Branches of Artificial Intelligence



Simon Puchner

BRANCHES OF ARTIFICIAL INTELLIGENCE

BRANCHES OF ARTIFICIAL INTELLIGENCE

Simon Puchner



Branches of Artificial Intelligence
by Simon Puchner

Copyright© 2021 BIBLIOTEX

www.bibliotex.com

All rights reserved. No part of this book may be reproduced or used in any manner without the prior written permission of the copyright owner, except for the use brief quotations in a book review.

To request permissions, contact the publisher at info@bibliotex.com

Ebook ISBN: 9781984656513



Published by:
Bibliotex
Canada
Website: www.bibliotex.com

Contents

Chapter 1 Sub-Symbolic 1

Chapter 2 Nouvelle AI and Computational Creativity 11

Chapter 3 Machine Learning 41

Chapter 4 Evolutionary Computation 117

Chapter 5 Metaheuristic and Swarm Intelligence 159

Chapter 1

Sub-symbolic

Behavior-based robotics

Behavior-based robotics (**BBR**) or **behavioral robotics** is an approach in robotics that focuses on robots that are able to exhibit complex-appearing behaviors despite little internal variable state to model its immediate environment, mostly gradually correcting its actions via sensory-motor links.

Principles

Behavior-based robotics sets itself apart from traditional artificial intelligence by using biological systems as a model. Classic artificial intelligence typically uses a set of steps to solve problems, it follows a path based on internal representations of events compared to the behavior-based approach. Rather than use preset calculations to tackle a situation, behavior-based robotics relies on adaptability. This advancement has allowed behavior-based robotics to become commonplace in researching and data gathering.

Most behavior-based systems are also reactive, which means they need no programming of internal representations of what a chair looks like, or what kind of surface the robot is moving on. Instead, all the information is gleaned from the input of the

robot's sensors. The robot uses that information to gradually correct its actions according to the changes in immediate environment.

Behavior-based robots (BBR) usually show more biological-appearing actions than their computing-intensive counterparts, which are very deliberate in their actions. A BBR often makes mistakes, repeats actions, and appears confused, but can also show the anthropomorphic quality of tenacity. Comparisons between BBRs and insects are frequent because of these actions.

BBRs are sometimes considered examples of weak artificial intelligence, although some have claimed they are models of all intelligence.

Features

Most behavior-based robots are programmed with a basic set of features to start them off. They are given a behavioral repertoire to work with dictating what behaviors to use and when, obstacle avoidance and battery charging can provide a foundation to help the robots learn and succeed.

Rather than build world models, behavior-based robots simply react to their environment and problems within that environment. They draw upon internal knowledge learned from their past experiences combined with their basic behaviors to resolve problems.

History

The school of behavior-based robots owes much to work undertaken in the 1980s at the Massachusetts Institute of Technology by Rodney Brooks, who with students and colleagues built a series of wheeled and legged robots utilizing the subsumption architecture. Brooks' papers, often written with lighthearted titles such as "*Planning is just a way of avoiding figuring out what to do next*", the anthropomorphic qualities of his robots, and the relatively low cost of developing such robots, popularized the behavior-based approach.

Brooks' work builds—whether by accident or not—on two prior milestones in the behavior-based approach. In the 1950s, W. Grey Walter, an English scientist with a background in neurological research, built a pair of vacuum tube-based robots that were exhibited at the 1951 Festival of Britain, and which have simple but effective behavior-based control systems.

The second milestone is Valentino Braitenberg's 1984 book, "*Vehicles – Experiments in Synthetic Psychology*" (MIT Press). He describes a series of thought experiments demonstrating how simply wired sensor/motor connections can result in some complex-appearing behaviors such as fear and love.

Later work in BBR is from the BEAM robotics community, which has built upon the work of Mark Tilden. Tilden was inspired by the reduction in the computational power needed for walking mechanisms from Brooks' experiments (which used one microcontroller for each leg), and further reduced the

computational requirements to that of logic chips, transistor-based electronics, and analog circuit design.

A different direction of development includes extensions of behavior-based robotics to multi-robot teams. The focus in this work is on developing simple generic mechanisms that result in coordinated group behavior, either implicitly or explicitly.

Subsumption architecture

Subsumption architecture is a reactive robotic architecture heavily associated with behavior-based robotics which was very popular in the 1980s and 90s. The term was introduced by Rodney Brooks and colleagues in 1986. Subsumption has been widely influential in autonomous robotics and elsewhere in real-time AI.

Subsumption architecture is a control architecture that was proposed in opposition to traditional AI, or GOFAI. Instead of guiding behavior by symbolic mental representations of the world, subsumption architecture couples sensory information to action selection in an intimate and bottom-up fashion.

It does this by decomposing the complete behavior into sub-behaviors. These sub-behaviors are organized into a hierarchy of layers. Each layer implements a particular level of behavioral competence, and higher levels are able to subsume lower levels (= integrate/combine lower levels to a more comprehensive whole) in order to create viable behavior. For example, a robot's lowest layer could be "avoid an object". The second layer would be "wander around", which runs beneath

the third layer "explore the world". Because a robot must have the ability to "avoid objects" in order to "wander around" effectively, the subsumption architecture creates a system in which the higher layers utilize the lower-level competencies. The layers, which all receive sensor-information, work in parallel and generate outputs. These outputs can be commands to actuators, or signals that suppress or inhibit other layers.

Goal

Subsumption architecture attacks the problem of intelligence from a significantly different perspective than traditional AI. Disappointed with the performance of Shakey the robot and similar conscious mind representation-inspired projects, Rodney Brooks started creating robots based on a different notion of intelligence, resembling unconscious mind processes. Instead of modelling aspects of human intelligence via symbol manipulation, this approach is aimed at real-time interaction and viable responses to a dynamic lab or office environment.

The goal was informed by four key ideas:

- Situatedness – A major idea of situated AI is that a robot should be able to react to its environment within a human-like time-frame. Brooks argues that situated mobile robot should not represent the world via an internal set of symbols and then act on this model. Instead, he claims that "the world is its own best model", which means that proper perception-to-action setups can be used to directly interact with the world as opposed to

modelling it. Yet, each module/behavior still models the world, but on a very low level, close to the sensorimotor signals. These simple models necessarily use hardcoded assumptions about the world encoded in the algorithms themselves, but avoid the use of memory to predict the world's behavior, instead relying on direct sensorial feedback as much as possible.

- Embodiment – Brooks argues building an embodied agent accomplishes two things. The first is that it forces the designer to test and create an integrated physical control system, not theoretic models or simulated robots that might not work in the physical world. The second is that it can solve the symbol grounding problem, a philosophical issue many traditional AIs encounter, by directly coupling sense-data to meaningful actions. "The world grounds regress," and the internal relation of the behavioral layers are directly grounded in the world the robot perceives.
- Intelligence – Looking at evolutionary progress, Brooks argues that developing perceptual and mobility skills are a necessary foundation for human-like intelligence. Also, by rejecting top-down representations as a viable starting point for AI, it seems that "intelligence is determined by the dynamics of interaction with the world."
- Emergence – Conventionally, individual modules are not considered intelligent by themselves. It is the interaction of such modules, evaluated by observing the agent and its environment, that is

> usually deemed intelligent (or not). "Intelligence," therefore, "is in the eye of the observer."

The ideas outlined above are still a part of an ongoing debate regarding the nature of intelligence and how the progress of robotics and AI should be fostered.

Layers and augmented finite-state machines

Each layer is made up by a set of processors that are augmented finite-state machines (AFSM), the augmentation being added instance variables to hold programmable data-structures. A layer is a module and is responsible for a single behavioral goal, such as "wander around." There is no central control within or between these behavioral modules. All AFSMs continuously and asynchronously receive input from the relevant sensors and send output to actuators (or other AFSMs). Input signals that are not read by the time a new one is delivered end up getting discarded. These discarded signals are common, and is useful for performance because it allows the system to work in real time by dealing with the most immediate information.

