning models for TF binding prediction with superior performance to that of several other benchmark methods

not based on deep learning. The experimental setting follows [19] with a slight modification. Specifically, we used ChIP-seq peaks from the ENCODE project [5] as the training, validating, and testing data. For each peak region, we utilized 100 bp of DNA sequence around the peak center as the positive peaks and randomly shuffled nucleotides with the number of dinucleotides matching those for each positive peak as the negative peaks. For each ChIP-seq dataset, DeepBind builds a separate model using even-numbered peaks from the top 1,000 peaks as the test set and all other peaks as the training set. For TFImpute, we used the same test set but employed only odd-numbered peaks from the top 1,000 peaks as the training set. In other words, for TFImpute, each ChIP-seq dataset contributed only 500 positive peaks and 500 negative peaks for training. Instead of training a separate model for each ChIP-seq dataset, we trained TFImpute for a single model based on all the ChIP-seq data. The expectation was that the binding information of one TF in one cell line could be helpful for predicting the binding of the same TF or other TFs in other cell lines. For example, the binding sequences of STAT1 and STAT2 should be similar across different cell lines so the training data in one condition may compensate for the lack of training data in another condition. A performance comparison between TFImpute and DeepBind, as shown in Fig 2A and 2B, confirmed the effectiveness of the joint training setting. Although TFImpute uses less training data, it gives better results without the possible drawback of over-training, a serious problem for DeepBind, which requires complicated methods to perform regularization and hyperparameter selection. The underlying reason for the TFImpute to avoid over-training is that TFImpute fits a single model using all the training data but DeepBind fits a separate model for each TF ChIP-seq data set. On the same dataset, we also compared TFImpute with gkm-SVM [15], a state-of-the-art TF binding prediction algorithm not based on deep learning in Fig 2C. Both methods are comparable.



B

TF	Cell line	DeepBind	TFImpute
BRCA1	HeLa-S3	0.4988	0.9490
PML	K562	0.5597	0.9642
STAT3	HeLa-S3	0.5814	0.9064
SP2	HepG2	0.5827	0.8536
NRF1	K562	0.6024	0.8602
NR2C2	K562	0.6045	0.8619
NR2F2	K562	0.6156	0.9972
POLR2A	A549	0.6173	0.8774
NRF1	GM12878	0.6196	0.9869
PML	GM12878	0.6244	0.9757

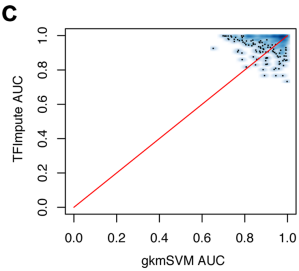


Figure 2: AUC comparison of TFImpute with DeepBind and gkm-SVM using shuffled sequences as negative instances.

(A) Comparison with DeepBind. Each point in the figure corresponds to a TF-cell line combination. (B) AUC for TF-cell line combinations in which DeepBind gives the lowest AUC. (C) Comparison with gkm-SVM using randomly shuffled sequences as negative instances. Each point in the figure corresponds to a TF-cell line combination.

Using DNase I hypersensitive sites as background

DeepBind defines negative peaks as random sequences in which the number of dinucleotides is matched to the positive set [19]. Another definition of random sequences requires sequence length, GC and repeat fractions that match the positive set [15]. However, in vivo DNA sequences are not random in above senses and the binding sites of TFs are mostly concentrated on active genomic regions, specifically the DNase I hypersensitive sites. Therefore, a set of union DHS sites provide a more biologically meaningful background for TF binding prediction. In this work, our method first defines unified DNase I hypersensitive sites using 122 ENCODE DNase-seq datasets [5] as the background. Then, for each TF ChIP-seq dataset, the cell-specific negative peaks are defined as the unified DNase-seq peak regions that do not overlap with any of the TF peaks in the dataset. Then, the top 5,000 peaks are taken as positive peaks, with each one being matched by a randomly selected negative peak.

We denote the set of TF-cell line combinations as F. To evaluate the performance of TFImpute, we first define 4 subsets (Base, TestSet2, ValidSet2 and TestSet3) of F in Table 1. Note that the TF-cell line combinations in TestSet3 do not appear in Base where the training set is

coming from. In other words, we are training a model using TF binding information in one cell line but verify the model in another cell line. This is biologically meaningful if there are multiple TFs and cell lines involved. For example, if two TFs (TF1 and TF2) were involved and a model learned that TF1 and TF2 are similar due to their similar binding profiles across many cell lines from the training data, it can give a better binding prediction for TF1 in a cell line if the training data contains the binding information of TF2 in this cell line. Therefore, we can evaluate whether a model can utilize such information by training a model for TF1 and TF2 in many cell lines excluding some cell line X and then evaluating the model's ability to predict TF1 binding in cell line X. The opposite extreme situation is that TF2 is very different from TF1. In this case, the best choice of the model is to ignore TF2 when predicting TF1 binding profile. Although the similarities between TFs or cell lines vary, we hope that the proper designed network structure of TFImpute can learn how to use TF2 in binding prediction of TF1. The designed TestSet3 in Table 1 is to verify whether TFImpute can gain such prediction ability.

Table 1: Subsets of TF-cell line combinations using DNase I hypersensitive sites as background

Definition	Set size	Description
$Base \subset F$	473	Would be further partitioned to get the training set.
$TestSet2 \subset Base$	70	Verify the model's imputation power on replicates.
$ValidSet2 \subset F - Base - TestSet3$	20	Monitor over-training for TF-cell line combinations not in the training set. All TF or cell lines in ValidSet2 are thoroughly represented in Base.
$TestSet3 \subset F - Base - ValidSet2$	69	Verify the model's imputation power on TF-cell line combinations not in the training set. All TF or cell lines in TestSet3 are thoroughly represented in Base but do not need to be in ValidSet2.

doi:10.1371/journal.pcbi.1005403.t001

Each TF-cell line combination corresponds to a set of instances for the input of TFImpute. Each instance is a <1/0, TF, cell line, DNA sequence> quadruple. 1/0 denotes whether the DNA sequence is a TF binding sequence. To simplify the annotation, we also use Base (or TestSet2, ValidSet2, TestSet3) to denote the union of instances corresponding to TF-cell line combinations in Base (or TestSet2, ValidSet2, TestSet3). Instances in Base are further partitioned into 3 disjoint subsets (TrainSet, TestSet1 and ValidSet1) in Table 2.

Table 2: Random partition of the instances in Base into 3 disjoint subsets

Name	#Instances	Description
TrainSet	3,708,593	The training set
ValidSet1	100,000	Monitor over-training for TF-cell line combinations in the training set.
TestSet1	900,000	Verify the model on TF-cell line combinations in the training set.

doi:10.1371/journal.pcbi.1005403.t002

During calculation, we first maximize the set size of the union of ValideSet2 and TestSet3. Then randomly selected 20 of them as the ValidSet2. Finally, we maximize the set size of TestSet2. Because ChIP-seq experiments are not evenly distributed for TFs or cell lines, subsets defined above are biased towards TFs and cell lines that have many ChIP-seq experiments in the dataset. This is a possible mechanism of overfitting.

Using GC matched negative instances

For a comprehensive comparison with gkm-SVM, we also applied TFImpute to the dataset in [15], where the GC matched sequences are used as negative instances. Similar to Tables 1and 2, we partitioned the dataset into 6 subsets: TrainSet, ValidSet1, TestSet1, ValidSet2, TestSet2, and TestSet3.

Table 3: Subsets of TF-cell line combinations using GC matched negative instances

Definition	Set size
Base	339
TestSet2	67
ValidSet2	20
TestSet3	41

doi:10.1371/journal.pcbi.1005403.t003

Table 4: Random partition of the instances in Base of Table 3 **into 3 disjoint subsets**

Name	#Instances
TrainSet	2000000
ValidSet1	200000
TestSet1	696076

doi:10.1371/journal.pcbi.1005403.t004

Comparisons with gkm-SVM and PIQ

To compare TFImpute with gkm-SVM [15] and PIQ [22], we apply the default parameters for both gkm-SVM and PIQ. gkm-SVM cannot accept inputs from multiple conditions; thus, we trained gkm-SVM separately on each TF-cell line combination on the union of the training and validation sets. PIQ does not need a training set but accepts a genome-wide DNase I hypersensitivity profile and a known motif to predict the binding of a TF. We apply PIQ on TestSet3 with matched cell line DNase I hypersensitivity profiles.

A comparison is shown in Fig 3A. For TestSet1, TFImpute achieved an average AUC above 0.94 and performed slightly better than gkm-SVM. For TestSet2, the set to verify the accuracy of the methods on ChIP-seq replicates, both methods were comparable with no statistically significant difference. The most important result was observed for TestSet3, in which the tested TF-cell line combinations were not in the training set. To evaluate the performance of gkm-SVM on TestSet3, we randomly selected a trained gkm-SVM model based on TrainSet with matched TFs for each dataset in TestSet3. For TestSet3, TFImpute performed much better (average AUC 0.895) than gkm-SVM (average AUC 0.852). The p-values (0.048 for TestSet1 and 0.06 for TestSet3) between TFImpute and gkm-SVM AUC indicate that the differences are not very significant. The main reason for this finding is that the numbers of data points are small. When the predictions were grouped by TF-cell line combinations, we obtained more significant p-values (4.446e-05 for TestSet1 and 0.001 for TestSet3; Fig 3B). To rule out the possibility that TFImpute outperforms gkm-SVM by chance, we randomly partition Base 5 extra times based on Table 2 and check the variability of TFImpute AUC. The results show that the variation of TFImpute AUC is very small. On TestSet3, PIQ achieves the lowest AUC. A possible reason is that PIQ does not give predictions on regions without strong motif or regions lack of DNase I signal. Therefore, we removed such regions before calculating the AUC (PIQNoNA in Fig 3A). We found that TFImpute still outperforms PIQ in this case.

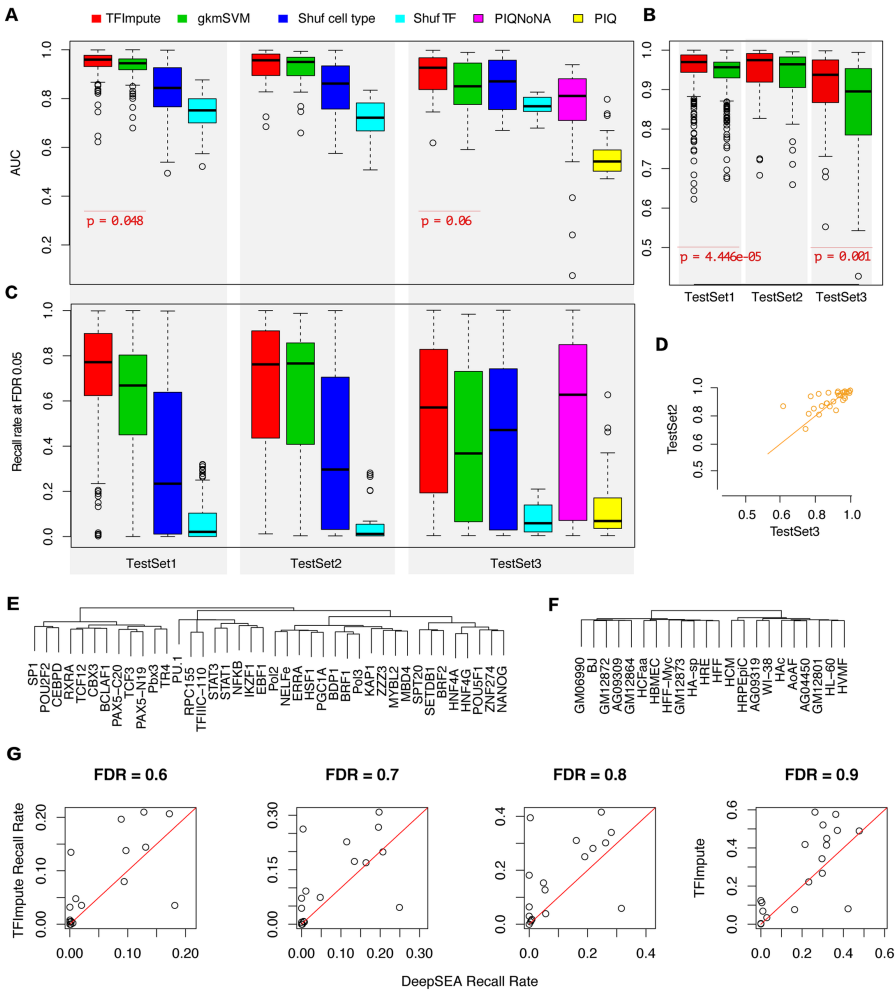


Figure 3: Comparison with gkm-SVM, PIQ, and DeepSEA.

(A) AUC comparison of TFImpute and gkm-SVM on TestSet1, TestSet2, and TestSet3. 'Shuf cell line indicates that the cell line of the corresponding test set was shuffled and that the trained TFImpute model was then applied to the shuffled dataset. Similarly, 'Shuf TF' indicates that the TFs were shuffled. For some of the given regions, PIQ give NA predictions. NA means that there is no motif based on log probability threshold of 5, or the region is lack of DNase I signal. PIQNoNA in this figure denotes the result after removing all NAs and PIQ denotes the result after treating NAs as no binding. To calculate the AUC, the

predictions were grouped by TFs. The middle bar in each box indicates the median. (B) AUC comparison based on predictions grouped by TF-cell line combinations. (C) The recall rates of different methods at FDR 0.05 (See Material and methods for more details). The predictions were grouped by TFs. (D) AUC comparison of TFImpute on TFs appearing in both TestSet2 and TestSet3. (E) Hierarchical clustering of a subset of the TFs based on the learned embedding by TFImpute. (F) Hierarchical clustering of a subset of cell lines based on the learned embedding by TFImpute. (G) The recall rate of TFImpute and DeepSEA at different FDR cutoffs on the datasets provided by DeepSEA.

Because a TF binds to only a small proportion of the chromatin accessible regions, methods that can impute TF binding with low FDR are more valuable. In Fig 3C, we compared the recall rates of different methods at fixed FDR 0.05. On TestSet1 and TestSet3, TFImpute outperforms gkm-SVM and PIQ. The conclusion also holds at FDR 0.01 and FDR 0.1. Another important application of TF binding prediction is to scan possible binding sites genome-widely. We evaluated the performances of the three methods on non-overlapping bins of length 300bp on chr8. Then we applied the trained models of TFImpute, gkmSVM on TrainSet to TF and cell line combinations in TestSet3 on chr8, and extracted the maximum score of PIQ calls for each bin on chr8. Note that PIQ needs associated DNase-seq data for each cell line but TFImpute and gkmSVM are solely based on DNA sequence. TFImpute and gkmSVM are comparable and relatively better than PIQ at large FDR cutoffs (95% and 99%). At FDR cutoff 50%, PIQ achieves higher recall rates. The low recall rate of TFImpute is partially due to the fact DNase region instead of the whole genome were used as the background during training. The following comparison with DeepSEA shows that the recall rate increases as wider regions were used as background.

To further confirm the super performance of TFImpute over gkm-SVM, we further applied both methods on the dataset used to validate gkm-SVM [15]. This dataset uses GC matched negative regions. The AUC and recall rate comparison demonstrate that TFImpute outperforms gkm-SVM consistently.

We also compared the running time of TFImpute, gkm-SVM and PIQ in Table 5. The total time costs of three methods are comparable. We didn't test TFImpute on CPU because CNN neural networks are

generally extremely slow on CPU. TFImpute doesn't have running time for each dataset separately because all the training data are supplied to TFImpute jointly. The running time of gkm-SVM and PIQ is the average time of processing all the datasets.

Table 5: Running time of different methods

	Hardware	Training Time
TFImpute	GPU GTX970	80h/all dataset
gkm-SVM	CPU AMD Phenom(tm) II X4 910e	38min/dataset
PIQ	CPU AMD Phenom(tm) II X4 910e	40min/dataset

doi:10.1371/journal.pcbi.1005403.t005

Finally, we compared the performance of TFImpute on TestSet2 and TestSet3. Because the TFs in TestSet3 were different from those in TestSet2, the differing performance of TFImpute on the two test sets may not be directly comparable with the overall AUC shown in Fig 3A. Therefore, we focused on transcription factors that appeared in both TestSet2 and TestSet3. For most of these TFs, TFImpute performed equally well (Fig 3D), indicating that the learned interactions of TFs and cell lines determined by TFImpute can be generalized to novel TF-cell line combinations. This benefit is highly valuable because most of the TF-cell line combinations don't have corresponding ChIP-seq data.

Comparisons with DeepSEA

Because the released DeepBind software package does not contain the training step, we could not train it on our dataset. Therefore, we compared TFImpute with DeepSEA on the datasets provided in [27]. This dataset defines the background as regions where at least one TF binds in at least one of the cell lines. Such a wide background gives us the opportunity to estimate the recall rate of a method on certain FDR cutoffs genome-widely.

To compare with DeepSEA genome-widely, we transformed the data from DeepSEA paper into the format required by TFImpute. We faced three obstacles. First, the input of TFImpute requires <DNA sequence, TF, cell line > triples. If we use all negative peaks in the data provided by DeepSEA to train TFImpute, the data would be too large to be trained. Second, given a <DNA sequence, TF, cell line> triple, the current TFImpute model predicts whether the TF would bind to the DNA

sequence in the cell line. It is too inefficient to cases if we want to predict the binding of a set of TFs on a DNA sequence in a cell line. Third, the input DNA sequence of DeepSEA is 1000bp and the DeepSEA model used three layers of convolutions to handle the long DNA sequence. The TFImpute model was designed for relatively short sequences and therefore only one layer of convolution was used to save computation time.

We found no easy way to solve above three problems simultaneously. The main reason is that the TF and cell line information have to be input to the model in some way to keep the imputation ability. This imputation ability dramatically increases the required computation. Therefore, we conducted the following procedures to partially solve the above problems and compared TFImpute with DeepSEA genome-widely.