Because there is no central control, AFSMs communicate with each other via inhibition and suppression signals. Inhibition signals block signals from reaching actuators or AFSMs, and suppression signals blocks or replaces the inputs to layers or their AFSMs. This system of AFSM communication is how

higher layers subsume lower ones (see figure 1), as well as how the architecture deals with priority and action selection arbitration in general.

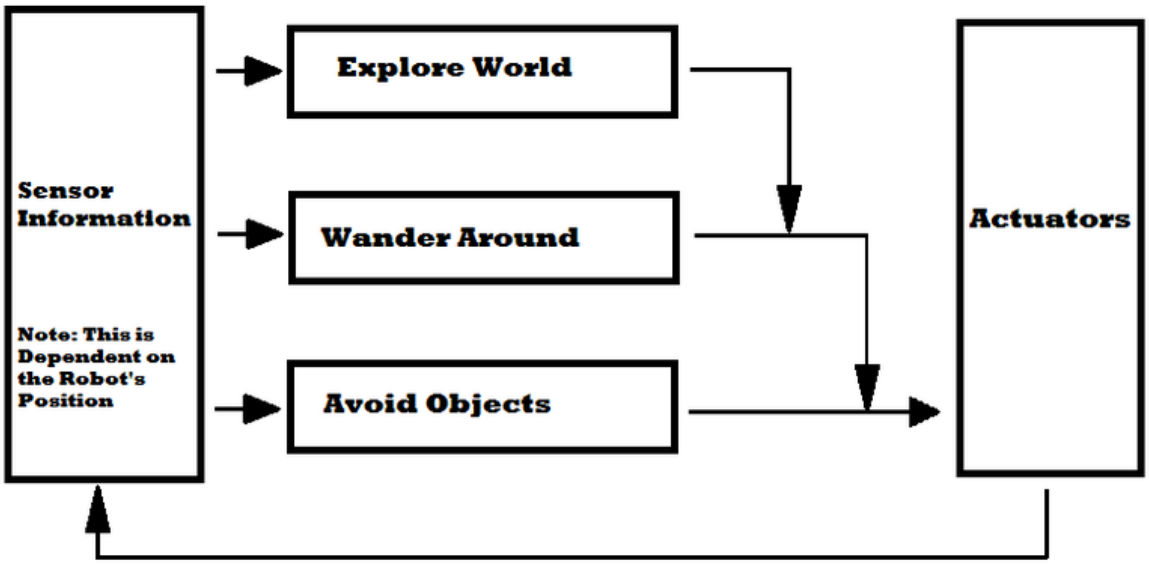


Figure: Abstract representation of subsumption architecture, with the higher level layers subsuming the roles of lower level layers when the sensory information determines it.

The development of layers follows an intuitive progression. First, the lowest layer is created, tested, and debugged.

Once that lowest level is running, one creates and attaches the second layer with the proper suppression and inhibition connections to the first layer.

After testing and debugging the combined behavior, this process can be repeated for (theoretically) any number of behavioral modules.

Robots

The following is a small list of robots that utilize the subsumption architecture.

- Allen (robot)
- Herbert, a soda can collecting robot (see external links for a video)
- Genghis, a robust hexapodal walker (see external links for a video)

The above are described in detail along with other robots in *Elephants Don't Play Chess*.

Strengths and weaknesses

The main advantages of the architecture are:

- the emphasis on iterative development and testing of real-time systems in their target domain;
- the emphasis on connecting limited, task-specific perception directly to the expressed actions that require it; and
- the emphasis on distributive and parallel control, thereby integrating the perception, control, and action systems in a manner similar to animals.

The main disadvantages of the architecture are:

- the difficulty of designing adaptable action selection through highly distributed system of inhibition and suppression; and
- the lack of large memory and symbolic representation, which seems to restrict the architecture from understanding language;

When subsumption architecture was developed, the novel setup and approach of subsumption architecture allowed it to be successful in many important domains where traditional AI had failed, namely real-time interaction with a dynamic environment. The lack of large memory storage, symbolic representations, and central control, however, places it at a disadvantage at learning complex actions, in-depth mapping, and understanding language.

Chapter 2

Nouvelle AI and Computational Creativity

Nouvelle AI

Nouvelle artificial intelligence (**AI**) is an approach to artificial intelligence pioneered in the 1980s by Rodney Brooks, who was then part of MIT artificial intelligence laboratory. Nouvelle AI differs from classical AI by aiming to produce robots with intelligence levels similar to insects. Researchers believe that intelligence can emerge organically from simple behaviors as these intelligences interacted with the "real world," instead of using the constructed worlds which symbolic AIs typically needed to have programmed into them.

Motivation

The differences between nouvelle AI and symbolic AI are apparent in early robots Shakey and Freddy. These robots contained an internal model (or "representation") of their micro-worlds consisting of symbolic descriptions. As a result, this structure of symbols had to be renewed as the robot moved or the world changed.

Shakey's planning programs assessed the program structure and broke it down into the necessary steps to complete the desired action. This level of computation required a large amount time to process, so Shakey typically performed its tasks very slowly.

Symbolic AI researchers had long been plagued by the problem of updating, searching, and otherwise manipulating the symbolic worlds inside their AIs. A nouvelle system refers continuously to its sensors rather than to an internal model of the world. It processes the external world information it needs from the senses when it is required. As Brooks puts it, "the world is its own best model--always exactly up to date and complete in every detail."

A central idea of nouvelle AI is that simple behaviors combine to form more complex behaviors over time. For example, simple behaviors can include elements like "move forward" and "avoid obstacles." A robot using nouvelle AI with simple behaviors like collision avoidance and moving toward a moving object could possibly come together to produce a more complex behavior like chasing a moving object.

The frame problem

The frame problem describes an issue with using first-order logic (FOL) to express facts about a robot in the world. Representing the state of a robot with traditional FOL requires the use of many axioms (symbolic language) to imply that things about an environment that do not change arbitrarily.

Nouvelle AI seeks to sidestep the frame problem by dispensing with filling the AI or robot with volumes of symbolic language

and instead letting more complex behaviors emerge by combining simpler behavioral elements.

Embodiment

The goal of traditional AI was to build intelligences without bodies, which would only have been able to interact with the world via keyboard, screen, or printer. However, nouvelle AI attempts to build embodied intelligence situated in the real world. Brooks quotes approvingly from the brief sketches that Turing gave in 1948 and 1950 of the "situated" approach. Turing wrote of equipping a machine "with the best sense organs that money can buy" and teaching it "to understand and speak English" by a process that would "follow the normal teaching of a child." This approach was contrasted to the others where they focused on abstract activities such as playing chess.

Brooks' robots

Insectoid robots

Brooks focused on building robots that acted like simple insects while simultaneously working to remove some traditional AI characteristics. He created insect-like robots called Allen and Herbert.

Brooks's insectoid robots contained no internal models of the world. Herbert, for example, discarded a high volume of the information received from its sensors and never stored information for more than two seconds.

Allen

Allen, named after Allen Newell, had a ring of twelve ultrasonic sonars as its primary sensors and three independent behavior-producing modules. These modules were programmed to avoid both stationary and moving objects. With only this module activated, Allen stayed in the middle of a room until an object approached and then it ran away while avoiding obstacles in its way.

Herbert

Herbert, named after Herbert A. Simon, used infrared sensors to avoid obstacles and a laser system to collect 3D data over a distance of about 12 feet. Herbert also carried a number of simple sensors in its "hand." The robot's testing ground was the real world environment of the busy offices and workspaces of the MIT AI lab where it searched for empty soda cans and carried them away, a seemingly goal-oriented activity that emerged as a result of 15 simple behavior units combining together.

Other insectoid robots

Other robots by Brooks' team were Genghis and Squirt. Genghis had six legs and was able to walk over rough terrain and follow a human. Squirt's behavior modules had it stay in dark corners until it heard a noise, then it would begin to follow the source of the noise.

Brooks agreed that the level of nouvelle AI had come near the complexity of a real insect, which raised a question about

whether or not insect level-behavior was and is a reasonable goal for nouvelle AI?

Humanoid robots

Brooks' own recent work has taken the opposite direction to that proposed by Von Neumann in the quotations "theorists who select the human nervous system as their model are unrealistically picking 'the most complicated object under the sun,' and that there is little advantage in selecting instead the ant, since any nervous system at all exhibits exceptional complexity."

Cog

In the 1990s, Brooks decided to pursue the goal of human-level intelligence and, with Lynn Andrea Stein, built a humanoid robot called Cog.

Cog is a robot with an extensive collection of sensors, a face, and arms (among other features) that allow it to interact with the world and gather information and experience so as to assemble intelligence organically in the manner described above by Turing.

The team believes that Cog will able to learn and able to find a correlation between the sensory information it receives and its actions. In the long term, the team wants Cog to be able to learn common sense knowledge on its own.

Computational creativity

Computational creativity (also known as **artificial creativity**, **mechanical creativity**, **creative computing** or **creative computation**) is a multidisciplinary endeavour that is located at the intersection of the fields of artificial intelligence, cognitive psychology, philosophy, and the arts.

The goal of computational creativity is to model, simulate or replicate creativity using a computer, to achieve one of several ends:

- To construct a program or computer capable of human-level creativity.
- To better understand human creativity and to formulate an algorithmic perspective on creative behavior in humans.
- To design programs that can enhance human creativity without necessarily being creative themselves.

The field of computational creativity concerns itself with theoretical and practical issues in the study of creativity. Theoretical work on the nature and proper definition of creativity is performed in parallel with practical work on the implementation of systems that exhibit creativity, with one strand of work informing the other.

The applied form of computational creativity is known as media synthesis.

Theoretical issues

As measured by the amount of activity in the field (e.g., publications, conferences and workshops), computational creativity is a growing area of research. But the field is still hampered by a number of fundamental problems. Creativity is very difficult, perhaps even impossible, to define in objective terms. Is it a state of mind, a talent or ability, or a process? Creativity takes many forms in human activity, some *eminent* (sometimes referred to as "Creativity" with a capital C) and some *mundane*.

These are problems that complicate the study of creativity in general, but certain problems attach themselves specifically to *computational* creativity:

- Can creativity be hard-wired? In existing systems to which creativity is attributed, is the creativity that of the system or that of the system's programmer or designer?
- How do we evaluate computational creativity? What counts as creativity in a computational system? Are natural language generation systems creative? Are machine translation systems creative? What distinguishes research in computational creativity from research in artificial intelligence generally?
- If eminent creativity is about rule-breaking or the disavowal of convention, how is it possible for an algorithmic system to be creative? In essence, this is a variant of Ada Lovelace's objection to machine

> intelligence, as recapitulated by modern theorists such as Teresa Amabile. If a machine can do only what it was programmed to do, how can its behavior ever be called *creative*?

Indeed, not all computer theorists would agree with the premise that computers can only do what they are programmed to do—a key point in favor of computational creativity.

Defining creativity in computational terms

Because no single perspective or definition seems to offer a complete picture of creativity, the AI researchers Newell, Shaw and Simon developed the combination of novelty and usefulness into the cornerstone of a multi-pronged view of creativity, one that uses the following four criteria to categorize a given answer or solution as creative:

- The answer is novel and useful (either for the individual or for society)
- The answer demands that we reject ideas we had previously accepted
- The answer results from intense motivation and persistence
- The answer comes from clarifying a problem that was originally vague
- Whereas the above reflects a "top-down" approach to computational creativity, an alternative thread

> has developed among "bottom-up" computational psychologists involved in artificial neural network research. During the late 1980s and early 1990s, for example, such generative neural systems were driven by genetic algorithms. Experiments involving recurrent nets were successful in hybridizing simple musical melodies and predicting listener expectations.

Concurrent with such research, a number of computational psychologists took the perspective, popularized by Stephen Wolfram, that system behaviors perceived as complex, including the mind's creative output, could arise from what would be considered simple algorithms. As neuro-philosophical thinking matured, it also became evident that language actually presented an obstacle to producing a scientific model of cognition, creative or not, since it carried with it so many unscientific aggrandizements that were more uplifting than accurate. Thus questions naturally arose as to how "rich," "complex," and "wonderful" creative cognition actually was.

Artificial neural networks

Before 1989, artificial neural networks have been used to model certain aspects of creativity. Peter Todd (1989) first trained a neural network to reproduce musical melodies from a training set of musical pieces. Then he used a change algorithm to modify the network's input parameters. The network was able to randomly generate new music in a highly uncontrolled manner. In 1992, Todd extended this work, using the so-called distal teacher approach that had been developed

by Paul Munro, Paul Werbos, D. Nguyen and Bernard Widrow, Michael I. Jordan and David Rumelhart. In the new approach there are two neural networks, one of which is supplying training patterns to another. In later efforts by Todd, a composer would select a set of melodies that define the melody space, position them on a 2-d plane with a mouse-based graphic interface, and train a connectionist network to produce those melodies, and listen to the new "interpolated" melodies that the network generates corresponding to intermediate points in the 2-d plane.

More recently a neurodynamical model of semantic networks has been developed to study how the connectivity structure of these networks relates to the richness of the semantic constructs, or ideas, they can generate. It was demonstrated that semantic neural networks that have richer semantic dynamics than those with other connectivity structures may provide insight into the important issue of how the physical structure of the brain determines one of the most profound features of the human mind – its capacity for creative thought.

Key concepts from the literature

Some high-level and philosophical themes recur throughout the field of computational creativity.

Important categories of creativity

Margaret Boden refers to creativity that is novel *merely to the agent that produces it* as "P-creativity" (or "psychological creativity"), and refers to creativity that is recognized as novel

by society at large as "H-creativity" (or "historical creativity"). Stephen Thaler has suggested a new category he calls "V-" or "Visceral creativity" wherein significance is invented to raw sensory inputs to a Creativity Machine architecture, with the "gateway" nets perturbed to produce alternative interpretations, and downstream nets shifting such interpretations to fit the overarching context. An important variety of such V-creativity is consciousness itself, wherein meaning is reflexively invented to activation turnover within the brain.

Exploratory and transformational creativity

Boden also distinguishes between the creativity that arises from an exploration within an established conceptual space, and the creativity that arises from a deliberate transformation or transcendence of this space. She labels the former as *exploratory creativity* and the latter as *transformational creativity*, seeing the latter as a form of creativity far more radical, challenging, and rarer than the former. Following the criteria from Newell and Simon elaborated above, we can see that both forms of creativity should produce results that are appreciably novel and useful (criterion 1), but exploratory creativity is more likely to arise from a thorough and persistent search of a well-understood space (criterion 3) -- while transformational creativity should involve the rejection of some of the constraints that define this space (criterion 2) or some of the assumptions that define the problem itself (criterion 4). Boden's insights have guided work in computational creativity at a very general level, providing more an inspirational touchstone for development work than a technical framework of algorithmic substance. However, Boden's insights are more

recently also the subject of formalization, most notably in the work by Geraint Wiggins.

Generation and evaluation

The criterion that creative products should be novel and useful means that creative computational systems are typically structured into two phases, generation and evaluation. In the first phase, novel (to the system itself, thus P-Creative) constructs are generated; unoriginal constructs that are already known to the system are filtered at this stage. This body of potentially creative constructs is then evaluated, to determine which are meaningful and useful and which are not. This two-phase structure conforms to the Geneplore model of Finke, Ward and Smith, which is a psychological model of creative generation based on empirical observation of human creativity.

Combinatorial creativity

A great deal, perhaps all, of human creativity can be understood as a novel combination of pre-existing ideas or objects. Common strategies for combinatorial creativity include:

- Placing a familiar object in an unfamiliar setting (e.g., Marcel Duchamp's *Fountain*) or an unfamiliar object in a familiar setting (e.g., a fish-out-of-water story such as *The Beverly Hillbillies*)
- Blending two superficially different objects or genres (e.g., a sci-fi story set in the Wild West, with robot cowboys, as in *Westworld*, or the reverse, as in *Firefly*; Japanese haiku poems, etc.)