First, to train TFImpute we excluded histone modifications and DNase-seq data and sampled one negative peak for each positive peak to reduce the data size. For DeepSEA, we used full dataset and reproduced the prediction results in the DeepSEA paper. Second, to compare with DeepSEA genome-widely, we queried the Cistrome Data Browser database [34] for another 19 TF ChIP-seq datasets not included in the training set (See Material and methods for more details). In the 19 datasets, the TF and cell line appears in the training set but the TF-cell line combinations are not contained in the training set. Then we partitioned chr8 into non-overlapping bins of length 1000bp and evaluated TFImpute and DeepSEA on chr8. For each ChIP-seq dataset, a bin is considered as having TF binding if there is a peak center in the bin. Because the TF-cell line combinations of the 19 datasets are not contained in the training set, DeepSEA cannot give predictions for the right cell line for each TF. Therefore, we selected the prediction of a random cell line in the training set for each TF to form the final prediction of DeepSEA. Third, we adjusted the network structure of TFImpute such that it can handle 1000bp DNA sequence without imposing extra computation needs and can predict the binding of a set of TFs on a DNA sequence in a cell line simultaneously (See Material and methods for more details).

We compared the recall rates of DeepSEA and TFImpute at different FDR cutoffs. Generally, the recall rates at low FDR are quite low for both methods. To get a reasonable recall rate (e.g. 20%), the FDR should be above 50%. The recall rates at 4 FDR cutoffs (60%, 70%, 80% and 90%)

are shown in Fig 3G. The TFImpute model outperforms DeepSEA for majority of the cases.

Sequence features are the primary factors influencing TF binding

The TFImpute model provides the ability to estimate whether sequence features or cell lines have a more significant influence on TF binding. Fig 3A and 3C show that if a TF does not match the DNA sequences and cell lines (by shuffling TFs in the test set), the prediction accuracy drops more statistically significantly than when cell lines are shuffled. This result indicates that sequence features are the dominant factor in determining TF binding compared to cell line. Because randomly selecting a trained model based on TrainSet with matched TFs for each dataset in TestSet3 is equivalent to ignoring cell- specificity, we observed that the performance of TFImpute with shuffled cell lines was comparable to that of gkm-SVM on TestSet3 (Fig 3A).

Because each TF or cell line had learned embedding, we could compare the similarities between embedded vectors. We observed that many TFs with similar known motifs were grouped together, such as HNF4G/HNF4A and STAT1/STAT3 (Fig 3E). The meaning of embedding for cell lines is subtler because it is not the primary factor influencing TF binding. Therefore, we did not observe clear patterns in clustering for cell lines (Fig 3F).

Enhancer activity is correlated with the binding prediction based on TFImpute

To systematically estimate the efficacy of TFImpute, we applied TFImpute on a massively parallel reporter assay that experimentally characterized the role of enhancer sequences by introducing target mutations in K562 and HepG2 cell lines [35]. For a specific cell line and each of the enhancer sequences, TFImpute gave TF binding predictions for all the TFs in the training set. To obtain a summarized enhancer activity prediction for each enhancer sequence, we performed principle component analysis (PCA) on the matrix of the predicted binding affinity for all TFs and enhancer sequences. The first principle component was taken as a signature of enhancer activity for each sequence. Then, we compared the signature

value distribution of the top 100 enhancer sequences from that of the bottom 100 enhancer sequences after all the enhancer sequences were ranked by their experimentally measured activities. Fig 4 shows that the difference is much more statistically significant if the cell lines are matched. When the cell lines are matched, the p values for the Spearman rank correlation are much more statistically significant than other cases. This is reasonable because we would expect that the set of TFs binding to the active enhancers of K562 is different from that of HepG2. It also means that the signature based on TF binding prediction should not be considered as a prediction of the enhancer activity.

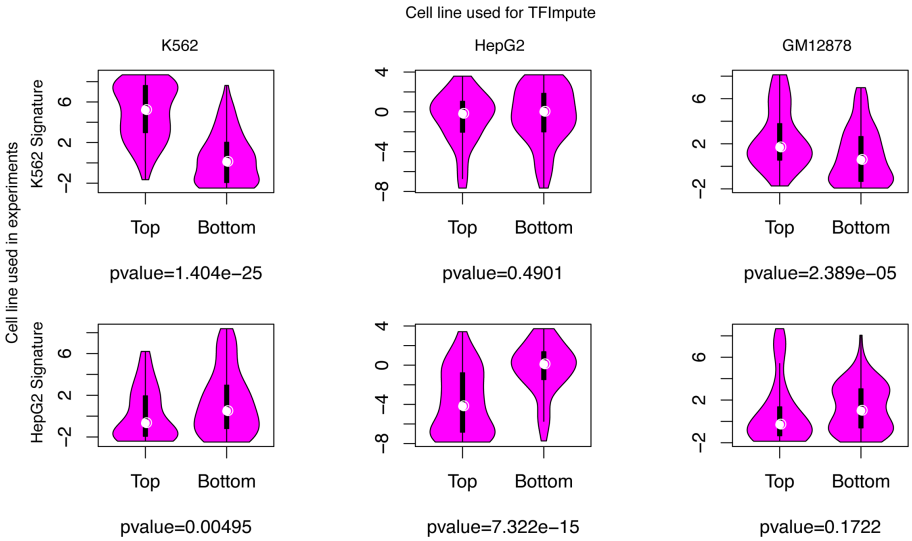


Figure 4: The distributions of the calculated enhancer signature for the top and bottom 100 enhancers.

The p value is calculated using t-test. We would like to emphasize the lack of data of the enhancer reporter assay of GM12878, which is a good control.

TFImpute accurately predicts the impact of SNPs on TF binding

Because TFImpute can accurately predict TF binding, we can use the trained TFImpute model to estimate TF binding differences in the minor and major alleles of individual SNPs for all possible TF-cell line combinations if the TF and the cell line appear in the training set. To validate the accuracy of TFImpute on mutated sequences, we filtered 10 SNPs from the literature that were reported to lead to differential binding for TFs and cell lines appearing in our training set. TFImpute gave predictions consistent with the literature for 7 of these SNPs. For example, in liver, the minor allele (T) of SNP rs12740374 created a binding site for C/EBP, and the major allele (G) disrupted the binding [24]. TFImpute accurately predicted that the C/EBP binding affinity would increase when the DNA sequence was changed from the major allele to the minor allele in HepG2, a liver hepatocellular carcinoma cell line (Fig 5). As another example, in SNP rs4784227 (C/T), a functional variant in breast cancer, the risk-associated allele (T) increases FoxA1 binding in breast cancer [36]. Although there is no ChIP-seq data for FoxA1 in MCF-7 in the training set, TFImpute predicted that the binding of FoxA1 would increase if the input sequence at the SNP location were changed from C to T in MCF-7. Moreover, for SNP rs4953223 (C/T), one of the DNase I—sensitivity quantitative trait loci (dsQTLs) in human lymphoblastoid cell lines (LCLs) [37], the T allele disrupts the binding of NF-kB and is strongly associated with low DNase I sensitivity [24]. TFImpute accurately predicted that the binding affinity of NF-kB would dramatically decrease in most of the cell lines, including all the GM cell lines (lymphoblastoid cell lines), if the DNA sequence at rs4953223 was changed from C to T. Moreover, dozens of other TFs, such as EBF1 and STAT1, also showed clear binding affinity decreases in most of the GM cell lines.

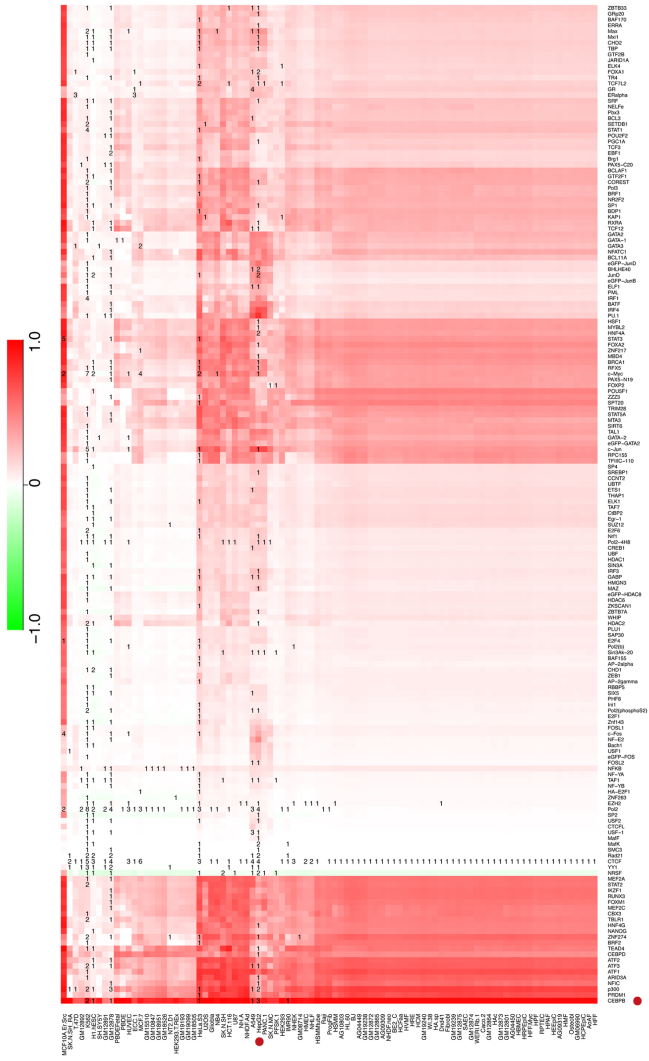


Figure 5: Predicted binding affinity change between two alleles of SNP rs12740374 (T/G).

The color in each cell represents the predicted binding affinity of allele G minus that of allele T for the corresponding TF and cell line. The number in each cell of the heatmap is the number of ChIP-seq datasets in the training set for the corresponding TF and cell line. If TFImpute predicted strong binding in the minor allele but no binding in the major allele, the score was 1. If TFImpute predicted no binding difference

between the two alleles, the score was 0.

DISCUSSION

The most efficient resource for studying TF binding is ChIP-seq. However, a ChIP-seq experiment is limited by its cost, the availability of proper antibodies, and the complexity of the protocol. Therefore, only a small fraction of possible ChIP-seq experiments have been performed, considering all possible TF-cell line combinations. Because TF binding is a cell-specific behavior, it is not optimal to transfer a trained TF binding prediction model without condition-specific features, such as sequence-only models of TF binding, from one condition to another. TFImpute provides an economical method for alleviating this problem. Compared with existing methods, the properly formed multi-tasking learning framework along with the embedding and the gating mechanism make TFImpute obtain more accurate predictions for TF-cell line combinations without ChIP-seq data, given that the TF and the cell line appear in the training set. As more ChIP-seq data are used to train the model, TFImpute will be more powerful because the number of TF-cell line combinations increases quadratically with the number of TFs and cell lines.

To summarize enhancer activities, we utilized the first PCA component of the predicted TF binding score for all TFs and enhancer sequences. This method represents a rather simplified approach. It is expected that the use of more PCA components would provide a better fit to the enhancer activity, which was observed. However, even when all the components were used to fit the activity of all enhancers using linear regression, we did not obtain a perfect fitting (r-squared are 0.1361 and 0.2359 for K562 and HepG2 respectively), for at least two underlying reasons. First, the measured enhancer activity was noisy [35]. Second, the activity of the enhancers was not linearly dependent on the TF binding signal. To address the non-linearity, a more dedicated model might be needed.

The trained TFImpute model based on non-mutated sequences is well suited for predicting the binding of TFs on mutated sequences such as SNPs. Similar results have also been observed in previous studies, such as [24,27]. Such models are valuable for studying the gene regulatory network *in silico*, which is more rapid and economical than

wet lab experiments. The unique imputation ability of TFImpute further extends the predictive power to some TF-cell line combinations without corresponding ChIP-seq data. Such an extension complements models based on sequence features and open chromatin profiling data, such as PIQ [22].

It is worth noting that the prediction reliabilities are different for different TF-cell line combinations without ChIP-seq data. Fig 5 clearly demonstrates this phenomenon. For approximately one half of the cell lines, only CTCF ChIP-seq data are available; therefore, the binding predictions are very similar for these cell lines. These predictions cannot intuitively reflect the cell-specificity. Important future work should include the quantitative estimation of the prediction reliability.

MATERIALS AND METHODS

The deep learning neural network

A deep learning neural network transforms inputs through multiple layers of simple computation units. Each unit provides very limited computations such as performing a summation or calculating the sigmoid function. Normally, computation units are connected only to adjacent layers. Similar to [19,27], our TFImpute model is also based on a convolutional neural network (CNN) [29]. However, our approach differs in that we also use TFs and cell lines as part of the input with embedding. Embedding is a technique that allows discrete features to be represented with continuous vectors. This technique has wide applications in natural language processing to represent words or relations [30]. With the embedding of TFs and cell lines, we can train the model to predict whether a sequence will exhibit cell-specific TF binding, even for TF-cell line combinations without data for all TFs and cell lines in the training set.

The 1-hot encoding method [19,27] represents each DNA sequence of length N by an N*4 matrix. Each column of the matrix corresponds to A, C, G, or T, and each row corresponds to a sequence location. For the i^{th} row of the matrix, we set the 1st (2nd, 3rd, or 4th) column to be 1 if the i^{th} j^{th} element of the sequence is A (C, G, or T), and all remaining elements of the matrix are set to 0.

A convolution layer first transforms the DNA sequence with multiple filters. Each filter can be conceptually considered as a motif. Specifically, the output of the convolution layer with input matrix X is another matrix Y of size $(N\text{-}F+1) * K$, where F is the length of the filter and K is the total number of filters. Y is defined as

$$Y_{r,k} = \sum_{i=1}^{F} \sum_{j=1}^{4} W_{i,j}^{(k)} X_{r+i-1,j}$$

where $W^{(k)}$ is a parameter matrix for the k^{th} filter and r is the filter position.

Next, a ReLU layer operates on the input matrix element by element. This layer filters out noise and keeps only significant signals by

$$\mathrm{ReLU}(x, b) = \begin{cases} x & if\ (x > b) \\ 0 & otherwise \end{cases}$$

where b is a parameter to reflect the fact that different motifs may have different binding affinities.

Then, a max-pooling layer follows. This layer takes the maximum value of the signal on non-overlapping windows. Intuitively, this layer detects whether a motif occurs in a window while ignoring the exact location of the occurrence.

After the pooling layer, the network goes through a gating layer. The gates are determined by the input TF and cell line. The purpose of gating is to further tune the signal such that only signals from motifs related to the current input TF and cell line pass through.

The calculation of gates is based on the embedding for the input TF and cell line, each of which is associated with a 1-dimensional continuous vector with 50 float elements. Given the input, the model first goes through a lookup table to obtain the embedding for the input TF and cell line. Then, a concatenated embedding vector goes through a one-layer neural network with sigmoid activations. In this way, the neural network can model complex interactions between TFs and cell lines to determine whether a TF will bind to a specific sequence in a specific cell line. This gating strategy differs from the method used by [38], which simply supplies the cell line or TF using a 1-of-N coding as an input to the network. The complete formula for describing the model is as follows:

$$G(X) = sigmoid(W_1 embed(X, E) + b_1)$$

$$M(S, X) = G(X) * Pooling(ReLU(conv(S, W), b))$$

$$F(S, X) = \max(W_2 M(S, X) + b_2)$$

$$Y = \max(F(S, X), F(revcomp(S, X)))$$

where X is the input TF and cell line; E is the embedding matrix for all TFs and cell lines; embed(X, E) simply concatenates the embedding vectors for the TF and the cell line from X; and S is the input sequence. *revcomp* takes the reverse complement of the input sequence, and E, W_1, W_2, b_1, b_2, b and $W = \{W^{(k)}, 1 \leq k \leq F\}$ are model parameters. Because the embedding for either the TF or the cell line is a 50-dimension vector, the output of embed(X, E) is a 100-dimension vector. Once the model has been trained, the embeddings for TFs and cell lines in the training set will be learned. For TFs or cell lines not in the training set, the model cannot be applied.

TFImpute achieves cell line specificity in a very different way from that of DeepSEA [27]. In that study, all the predictions shared the whole network, except for the last two layers. Therefore, the ability to differentiate various TFs and cell lines was limited by the model ability of the last two layers. Moreover, DeepSEA can model only TF-cell line combinations that appear in the training data and cannot extrapolate to cases for further TF-cell line combinations.

The revised neural network structure for comparison with DeepSEA

To compare with DeepSEA on long DNA sequence, we revised the network structure of TFImpute as follows. First, each cell line is associated with an embedding with length equal to the number of motifs (2000). Then a sigmoid function is applied on the embedding to get the gates. Second, F(S, X) is replaced by taking maximum over non-overlapping window with size 50 followed by GRU [39], a recurrent layer, with hidden size 1000. The hidden state of the GRU at the last time step is connected to the output. Third, the number of outputs is equal to the number of TFs in the training set. In other words, the revised model is a CNN-RNN structure with cell line embedding as gates. Such a model keeps

the imputation ability, gives predictions for multiple TFs in a cell line simultaneously and handles long DNA sequences without imposing extra computation needs. During training, the learning rate decay was 0.01 for every 200,000 instances and the best model is selected based on valid set.

Data source for generating datasets

Thanks the authors of [40], we get a copy of the processed union of DHS. Briefly speaking, this union DHS are constructed based on the 122 ENCODE [5] DNaseI-seq data. Each DNase peak was trimmed to 300 bp centered at its summit. Then, the peaks are merged to get the union DHS regions. The final DHS region contains 409303942 bps, approximately 13% of the genome. We downloaded 690 TF ChIP-seq datasets of the ENCODE project [5] from http://hgdownload.cse.ucsc.edu/goldenPath/hg19/encodeDCC/wgEncodeAwgTfbsUniform/. This dataset contained 562 unique TF-cell combinations, with 172 TFs and 91 cell lines. For datasets with GC matched negative instances [15], the data has been downloaded from http://www.beerlab.org/gkmsvm/downloads/sequences_gkmsvm.tar.gz.

Model training for TFImpute

To train the model based on random sequences, the odd-numbered peaks in the top 1,000 peaks for each dataset were randomly partitioned into a training set (280,000 instances) and a validation set (31,102). The neural network was trained by scanning the training set over 50 epochs, and the performance on the validation set after each epoch was recorded. Finally, the best-performing model was applied to the test set.

To train the model based on DNase I hypersensitive sites, the performance of the model was evaluated on ValidSet1 and ValidSet2 for every 50,000 instances. Then, the best-performing models over 10 epochs were taken as the final models. The best model based on ValidSet1 was used on TestSet1 and TestSet2, and the best model based on ValidSet2 was used on TestSet3. When applying TFImpute to estimate the effect of SNPs on TF binding affinity and to estimate the enhancer activity, TrainSet, TestSet1, ValidSet1, TestSet2, and TestSet3 were merged and used for training, and ValidSet2 was used for validation to select the optimal early stop point. Then, the stop point and all the data including

ValidSet2 were used to train the final model. The model was trained on a GPU (GTX970) for 80 hours to finish 10 epochs.