- Comparing a familiar object to a superficially unrelated and semantically distant concept (e.g., "Makeup is the Western burka"; "A zoo is a gallery with living exhibits")
- Adding a new and unexpected feature to an existing concept (e.g., adding a scalpel to a Swiss Army knife; adding a camera to a mobile phone)
- Compressing two incongruous scenarios into the same narrative to get a joke (e.g., the Emo Philips joke "Women are always using men to advance their careers. Damned anthropologists!")
- Using an iconic image from one domain in a domain for an unrelated or incongruous idea or product (e.g., using the Marlboro Man image to sell cars, or to advertise the dangers of smoking-related impotence).

The combinatorial perspective allows us to model creativity as a search process through the space of possible combinations. The combinations can arise from composition or concatenation of different representations, or through a rule-based or stochastic transformation of initial and intermediate representations. Genetic algorithms and neural networks can be used to generate blended or crossover representations that capture a combination of different inputs.

Conceptual blending

Mark Turner and Gilles Fauconnier propose a model called Conceptual Integration Networks that elaborates upon Arthur Koestler's ideas about creativity as well as more recent work by Lakoff and Johnson, by synthesizing ideas from Cognitive

Linguistic research into mental spaces and conceptual metaphors. Their basic model defines an integration network as four connected spaces:

- A first input space (contains one conceptual structure or mental space)
- A second input space (to be blended with the first input)
- A *generic space* of stock conventions and image-schemas that allow the input spaces to be understood from an integrated perspective
- A *blend space* in which a selected projection of elements from both input spaces are combined; inferences arising from this combination also reside here, sometimes leading to emergent structures that conflict with the inputs.

Fauconnier and Turner describe a collection of optimality principles that are claimed to guide the construction of a well-formed integration network. In essence, they see blending as a compression mechanism in which two or more input structures are compressed into a single blend structure. This compression operates on the level of conceptual relations. For example, a series of similarity relations between the input spaces can be compressed into a single identity relationship in the blend.

Some computational success has been achieved with the blending model by extending pre-existing computational models of analogical mapping that are compatible by virtue of their emphasis on connected semantic structures. More recently, Francisco Câmara Pereira presented an implementation of blending theory that employs ideas both

from GOFAI and genetic algorithms to realize some aspects of blending theory in a practical form; his example domains range from the linguistic to the visual, and the latter most notably includes the creation of mythical monsters by combining 3-D graphical models.

Linguistic creativity

Language provides continuous opportunity for creativity, evident in the generation of novel sentences, phrasings, puns, neologisms, rhymes, allusions, sarcasm, irony, similes, metaphors, analogies, witticisms, and jokes. Native speakers of morphologically rich languages frequently create new word-forms that are easily understood, and some have found their way to the dictionary. The area of natural language generation has been well studied, but these creative aspects of everyday language have yet to be incorporated with any robustness or scale.

Hypothesis of creative patterns

In the seminal work of applied linguist Ronald Carter, he hypothesized two main creativity types involving words and word patterns: pattern-reforming creativity, and pattern-forming creativity. Pattern-reforming creativity refers to creativity by the breaking of rules, reforming and reshaping patterns of language often through individual innovation, while pattern-forming creativity refers to creativity via conformity to language rules rather than breaking them, creating convergence, symmetry and greater mutuality between

interlocutors through their interactions in the form of repetitions.

Story generation

Substantial work has been conducted in this area of linguistic creation since the 1970s, with the development of James Meehan's TALE-SPIN system. TALE-SPIN viewed stories as narrative descriptions of a problem-solving effort, and created stories by first establishing a goal for the story's characters so that their search for a solution could be tracked and recorded. The MINSTREL system represents a complex elaboration of this basic approach, distinguishing a range of character-level goals in the story from a range of author-level goals for the story. Systems like Bringsjord's BRUTUS elaborate these ideas further to create stories with complex inter-personal themes like betrayal. Nonetheless, MINSTREL explicitly models the creative process with a set of Transform Recall Adapt Methods (TRAMs) to create novel scenes from old. The MEXICA model of Rafael Pérez y Pérez and Mike Sharples is more explicitly interested in the creative process of storytelling, and implements a version of the engagement-reflection cognitive model of creative writing.

The company Narrative Science makes computer generated news and reports commercially available, including summarizing team sporting events based on statistical data from the game. It also creates financial reports and real estate analyses.

Metaphor and simile

Example of a metaphor: *"She was an ape."*

Example of a simile: *"Felt like a tiger-fur blanket."* The computational study of these phenomena has mainly focused on interpretation as a knowledge-based process. Computationalists such as Yorick Wilks, James Martin, Dan Fass, John Barnden, and Mark Lee have developed knowledge-based approaches to the processing of metaphors, either at a linguistic level or a logical level. Tony Veale and Yanfen Hao have developed a system, called Sardonicus, that acquires a comprehensive database of explicit similes from the web; these similes are then tagged as bona-fide (e.g., "as hard as steel") or ironic (e.g., "as hairy as a bowling ball", "as pleasant as a root canal"); similes of either type can be retrieved on demand for any given adjective. They use these similes as the basis of an on-line metaphor generation system called Aristotle that can suggest lexical metaphors for a given descriptive goal (e.g., to describe a supermodel as skinny, the source terms "pencil", "whip", "whippet", "rope", "stick-insect" and "snake" are suggested).

Analogy

The process of analogical reasoning has been studied from both a mapping and a retrieval perspective, the latter being key to the generation of novel analogies. The dominant school of research, as advanced by Dedre Gentner, views analogy as a structure-preserving process; this view has been implemented in the structure mapping engine or SME, the MAC/FAC retrieval engine (Many Are Called, Few Are Chosen), ACME (Analogical Constraint Mapping Engine) and ARCS (Analogical Retrieval Constraint System). Other mapping-based approaches include Sapper, which situates the mapping process in a semantic-network model of memory. Analogy is a very active

sub-area of creative computation and creative cognition; active figures in this sub-area include Douglas Hofstadter, Paul Thagard, and Keith Holyoak. Also worthy of note here is Peter Turney and Michael Littman's machine learning approach to the solving of SAT-style analogy problems; their approach achieves a score that compares well with average scores achieved by humans on these tests.

Joke generation

Humour is an especially knowledge-hungry process, and the most successful joke-generation systems to date have focussed on pun-generation, as exemplified by the work of Kim Binsted and Graeme Ritchie. This work includes the JAPE system, which can generate a wide range of puns that are consistently evaluated as novel and humorous by young children. An improved version of JAPE has been developed in the guise of the STANDUP system, which has been experimentally deployed as a means of enhancing linguistic interaction with children with communication disabilities. Some limited progress has been made in generating humour that involves other aspects of natural language, such as the deliberate misunderstanding of pronominal reference (in the work of Hans Wim Tinholt and Anton Nijholt), as well as in the generation of humorous acronyms in the HAHAcronym system of Oliviero Stock and Carlo Strapparava.

Neologism

The blending of multiple word forms is a dominant force for new word creation in language; these new words are commonly called "blends" or "portmanteau words" (after Lewis Carroll). Tony Veale has developed a system called ZeitGeist that

harvests neological headwords from Wikipedia and interprets them relative to their local context in Wikipedia and relative to specific word senses in WordNet. ZeitGeist has been extended to generate neologisms of its own; the approach combines elements from an inventory of word parts that are harvested from WordNet, and simultaneously determines likely glosses for these new words (e.g., "food traveller" for "gastronaut" and "time traveller" for "chrononaut"). It then uses Web search to determine which glosses are meaningful and which neologisms have not been used before; this search identifies the subset of generated words that are both novel ("H-creative") and useful. Neurolinguistic inspirations have been used to analyze the process of novel word creation in the brain, understand neurocognitive processes responsible for intuition, insight, imagination and creativity and to create a server that invents novel names for products, based on their description. Further, the system Nehovah blends two source words into a neologism that blends the meanings of the two source words. Nehovah searches WordNet for synonyms and TheTopTens.com for pop culture hyponyms. The synonyms and hyponyms are blended together to create a set of candidate neologisms. The neologisms are then scored based on their word structure, how unique the word is, how apparent the concepts are conveyed, and if the neologism has a pop culture reference. Nehovah loosely follows conceptual blending.