Hyperparameters for TFImpute

For comparison with DeepBind, the input sequence length was 100. The motif length was set to 20, and the number of motifs was set to 600. The window size for the max-pooling layer was 100; thus, only one maximum value was calculated for each motif. The full connection layer size was set to 50. The learning rate decay was 0.03 for each epoch. For the model based on DNase I background, the input sequence length was 300. The motif length was set to 20, and the number of motifs was set to 2,000. The window size for the max-pooling layer was 100; thus, three maximum values were calculated for each motif. For each situation, all the weights were randomly initialized with a Gaussian distribution of mean 0 and standard deviation 0.01. The bias b for the rectification layer was initialized to -4. All other biases were initialized to 0. The learning rate decay was 0.03 for every 50,000 instances.

Procedures of applying PIQ

We compared TFimpute with the latest version of PIQ (http://bitbucket.org/thashim/piq-single/src). We consulted the author of PIQ for the parameters and usage of PIQ. The procedure is as follows. First, the DNase-seq data was obtained from http://hgdownload.cse.ucsc.edu/goldenPath/hg19/encodeDCC/wgEncodeUwDgf, http://hgdownload.cse.ucsc.edu/goldenPath/hg19/encodeDCC/wgEncodeOpenChromDnase, and http://hgdownload.cse.ucsc.edu/goldenPath/hg19/encodeDCC/wgEncodeUwDnase. Replicate DNase-seq data from the same ENCODE participant lab were merged. For the DNase-seq experiments in the same cell line performed by different labs, the one with the maximum number of mappable reads was selected. Second, we used the motif reference table from PIQ (http://piq.csail.mit.edu/data/misc/integration.test.csv) to match the transcription factor name with PWM id in JASPAR database. For transcription factor names without PWM annotations, we combined the PIQ transcription factor calls from http://piq.csail.mit.edu/data/141105-3618f89-hg19k562.calls/141105-3618f89-hg19k562.calls.tar.gz and ENCODE ChIP-seq peaks to follow the routine suggested by the PIQ author to get the most predictive PWM of the ChIP-seq peaks

as in http://piq.csail.mit.edu/data/misc/compare.tar.gz. Last, we used the default parameters to run PIQ for the 69 <TF, PWM, DNase-seq, cell line > combinations whose TF-cell line combinations are in TestSet3, and got the maximum purity score of the positive and negative strand for each region. For genome wide transcription factor binding prediction, we run the same procedure as above, but replaced the background input region with the 300bp windows on chr8 for the TF-cell combinations in TestSet3.

External dataset from Cistrome Data Browser

We queried the Cistrome Data Browser database [34] for human ChIP-seq datasets satisfying the following two conditions: First, the TF and the cell line combinations are not in the TF ChIP-seq datasets from DeepSEA [27], but the TF or the cell line separately should be in the datasets. In this step, both the official name and its aliases for a TF are considered. Second, the number of peaks with good quality (Q-value < = 0.01) is larger than 20,000. The resulted datasets were manually selected to make the TF and cell line name as unique as possible. Finally, 19 ChIP-seq datasets remain.

The calculation of recall rate at certain FDR cutoff

We calculated the threshold for the predicted score based on the test data for a certain FDR cutoff. Then, the recall rate is calculated based on the threshold. We didn't calculate the FDR using training data to make the metric definition consistent for all methods compared in this work including PIQ that does not need a separate training set.

TFImpute package

The TFImpute method was implemented using Theano [41,42]. The software is available at https://bitbucket.org/feeldead/tfimpute.

ACKNOWLEDGMENTS

We thank Professor Xiaole Shirley Liu and Dr. Clifford A. Meyer for their highly valuable discussions and suggestions. We thank Shenglin Mei for the DNase I background data. We also thank the author of PIQ for detailed help about the usage of PIQ.

AUTHOR CONTRIBUTIONS

- **Conceived and designed the experiments:** JF.
- **Performed the experiments:** JF QQ.
- **Analyzed the data:** JF QQ.
- **Contributed reagents/materials/analysis tools:** JF.
- **Wrote the paper:** JF QQ.

REFERENCES

1. Mathelier A, Shi W, Wasserman WW. Identification of altered cis-regulatory elements in human disease. Trends in Genetics. 2015;31: 67–76. pmid:25637093
2. Weinhold N, Jacobsen A, Schultz N, Sander C, Lee W. Genome-wide analysis of noncoding regulatory mutations in cancer. Nat Genet. 2014;46: 1160–1165. pmid:25261935
3. Friedensohn S, Sawarkar R. Cis-regulatory variation: significance in biomedicine and evolution. Cell Tissue Res. 2014;356: 495–505. pmid:24744265
4. Johnson DS, Mortazavi A, Myers RM, Wold B. Genome-wide mapping of in vivo protein-DNA interactions. Science. 2007;316: 1497–1502. pmid:17540862
5. ENCODE Project Consortium. An integrated encyclopedia of DNA elements in the human genome. Nature. 2012;489: 57–74. pmid:22955616
6. Warner JB, Philippakis AA, Jaeger SA, He FS, Lin J, Bulyk ML. Systematic identification of mammalian regulatory motifs' target genes and functions. Nat Methods. 2008;5: 347–353. pmid:18311145
7. Badis G, Chan ET, van Bakel H, Pena-Castillo L, Tillo D, Tsui K, et al. A Library of Yeast Transcription Factor Motifs Reveals a Widespread Function for Rsc3 in Targeting Nucleosome Exclusion at Promoters. Molecular Cell. 2008;32: 878–887. pmid:19111667
8. Stormo GD. Modeling the specificity of protein-DNA interactions. Quant Biol. 2013;1: 115–130. pmid:25045190
9. Weirauch MT, Cote A, Norel R, Annala M, Zhao Y, Riley TR, et al. Evaluation of methods for modeling transcription factor sequence

specificity. Nat Biotech. 2013;31: 126–134.

10. Wang J, Zhuang J, Iyer S, Lin X-Y, Greven MC, Kim B-H, et al. Factorbook.org: a Wiki-based database for transcription factor-binding data generated by the ENCODE consortium. Nucleic Acids Res. 2013;41: D171–176. pmid:23203885
11. Mathelier A, Zhao X, Zhang AW, Parcy F, Worsley-Hunt R, Arenillas DJ, et al. JASPAR 2014: an extensively expanded and updated open-access database of transcription factor binding profiles. Nucleic Acids Res. 2014;42: D142–147. pmid:24194598
12. von Hippel PH. Increased subtlety of transcription factor binding increases complexity of genome regulation. Proc Natl Acad Sci U S A. 2014;111: 17344–17345. pmid:25468983
13. Siggers T, Gordân R. Protein—DNA binding: complexities and multi-protein codes. Nucl Acids Res. 2014;42: 2099–2111. pmid:24243859
14. Afek A, Schipper JL, Horton J, Gordân R, Lukatsky DB. Protein—DNA binding in the absence of specific base-pair recognition. PNAS. 2014;111: 17140–17145. pmid:25313048
15. Ghandi M, Lee D, Mohammad-Noori M, Beer MA. Enhanced regulatory sequence prediction using gapped k-mer features. PLoS Comput Biol. 2014;10: e1003711. pmid:25033408
16. Ghandi M, Mohammad-Noori M, Ghareghani N, Lee D, Garraway L, Beer MA. gkmSVM: an R package for gapped-kmer SVM. Bioinformatics. 2016; btw203.
17. Lee D, Karchin R, Beer MA. Discriminative prediction of mammalian enhancers from DNA sequence. Genome Res. 2011;21: 2167–2180. pmid:21875935
18. Zeng H, Hashimoto T, Kang DD, Gifford DK. GERV: a statistical method for generative evaluation of regulatory variants for transcription factor binding. Bioinformatics. 2015; btv565.
19. Alipanahi B, Delong A, Weirauch MT, Frey BJ. Predicting the sequence specificities of DNA- and RNA-binding proteins by deep learning. Nat Biotech. 2015;33: 831–838.
20. Xie D, Boyle AP, Wu L, Zhai J, Kawli T, Snyder M. Dynamic trans-Acting Factor Colocalization in Human Cells. Cell. 2013;155: 713–724. pmid:24243024

21. Pique-Regi R, Degner JF, Pai AA, Gaffney DJ, Gilad Y, Pritchard JK. Accurate inference of transcription factor binding from DNA sequence and chromatin accessibility data. Genome Res. 2011;21: 447–455. pmid:21106904
22. Sherwood RI, Hashimoto T, O'Donnell CW, Lewis S, Barkal AA, van Hoff JP, et al. Discovery of directional and nondirectional pioneer transcription factors by modeling DNase profile magnitude and shape. Nat Biotech. 2014;32: 171–178.
23. He HH, Meyer CA, Hu SS, Chen M-W, Zang C, Liu Y, et al. Refined DNase-seq protocol and data analysis reveals intrinsic bias in transcription factor footprint identification. Nat Meth. 2014;11: 73–78.
24. Lee D, Gorkin DU, Baker M, Strober BJ, Asoni AL, McCallion AS, et al. A method to predict the impact of regulatory variants from DNA sequence. Nat Genet. 2015;47: 955–961. pmid:26075791
25. LeCun Y, Bengio Y, Hinton G. Deep learning. Nature. 2015;521: 436–444. pmid:26017442
26. Schmidhuber J. Deep Learning in Neural Networks: An Overview. Neural Networks. 2015;61: 85–117. pmid:25462637
27. Zhou J, Troyanskaya OG. Predicting effects of noncoding variants with deep learning-based sequence model. Nat Meth. 2015;advance online publication.
28. Ernst J, Kellis M. Large-scale imputation of epigenomic datasets for systematic annotation of diverse human tissues. Nat Biotech. 2015;33: 364–376.
29. Lecun Y, Bottou L, Bengio Y, Haffner P. Gradient-based learning applied to document recognition. Proceedings of the IEEE. 1998;86: 2278–2324.
30. Bengio Y, Ducharme R, Vincent P, Jauvin C. A Neural Probabilistic Language Model. JOURNAL OF MACHINE LEARNING RESEARCH. 2003;3: 1137–1155.
31. Zeiler MD. ADADELTA: An Adaptive Learning Rate Method. arXiv:12125701 [cs]. 2012; http://arxiv.org/abs/1212.5701
32. Ernst J, Plasterer HL, Simon I, Bar-Joseph Z. Integrating multiple evidence sources to predict transcription factor binding in the human

genome. Genome Res. 2010;20: 526–536. pmid:20219943

33. Whitington T, Perkins AC, Bailey TL. High-throughput chromatin information enables accurate tissue-specific prediction of transcription factor binding sites. Nucleic Acids Res. 2009;37: 14–25. pmid:18988630
34. Mei S, Qin Q, Wu Q, Sun H, Zheng R, Zang C, et al. Cistrome Data Browser: a data portal for ChIP-Seq and chromatin accessibility data in human and mouse. Nucleic Acids Res. 2016;
35. Kheradpour P, Ernst J, Melnikov A, Rogov P, Wang L, Zhang X, et al. Systematic dissection of regulatory motifs in 2000 predicted human enhancers using a massively parallel reporter assay. Genome Res. 2013;23: 800–811. pmid:23512712
36. Cowper-Sal lari R, Zhang X, Wright JB, Bailey SD, Cole MD, Eeckhoute J, et al. Breast cancer risk-associated SNPs modulate the affinity of chromatin for FOXA1 and alter gene expression. Nat Genet. 2012;44: 1191–1198. pmid:23001124
37. Degner JF, Pai AA, Pique-Regi R, Veyrieras J-B, Gaffney DJ, Pickrell JK, et al. DNase I sensitivity QTLs are a major determinant of human expression variation. Nature. 2012;482: 390–394. pmid:22307276
38. Leung MKK, Xiong HY, Lee LJ, Frey BJ. Deep learning of the tissue-regulated splicing code. Bioinformatics. 2014;30: i121–i129. pmid:24931975
39. Cho K, van Merrienboer B, Bahdanau D, Bengio Y. On the Properties of Neural Machine Translation: Encoder-Decoder Approaches. arXiv:14091259 [cs, stat]. 2014; http://arxiv.org/abs/1409.1259
40. Zang C, Wang T, Deng K, Li B, Hu S, Qin Q, et al. High-dimensional genomic data bias correction and data integration using MANCIE. Nat Commun. 2016;7: 11305. pmid:27072482
41. Bergstra J, Breuleux O, Bastien F, Lamblin P, Pascanu R, Desjardins G, et al. Theano: a CPU and GPU Math Expression Compiler. Proceedings of the Python for Scientific Computing Conference (SciPy). Austin, TX; 2010.

42. Bastien F, Lamblin P, Pascanu R, Bergstra J, Goodfellow IJ, Bergeron A, et al. Theano: new features and speed improvements. 2012.

CHAPTER 10

Deep Learning of Orthographic Representations in Baboons

Thomas Hannagan, Johannes C. Ziegler, Ste´phane Dufau, Joe¨l Fagot, Jonathan Grainger

Laboratoire de Psychologie Cognitive, Aix-Marseille University, & CNRS (Centre National de la Recherche Scientifique), Marseille, France

ABSTRACT

What is the origin of our ability to learn orthographic knowledge? We use deep convolutional networks to emulate the primate's ventral visual stream and explore the recent finding that baboons can be trained to discriminate English words from nonwords [1]. The networks were exposed to the exact same sequence of stimuli and reinforcement signals as the baboons in the experiment, and learned to map real visual inputs (pixels) of letter strings onto binary word/nonword responses. We show that the networks› highest levels of representations were indeed sensitive to letter combinations as postulated in our previous research. The model also captured the key empirical findings, such as generalization to

novel words, along with some intriguing inter-individual differences. The present work shows the merits of deep learning networks that can simulate the whole processing chain all the way from the visual input to the response while allowing researchers to analyze the complex representations that emerge during the learning process.

INTRODUCTION

Baboons can learn to classify strings of letters as real English words or not [1]. Generalization performance to novel words [1] and the existence of transposed letter effects in baboons [2]clearly suggested that the monkeys had learned some kind of orthographic information. However, there is still considerable debate about the precise nature of the information that allows monkeys to achieve such remarkable performance [3], [4]. Are they using purely visual information, unordered strings of letters, or higher-level abstract letter combinations? To shed light on these mechanisms, we used state-of-the-art neural networks of invariant object identification that were designed to simulate the primate›s visual ventral stream. The networks were trained with an incremental supervised procedure to perform a lexical decision task on realistic, pixel-based strings of letters using the exact same stimuli and training regime as the baboons in the experiment. We first show that the networks can account for several empirical effects observed in baboons as well as for intriguing inter-individual differences. We then establish that the networks developed ordered letter representations, therefore demonstrating the efficiency of such representations for a purely visually driven system that must learn to tell words and nonwords apart. This study allows us to move one step further towards the goal of separating the “visual” from the “linguistic” in human reading, while setting a new standard for more realistic computational models of visual word recognition and learning.

Because reading is acquired late in development compared to speech, visual words must map onto pre-existing linguistic knowledge [5]. Not surprisingly, theories of reading have assumed an important role of phonology and semantics in orthographic development [6]. The key finding of the Grainger et al. [1] experiment is that orthography, that is, information about letter identity and letter order, can be acquired in the total absence of prior linguistic knowledge in the primate (because

neither sounds nor meanings were associated with the letter strings processed by the baboons). How baboons actually performed the task is not known. Current computational models of orthographic coding in human adults essentially involve either letter-based schemes, whereby letters are being assigned positions in the word in more or less flexible ways, or schemes based on letter combinations, whereby, for instance, the word "LIFE" would be represented as the set of bigrams LI, LF, LE, IF, IE, FE (see [7], [8] for reviews). A consensus has not yet emerged within cognitive psychology as to which scheme captures the data in the most parsimonious way. However cognitive neuroscience has provided direct evidence [9], [10]that letter combinations, so-called n-grams, are processed in the Visual Word Form Area - VWFA hereafter, a brain region that specifically responds to orthographic information in humans [11]. Given the solid and long standing evidence that primates represent visual objects by means of combinations of features [12], [13], it appears plausible that baboons might use letter combinations or ngrams to represent written words.

Deep learning by convolutional networks

In contrast to our claim that baboons use letter combinations or ngrams to perform word/nonword classifications, several authors have now suggested that baboons in the Grainger et al. experiment used exclusively letter-based schemes. For instance, Frost and Keuleers [4] have argued, without a computational proof, that the task could have been performed simply based on the detection of unordered sets of letters in the stimuli. Others have implemented offline classification systems that can learn to classify the letter strings based on letter positions alone, without invoking any type of ngrams [3]. There is often a host of possible explanations to any given phenomenon, and orthographic coding in baboons is no exception. But none of the explanations described above, when they are actually implemented, brings us any closer to understanding how baboons actually solve the task. This is because they all assume either unlimited computational power, or perfect memory resources, or the availability of all training stimuli at once. In addition, these explanations do not attempt to capture the full range of behavior that has now been documented, but instead, focus on one aspect of the data at best. What

is needed is a comprehensive computational account consistent with resource-limited brains, with what we know of the primate›s visual system, and with systems being subjected to real visual stimuli in an incremental supervised training procedure. Deep convolutional networks possess many if not all of these properties, and are therefore natural candidates for our purpose.

Figure 1 illustrates our general approach. The top left panel (1A) describes the operant-conditioning experiment simulated in this article, in which baboons were shown strings of letters and were rewarded when they classified them accurately as words or nonwords. The top right panel (1B) shows the main pathway presumed to be involved in the baboon›s brain when solving this task [14], constituted by a succession of processing stages within the ventral visual stream from V1 to TEO (posterior inferior temporal cortex) and from there to the orbito-frontal cortex. The lower panel (1C) illustrates the convolutional network architecture of the model.

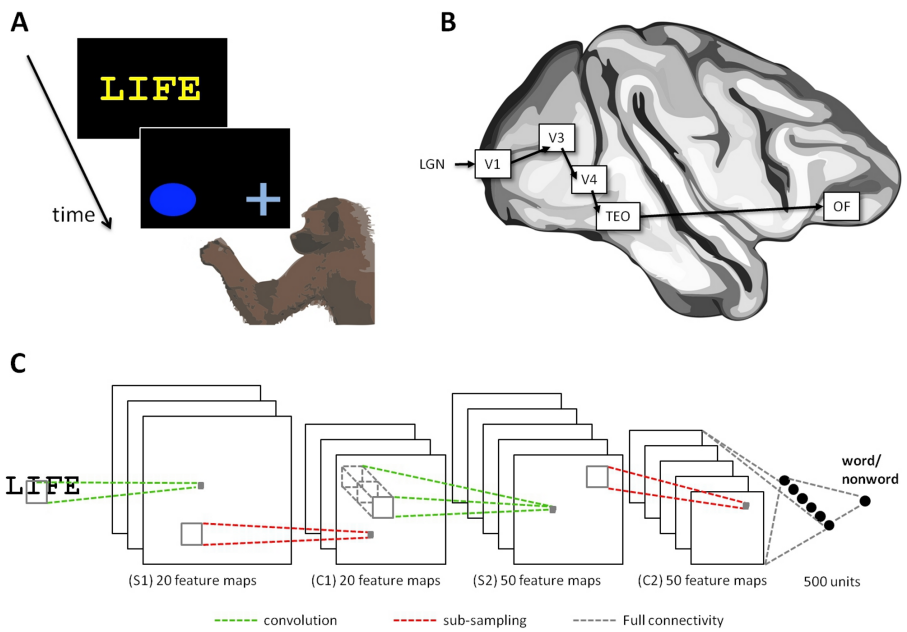


Figure 1: (A) Word-Nonword classification by operant conditioning in baboons [1].

Six baboons were trained to distinguish English words (such as "LIFE") from nonwords (such as "LEFE"), by touching either the cross or the oval shape presented immediately after the word or nonword. A correct response was followed by a blank screen and a food reward, whereas an incorrect response prompted a green screen to appear for 3s. (B) Main cortical network presumed to be involved in distinguishing between words and nonwords, formed by a succession of increasingly selective regions throughout the ventral stream (LGN-V1-V3-V4-TEO) towards the orbito-frontal cortex (OF). (C) Deep learning convolutional network implementing the local and hierarchical connectivity found in the ventral pathway, through a succession of alternating simple convolution and complex pooling levels (cortical regions not mapped to network levels).

Convolutional networks were initially inspired by the primate›s visual system [15], [16]. This is visible in the hierarchical organization of the network Figure 1C, in the restricted connectivity and local sampling of units that captures the increasingly large receptive fields observed from V1 to TEO [17], and in the alternation of convolutional and sampling layers, which mimic the simple and complex cells found in the primary visual cortex. These characteristics have met tremendous success in computer vision and machine learning, and it is fair to say that today convolutional networks display state-of-the-art performance [18], including on tasks such as handwritten character recognition [16], and scene labeling [19]. Closer to our work, the HMAX model, which has been presented in computational neuroscience as the "standard model" of the visual system [20], is very similar in design to convolutional networks.

The model

The model is a convolutional network with a total of 7 layers, a simplified version of the LeNet5 network [16] built using the Theano deep learning libraries [21]. The first layer receives input images (150×30 pixel units) and the output layer produces a binary decision: Word or Nonword. In between, the S1,C1,S2 and C2 layers implement the succession of convolutional (hereafter S, for simple) and sampling (hereafter C, for complex) stages discussed above.

A convolutional layer consists of a number of feature maps (here, 20 for S1 and 50 for S2). Each unit in each feature map is connected to a restricted patch of units lying directly underneath it in all the maps forming the previous layer (for S1, there is only one map in the preceding layer: the input). Although units in the same feature map are connected to different patches of the previous layer, they all share the same connection weight values, according to the principle that what is learned at one location must hold for all locations (i.e. the principle of "weight sharing").

Contrary to convolutional layers, sampling layers reduce the dimensionality of the preceding layer. In a sampling layer, each unit in each feature map represents a larger patch from the preceding layer (in our model, patches of dimensions 2×2 units were used), and together all patches form a partition of the sampled layer. The activation of a sampling unit is obtained in the model by a max pooling operator, which outputs the maximally activated unit within the patch considered. After two series of convolution and sampling, the C2 layer is eventually used as input to a standard multilayer perceptron, which has full connectivity from C2 to the hidden layer, and from the hidden layer to the output layer.

Because baboons in [1] were reinforced on each trial during training, an incremental supervised learning procedure was also needed for the networks and we used a stochastic gradient descent algorithm, with the error gradient being evaluated on each trial and a fixed learning rate used at all epochs (annealing of the learning rate may be introduced in future work but was not found necessary to reproduce the baboons learning curves). The error signal in the case of a binary word/nonword classification task like ours amounts to providing always the same reward in case of success (error of 0) and no reward in case of failure (error of 1), which was precisely the training by reinforcement that the baboons had received. In the experiment, new words were introduced one at a time and interleaved with old words and with nonwords. They were shown repeatedly until the baboon succeeded in categorizing a new word more than 80% of the time, at which point the exemplar became a known word and a new word was introduced. Likewise, each network was exposed to exactly the same sequence of word and nonword stimuli as presented to

the corresponding baboon. For instance, the "DAN network" (simulating the baboon of the same name) was exposed to 56 689 strings, half being words and half nonwords, with these categories being balanced for positional letter frequencies but differenciated by positional bigram frequency distributions. It should be noted, however, that the six baboons had received plenty of visual experience before the experiment, whereas our networks started out as a blank slate, with random connection weights. We will return to this limit of our model in the Discussion. For this study, assessment of classification accuracy for both baboons and networks was done every 1000 trials during training.

Six networks were instantiated, one for each of the six baboons (DAN, ART, CAU, DOR, VIO, and ARI) trained in the original study [1]. All networks had two free parameters: learning rate and convolution filter size. We launched a brute force exploration of this parameter space on six possible learning rates (from 10^{-1} to 10^{-6}) and eight possible filter sizes (from 3 to 10 units). Recognition accuracy was collected every 1000 training trials, compared to the corresponding baboon performance, and the parameter values that minimized this distance for each monkey/model pair were selected. Therefore, the models were not fitted to display any generalization, transposition, or letter similarity effect, but only to match the global performance of the baboons during training.

RESULTS

Figure 2A shows the evolution of performance across the first 40 000 trials, averaged over all fitted networks (black crosses) or baboons (gray filled circles). The main feature of a noisy overall increase in correct responses is captured, and a simple similarity index between model and data based on the absolute difference in performance every 1000 trials approaches the maximal value of 1. By the end of training (not shown in Figure 2A), performance levels were similar though slightly higher for networks (mean accuracy 80.9%) than for baboons (mean accuracy 73.6%).

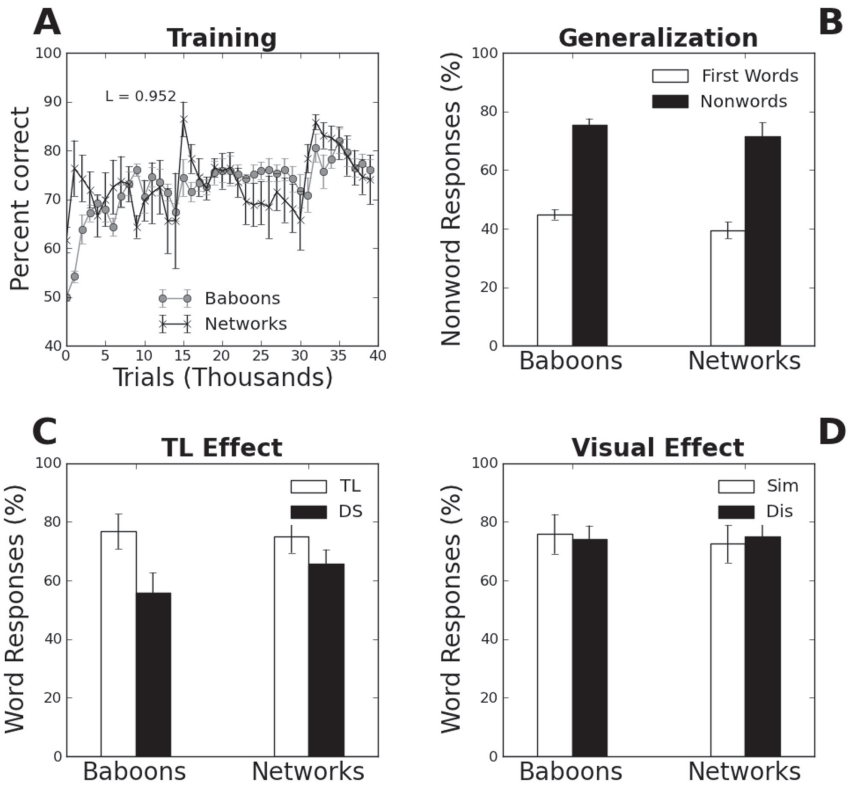


Figure 2: **(A) Learning curves in the fitted model, showing the evolution in performance across** traning trials in the networks (black crosses) and the baboons (gray dots).

Only the first 40 000 trials are shown. (B) Generalization effect: percentage of nonword responses for baboons and networks upon presentation of a new word (white) during training, as opposed to the next nonword (black) in the sequence (experimental data from [1]). (C) Transposition effect: percentage of word responses produced in the networks and in the baboons upon presentation of nonword stimuli that differed from known words by two transposed letters (TL, white bars) or two letter substitutions (DS, black bars) (experimental data from [2]). (D) Visual similarity effect: word responses made on nonword stimuli that were visually similar (Sim, gray bars) or dissimilar (Dis, white bars) to known words (experimental data from [2]).

Figure 2B reports on generalization effects in the networks, that is, the finding that baboons were more likely to classify a real word seen for the first time as a word than a nonword. It can be seen that networks slightly overestimate baboons› performance, but that both are in very close agreement, with much less nonword responses being made upon encountering for the first time a new word (mean baboons 44.8%, mean networks 39.5%) than the next nonword in the sequence (mean baboons 75.6%, mean networks 71.4%). A one-way analysis of variance (ANOVA) on nonword responses for new word stimuli versus nonword stimuli yielded significant differences both for baboons and the model (both ps<.0001). Although a strong similarity in accuracy during training was expected from (indeed the purpose of) our parameter fitting phase, we note that there was no a priori reason why the networks should have displayed quantitatively similar generalization behavior as the baboons, as networks were not fitted to do so.

We then tested whether the networks displayed letter transposition or letter similarity effects, using the exact same stimuli and conditions as in [2]. Letter transposition effects reflect the finding that transposed-letter (TL) nonwords, such as “caniso”, are more likely to be misclassified as words than controls like “caviro” [22]. 80 nonwords were constructed from four-letter English words for each monkey: 20 were obtained by substituting two letters (condition DS, e.g. LGAE), 20 by transposing two letters (condition TL, e.g. LFIE), 20 by substituting a single letter by another that was visually dissimilar (condition Dis, e.g. LOFE), and 20 by substituting a single letter by a visually similar one (condition Sim, e.g. LJFE). Figure 2C shows the results of our TL simulations, along with the experimental data. The baboons showed a significant TL effect, making more word responses in the TL condition than in the DS but no Visual similarity effect ($F(1,5)=5.2$, $p=0.046$). The networks also made on average more word responses in the TL than in the DS condition, although this did not reach significance ($F(1,5)=1.56$, $p=0.24$). Figure 3D reports on the Visual similarity effect: the results show that an equal number of word responses in the Sim and Dis conditions both by baboons ($F(1,5)=0.04$, $p=0.84$) and by the networks ($F(1,5)=0.09$, $p=0.77$). Table 1 presents the individual parameters of the networks, their performance on accuracy, generalization, transposition effect, and visual similarity effects, along with the corresponding baboon performance.

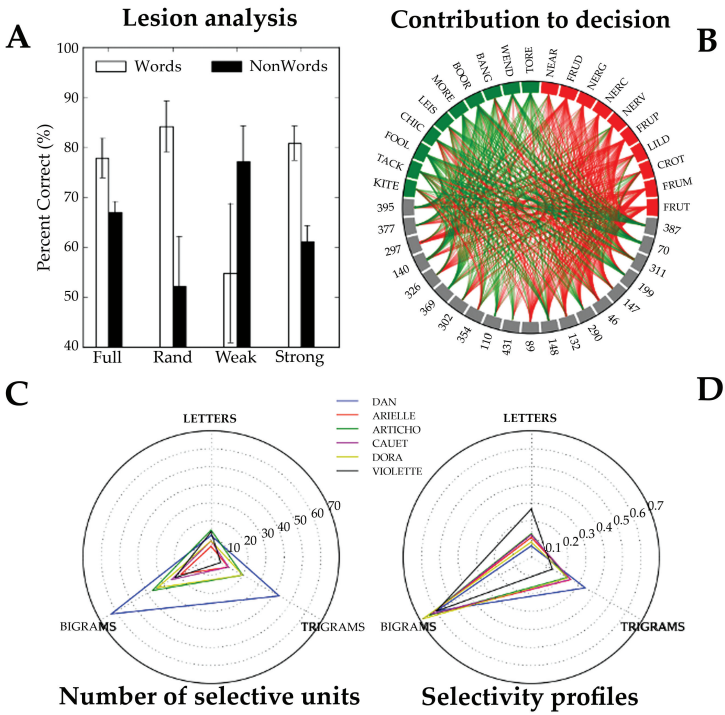


Figure 3: (A) Performance in the normal (Full) and lesioned model, when the 20% intact connections are either chosen at random (Rand), or to be the weakest (Weak) or as the strongest (Strong) connections.

(B) Contribution of extremal network units to the lexical decision. Word stimuli occupy the upper left half of the circle, nonwords the upper right half, and the "extremal" units from the last-but-one layer occupy the lower half. A green connection (resp. red) between a unit and a stimulus signifies that this unit contributes a word signal (resp. nonword) to the decision node upon presentation of this stimulus —only 20 units out of 100 are shown. (C) Number of extremal units selective for at least one letter, bigram or trigram, according to a d-prime statistics. A unit was deemed selective for an entity (letter, bigram or trigram) if upon presentation of a word containing or not this entity, the d-prime defined as the difference of z-values for hit rate and false alarm rate reached an arbitrary common threshold. (D) Global proportion of letters, bigrams and trigrams coded for across all extremal units.

Table 1: Summary of networks parameters and performance

	Train. Base		Network Parameters		Network Selectivity			Network Performance				Baboon Performance			
	Trials	*Voc.*	*CFilter*	*LRate*	*Let.*	*Big.*	*Tri.*	*Acc.*	*Gen.*	*TL*	*VS*	*Acc.*	*Gen.*	*TL*	*VS*
DAN	56 689	308	9	1E-4	12	63	43	91.5	57.09	5	5	79.81	33.7	35	−25
ARIELLE	55 407	87	3	1E-5	6	19	11	68.4	9.20	−10	−4.74	71.14	29.7	30	20
ARTICHO	61 142	112	3	1E-4	15	37	20	92	40.98	20	20	73.41	26.4	−5	−15
CAUET	49 608	121	6	1E-5	9	25	11	70.4	15.18	30	−10	72.43	27.0	0	20
DORA	43 041	81	3	1E-4	9	34	20	82.1	35.54	10	−10	73.15	29.5	40	20
VIOLETTE	50 985	125	3	1E-4	14	23	6	80.8	33.33	0	15	71.55	38.0	25	−10

Trials: number of trials. Voc: number of words in the training base. Conv. Filter: convolutional filter size. Learn. rate: learning rate. Let, Big and Tri: number of units respectively selective to letters, bigrams and trigrams in the penultimate layer. Acc: accuracy. Gen: generalization effect. TL: transposed letter effect. VS: visual similarity effect.

doi:10.1371/journal.pone.0084843.t001

ANALYSIS

The fact that deep learning convolutional networks can reproduce several of our key experimental findings, without being fitted to do so, forces us to ask how they achieved this feat. There are a number of properties over and above a deep architecture and supervised learning that might explain the success of these networks in capturing the results —namely the use of weight sharing, feature maps, and interleaved convolutional/pooling stages involving in particular the Max operator. Some of these assumptions are not beyond criticism from the point of view of neuroscience. For instance, doubts have been raised about the classification of cells into simple and complex types [23] —a classification which inspired the convolutional and pooling stages— and even assuming such a classification, there is no evidence for layers of simple and complex cells being interleaved throughout the ventral stream [24], [25]. Other assumptions of convolutional networks appear to be better motivated. For example, Léveillé and Hannagan [26] recently provided a formal analysis of how connection weights suitable for a max operation could arise in networks trained under a Hebbian learning rule with a temporal window. Yet other assumptions, such as weight sharing, are intriguing and would deserve empirical investigation. But regardless of how well-established or plausible they are however, all of these network characteristics fall short of describing the exact learned mechanism or coding scheme whereby the model succeeds in solving the task. Hence a dedicated analysis is called for.

In recent years, the representations formed by deep learning networks have been analyzed in a number of ways. Erhan, Courville,

and Bengio [27] proposed, among other strategies, to circumscribe the function of any network unit by launching a systematic exploration of input space in search of the patterns that would maximize the unit's activation. Stoianov and Zorzi [28] were able to determine the selectivity of network units by regressing their activations against some continuous properties of the stimulus, in their case numerosity and cumulative surface area. In what follows we choose a different approach. We contend that a key step towards understanding the model can be taken by partly destroying it. In neuropsychology and computational modeling, lesion studies have provided important insights into cognition [29]–[31], and accordingly we start by asking how the model performs when deprived of 80% of its connections in the penultimate layer — that is, the connections coming from the 500 units that entirely and directly subserve lexical decision.

Figure 3A shows the results of our simulated lesion study. We first tested each trained network's performance on all the words in its training base, and on an equal number of nonwords (condition "Full"). Intact networks reached performance levels that were quantitatively very close to observed accuracies for baboons, both for words and for nonwords (Network mean word accuracy 77.9%, Network mean nonword accuracy 67.0%; Baboons mean word accuracy 76.6%, Baboons mean nonword accuracy 70.0%). We then considered three lesioning conditions, "Rand", "Weak", and "Strong", which had in common the removal of 400 connections and only differed in the 100 being left intact. It can be seen that a lesioned model with all but its strongest connections lesioned (condition Strong) retained essentially the same performance on the training base as an intact model. If anything, performance was slightly improved for words and diminished for nonwords (mean word accuracy 80.8%, mean nonword accuracy 61.2%). When all but 100 randomly selected connections were lesioned (condition Rand), network performance on both words and nonwords was impaired (mean word accuracy 84.1%, mean nonword accuracy 52.2%). Finally when all but the 100 weakest connections were lesioned, performance on words approached chance level whereas performance on nonwords increased (condition Weak, mean word accuracy 54.9%, mean nonword accuracy 77.1%). This analysis suggests that the knowledge used by the model in order to tell whether a stimulus is a word or not is essentially embodied in the 100 strongest connections. Therefore, in order to characterize this

knowledge, it is sufficient to restrict the analysis to these 100 "extremal" units that correspond to the strongest connections to the output node. The fact that networks restricted to extremal units can still operate very well is less surprising than it may seem at first blush, being in line with at least one deep learning study, which has found that adequate selection of features in the penultimate stage could actually improve network performance on a task of generic visual object classification [32].