A corpus linguistic approach to the search and extraction of neologism have also shown to be possible. Using Corpus of Contemporary American English as a reference corpus, Locky Law has performed an extraction of neologism, portmanteaus and slang words using the hapax legomena which appeared in the scripts of American TV drama House M.D.

In terms of linguistic research in neologism, Stefan Th. Gries has performed a quantitative analysis of blend structure in English and found that "the degree of recognizability of the source words and that the similarity of source words to the blend plays a vital role in blend formation." The results were validated through a comparison of intentional blends to speech-error blends.

Poetry

Like jokes, poems involve a complex interaction of different constraints, and no general-purpose poem generator adequately combines the meaning, phrasing, structure and rhyme aspects of poetry. Nonetheless, Pablo Gervás has developed a noteworthy system called ASPERA that employs a case-based reasoning (CBR) approach to generating poetic formulations of a given input text via a composition of poetic fragments that are retrieved from a case-base of existing poems. Each poem fragment in the ASPERA case-base is annotated with a prose string that expresses the meaning of the fragment, and this prose string is used as the retrieval key for each fragment. Metrical rules are then used to combine these fragments into a well-formed poetic structure. Racter is an example of such a software project.

Musical creativity

Computational creativity in the music domain has focused both on the generation of musical scores for use by human musicians, and on the generation of music for performance by computers. The domain of generation has included classical

music (with software that generates music in the style of Mozart and Bach) and jazz. Most notably, David Cope has written a software system called "Experiments in Musical Intelligence" (or "EMI") that is capable of analyzing and generalizing from existing music by a human composer to generate novel musical compositions in the same style. EMI's output is convincing enough to persuade human listeners that its music is human-generated to a high level of competence.

In the field of contemporary classical music, Iamus is the first computer that composes from scratch, and produces final scores that professional interpreters can play. The London Symphony Orchestra played a piece for full orchestra, included in Iamus' debut CD, which *New Scientist* described as "The first major work composed by a computer and performed by a full orchestra". Melomics, the technology behind Iamus, is able to generate pieces in different styles of music with a similar level of quality.

Creativity research in jazz has focused on the process of improvisation and the cognitive demands that this places on a musical agent: reasoning about time, remembering and conceptualizing what has already been played, and planning ahead for what might be played next. The robot Shimon, developed by Gil Weinberg of Georgia Tech, has demonstrated jazz improvisation. Virtual improvisation software based on researches on stylistic modeling carried out by Gerard Assayag and Shlomo Dubnov include OMax, SoMax and PyOracle, are used to create improvisations in real-time by re-injecting variable length sequences learned on the fly from live performer.

In 1994, a Creativity Machine architecture (see above) was able to generate 11,000 musical hooks by training a synaptically perturbed neural net on 100 melodies that had appeared on the top ten list over the last 30 years. In 1996, a self-bootstrapping Creativity Machine observed audience facial expressions through an advanced machine vision system and perfected its musical talents to generate an album entitled "Song of the Neurons"

In the field of musical composition, the patented works by René-Louis Baron allowed to make a robot that can create and play a multitude of orchestrated melodies so-called "coherent" in any musical style. All outdoor physical parameter associated with one or more specific musical parameters, can influence and develop each of these songs (in real time while listening to the song). The patented invention *Medal-Composer* raises problems of copyright.

Visual and artistic creativity

Computational creativity in the generation of visual art has had some notable successes in the creation of both abstract art and representational art. The most famous program in this domain is Harold Cohen's AARON, which has been continuously developed and augmented since 1973. Though formulaic, Aaron exhibits a range of outputs, generating black-and-white drawings or colour paintings that incorporate human figures (such as dancers), potted plants, rocks, and other elements of background imagery. These images are of a sufficiently high quality to be displayed in reputable galleries.

Other software artists of note include the NEvAr system (for "Neuro-Evolutionary Art") of Penousal Machado. NEvAr uses a genetic algorithm to derive a mathematical function that is then used to generate a coloured three-dimensional surface. A human user is allowed to select the best pictures after each phase of the genetic algorithm, and these preferences are used to guide successive phases, thereby pushing NEvAr's search into pockets of the search space that are considered most appealing to the user.

The Painting Fool, developed by Simon Colton originated as a system for overpainting digital images of a given scene in a choice of different painting styles, colour palettes and brush types. Given its dependence on an input source image to work with, the earliest iterations of the Painting Fool raised questions about the extent of, or lack of, creativity in a computational art system. Nonetheless, in more recent work, The Painting Fool has been extended to create novel images, much as AARON does, from its own limited imagination. Images in this vein include cityscapes and forests, which are generated by a process of constraint satisfaction from some basic scenarios provided by the user (e.g., these scenarios allow the system to infer that objects closer to the viewing plane should be larger and more color-saturated, while those further away should be less saturated and appear smaller). Artistically, the images now created by the Painting Fool appear on a par with those created by Aaron, though the extensible mechanisms employed by the former (constraint satisfaction, etc.) may well allow it to develop into a more elaborate and sophisticated painter.

The artist Krasi Dimtch (Krasimira Dimtchevska) and the software developer Svillen Ranev have created a computational system combining a rule-based generator of English sentences and a visual composition builder that converts sentences generated by the system into abstract art. The software generates automatically indefinite number of different images using different color, shape and size palettes. The software also allows the user to select the subject of the generated sentences or/and the one or more of the palettes used by the visual composition builder.

An emerging area of computational creativity is that of video games. ANGELINA is a system for creatively developing video games in Java by Michael Cook. One important aspect is Mechanic Miner, a system that can generate short segments of code that act as simple game mechanics. ANGELINA can evaluate these mechanics for usefulness by playing simple unsolvable game levels and testing to see if the new mechanic makes the level solvable. Sometimes Mechanic Miner discovers bugs in the code and exploits these to make new mechanics for the player to solve problems with.

In July 2015 Google released *DeepDream* – an open source computer vision program, created to detect faces and other patterns in images with the aim of automatically classifying images, which uses a convolutional neural network to find and enhance patterns in images via algorithmic pareidolia, thus creating a dreamlike psychedelic appearance in the deliberately over-processed images.

In August 2015 researchers from Tübingen, Germany created a convolutional neural network that uses neural representations

to separate and recombine content and style of arbitrary images which is able to turn images into stylistic imitations of works of art by artists such as a Picasso or Van Gogh in about an hour. Their algorithm is put into use in the website DeepArt that allows users to create unique artistic images by their algorithm.

In early 2016, a global team of researchers explained how a new computational creativity approach known as the Digital Synaptic Neural Substrate (DSNS) could be used to generate original chess puzzles that were not derived from endgame databases. The DSNS is able to combine features of different objects (e.g. chess problems, paintings, music) using stochastic methods in order to derive new feature specifications which can be used to generate objects in any of the original domains. The generated chess puzzles have also been featured on YouTube.

Creativity in problem solving

Creativity is also useful in allowing for unusual solutions in problem solving. In psychology and cognitive science, this research area is called creative problem solving. The Explicit-Implicit Interaction (EII) theory of creativity has recently been implemented using a CLARION-based computational model that allows for the simulation of incubation and insight in problem solving. The emphasis of this computational creativity project is not on performance per se (as in artificial intelligence projects) but rather on the explanation of the psychological processes leading to human creativity and the reproduction of data collected in psychology experiments. So far, this project

has been successful in providing an explanation for incubation effects in simple memory experiments, insight in problem solving, and reproducing the overshadowing effect in problem solving.

Debate about "general" theories of creativity

Some researchers feel that creativity is a complex phenomenon whose study is further complicated by the plasticity of the language we use to describe it.

We can describe not just the agent of creativity as "creative" but also the product and the method. Consequently, it could be claimed that it is unrealistic to speak of a *general theory of creativity*. Nonetheless, some generative principles are more general than others, leading some advocates to claim that certain computational approaches are "general theories". Stephen Thaler, for instance, proposes that certain modalities of neural networks are generative enough, and general enough, to manifest a high degree of creative capabilities. Likewise, the Formal Theory of Creativity is based on a simple computational principle published by Jürgen Schmidhuber in 1991. The theory postulates that creativity and curiosity and selective attention in general are by-products of a simple algorithmic principle for measuring and optimizing learning progress.