We analyze the connectivity pattern of these extremal units in Figure 3B, where extremal units all lie in the lower half of the circle, word stimuli are in the upper left half, and nonword stimuli in the upper right half. The color of the connection between a given unit and stimulus shows the kind of signal sent by this unit to the decision output upon presentation of the stimulus: a green connection for a positive word signal, and a red connection for a negative word signal. The one fact that immediately stands out in Figure 3B is that word and nonword signals coming from this sample of extremal units are very well matched to word and nonword stimuli. The fact that extremal signals are well matched to lexical status explains why these units are so important in decision making.

Next we assessed the selectivity of extremal units. In this analysis, we exposed each network successively to all the words it had been exposed to and asked whether the observed activity patterns in extremal units showed any regularity with the presence of letters, bigrams, and trigrams in the stimuli at specific positions. We restricted our search to position-specific letters, bigrams, and trigrams, and we adopted two operational definitions for the activity and selectivity of a unit. First, we called a unit *active* if upon presentation of a stimulus, the signal it sent forward to the output layer had a magnitude three standard deviations above the mean across all units and stimuli. Under this count, we found that 15 units were active on average upon presentation of any stimulus, and that a unit was active for on average 22 stimuli. On the other hand, a unit was deemed *selective* for a given entity (letter, bigram, or trigram) if it was mostly active for stimuli that contained this entity and inactive for stimuli that did not contain this entity. We formalized this notion using the d-prime statistic, and granted selectivity to a unit if the difference in Z-scores between hit rate and false alarm rate for the considered entity was superior to an arbitrary and constant threshold (see Methods section for more details).

Figure 3C reports the number of units selective for any letters, bigrams, or trigrams, for each network. It can be seen that every simulated baboon had selective units, with some networks having a majority of their extremal units selective (as in the case of DAN). It is also apparent in this case, because the total sums up to more than 100, that the same same unit can be multiplex, in the sense that it can be selective for several different entities simultaneously. Notably enough, the number of bigram and trigram-selective units in a network did not correlate with the size of the training sets (r_{bi}=0.29; r_{tri}=0.29), but it strongly correlated with the number of words in them (r_{bi}=0.87; r_{tri}=0.84), with network accuracy (r_{bi}=0.78; r_{tri}=0.68), and with generalization performance (r_{bi}=0.88; r_{tri}=0.79). These relations extended to the experimentally observed values in baboons as for accuracy (r_{bi}=0.98; r_{tri}=0.97), but not for generalization (r_{bi}=0.12; r_{tri}=0.03). The number of letter-selective units correlated globally in the same way with these variables, only consistently weaker (number of words, r_{let}=0.3; model accuracy, r_{let}=0.79; baboon accuracy, r_{let}=0.27; model generalization r_{let}=0.69; baboon generalization r_{let}=0.30). Finally Figure 3D provides an overview of selectivity for each baboon, by displaying the probability that a given selective unit will code for a letter, a bigram, or a trigram. Unlike the absolute number of selective units, it can be seen that the portfolio of entities selected for in each model is quite similar.

Individual differences: The case of Dan

In our experimental studies, one baboon, Dan, stood out as having acquired the largest vocabulary of all (308 words, more than twice the vocabulary of any other subject), reaching the best accuracy on words (80.0%) and nonwords (79.6%), and showing massive generalization and strong TL effects (33.7 and 35 percentage points, respectively). This is especially puzzling given that Dan was not the baboon who was trained the most (see Table 1). It is striking in Figure 3C and Table 1 that the Dan network is precisely the one that has developed the largest number of bigram and trigram-selective units (resp. 63 and 43 units out of the 100 considered). It appears that the uniqueness of this network can be traced back to the parameter search phase. Although the search yielded a perfectly unremarkable learning rate of 0.0001 for Dan, the size of the convolutional filter was by far the largest, covering 9 units in radius. The

straightforward consequence of using a wide convolutional filter is that it allows for a much larger context to be integrated at every convolutional stage, thereby explaining the high number of bigram and trigram selective units that are ultimately available for the Dan network to take decisions upon. Indeed convolutional filter size in the model bore no relationship to the final number of letter-selective units (r_{let}=0.03), but correlated strongly with the number of bigram (r_{bi}=0.76) and trigram (r_{tri}=0.75) selective units.

DISCUSSION

This computational study allowed us to propose a general explanation for how baboons can learn to distinguish words from nonwords. Our results suggest that after tens of thousands of training trials the baboons have developed banks of position-specific letter combination detectors. Because words and nonwords in the training base were explicitly designed so as to maximize the difference in bigram frequencies at specific positions, such a strategy explains the baboons' good performance on the training base as well as their ability to generalize to new stimuli, and is well supported by the strong correlation between both of these abilities and the number of bigram selective units. Likewise, our modeling explains why no visual effect was obtained in baboons: if the final decisions were only based on highly selective letter combination detectors in the top layer of the network, then one would expect that the details pertaining to the specific shapes of the letters would have disappeared at this late stage of integration.

Let us now discuss our findings in the light of other recent computational studies of orthographic coding. Two independent analyses on different word recognition networks have now failed to found any support for the emergence of letter combinations, and have instead reported solid evidence for letter-based schemes [33], [34]. Although both networks used artificial letter strings as inputs, the first had a shallow architecture [33], whereas the second used a deep architecture [34]. This suggests that in itself a deep learning design is not sufficient to trigger the emergence of such detectors, and the use of pixel-based visual inputs is unlikely to make a difference in this respect because of one very constraining and critical property shared by these networks: the use of full connectivity between layers. Indeed in both networks, every unit at

a given layer was connected to every unit in adjacents layers, effectively forcing every unit to take all the information from the preceding layer into account. In contrast the sparse connectivity used in convolutional networks and in the primate ventral visual stream frees the network from this constraint, and allows each unit to only analyze a limited patch of information. This encourages the specialization of units on different fragments of the pattern from the previous layer, and these fragment detectors in turn get combined to form more complex fragment detectors at the next convolutional step. According to this view, the presence of two critical factors, sparse connectivity and deep architecture, is required in order to produce letter combination detectors.

It coud also be argued that a position-specific letter combination account of orthographic coding in baboons is not entirely compelling in that it does not a priori predict any TL effect, and indeed the networks only showed a non-significant trend towards one, at odds with the observed data. One way to make sense of this conflict is that we may be dealing with a very volatile effect, whose magnitude would depend on minute factors that are not necessarily captured in the model. According to the model, what ultimately determines the presence or absence of a TL effect is not whether the code is position specific or position invariant, but rather, how many more selective units will send a "word" signal to the output layer in the TL condition relative to the DS one. This would depend, first, on the differences in letter, bigram and trigram frequencies between TL and DS stimuli, and second on the exact selectivity profile of the network's units which is a result of the whole training history. It is possible that we made some implausible simplifications during the training phase, for instance by keeping the learning rate and convolution kernel parameters constant throughout training, or by specifying that the perceived reward was always constant. One obvious implausible simplification of this model is that stimuli were always presented in the same position and size on the simplistic simulated retina, whereas actual fixations made by baboons on the letter strings were bound to have varied in locations. Stochastic presentation went beyond the main point of our model, which was essentially to acknowledge the limited resources and hierarchical organization of the primate's visual system. In addition, and due to their built-in weight sharing mechanism for location invariance, deep convolutional networks may not be the best-suited framework for studying the effects of stochastic stimulus presentation.

Regardless of this, the distribution of words on the retina may affect orthographic representations, and indeed the finding that orthographic representations in the VWFA are location sensitive has lately been a topic of much interest [35], which some of our computational work has aimed at explaining [36].

We have also trained our networks on a lexical decision task without the benefit of any previous visual expertise, whereas baboons were already experts at recognizing all sorts of visual objects by the time they started experimental training. Introducing previous visual training in this model would be very interesting for a number of reasons, not least because it would allow for the study of mirror invariance, a general characteristic of the expert visual system that has been theorized to be lost during the acquisition of reading [37]. Would the deep learning networks develop mirror-invariance for generic visual stimuli, and would they lose it upon exposure to strings of letters? Regardless of what these answers turn out to be, previous visual training would almost surely impact on the type of features used to perform lexical decision, but we would argue that it is unlikely to threaten our main conclusions here. Indeed, a developmental-like, unsupervised training stage has already been introduced in extant work with deep learning networks on visual categorization, and suggests that such training would result in a dictionary of feature combinations [20]. It is hard to see how the presence of feature combinations at the beginning of training on the lexical task could prevent the emergence of letter combination detectors later on, or could prevent these more developmentally realitistic networks to display any of the effects reported in this study.

We may finally speculate on how the penultimate layer, that was the focus of our analysis, relates to the baboon's brain or to the VWFA. The penultimate layer appears late in the processing stream and its units stand out as the only ones having access to all the information across all feature maps and all locations in the preceeding layer. For that reason, it is perhaps more plausibly related to a region lying between TEO and the orbital frontal cortex. Alternatively, because clear evidence for letter, bigram and quadrigram representations has been found in the VWFA [10], the penultimate layer could be a simplified, structureless homologue of the VWFA itself, whose detailed connectivity is not yet known. In this case, assuming that the same causes produce the same effects in a neural

hardware shared across baboons and humans, the model's prediction would be that a region analogue to the VWFA should be observed in baboons that would only be activated for words and legal nonwords but not for false-font letter strings or for illegal, random strings of letters.

CONCLUSION

We have used deep learning networks to uncover the nature of the orthographic code which allows baboons to perform word-nonword classifications at a high level of accuracy. To do so, we have applied a range of techniques inspired from neuroscience, especially lesion studies and single cell recordings. The understanding of the model's inner workings gained hereby provides new insights into the nature of the orthographic code in humans. Under the dual pressure of visual processing constraints shared by all primates, and of an incremental training procedure, the model has converged towards a code that uses an assortment of position-specific letter combination detectors in order to categorize letter strings. This is in line with a large but still controversial class of theoretical models of visual word recognition proposed over the last decade [8], [13], [38]–[40], and weighs in favor of similar letter combinations being involved in the human faculty of reading.

METHODS

No new data was collected for this study, this being a computational modeling article that aims at explaining data already published [1], [2]. The previous experiments on baboons had the following characteristics:

- Name of the Institutional Animal Care and Use Committee (IACUC) or ethics committee that approved this study: "comité d'éthique de Marseille pour l'expérimentation animale".
- Details of animal welfare: Housing conditions; 650 m^2 outdoor enclosure with various climbing structures connected by tunnels to an indoor enclosure.
- Feeding regimens: Baboons were not food deprived, and received daily food ratio (monkey chow, fruits, vegetables) at 5 pm, water was provided ad libitum.
- Environmental enrichment: Climbing structures, live in social

group. Use of the ALDM ("Automated Learning Device for Monkeys") is a form of behavioral enrichment.

- Steps taken to alleviate suffering: no suffering.
- Methods of sacrifice: none.

Summary of the experiments

Here, we provide a summary of the two experiments modeled in this paper, and we refer the interested reader to the original studies for more details about the material and procedure [1], [2] Six guinea baboons (Papio papio; 3 females) living in semi-freedom participated in the original word learning experiment of [1]. Baboons were free to access and leave the automated experimental booths whenever they wanted. Self-initiated trials began with the presentation of a stimulus (word or nonword) until the baboon touched the screen, at which point the screen displayed a dark blue cross on the left (correct nonword response) and a light blue oval shape on the right (correct word response). Response was followed by a food reward (dry wheat) when a correct word or nonword response had been given, or followed by a 3 sec delay with a green screen if the response was incorrect. Words and nonwords were four letter strings designed so as to differ in bigram frequencies. Each baboon saw an equal number of words and nonwords, but because of the self-initiated nature of the trials, their total number differed across baboons (DAN: 56 689, ART: 50 985, CAU: 61 142, DOR: 49 608, VIO: 43 041, ARI: 55 407).

Immediately after training, baboons were tested for effects of visual similarity and transposition [2]. The test items were constructed individually for each monkey on the basis of the words that he or she had learned (e.g. LIFE), by substituting two letters (condition DS, e.g. LGAE), transposing two letters (condition TL, e.g. LFIE), by substituting a single letter with another that was visually dissimilar (condition Dis, e.g. LOFE), and by substituting a single letter with a visually similar one (condition Sim, e.g. LJFE). All of these items were reinforced as nonword trials.

Statistical analysis of the models

For each model, accuracy on word/nonword discrimination and generalization performance were calculated as described in [1], with

the only difference being that for practical reasons in our paper these calculations were done every 1000 trials, both for networks and for baboons. Generalization scores were computed as the proportion of nonword responses made for first words minus the proportion of nonword responses made for nonwords. Transposed-letter effects were calculated as the number of word responses produced by TL nonwords minus the number of word responses produced by double substitution nonwords. Visual similarity effects were obtained as the number of word responses made on nonword stimuli that were visually similar to a word, minus the number of word responses made on nonwords that were visually dissimilar to known words. Following Ziegler et al. [2], the statistical analysis performed across models was a standard one-way analysis of variance.

Lesion analyses

Networks were lesioned after training was complete, by removing 400 connections out of the 500 arriving at the output node. The 100 connections left intact were either chosen to be the strongest (condition "strong"), the weakest (condition "weak"), or random (condition "random"). Strong and weak connections were defined by the absolute value of their weights: the weight of a strong connection had a large absolute value whereas the weight of a weak connection had a small absolute value (i.e. close to zero).

Sparseness, activity, and selectivity analyses

For each model, we first collected the activation patterns in all 100 extremal units in the last-but-one layer, upon presentation of all the words in the dataset, and of an equal number of nonwords. Because datasets differed across baboons but were exactly identical for a baboon and for its model, the number N of pairs of word/nonword presented varied across models (N_{DAN} = 308; N_{ART} = 125; N_{CAU} = 112; N_{DOR} = 121; N_{VIO} = 81; N_{ARI} = 87). We then computed the mean activation level and standard deviation of all activities across all extremal units and all patterns, and thresholded activation patterns to three standard deviations above the mean - an arbitrary threshold for what counts as an active unit in the model. The resulting Nx100 binary matrix of words-by-units activity served as a base for the computation of sparseness and selectivity in each

model (see Figures 3B, 3C and 3D).

Selectivity of a unit to a particular entity (letter, bigram or trigram) was quantified by the d-prime statistic, a signal detection procedure based on the calculation of two quantities: the hit rate (HR), defined as the number of times the unit was active upon presentation of this specific entity divided by the total number of times the entity had been seen, and the false alarm rate (FAR), defined as the number of times the unit was active despite the absence of the entity in the stimulus, divided by this number plus the number of times the unit was not active in the absence of the stimulus (correct rejection). The d-prime was then simply obtained as the difference of z-scores for HR and for FAR, and a unit was deemed selective for an entity if its d-prime was above a common arbitrary value of 2.

All networks were built, trained, and tested using the Theano library [21] running on the Enthought Python Distribution.

ACKNOWLEDGMENTS

The first author would like to thank Anne Warlaumont for stimulating discussions.

AUTHOR CONTRIBUTIONS

Conceived and designed the experiments: TH JZ JG. Performed the experiments: TH. Analyzed the data: TH JZ JF JG. Contributed reagents/materials/analysis tools: TH. Wrote the paper: TH JZ JF SD JG.

REFERENCES

1. Grainger J, Dufau S, Montant M, Ziegler JC, Fagot J (2012) Orthographic Processing in Baboons (Papio papio). Science 336: , 245–248.
2. Ziegler JC, Hannagan T, Dufau S, Montant M, Fagot J, et al.. (2013) Transposed Letter Effects Reveal Orthographic Processing in Baboons Psychological Science 24(8): , 1609–1611.
3. Bains W (2012) Comment on “Orthographic Processing in Baboons (Papio papio)”. Science 337: , 1173. Available: www.sciencemag.

org/cgi/content/full/337/6099/1173-b.

4. Frost R, Keuleers E (2013) What can we learn from monkeys about orthographic processing in humans? A reply to Ziegler et al. Psychological Science 24(9): , 1868–1869.
5. Ziegler JC, Goswami U (2005) Reading acquisition, developmental dyslexia, and skilled reading across languages: a psycholinguistic grain size theory. Psychological Bulletin 131(1): , 3–29.
6. Ziegler JC, Perry C, Zorzi M (in press) Modelling reading development through phonological decoding and self-teaching: Implications for dyslexia. Philosophical Transactions of the Royal Society B.
7. Grainger J (2008) Cracking the orthographic code: An introduction. Language and Cognitive Processes 23(1): , 1–35.
8. Hannagan T, Grainger J (2012) Protein Analysis Meets Visual Word Recognition: A Case for String Kernels in the Brain. Cognitive Science 36: , 575–606.
9. Binder JR, Medler DA, Westbury CF, Liebenthal E, Buchanan L (2006) Tuning of the human left fusiform gyrus to sublexical orthographic structure. Neuroimage 33: , 739–748.
10. Vinckier F, Dehaene S, Jobert A, Dubus JP, Sigman M, et al. (2007) Hierarchical coding of letter strings in the ventral stream: dissecting the inner organization of the visual word-form system. Neuron 55(1): 143–156.
11. Cohen L, Dehaene S, Naccache L, Lehéricy S, Dehaene-Lambertz G, et al. (2000) The visual word form area: Spatial and temporal characterization of an initial stage of reading in normal subjects and posterior split-brain patients. Brain 123: 291–307.
12. Brincat SL, Connor CE (2006) Dynamic Shape Synthesis in Posterior Inferotemporal Cortex. Neuron 49(1): , 17–24.
13. Dehaene S, Cohen L, Sigman M, Vinckier F (2005) The neural code for written words: a proposal. Trends Cogn Sci 9: 335–341.
14. Blaizot X, Landeau B, Baron JC, Chavoix C (2000) Mapping the visual recognition memory network with PET in the behaving baboon. Journal of Cerebral Blood Flow and Metabolism 20: , 213–219.

15. Fukushima K (1980) Neocognitron: A self-organizing neural network for a mechanism of pattern recognition unaffected by shift in position. Biological Cybernetics 36(4): , 193–202.
16. LeCun Y, Bottou L, Bengio Y, Haffner P (1998) Gradient-based learning applied to document recognition. Proceedings of the IEEE 86(11): , 2278–2324.
17. Van Essen DC, Gallant JL (1994) Neural mechanisms of form and motion processing in the primate visual system. Neuron 13: 1–10.
18. Ciresan DC, Meier U, Schmidhuber J (2012) Multi-column Deep Neural Networks for Image Classification. IEEE Conf. on Computer Vision and Pattern Recognition.
19. Farabet C, Couprie C, Najman L, LeCun Y (2013) Learning hierarchical features for scene labeling. IEEE Transactions on Pattern Analysis and Machine Intelligence.
20. Serre T, Oliva A, Poggio T (2007) A Feedforward Architecture Accounts for Rapid Categorization. Proc Natl Acad Sci USA 104(15): 6424–6429.
21. Bergstra J, Bastien F, Breuleux O, Lamblin P, Pascanu R, Delalleau O, Desjardins G, Warde-Farley D, Goodfellow I, Bergeron A, Bengio Y (2011) Theano: Deep learning on GPUs with python. In NIPS 2011, BigLearning Workshop, Granada, Spain.
22. Perea M, Lupker SJ (2004) Can CANISO activate CASINO? Transposed-letter similarity effects with nonadjacent letter positions. Journal of Memory and Language 51: , 231–246.
23. Mechler F, Ringach DL (2002) On the classification of simple and complex cells. Vision research 42: , 1017–1033.
24. Hubel DH, Wiesel TN (1968) Receptive fields and functional architecture of monkey striate cortex. Journal of Physiology-London 195: , 215–243.
25. Skottun BC, De Valois RL, Grosof DH, Movshon JA, Albrecht DG, et al.. (1991) Classifying simple and complex cells on the basis of response modulation. Vision Research 31: , 1079–1086.
26. Léveillé J, Hannagan T (2012) Learning spatial invariance with the trace rule in non-uniform distributions. Neural Computation 25(5): , 1261–1276.

27. Erhan D, Courville A, Bengio Y (2010) Understanding Representations Learned in Deep Architectures. Technical Report 1355, Université de Montréal.
28. Stoianov I, Zorzi M (2012) Emergence of a "visual number sense" in hierarchical generative models. Nature neuroscience 15(2): 194–6.
29. Plaut DC, Shallice T (1993) Deep dyslexia: A case study of connectionist neuropsychology. Cognitive Neuropsychology 10: , 377–500.
30. Cohen G, Johnston RA, Plunkett K (2000) Exploring cognition: Damaged brains and neural networks, readings in cognitive neuropsychology and connectionist modelling. Hove, UK: Psychology Press.
31. Thomas MSC, Purser HRM, Tomlinson S, Mareschal D (2011) Are imaging and lesioning convergent methods for assessing functional specialisation? Investigations using an artificial neural network. Brain and Cognition 78(1): 38–49.
32. Mutch J (2008) Lowe DG (2008) Object class recognition and localization using sparse features with limited receptive fields. International Journal of Computer Vision 80(1): 45–57.
33. Hannagan T, Dandurand F, Grainger J (2011) Broken symmetries in a location-invariant word recognition network. Neural Comput 23: 251–283.
34. Di Bono MG, Zorzi M (2013) Deep generative learning of location-invariant visual word recognition. Front. Psychol 4: 635.
35. Rauschecker AM, Bowen RF, Parvizi J, Wandell BA (2012) Position sensitivity in the visual word form area. Proc Natl Acad Sci USA 109(24): 1568–1577.
36. Hannagan T, Grainger J (2013) The Lazy Visual Word Form Area: Computational Insights into Location-Sensitivity. PLoS Comput Biol 9(10): e1003250.
37. Pegado F, Nakamura K, Cohen L, Dehaene S (2011) Breaking the Symmetry: Mirror discrimination for single letters but not for pictures in the Visual Word Form Area. NeuroImage 55: 742–9.
38. Mozer M (1987) Early parallel processing in reading: A connectionist approach. In M. Coltheart (Ed.) Attention and Perfomance XII: The

Psychology of Reading. 83–104. Hillsdale, NJ: Lawrence Erlbaum.

39. Whitney CS, Berndt RS (1999) A new model of letter string encoding: Simulating right neglect dyslexia. Progress in Brain Research 121: 143–163.
40. Grainger J, van Heuven W (2003) Modeling letter position coding in printed word perception. In Bonin P. (Ed.), Mental lexicon: "Some words to talk about words" 1–23. New York: Nova Science.

CHAPTER 11

Large-Scale Transportation Network Congestion Evolution Prediction Using Deep Learning Theory

Xiaolei Ma[1,2], Haiyang Yu[1,2], Yunpeng Wang[1,2], Yinhai Wang[3]

[1] School of Transportation Science and Engineering, Beijing Key Laboratory for Cooperative Vehicle Infrastructure, Systems, and Safety Control, Beihang University, Beijing, China

[2] Jiangsu Province Collaborative Innovation Center of Modern Urban Traffic Technologies, SiPaiLou #2, Nanjing, China

[3] Department of Civil and Environmental Engineering, University of Washington, Seattle, Washington, United States of America

ABSTRACT

Understanding how congestion at one location can cause ripples throughout large-scale transportation network is vital for transportation researchers and practitioners to pinpoint traffic bottlenecks for congestion mitigation. Traditional studies rely on either mathematical equations or simulation techniques to model traffic congestion dynamics. However, most of the approaches have limitations, largely due to unrealistic assumptions and cumbersome parameter calibration process. With the

development of Intelligent Transportation Systems (ITS) and Internet of Things (IoT), transportation data become more and more ubiquitous. This triggers a series of data-driven research to investigate transportation phenomena. Among them, deep learning theory is considered one of the most promising techniques to tackle tremendous high-dimensional data. This study attempts to extend deep learning theory into large-scale transportation network analysis. A deep Restricted Boltzmann Machine and Recurrent Neural Network architecture is utilized to model and predict traffic congestion evolution based on Global Positioning System (GPS) data from taxi. A numerical study in Ningbo, China is conducted to validate the effectiveness and efficiency of the proposed method. Results show that the prediction accuracy can achieve as high as 88% within less than 6 minutes when the model is implemented in a Graphic Processing Unit (GPU)-based parallel computing environment. The predicted congestion evolution patterns can be visualized temporally and spatially through a map-based platform to identify the vulnerable links for proactive congestion mitigation.

INTRODUCTION

Traffic congestion costs billions of dollars in each year due to lost time, wasted fuel, excessive air pollution, and reduced productivity. The 2012 Urban Mobility Report indicates that the annual average delay per person was 38 hours in 2011 for the 498 surveyed urban areas, which is equivalent to a 238% increase compared to that in 1982. Traffic congestion incurred a total of 5.5 billion hours of travel delays and 2.9 billion gallons of extra fuel consumption in 2009, which corresponds to a congestion cost of 121 billion dollars [1].

Diagnosing congestion onset and predicting congestion evolution patterns are considered strategic countermeasures to locate traffic bottlenecks and adopt proactive measures for congestion mitigation. Many research efforts have been made to achieve these goals [2, 3]. However, most of previous studies tend to view congestion spots separately in a small-scale network. As mentioned by Yang[4], the number of network links for almost all the existing traffic congestion prediction methods do not exceed 100. In addition, these studies rely on either mathematical equations or simulation techniques to depict the

network congestion evolution. This often results in suboptimality since transportation activities involve human factors which are difficult to represent or model accurately using mathematics-driven approaches. Previous network-wide congestion studies mainly resort to either complex network theory [5–11] or visualization techniques [12] to understand the evolution of network-wide traffic congestion. In complex network theory, transportation networks can be abstracted as scale-free networks [9], and traffic flow dynamics over the network are generated based on the power law distribution [11] However, these assumptions are not always adherent to the reality, and lack sufficient traffic sensor data to validate their findings. Visualization techniques can intuitively display the spatial and temporal distribution of network congestion through a map-based platform, but are incapable of explaining the mechanism of congestion generation and predicting future trend of congestion evolution.

Over the past decades, tremendous traffic sensors have been deployed on the existing freeway networks, generating a huge amount of data at relatively high time resolutions. The increasing availability of network data makes it possible to simultaneously examine traffic flows on a large-scale roadway network and observe the evolution of congestion on that network through data mining techniques. Transportation network consists of thousands of links with changing traffic condition over time. This is equivalent to a high-dimensional space where congestion prorogates temporally and spatially. This is challenging to model using traditional data mining approaches due to the curse of dimensionality: when the input dimensionality increases, the required training data grow exponentially [13]. The recent emergence of deep learning theory can address the curse of dimensionality issue through distributed representations, and thereby holds great promise in learning high-dimensional features with tremendous data. Compared to those shallow learning architectures, deep learning is able to model complex non-linear phenomenon using distributed and hierarchical feature representation [14, 15], and has received numerous success in the domain of computer vision, speech recognition, natural language processing and music composition [16]. Deep learning theory began to exhibits its superiority of predicting traffic flow over a single road segment [14]. However, to the best of our knowledge, no research has been conducted to apply deep learning theory into large-scale transportation network modeling and

analysis. To assist transportation professionals in congestion diagnosis and operational strategy assessment, this study aims at developing a data-driven and network-wide congestion analysis paradigm based on traffic sensor measurements using deep learning theory.

The remainder of this paper is organized as follows: Section 2 presents a methodological framework to model network-wide traffic congestion, where GPS data are collected to depict the roadway traffic condition and a deep learning architecture Recurrent Neural Network and Restricted Boltzmann Machine (RNN-RBM) is introduced to predict the temporal-spatial congestion evolution pattern. To validate the effectiveness of the proposed algorithm, a numerical experiment with a city roadway network of over 500 links is undertaken in Section 3. The prediction results are demonstrated through a GIS platform to observe the future congestion evolution, followed by a sensitivity analysis with various congestion speed thresholds. In addition, a comparative study with conventional neural network and support vector machine methods is conducted to demonstrate the advantage of the proposed algorithm. Finally, discussion and conclusion are made in the last section.

METHODOLOGY

Traffic condition on each link is classified into a binary state, where 1 represents congested and 0 represents uncongested. The link traffic condition changes over time, and essentially becomes a binary sequence with varying length. Considering the size of a typical transportation network, the network-wide congestion pattern is equivalent to a high-dimensional matrix arranged in time (number of time steps) and space (number of links). The key of network-wide traffic congestion evolution prediction model is to understand how each matrix element evolves spatially and temporally. The initial step is to identify the traffic condition of each link using taxi GPS data. Missing and erroneous data will be reviewed and corrected to ensure satisfying quality. Finally, processed GPS data are aggregated into varying time intervals for representing network-wide traffic patterns. Inspired by polyphonic music generation and transcription approach [17](Boulanger-Lewandowski, 2012), a combination of Restricted Boltzmann Machine (RBM) and Recurrent Neural Network (RNN) is then utilized to learn the high-dimensional

traffic congestion patterns. To fully take advantages of Graphics Processing Unit (GPU)-based parallel computing architecture, the proposed algorithm is implemented in the NVIDIA Compute Unified Device Architecture (CUDA) environment to accelerate the model learning procedure.

Transportation Network Representation

A GPS-equipped taxi can record its location and timestamp information, and the travel speed can be directly measured. For each link n, the average speed at time t is denoted as c_n^t, If the average speed is lower than a threshold (20 km/hour), the link is considered congested, and thereby is set as 1. Due to the sampling error, GPS-equipped taxis may not fully travel along the entire network at any time. For those links without GPS data, vehicle speed is calculated by using the historical records for the same link. Based on the above discussion, traffic congestions for a network with N links within T time intervals can be expressed as:

$$\begin{bmatrix} c_1^1 & c_1^2 & \cdots & c_1^T \\ c_2^1 & c_2^1 & \cdots & c_2^T \\ \vdots & \vdots & \cdots & \vdots \\ c_N^1 & c_N^2 & \cdots & c_N^T \end{bmatrix} \quad (1)$$

Where c_n^t represents the traffic congestion condition on nth link at time t, which is a binary value. The next step is to predict the elements in each row (link). This can be treated as the single sequence learning problem. When multiple rows (links) are predicted simultaneously, this is known as the high-dimensional sequence learning problem. A deep learning architecture with temporal processing capabilities is desired.

RNN-RBM: A deep learning architecture for high-dimensional temporal sequence prediction

Restricted Boltzmann Machine (RBM)

Restricted Boltzmann Machine (RBM) is an energy based model that includes one visible layer and one hidden layer [15]. As demonstrated

in Fig. 1, units in both visible layer and hidden layer are mutually connected.

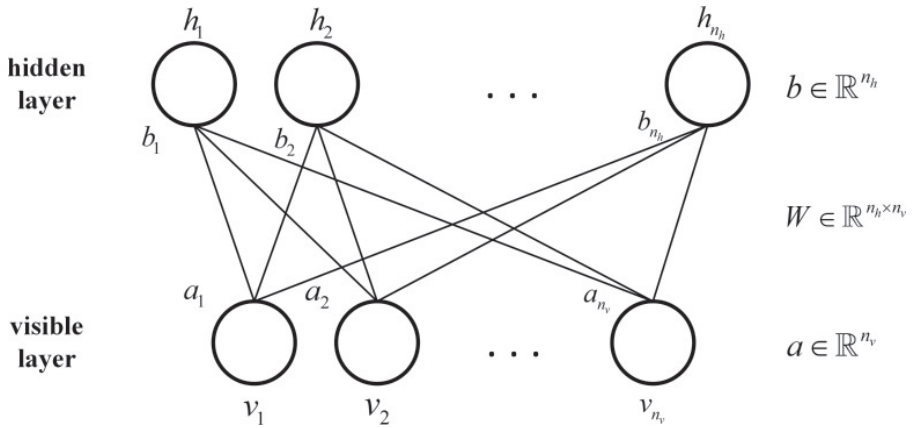


Figure 1: RBM Structure.

In Fig. 1, n_v and n_h respectively represent the number of units in hidden layer and visible layer. $v = (v_1, v_2, \ldots v_{nv})^T$ represents the state vector for visible layer, where v_i is the state of the ith unit in visible layer. $h = (h_1, h_2, \ldots h_{nh})^T$ represents the state vector value for hidden layer, where h_i is the state of the ith unit in hidden layer. $a = (a_1, a_2, \ldots a_{nv})^T$ represents the bias vector for visible layer, where a_i is the bias of the ith unit in visible layer. $b = (b_1, b_2, \ldots b_{nh})^T$r epresents the bias vector for hidden layer, where b_i is the bias of the ith unit in hidden layer. $W = (w_{i,j}) \in \mathbb{R}^{n_h \times n_v}$ represents the weight matrix between visible layer and hidden layer, where $w_{i,j}$ is the connecting weight for the ith unit in visible layer and the jth unit in hidden layer. The units in both hidden and visible layers are binary-valued.

Based on the above notations, the energy function for a given (**v,h**) is defined as:

$$E_\theta(\mathbf{v}, \mathbf{h}) = -\mathbf{a}^T\mathbf{v} - \mathbf{b}^T\mathbf{v} - \mathbf{h}^T W\mathbf{v} \tag{2}$$

The joint probability distribution function can be expressed as:

$$P_\theta(\mathbf{v}, \mathbf{h}) = \frac{1}{Z_\theta} e^{-E_\theta(\mathbf{v},\mathbf{h})} \tag{3}$$

Where

$$\theta = (W, \mathbf{a}, \mathbf{b}), Z_\theta = \sum_{\mathbf{v},\mathbf{h}} e^{-E_\theta(\mathbf{v},\mathbf{h})}$$

is defined as the partition function to normalize the probability function.

The conditional probability that hidden unit h_k is activated given hidden vector $\mathbf{v}$ and the conditional probability that visible unit v_k is activated given visible vector $\mathbf{h}$ can be respectively calculated as:

$$P(h_k = 1|\mathbf{v}) = sigmoid(b_k + \sum_{i=1}^{n_v} w_{k,i} v_i) \quad (4)$$

$$P(v_k = 1|\mathbf{h}) = sigmoid(a_k + \sum_{i=1}^{n_h} w_{k,i} h_i) \quad (5)$$

sigmoid means the sigmoid activation function. Because there is no dependency within visible layer and hidden layer, both conditional probability functions can be written as:

$$P(\mathbf{h}|\mathbf{v}) = \prod_{k=1}^{n_h} P(h_k = 1|\mathbf{v}) \quad (6)$$

$$P(\mathbf{v}|\mathbf{h}) = \prod_{k=1}^{n_v} P(v_k = 1|\mathbf{h}) \quad (7)$$

To find the parameters θ, RBM is required to maximize the probability of training set V:

$$\arg\max_\theta \prod_{v \in V} P(v) \quad (8)$$

This is equal to maximize the log likelihood of $P(\mathrm{v})$. A common optimization method is gradient descent algorithm. The mathematical form can be expressed in (Equation 9), where the derivative of ln $P(v)$ should be calculated as (Equation 10) in advance.

$$\theta = \theta + \eta \frac{\partial lnP(v)}{\partial \theta} \quad (9)$$

$$\frac{\partial lnP(v)}{\partial \theta} = -\langle \frac{\partial E(\mathbf{v},\mathbf{h})}{\partial \theta} \rangle_{P(\mathbf{h}|v)} + \langle \frac{\partial E(\mathbf{v},\mathbf{h})}{\partial \theta} \rangle_{P(\mathbf{v},\mathbf{h})} \quad (10)$$

Where η is known as the learning rate, and it is responsible to control the speed of convergence to find the optimal solution. $\langle\cdot\rangle_P$denotes the expectation value with respect to probability distribution P. Solving (equation 8) is computational intensive, Hilton [18] proposed the famous Contrastive Divergence (CD) approach to train RBM in an efficient fashion. The key of this algorithm is to perform n-step Gibbs sampling for approximating the expectation value of (Equation 10), the equation can be further simplified into the following three equations:

$$\frac{\partial lnP(v)}{\partial \theta} = \begin{cases} \dfrac{\partial lnP(\mathbf{v})}{\partial w_{i,j}} \approx P(h_i = 1|\mathbf{v}^{(0)})v_j^{(0)} - P(h_i = 1|\mathbf{v}^{(n)})v_j^{(n)} \\ \dfrac{\partial lnP(\mathbf{v})}{\partial a_i} \approx v_i^{(0)} - v_i^{(n)} \\ \dfrac{\partial lnP(\mathbf{v})}{\partial b_i} \approx P(h_i = 1|\mathbf{v}^{(0)}) - P(h_i = 1|\mathbf{v}^{(n)}) \end{cases} \tag{11}$$

The probability distribution function for RBM can be modified as a conditional probability distribution function depending on the previous state of RBM [19], and this leads to a variant of RBM model for efficiently modeling complex time series [20].