Criticism of Computational Creativity

Traditional computers, as mainly used in the computational creativity application, do not support creativity, as they fundamentally transform a set of discrete, limited domain of input parameters into a set of discrete, limited domain of output parameters using a limited set of computational functions. As such, a computer cannot be creative, as everything in the output must have been already present in the input data or the algorithms. For some related discussions and references to related work are captured in some recent work on philosophical foundations of simulation.

Mathematically, the same set of arguments against creativity has been made by Chaitin. Similar observations come from a Model Theory perspective. All this criticism emphasizes that computational creativity is useful and may look like creativity, but it is not real creativity, as nothing new is created, just transformed in well defined algorithms.

Events

The International Conference on Computational Creativity (ICCC) occurs annually, organized by The Association for Computational Creativity. Events in the series include:

- ICCC 2018, Salamanca, Spain

- ICCC 2017, Atlanta, Georgia, USA
- ICCC 2016, Paris, France
- ICCC 2015, Park City, Utah, USA. Keynote: Emily Short
- ICCC 2014, Ljubljana, Slovenia. Keynote: Oliver Deussen
- ICCC 2013, Sydney, Australia. Keynote: Arne Dietrich
- ICCC 2012, Dublin, Ireland. Keynote: Steven Smith
- ICCC 2011, Mexico City, Mexico. Keynote: George E Lewis
- ICCC 2010, Lisbon, Portugal. Keynote/Invited Talks: Nancy J Nersessian and Mary Lou Maher

Previously, the community of computational creativity has held a dedicated workshop, the International Joint Workshop on Computational Creativity, every year since 1999. Previous events in this series include:

- IJWCC 2003, Acapulco, Mexico, as part of IJCAI'2003
- IJWCC 2004, Madrid, Spain, as part of ECCBR'2004
- IJWCC 2005, Edinburgh, UK, as part of IJCAI'2005
- IJWCC 2006, Riva del Garda, Italy, as part of ECAI'2006
- IJWCC 2007, London, UK, a stand-alone event
- IJWCC 2008, Madrid, Spain, a stand-alone event

The 1st Conference on Computer Simulation of Musical Creativity will be held

- CCSMC 2016, 17–19 June, University of Huddersfield, UK. Keynotes: Geraint Wiggins and Graeme Bailey.

Publications and forums

Design Computing and Cognition is one conference that addresses computational creativity. The ACM Creativity and Cognition conference is another forum for issues related to computational creativity. Journées d'Informatique Musicale 2016 keynote by Shlomo Dubnov was on Information Theoretic Creativity.

A number of recent books provide either a good introduction or a good overview of the field of Computational Creativity. These include:

- Pereira, F. C. (2007). "Creativity and Artificial Intelligence: A Conceptual Blending Approach". Applications of Cognitive Linguistics series, Mouton de Gruyter.
- Veale, T. (2012). "Exploding the Creativity Myth: The Computational Foundations of Linguistic Creativity". Bloomsbury Academic, London.
- McCormack, J. and d'Inverno, M. (eds.) (2012). "Computers and Creativity". Springer, Berlin.
- Veale, T., Feyaerts, K. and Forceville, C. (2013, forthcoming). "Creativity and the Agile Mind: A

Multidisciplinary study of a Multifaceted phenomenon". Mouton de Gruyter.

In addition to the proceedings of conferences and workshops, the computational creativity community has thus far produced these special journal issues dedicated to the topic:

- *New Generation Computing*, volume 24, issue 3, 2006
- *Journal of Knowledge-Based Systems*, volume 19, issue 7, November 2006
- *AI Magazine*, volume 30, number 3, Fall 2009
- *Minds and Machines*, volume 20, number 4, November 2010
- *Cognitive Computation*, volume 4, issue 3, September 2012
- *AIEDAM*, volume 27, number 4, Fall 2013
- *Computers in Entertainment*, two special issues on Music Meta-Creation (MuMe), Fall 2016 (forthcoming)

In addition to these, a new journal has started which focuses on computational creativity within the field of music.

- JCMS 2016, *Journal of Creative Music Systems*

Chapter 3

Machine Learning

Machine learning (**ML**) is the study of computer algorithms that improve automatically through experience. It is seen as a subset of artificial intelligence. Machine learning algorithms build a model based on sample data, known as "training data", in order to make predictions or decisions without being explicitly programmed to do so. Machine learning algorithms are used in a wide variety of applications, such as email filtering and computer vision, where it is difficult or unfeasible to develop conventional algorithms to perform the needed tasks.

A subset of machine learning is closely related to computational statistics, which focuses on making predictions using computers; but not all machine learning is statistical learning. The study of mathematical optimization delivers methods, theory and application domains to the field of machine learning. Data mining is a related field of study, focusing on exploratory data analysis through unsupervised learning. In its application across business problems, machine learning is also referred to as predictive analytics.

Overview

Machine learning involves computers discovering how they can perform tasks without being explicitly programmed to do so. It

involves computers learning from data provided so that they carry out certain tasks. For simple tasks assigned to computers, it is possible to program algorithms telling the machine how to execute all steps required to solve the problem at hand; on the computer's part, no learning is needed. For more advanced tasks, it can be challenging for a human to manually create the needed algorithms. In practice, it can turn out to be more effective to help the machine develop its own algorithm, rather than having human programmers specify every needed step.

The discipline of machine learning employs various approaches to teach computers to accomplish tasks where no fully satisfactory algorithm is available. In cases where vast numbers of potential answers exist, one approach is to label some of the correct answers as valid. This can then be used as training data for the computer to improve the algorithm(s) it uses to determine correct answers. For example, to train a system for the task of digital character recognition, the MNIST dataset of handwritten digits has often been used.

Machine learning approaches

Machine learning approaches are traditionally divided into three broad categories, depending on the nature of the "signal" or "feedback" available to the learning system:

- Supervised learning: The computer is presented with example inputs and their desired outputs, given by a "teacher", and the goal is to learn a general rule that maps inputs to outputs.
- Unsupervised learning: No labels are given to the learning algorithm, leaving it on its own to find

structure in its input. Unsupervised learning can be a goal in itself (discovering hidden patterns in data) or a means towards an end (feature learning).

- Reinforcement learning: A computer program interacts with a dynamic environment in which it must perform a certain goal (such as driving a vehicle or playing a game against an opponent). As it navigates its problem space, the program is provided feedback that's analogous to rewards, which it tries to maximize.

Other approaches have been developed which don't fit neatly into this three-fold categorisation, and sometimes more than one is used by the same machine learning system. For example topic modeling, dimensionality reduction or meta learning.

As of 2020, deep learning has become the dominant approach for much ongoing work in the field of machine learning.

History and relationships to other fields

The term *machine learning* was coined in 1959 by Arthur Samuel, an American IBMer and pioneer in the field of computer gaming and artificial intelligence. A representative book of the machine learning research during the 1960s was the Nilsson's book on Learning Machines, dealing mostly with machine learning for pattern classification. Interest related to

pattern recognition continued into the 1970s, as described by Duda and Hart in 1973. In 1981 a report was given on using teaching strategies so that a neural network learns to recognize 40 characters (26 letters, 10 digits, and 4 special symbols) from a computer terminal.

Tom M. Mitchell provided a widely quoted, more formal definition of the algorithms studied in the machine learning field: "A computer program is said to learn from experience *E* with respect to some class of tasks *T* and performance measure *P* if its performance at tasks in *T*, as measured by *P*, improves with experience *E*." This definition of the tasks in which machine learning is concerned offers a fundamentally operational definition rather than defining the field in cognitive terms. This follows Alan Turing's proposal in his paper "Computing Machinery and Intelligence", in which the question "Can machines think?" is replaced with the question "Can machines do what we (as thinking entities) can do?".