Recurrent Neural Network (RNN)

Recurrent Neural Network (RNN) is a special form of neural network family, and contains at least one feed-back connection as an internal state from neurons' outputs to their inputs. This loop structure grants the capability of temporal processing and sequence learning for the network. Due to RNN's short-term memory, it is widely used to model non-linear time series data [21]. Fig. 2 presents a State Space Neural Network (SSNN), which a widely used structure of RNN. Delay units are introduced to feed back the previous hidden unit activations with the inputs into the neural network. $x(t)$ and $h(t)$ respectively represent the vectors in input layer as well as delay units at time t. $h(t+l)$ and $y(t+l)$ respectively represent the vectors in hidden layer and output layer at time $t+l$. W_{IH} is the weight matrix from input layer to hidden layer. W_{HH} is the weight matrix from the delay units to hidden layer. W_{HO} is the weight matrix from hidden layer to output layer. A hidden layer connects both input layer and output layer. The hidden states are delayed by l time step,

and fed back to the network as additional inputs. This procedure can be described in (Equation 12) and (13).

$$h(t+l) = \sigma(W_{IH}x(t) + W_{HH}h(t)) \tag{12}$$

$$y(t+l) = \sigma(W_{HO}h(t+l)) \tag{13}$$

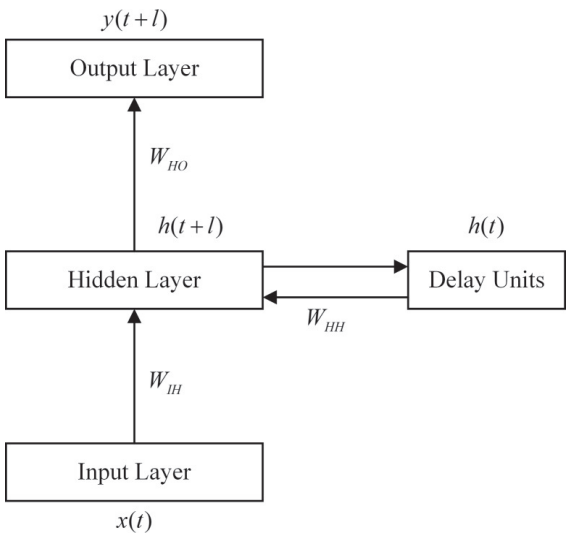


Figure 2: State Space Neural Network Structure.

Where $\sigma(\bullet)$ denote the activation function such as sigmoid function, tangent function and linear function.

Training RNN is similar with training traditional multilayer Feed Forward Neural Network (FFNN). By unfolding the RNN structure over time, each feedback loop will be expanded as single layer feed-forward neural network at each time stamp in Fig. 3. In this case, RNN can be efficiently trained using the well-known back prorogation algorithm over time. This is also known as BackPropagation Through Time (BPTT). However, BPTT has difficulty training the sequence with long-term dependency since it can be converted into a deep architecture with multiple layers, and results in vanishing gradient as well as exploding gradient issues [22]. Although there are a certain number of algorithms proposed to solve these issues such as long short term memory network [23], it is still worth investigating in the context of deep learning theory.

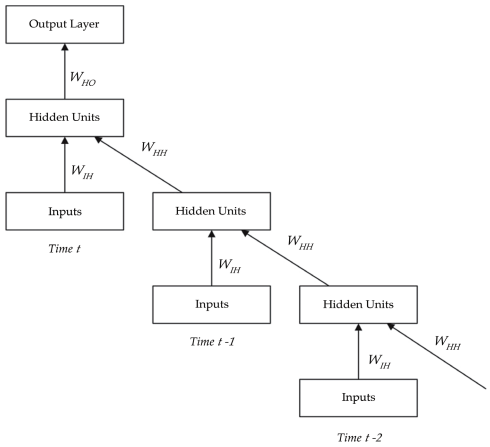


Figure 3: BackPropagation Through Time for RNN.

RNN-RBM

Both RBM and RNN models have the capability of predicting a temporal sequence. A natural thought is to take advantage of the merits from both models. This triggers a deep architecture called RNN-RBM model to depict the temporal dependencies in high-dimensional sequences [17]. The architecture of RBB-RBM is given in Fig. 4:

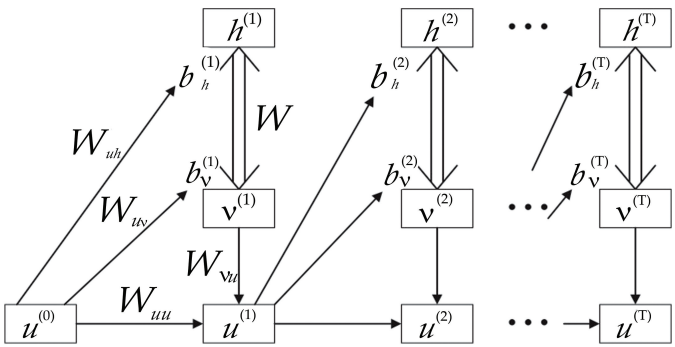


Figure 4: RBM-RNN Architecture.

The conditional RBM and RNN are stacked to construct a RNN-RBM model. Conditional RBM is an extension of conventional RBM, and it is designed to process temporal sequence by providing a feedback loop

between visible layer and hidden layer. The bias values for both visible layer and hidden layer are updated based on the previous visible units [24]. The similar concept can be applied in the RNN-RBM model. b(t)vandb(t) hrespectively represent the bias vectors for visible layer and hidden layer in RBM model at time t, and are updated through the hidden units $u^{(t-1)}$ in RNN model at time t-1. Weight matrices W_{uv} and W_{uh} are provided to connect the RNN model and RBM model. The above procedure can be described in (Equation 14) and (Equation 15).

$$b_v^{(t)} = b_v + W_{uu}u^{(t-1)} \tag{14}$$

$$b_h^{(t)} = b_h + W_{uh}u^{(t-1)} \tag{15}$$

Where b_v and b_h are the initial biases in visible layer and hidden layer for RBM model. The RNN model is unfolded over time, and is utilized to generate the previous hidden states in RBM model based on the input layer $v^{(t)}$ and hidden layer $u^{(t)}$ in RNN model. The activation of hidden units in the hidden layer can be then calculated as:

$$u^{(t)} = sigmoid(b_u + W_{uu}u^{(t-1)} + W_{vu}v^{(t)}) \tag{16}$$

The algorithm execution is summarized below:

Step 1: Generate the value of hidden units in RNN model using (Equation 16).

Step 2: Update the biases in the RBM model using (Equation 14) and (15) based on the estimated $u^{(t-1)}$ in *Step 1*, and calculated the RBM parameters.

Step 3: Calculate the log-likelihood gradient $\frac{\partial lnP(v)}{\partial \theta}$ in RBM model using (Equation 11)

Step 4: The estimated gradient is propagated into RNN model, and the weight W_{uv} and W_{vu} are updated over time to train RNN model for prediction.

Temporal-Spatial Congestion Evolution Prediction

As previously mentioned, traffic condition on each roadway segment can be converted into a binary value denoting congestion status. Considering historical traffic patterns and number of links in a large-

scale traffic network, the congestion evolution prediction problem becomes a high-dimensional temporal sequence learning problem. The RNN-RBM model can be applied to predict the temporal-spatial network congestion evolution pattern. To implement the RNN-RBM model in an efficient manner, the mini-batches gradient descent optimization method is applied: the training set is divided into a number of small samples, and each sample set can be parallelly processed using standard gradient descent optimization method. GPU is designed on a parallel throughput architecture that can execute multiple tasks concurrently [25]. This feature is especially suitable to accelerate the computation of RNN-RBM model. CUDA is a parallel computing platform developed by NVIDIA [26], and it offers a flexible programming library for users to manipulate the computational elements in GPUs.

NUMERICAL STUDY

Model Development

To validate the effectiveness and efficiency of the proposed network-wide congestion prediction approach, a transportation network in Ningbo City, China is used for the numerical experiment. The network consists of 515 road links where approximately 4000 GPS-equipped taxis traveled from April 13, 2014 to May 9, 2014. The GPS data are updated every 2 minutes. A series of data quality control methods are applied to remedy the missing or erroneous data, and this generates 5,521,294 GPS records with speed information. Each GPS record is associated with a unique link ID, and this link ID corresponds to a specific geospatial polyline in a GIS platform. These GPS data are further aggregated into 5 minutes, 10 minutes, 30 minutes and 60 minutes time intervals to evaluate the network traffic congestion prediction performance. The data from the first 22 days (i.e. from April 13, 2014 to May 4, 2014) are utilized for training the RNN-RBM model, and the remaining data are used for testing.

Minimizing cross-entropy error is set as the optimization objective during the RNN-RBM model training procedure. This is because that the cross-entropy indicates the distance between the probability distributions of computed outputs and target outputs, and thus it is preferable to using

mean squared error for a neural network classifier with binary values [27]. The cross-entropy error is defined in (Equation 17).

$$\text{Cross} - \text{Entropy Error (CEE)} = -\frac{1}{T}\sum_{n=1}^{N}\sum_{t=1}^{T} c_n^t ln(\hat{c}_n^t) + (1 - c_n^t) ln(1 - \hat{c}_n^t) \tag{17}$$

In addition, both training accuracy and testing accuracy are presented to measure the algorithm effectiveness. The estimated congestion status for link n at time t is denoted as $\hat{c}_n^t$. According to (Equation 1), the ground-true congestion status for link n at time t is denoted asctn. Because both $\hat{c}_n^t$ and c_n^t are binary values, the prediction accuracy can be defined in (Equation 18) given a transportation network with N links within T time intervals:

$$\text{Accuacy (Acc)} = \left(1 - \frac{\sum_{n=1}^{N}\sum_{t=1}^{T} |c_n^t - \hat{c}_n^t|}{NT}\right) \times 100\% \tag{18}$$

If a positive result refers to each link that is correctly identified as either congested or uncongested, then the sensitivity and specificity can be respectively defined as:

$$\text{Sensitivity} = \frac{\text{number of links correctly identifed as congested}}{\text{total number of congested links}} \tag{19}$$

$$\text{Specificity} = \frac{\text{number of links correctly identifed as uncongested}}{\text{total number of uncongested links}} \tag{20}$$

Algorithm runtime is used to evaluate the efficiency of the proposed approach.

Parameters in the RNN-RBM model are set optimally as follows:

Number of hidden units in the RNN model = 100

Number of hidden units in the RBM model = 150

Learning rate for gradient descent optimization method = 0.05

The weight matrix W for RNN-RBM model is initialized by following a Normal distribution with mean = 0 and variance = 0.01, and other matrices W_{uu}, W_{uv}, W_{vu} and W_{uh} are initialized by following a Normal distribution with mean = 0 and variance = 0.0001. All the bias vectors in the model are set as zero initially.

Result Analysis

GPS speed data with four different aggregation levels were tested using the RNN-RBM model, and the trained model was then utilized to generate the future network-wide congestion evolution patterns from May 5, 2014 to May 9, 2014 for prediction. Each model was run for 200 iterations to ensure the algorithm's successful convergence. The algorithm was implemented using Python Theano [28] and was executed on a desktop computer with Intel i7 3.4GHz CPU, 8GB memory and NVIDIA GeForce GTX650 GPU (2GB RAM). The comparison results are demonstrated in Table 1:

Table 1: Comparison of traffic congestion prediction performance with different data aggregation levels

Aggregation Level	Training Accuracy (%)	Cross-Entropy Error	Runtime (seconds)	Testing Accuracy (%)
60 minutes	95.1	-38.7	354	88.2%
30 minutes	88.4	-102.52	643	80.8%
10 minutes	80.2	-172.3	1729	73.4%
5 minutes	72.2	-239.8	4642	68.9%

Results in Table 1 demonstrate that the overall algorithm prediction performance improves as the data aggregation level increases. This is probably due to the less data fluctuation (i.e. traffic condition does not change dramatically) as the time interval becomes longer. When the aggregation level achieves to one hour, the training accuracy is 95%, and more than 88% of congestion conditions can be correctly identified, while the algorithm can be successfully executed within 6 minutes. Both testing accuracy and training accuracy degrade when the data resolution becomes higher. However, the sensitivity and specificity do not straightly increase as the data aggregation level becomes larger. For instance, only 43% of congested linked can be correctly predicted when the aggregation level is set as 10 minutes. This implies that the data aggregation level influences the prediction outcomes, and should be carefully selected.

In addition, CUDA-based GPU parallel computing engine presents its superior capability of concurrently accelerating the computational intensive tasks. Compared with the CPU-based algorithm implementation, the proposed GPU-based framework can approximately improve the algorithm efficiency by 5 times.

The training accuracy changing curves for the above four scenarios are plotted in Fig. 5:

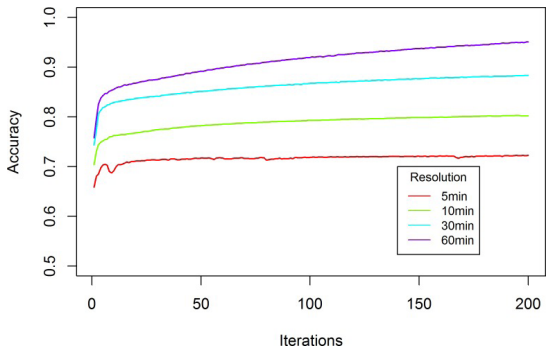


Figure 5: Training Accuracy Changing Curves with Different Data Aggregation Levels.

RNN-RBM models with varying data aggregation levels can converge after 150 iterations, and the training accuracy remains almost constant. This indicates that the RNN-RBM model is able to learn the network-wide traffic congestion patterns in a fast and reliable fashion.

Visualization

Based on the trained RNN-RBM models, the network congestion evolution patterns from May 5, 2014 to May 9, 2014 can be thereby predicted. As previously mentioned, each binary congestion status can be mapped on a GIS platform using the unique link ID. To visualize the spatial and temporal network congestion onset and growth, a series of dynamic GIS networks with colored links were created in Fig. 6A-Fig. 6D based on the predicted traffic congestion patterns on May 9, 2014.

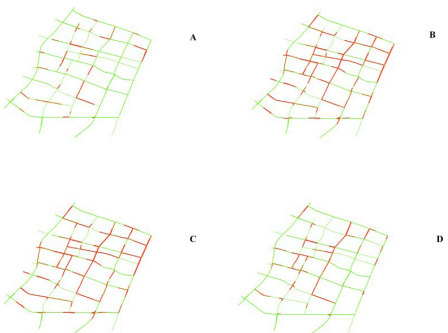


Figure 6: Predicted Network Congestion Evolution Patterns on May 09, 2014 with Varying Times of Day.

The congested link (less than 20 km/hour) is marked in red, and the uncongested link (equal to or greater than 20 km/hour) is marked in green. The traffic condition in the early morning was generally good with only a small number of congested links, and the network became more and more congested during morning peak hours. Similarly, when most people commuted during evening peak hours, network-wide traffic congestion reached highest. Finally, traffic jam dissipated at midnight since the majority of passengers engaged in less or no activities during that time. As presented in Fig. 6A-Fig. 6D, traffic congestion initially concentrated in the central of network, and gradually spread to the network boundary. There are several links that consistently experience traffic congestion, and should be taken special attention for congestion mitigation in advance.

The temporal distribution for number of congested links is summarized in Table 2 and visualized in Fig. 7. There are 35.9% of road links experiencing traffic congestion during morning peak hour (9:00AM-10:00AM), while 41.1% of network was congested during evening peak hour (5:00PM-6:00PM). This indicates that the evening peak traffic is worse than morning peak traffic, and causes more delay. People may tend to return home earlier than usual week days for weekend activities, and thus higher traffic volumes on Friday's evening peak hours can be observed.

Table 2: Statistics for number of congested links on May 9, 2014

Time	Number of congested links	Percentage (%)
5:00–6:00	53	10.3
6:00–7:00	72	14.0
7:00–8:00	121	23.5
8:00–10:00	162	31.5
9:00–10:00	185	35.9
10:00–11:00	168	32.6
11:00–12:00	163	31.7
12:00–13:00	168	32.6
13:00–14:00	146	28.3
14:00–15:00	160	31.1
15:00–16:00	178	34.6
16:00–17:00	183	35.5
17:00–18:00	213	41.4
18:00–19:00	189	36.7
19:00–20:00	174	33.8
20:00–21:00	153	29.7
21:00–22:00	119	23.1
22:00–23:00	111	21.6
23:00–00:00	94	18.3

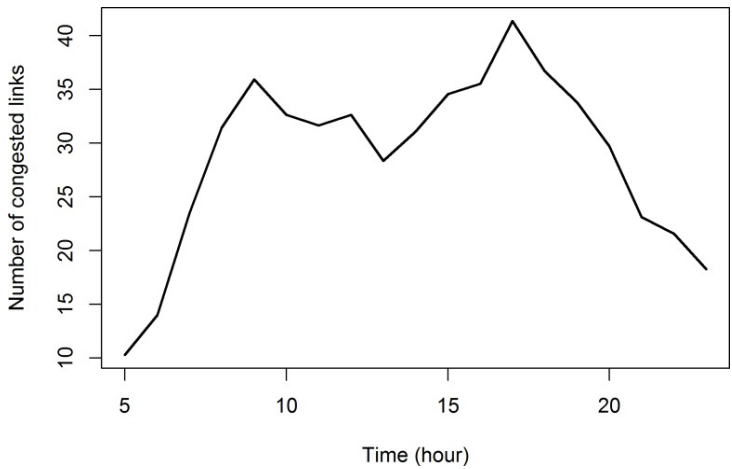


Figure 7: Temporal distribution for number of congested links on May 9, 2014.

Comparison

To further evaluate the advantages of RNN-RBM algorithm for large-scale transportation network congestion prediction, a study was conducted by comparing RNN-RBM, Back Propagation Neural Network (BPNN) and Support Vector Machine (SVM) methods. To remain a fair comparison environment, the same dataset and computing platform was adopted. The topology for BPNN was set as one input layer, one hidden layers with 10 processing units and one output layers\. For the SVM method, Radial Basis Function (RBF) was utilized with three adjustable parameters: cost C, width parameter g, and epsilon ε. These parameters were calibrated using 5-fold cross validation. The rolling horizon (i.e. the size of moving window defining previous network congestion statuses) was set as 2 time steps. To eliminate the randomness, both BPNN and SVM were executed for 10 times. Table 3 presents the performances for three algorithms with 60-minute data aggregation level.

Table 3: Comparison of traffic congestion prediction performance for different algorithms with 60-minute data aggregation level

Algorithms	Runtime (seconds)	Prediction Accuracy (%)	Sensitivity (%)	Specificity (%)
RNN-RBM	354	88.2%	64.1%	91.1%
BPNN	13498	69.7%	38.2%	77.5%
SVM	14979	71.0%	36.6%	80.3%

Due to the limit of computational resources, only 60-minute data aggregation level is utilized for testing. As demonstrated in table 3, the RNN-RBM outperforms the other two algorithm in terms of efficiency and effectiveness: The algorithm execution time for RNN-RBM is only 3% of that for BPNN, and 2.3% of that for SVM. The gain in computational efficiency for RNN-RBM does not sacrifice the algorithm accuracy. Compared with BPNN and SVM, the prediction accuracy for RNN-RBM increases by at least 17%. Moreover, both sensitivity and specificity for RNN-RBM are significantly higher than those for BPNN and SVM. This is owing to the deep and parallel architecture of RNN-RBM for learning multidimensional features in an efficient and effective manner.

Sensitivity Analysis

Congestion identification relies on the setting of speed threshold when traffic congestion occurs. To understand how various speed thresholds influence the congestion pattern prediction performance, a sensitivity analysis was conducted to examine the prediction capability of RNN-RBM model. Three different speed settings were adopted: 10 minutes, 20 minutes and 30 minutes. When the calculated average GPS link speed is lower than the specified speed threshold, the link is considered congested. The GPS speed data were aggregated into 60-minute interval, and followed the similar model training and testing procedure in the model development. The results are presented in table 4:

Table 4: Sensitivity analysis of congestion evolution prediction performance with various speed thresholds

Speed Threshold	Number of Congested Links	Training Accuracy (%)	Runtime (seconds)	Testing Accuracy (%)
10 km/hour	1895	98.6%	290	93.8%
20 km/hour	13611	95.1%	354	88.2%
30 km/hour	20238	84.0%	419	79.9%

There are a total of 59,225 traffic conditions from May 5, 2014 to May 9, 2014 for GPS data with 60-minute aggregation level. When the speed threshold is set as 10 km/hour, only 1895 congested links can be detected. Surprisingly, the model training and testing accuracies are not significantly higher than those with 20km/hour speed threshold, even though the number of congested links with 20km/hour speed threshold is approximately 7 times more than that with 10km/hour speed threshold. However, when the speed threshold increases to 30 km/hour, more than 30% of network links are congested. The training accuracy decreases to 84%, which is much lower than the scenario with 20 km/hour speed threshold. This is probably because that traffic congestion temporal patterns fluctuate more significantly as the speed threshold is higher, and thus incurs difficulties in predicting congestion evolution. In practice, the setting of congestion speed threshold should be carefully determined.

CONCLUSION

Properly understanding the temporal and spatial patterns of congestion evolution is crucial to effectively mitigate congestion. This study proposed a data-driven method to predict the network traffic congestion onset and evolution patterns. Based on the tremendous GPS taxi speed data, the network congestion onset, propagation and dissipation can be modeled using a deep RNN-RBM architecture. To the best of our knowledge, this is the first study to utilize the deep learning theory into large-scale transportation network analysis. Compared with the traditional equation-driven or simulation-driven network studies, the proposed method relies on less assumption to model traffic congestion dynamics. A numerical experiment in Ningbo, China is conducted to validate the effectiveness and efficiency of the proposed method. A transportation network with 515 links is constructed, where the speed information generated from 4000 taxis are mapped and converted into a binary value to represent traffic congestion for each link. The RNN-RBM model has the capability of learning high-dimensional temporal sequences in an accurate and efficient manner, and is implemented in a CUDA-based parallel environment to accelerate the computational procedure. The training accuracy can achieve as high as 95.1% using the GPS data with one hour aggregation level while the algorithm execution time is only less than 6 minutes, and the trained model can be utilized

to correctly predict network-wide traffic congestion on more than 88% links. The inferred network congestion patterns are further visualized on a GIS-based map platform to investigate the temporal and spatial evolution of traffic congestion. By comparing with conventional data mining approaches (i.e. BPNN and SVM), the RNN-RBM algorithm demonstrates its superiority in terms of effectiveness and efficiency. In addition, a sensitivity analysis is conducted to understand how different speed thresholds impact the congestion prediction accuracy. The finding of this study is important for large-scale roadway network planning, operations, and investment decisions. Understanding which congested locations are autonomously generated and how congestion propagates over transportation network will allow researchers and practitioners to focus the limited resources on the primary congestion locations, and adopt the proactive countermeasures to mitigate congestion.

Although the proposed method is promising to model and predict large-scale transportation network congestion, there is still plenty of room to be improved in the future research. For instances, the pretraining techniques such as Hessian-free optimization method should be considered to initialize the parameters more rationally. In addition, the spatial interaction among the adjacent roadway segments should be taken into account to increase training and prediction accuracy.

AUTHOR CONTRIBUTIONS

Conceived and designed the experiments: XM Yinhai Wang. Performed the experiments: XM HY. Analyzed the data: XM HY. Contributed reagents/materials/analysis tools: Yunpeng Wang. Wrote the paper: XM.

REFERENCES

1. Schrank D, Eisele B, Lomax T. TTI's 2012 urban mobility report. Proceedings of the 2012 annual urban mobility report Texas A&M Transportation Institute, Texas, USA. 2012.
2. Aligawesa AK. The detection and localization of traffic congestion for highway traffic systems using hybrid estimation techniques: Purdue University; 2009.

3. Long J, Gao Z, Ren H, Lian A. Urban traffic congestion propagation and bottleneck identification. Science in China Series F: Information Sciences. 2008;51(7):948–64.
4. Yang S. On feature selection for traffic congestion prediction. Transportation Research Part C: Emerging Technologies. 2013;26:160–9.
5. Wang J, Li Y, Liu J, He K, Wang P. Vulnerability analysis and passenger source prediction in urban rail transit networks. PloS one. 2013;8(11):e80178 doi: 10.1371/journal.pone.0080178
6. Wang J, Wang L. Congestion analysis of traffic networks with direction-dependant heterogeneity. Physica A: Statistical Mechanics and its Applications. 2013;392(2):392–9.
7. Liu C, Zhang Q-p, Zhang X. Emergence and disappearance of traffic congestion in weight-evolving networks. Simulation Modelling Practice and Theory. 2009;17(10):1566–74.
8. Colak S, Schneider CM, Wang P, González MC. On the role of spatial dynamics and topology on network flows. New Journal of Physics. 2013;15(11):113037.
9. Zhao L, Lai Y-C, Park K, Ye N. Onset of traffic congestion in complex networks. Physical Review E. 2005;71(2):026125.
10. Wang P, Hunter T, Bayen AM, Schechtner K, González MC. Understanding road usage patterns in urban areas. Scientific reports. 2012;2.
11. Chen S, Huang W, Cattani C, Altieri G. Traffic dynamics on complex networks: a survey. Mathematical Problems in Engineering. 2011;2012.
12. Cheng T, Tanaksaranond G, Brunsdon C, Haworth J. Exploratory visualisation of congestion evolutions on urban transport networks. Transportation Research Part C: Emerging Technologies. 2013;36:296–306.
13. Bellman R, Bellman RE, Bellman RE, Bellman RE. Adaptive control processes: a guided tour: Princeton University Press Princeton; 1961.
14. Huang WH, Song GJ, Hong HK, Xie KQ. Deep Architecture for Traffic Flow Prediction: Deep Belief Networks With Multitask

Learning. Ieee Transactions on Intelligent Transportation Systems. 2014;15(5):2191–201.

15. Hinton GE, Salakhutdinov RR. Reducing the dimensionality of data with neural networks. Science. 2006;313(5786):504–7.
16. Bengio Y. Learning deep architectures for AI. Foundations and trends® in Machine Learning. 2009;2(1):1–127.
17. Boulanger-Lewandowski N, Bengio Y, Vincent P. Modeling temporal dependencies in high-dimensional sequences: Application to polyphonic music generation and transcription. arXiv preprint arXiv:12066392. 2012.
18. Hinton GE. Training products of experts by minimizing contrastive divergence. Neural computation. 2002;14(8):1771–800.
19. Längkvist M, Karlsson L, Loutfi A. A review of unsupervised feature learning and deep learning for time-series modeling. Pattern Recognition Letters. 2014;42:11–24.
20. Sutskever I, Hinton GE, Taylor GW. The recurrent temporal restricted boltzmann machine. Advances in Neural Information Processing Systems; 2009.
21. Hüsken M, Stagge P. Recurrent neural networks for time series classification. Neurocomputing. 2003;50:223–35.
22. Bengio Y, Simard P, Frasconi P. Learning long-term dependencies with gradient descent is difficult. Neural Networks, IEEE Transactions on. 1994;5(2):157–66.
23. Hochreiter S, Schmidhuber J. Long short-term memory. Neural computation. 1997;9(8):1735–80.
24. Sutskever I, Hinton GE, editors. Learning multilevel distributed representations for high-dimensional sequences. International Conference on Artificial Intelligence and Statistics; 2007.
25. Schatz MC, Trapnell C, Delcher AL, Varshney A. High-throughput sequence alignment using Graphics Processing Units. BMC bioinformatics. 2007;8(1):474.
26. Documentation CT. v6. 0. Santa Clara (CA, USA): NVIDIA Corporation. 2014.
27. Golik P, Doetsch P, Ney H, editors. Cross-entropy vs. squared error training: a theoretical and experimental comparison.

INTERSPEECH; 2013.

28. Bergstra J, Breuleux O, Bastien F, Lamblin P, Pascanu R, Desjardins G, et al. . Theano: a CPU and GPU math expression compiler. Proceedings of the Python for scientific computing conference (SciPy); 2010.

Citations

CHAPTER 2

Sheehan S, Song YS (2016) Deep Learning for Population Genetic Inference. PLoS Comput Biol 12(3): e1004845. https://doi.org/10.1371/journal.pcbi.1004845.

CHAPTER 3

Van Valen DA, Kudo T, Lane KM, Macklin DN, Quach NT, DeFelice MM, et al. (2016) Deep Learning Automates the Quantitative Analysis of Individual Cells in Live-Cell Imaging Experiments. PLoS Comput Biol 12(11): e1005177. https://doi.org/10.1371/journal.pcbi.1005177.

CHAPTER 4

Wang S, Sun S, Li Z, Zhang R, Xu J (2017) Accurate De Novo Prediction of Protein Contact Map by Ultra-Deep Learning Model. PLoS Comput Biol 13(1): e1005324. doi:10.1371/journal. pcbi.1005324.

CHAPTER 5

Kim YB, Park N, Zhang Q, Kim JG, Kang SJ, Kim CH (2016) Predicting Virtual World User Population Fluctuations with Deep Learning. PLoS ONE 11(12): e0167153. https://doi.org/10.1371/journal.pone.0167153.

CHAPTER 6

Umarov RK, Solovyev VV (2017) Recognition of prokaryotic and eukaryotic promoters using convolutional deep learning neural networks. PLoS ONE 12(2): e0171410. https://doi.org/10.1371/journal.pone.0171410.

CHAPTER 7

Kang M-J, Kang J-W (2016) Intrusion Detection System Using Deep Neural Network for In-Vehicle Network Security. PLoS ONE 11(6): e0155781. https://doi.org/10.1371/journal.pone.0155781.

CHAPTER 8

Zhang Y, Liu Z, Zhou W, National Natural Science Foundation of China (NSFC) (2016) Event Recognition Based on Deep Learning in Chinese Texts. PLoS ONE 11(8): e0160147. https://doi.org/10.1371/journal.pone.0160147.

CHAPTER 9

Qin Q, Feng J (2017) Imputation for transcription factor binding predictions based on deep learning. PLoS Comput Biol 13(2): e1005403. https://doi.org/10.1371/journal.pcbi.1005403.

CHAPTER 10

Hannagan T, Ziegler JC, Dufau S, Fagot J, Grainger J (2014) Deep Learning of Orthographic Representations in Baboons. PLoS ONE 9(1): e84843. https://doi.org/10.1371/journal.pone.0084843.

CHAPTER 11

Ma X, Yu H, Wang Y, Wang Y (2015) Large-Scale Transportation Network Congestion Evolution Prediction Using Deep Learning Theory. PLoS ONE 10(3): e0119044. doi:10.1371/journal. pone.0119044.

Index

A

Advanced driver assistance systems (ADAS) 208
Advanced Driver Assistance Systems (ADAS) 1
Annealing 294
Approximate Bayesian Computation (ABC) 16
Artificial intelligence 4, 5, 6
Artificial neural network (ANN) 209, 222

B

Bone marrow derived macrophages (BMDMs) 66

C

CASP (Critical Assessment of Structure Prediction) 92
Chinese emergency event recognition model based on deep learning (CEERM) 231
Chinese emergency event recognition model (CEERM) 233
Computer data 1, 2, 3, 5, 7, 9
Compute Unified Device Architecture (CUDA) 319
Contrastive divergence (CD) 238
Contrastive Divergence (CD) 322
Controller area network (CAN) 207
Controller Area Network (CAN) 208
Convolutional network 292, 293
Convolutional neural network (CNN) 278
Convolutional Neural Networks (CNN) 187, 191
Convolutional neural system 2
Correlation coefficient (CC) 189, 196
Cyclic redundancy check (CRC) 210

D

Deep belief network (DBN) 210, 232
Deep belief networks (DBN) 207, 213, 233
Deep learning (DL) 232
Deep neural network (DNN) 207, 209, 212, 224
Direct evolutionary coupling analysis (DCA) 93
Downstream promoter element (DPE)

188

E

Electronics Control Unit (ECU) 208
Evolutionary coupling analysis (ECA) 93
Evolutionary coupling (EC) 91

F

False positive (FP) 189
Fluorescence channel 59, 76

G

Gene expression 188, 189
Gene-recognition 200
Genomic region 20, 31, 45
Global Positioning System (GPS) 316
Grand Cooperative Driving Challenge (GCDC) 208
Graphic Processing Unit (GPU) 316
Graphics Processing Unit (GPU) 319

I

Identity-by-state (IBS) 30, 45
Image segmentation 61, 63, 75, 85
Intelligent Transportation Systems (ITS) 316
Internet of Things (IoT) 316
Intrusion detection sensor (IDS) 209
Intrusion detection system (IDS) 207, 224

K

Kinase translocation reporters (KTRs) 61

L

Linkage disequilibrium (LD) 45

M

Markov chain Monte Carlo(MCMC) 237
Massively multiplayer online games (MMOGs) 140
Mathematical structure 61, 63, 64
Matthews correlation coefficient (MCC) 147
Multiple sequence alignment (MSA) 93, 98

N

Natural language processing (NLP) 232
Network 15, 17, 18, 22, 23, 25, 26, 27, 30, 33, 34, 36, 37, 40, 41, 46, 47
Network technology 140
Neural network 2, 3
Nonlinear transformation 95, 125

O

Open Car Test-bed and Network Experiments (OCTANE) 220
Orthographic knowledge 289

P

Pearson correlation coefficient 144, 154
Personal information 142
Pheromone detection 29
Position weight matrices (PWMs) 260
Posterior distribution 16, 17, 19, 36
Principle component analysis (PCA) 273

R

Receiver Operating Characteristic (ROC) 222
Recurrent Neural Network and Restricted Boltzmann Machine (RNN-RBM) 318
Recurrent Neural Network (RNN) 318, 322
Restricted Boltzmann machine(RBM) 236
Restricted Boltzmann Machine (RBM) 318, 319

Restricted Boltzmann Machines (RBM) 212

S

Sampling 293, 294
Self-organized feature map (SOM) 211
Site frequency spectrum (SFS) 30, 44
Statistics 14, 15, 16, 17, 18, 19, 20, 22, 23, 27, 28, 30, 31, 32, 33, 34, 36, 37, 38, 43, 44, 45, 46, 47, 49, 50, 51, 52
Stochastic gradient descent (SGD) 5
Support vector machines (SVMs) 16
Support vector machine (SVM) 211, 222, 233

T

Template-based models (TBMs) 102, 132
Transcription 187, 188, 189, 190, 197, 200, 202, 204
Transcriptional start sites (TSS) 189
Transcription factors (TFs) 259, 260
Transcription start site (TSS) 188
Transposed-letter (TL) 297

V

Vehicle-to-Infrastructure (V2I) 208
Vehicle-to-Vehicle (V2V) 208
Vocabulary 302

W

Written communication 1

Deep Learning Technologies and Applications

Deep learning tools could very well end up with a substantially higher standard of accuracy and reliability in the recognition of physical objects, in some instances more advanced than individual overall human performance. Deep learning is an outlet of understanding, or perhaps studying, that takes advantage of a number of levels of non-linear processor jobs to discover how you can make representations of highly effective daily processes unswervingly from computer data. The first chapter refers to deep learning. Chapter 2 shows that when provided with genomic variance computer data from a variety of people, calculating the chance of complicated populace hereditary designs can often be improbable. Chapter 3 looks at how live-cell imaging provides you with started out a thrilling range into the function cellular heterogeneity performs in vibrant, subsistence devices. Chapter 4 looks at how protein contacts provide you with crucial information and facts for the comprehension of protein frameworks. Chapter 5 suggests a structure for foretelling updates in electronic community end user behavior. Chapter 6 looks at how precise computational recognition of promoters continues to be an issue. Chapter 7 shows that an innovative intrusion detection system (IDS) making use of a deep neural network (DNN) is offered to improve the safety of in-vehicular system. Chapter 8 looks at how event identification is easily the most basic and also crucial job in event-based all-natural vocabulary processing devices. Chapter 9 looks at getting a grasp on the cell-specific merging designs of transcription factors. Chapter 10 looks at what exactly is the source of our capability to understand orthographic information. Chapter 11 displays comprehending the simplest way to understand blockage at one area.



Gerard I. Prudhomme has a graduate degree (M.S.) for Computer Science from University College London (UCL). He has also worked as a software programmer and tech writer for different Fortune 500 companies, and studied at UCL, Harvard, and Oxford.

ISBN 978-1-77361-274-4

ARCLER PRESS