Artificial intelligence

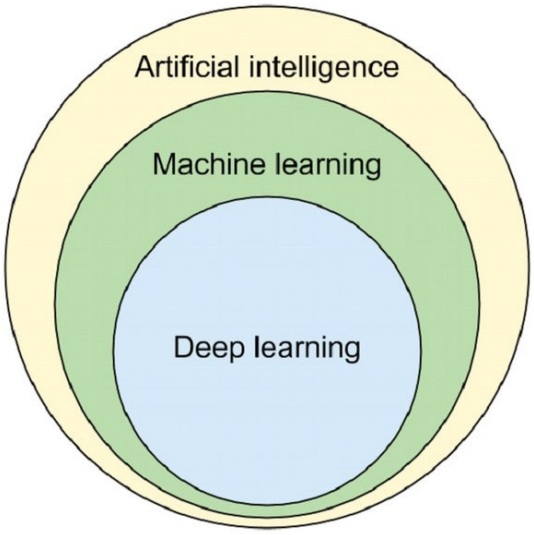


Fig: Machine Learning as subfield of AI

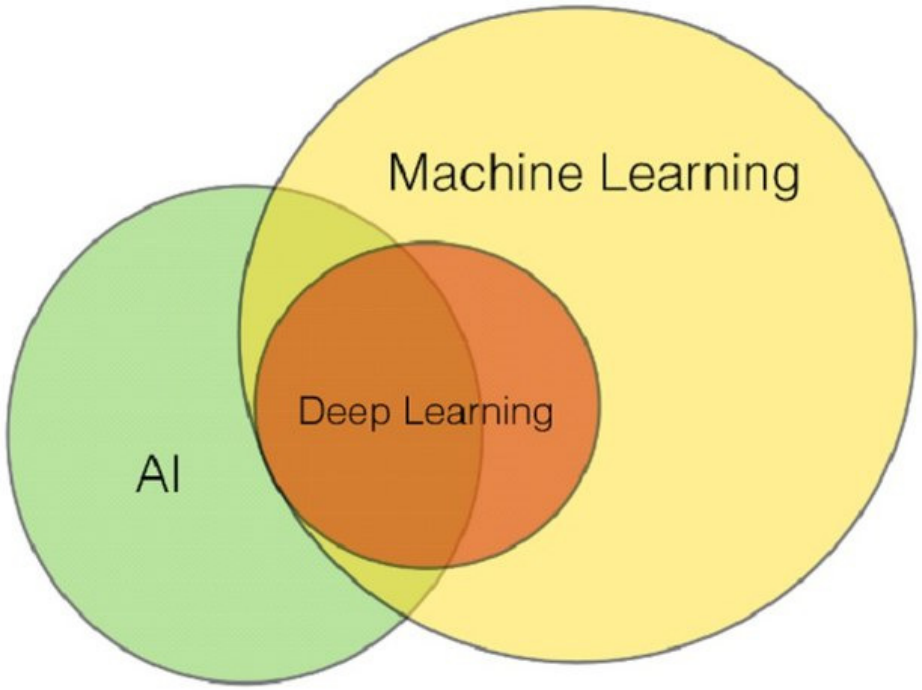


Fig: Part of Machine Learning as subfield of AI or part of AI as subfield of Machine Learning

As a scientific endeavor, machine learning grew out of the quest for artificial intelligence. In the early days of AI as an academic discipline, some researchers were interested in having machines learn from data. They attempted to approach the problem with various symbolic methods, as well as what was then termed "neural networks"; these were mostly perceptrons and other models that were later found to be reinventions of the generalized linear models of statistics. Probabilistic reasoning was also employed, especially in automated medical diagnosis.

However, an increasing emphasis on the logical, knowledge-based approach caused a rift between AI and machine learning. Probabilistic systems were plagued by theoretical and practical problems of data acquisition and representation. By 1980, expert systems had come to dominate AI, and statistics was

out of favor. Work on symbolic/knowledge-based learning did continue within AI, leading to inductive logic programming, but the more statistical line of research was now outside the field of AI proper, in pattern recognition and information retrieval. Neural networks research had been abandoned by AI and computer science around the same time. This line, too, was continued outside the AI/CS field, as "connectionism", by researchers from other disciplines including Hopfield, Rumelhart and Hinton. Their main success came in the mid-1980s with the reinvention of backpropagation.

Machine learning (ML), reorganized as a separate field, started to flourish in the 1990s. The field changed its goal from achieving artificial intelligence to tackling solvable problems of a practical nature. It shifted focus away from the symbolic approaches it had inherited from AI, and toward methods and models borrowed from statistics and probability theory.

As of 2020, many sources continue to assert that machine learning remains a subfield of AI. The main disagreement is whether all of ML is part of AI, as this would mean that anyone using ML could claim they are using AI. Others have the view that not all of ML is part of AI where only an 'intelligent' subset of ML is part of AI.

The question to what is the difference between ML and AI is answered by Judea Pearl in *The Book of Why*. Accordingly ML learns and predicts based on passive observations, whereas AI implies an agent interacting with the environment to learn and take actions that maximize its chance of successfully achieving its goals.

Data mining

Machine learning and data mining often employ the same methods and overlap significantly, but while machine learning focuses on prediction, based on *known* properties learned from the training data, data mining focuses on the discovery of (previously) *unknown* properties in the data (this is the analysis step of knowledge discovery in databases). Data mining uses many machine learning methods, but with different goals; on the other hand, machine learning also employs data mining methods as "unsupervised learning" or as a preprocessing step to improve learner accuracy. Much of the confusion between these two research communities (which do often have separate conferences and separate journals, ECML PKDD being a major exception) comes from the basic assumptions they work with: in machine learning, performance is usually evaluated with respect to the ability to *reproduce known* knowledge, while in knowledge discovery and data mining (KDD) the key task is the discovery of previously *unknown* knowledge. Evaluated with respect to known knowledge, an uninformed (unsupervised) method will easily be outperformed by other supervised methods, while in a typical KDD task, supervised methods cannot be used due to the unavailability of training data.

Optimization

Machine learning also has intimate ties to optimization: many learning problems are formulated as minimization of some loss function on a training set of examples. Loss functions express the discrepancy between the predictions of the model being trained and the actual problem instances (for example, in

classification, one wants to assign a label to instances, and models are trained to correctly predict the pre-assigned labels of a set of examples). The difference between the two fields arises from the goal of generalization: while optimization algorithms can minimize the loss on a training set, machine learning is concerned with minimizing the loss on unseen samples.

Statistics

Machine learning and statistics are closely related fields in terms of methods, but distinct in their principal goal: statistics draws population inferences from a sample, while machine learning finds generalizable predictive patterns. According to Michael I. Jordan, the ideas of machine learning, from methodological principles to theoretical tools, have had a long pre-history in statistics. He also suggested the term data science as a placeholder to call the overall field.

Leo Breiman distinguished two statistical modeling paradigms: data model and algorithmic model, wherein "algorithmic model" means more or less the machine learning algorithms like Random forest.

Some statisticians have adopted methods from machine learning, leading to a combined field that they call *statistical learning*.

Theory

A core objective of a learner is to generalize from its experience. Generalization in this context is the ability of a learning machine to perform accurately on new, unseen examples/tasks after having experienced a learning data set. The training examples come from some generally unknown probability distribution (considered representative of the space of occurrences) and the learner has to build a general model about this space that enables it to produce sufficiently accurate predictions in new cases.

The computational analysis of machine learning algorithms and their performance is a branch of theoretical computer science known as computational learning theory. Because training sets are finite and the future is uncertain, learning theory usually does not yield guarantees of the performance of algorithms. Instead, probabilistic bounds on the performance are quite common. The bias–variance decomposition is one way to quantify generalization error.

For the best performance in the context of generalization, the complexity of the hypothesis should match the complexity of the function underlying the data. If the hypothesis is less complex than the function, then the model has under fitted the data. If the complexity of the model is increased in response, then the training error decreases. But if the hypothesis is too complex, then the model is subject to overfitting and generalization will be poorer.

In addition to performance bounds, learning theorists study the time complexity and feasibility of learning. In computational learning theory, a computation is considered feasible if it can be done in polynomial time. There are two kinds of time complexity results. Positive results show that a certain class of functions can be learned in polynomial time. Negative results show that certain classes cannot be learned in polynomial time.

Approaches

Types of learning algorithms

The types of machine learning algorithms differ in their approach, the type of data they input and output, and the type of task or problem that they are intended to solve.

Supervised learning

Supervised learning algorithms build a mathematical model of a set of data that contains both the inputs and the desired outputs. The data is known as training data, and consists of a set of training examples. Each training example has one or more inputs and the desired output, also known as a supervisory signal. In the mathematical model, each training example is represented by an array or vector, sometimes called a feature vector, and the training data is represented by a matrix. Through iterative optimization of an objective function, supervised learning algorithms learn a function that can be used to predict the output associated with new inputs. An optimal function will allow the algorithm to correctly determine

the output for inputs that were not a part of the training data. An algorithm that improves the accuracy of its outputs or predictions over time is said to have learned to perform that task.

Types of supervised learning algorithms include active learning, classification and regression. Classification algorithms are used when the outputs are restricted to a limited set of values, and regression algorithms are used when the outputs may have any numerical value within a range. As an example, for a classification algorithm that filters emails, the input would be an incoming email, and the output would be the name of the folder in which to file the email.

Similarity learning is an area of supervised machine learning closely related to regression and classification, but the goal is to learn from examples using a similarity function that measures how similar or related two objects are. It has applications in ranking, recommendation systems, visual identity tracking, face verification, and speaker verification.

Unsupervised learning

Unsupervised learning algorithms take a set of data that contains only inputs, and find structure in the data, like grouping or clustering of data points. The algorithms, therefore, learn from test data that has not been labeled, classified or categorized. Instead of responding to feedback, unsupervised learning algorithms identify commonalities in the data and react based on the presence or absence of such commonalities in each new piece of data. A central application of unsupervised learning is in the field of density estimation in

statistics, such as finding the probability density function. Though unsupervised learning encompasses other domains involving summarizing and explaining data features.

Cluster analysis is the assignment of a set of observations into subsets (called *clusters*) so that observations within the same cluster are similar according to one or more predesignated criteria, while observations drawn from different clusters are dissimilar. Different clustering techniques make different assumptions on the structure of the data, often defined by some *similarity metric* and evaluated, for example, by *internal compactness*, or the similarity between members of the same cluster, and *separation*, the difference between clusters. Other methods are based on *estimated density* and *graph connectivity*.

Semi-supervised learning

Semi-supervised learning falls between unsupervised learning (without any labeled training data) and supervised learning (with completely labeled training data). Some of the training examples are missing training labels, yet many machine-learning researchers have found that unlabeled data, when used in conjunction with a small amount of labeled data, can produce a considerable improvement in learning accuracy.

In weakly supervised learning, the training labels are noisy, limited, or imprecise; however, these labels are often cheaper to obtain, resulting in larger effective training sets.

Reinforcement learning

Reinforcement learning is an area of machine learning concerned with how software agents ought to take actions in an environment so as to maximize some notion of cumulative reward. Due to its generality, the field is studied in many other disciplines, such as game theory, control theory, operations research, information theory, simulation-based optimization, multi-agent systems, swarm intelligence, statistics and genetic algorithms. In machine learning, the environment is typically represented as a Markov decision process (MDP). Many reinforcement learning algorithms use dynamic programming techniques. Reinforcement learning algorithms do not assume knowledge of an exact mathematical model of the MDP, and are used when exact models are infeasible. Reinforcement learning algorithms are used in autonomous vehicles or in learning to play a game against a human opponent.

Self learning

Self-learning as a machine learning paradigm was introduced in 1982 along with a neural network capable of self-learning named *crossbar adaptive array* (CAA). It is a learning with no external rewards and no external teacher advice. The CAA self-learning algorithm computes, in a crossbar fashion, both decisions about actions and emotions (feelings) about consequence situations. The system is driven by the interaction between cognition and emotion. The self-learning algorithm updates a memory matrix W =||w(a,s)|| such that in each iteration executes the following machine learning routine:

In situation s perform an action a;

Receive consequence situation s';

Compute emotion of being in consequence situation v(s');

Update crossbar memory w'(a,s) = w(a,s) + v(s').

It is a system with only one input, situation s, and only one output, action (or behavior) a. There is neither a separate reinforcement input nor an advice input from the environment. The backpropagated value (secondary reinforcement) is the emotion toward the consequence situation. The CAA exists in two environments, one is the behavioral environment where it behaves, and the other is the genetic environment, wherefrom it initially and only once receives initial emotions about situations to be encountered in the behavioral environment. After receiving the genome (species) vector from the genetic environment, the CAA learns a goal-seeking behavior, in an environment that contains both desirable and undesirable situations.

Feature learning

Several learning algorithms aim at discovering better representations of the inputs provided during training. Classic examples include principal components analysis and cluster analysis. Feature learning algorithms, also called representation learning algorithms, often attempt to preserve the information in their input but also transform it in a way that makes it useful, often as a pre-processing step before performing classification or predictions. This technique allows reconstruction of the inputs coming from the unknown data-

generating distribution, while not being necessarily faithful to configurations that are implausible under that distribution. This replaces manual feature engineering, and allows a machine to both learn the features and use them to perform a specific task.

Feature learning can be either supervised or unsupervised. In supervised feature learning, features are learned using labeled input data. Examples include artificial neural networks, multilayer perceptrons, and supervised dictionary learning. In unsupervised feature learning, features are learned with unlabeled input data. Examples include dictionary learning, independent component analysis, autoencoders, matrix factorization and various forms of clustering.

Manifold learning algorithms attempt to do so under the constraint that the learned representation is low-dimensional. Sparse coding algorithms attempt to do so under the constraint that the learned representation is sparse, meaning that the mathematical model has many zeros. Multilinear subspace learning algorithms aim to learn low-dimensional representations directly from tensor representations for multidimensional data, without reshaping them into higher-dimensional vectors. Deep learning algorithms discover multiple levels of representation, or a hierarchy of features, with higher-level, more abstract features defined in terms of (or generating) lower-level features. It has been argued that an intelligent machine is one that learns a representation that disentangles the underlying factors of variation that explain the observed data.

Feature learning is motivated by the fact that machine learning tasks such as classification often require input that is mathematically and computationally convenient to process. However, real-world data such as images, video, and sensory data has not yielded to attempts to algorithmically define specific features. An alternative is to discover such features or representations thorough examination, without relying on explicit algorithms.

Sparse dictionary learning

Sparse dictionary learning is a feature learning method where a training example is represented as a linear combination of basis functions, and is assumed to be a sparse matrix. The method is strongly NP-hard and difficult to solve approximately. A popular heuristic method for sparse dictionary learning is the K-SVD algorithm. Sparse dictionary learning has been applied in several contexts. In classification, the problem is to determine the class to which a previously unseen training example belongs. For a dictionary where each class has already been built, a new training example is associated with the class that is best sparsely represented by the corresponding dictionary. Sparse dictionary learning has also been applied in image de-noising. The key idea is that a clean image patch can be sparsely represented by an image dictionary, but the noise cannot.

Anomaly detection

In data mining, anomaly detection, also known as outlier detection, is the identification of rare items, events or observations which raise suspicions by differing significantly

from the majority of the data. Typically, the anomalous items represent an issue such as bank fraud, a structural defect, medical problems or errors in a text. Anomalies are referred to as outliers, novelties, noise, deviations and exceptions.

In particular, in the context of abuse and network intrusion detection, the interesting objects are often not rare objects, but unexpected bursts of inactivity. This pattern does not adhere to the common statistical definition of an outlier as a rare object, and many outlier detection methods (in particular, unsupervised algorithms) will fail on such data unless it has been aggregated appropriately. Instead, a cluster analysis algorithm may be able to detect the micro-clusters formed by these patterns.

Three broad categories of anomaly detection techniques exist. Unsupervised anomaly detection techniques detect anomalies in an unlabeled test data set under the assumption that the majority of the instances in the data set are normal, by looking for instances that seem to fit least to the remainder of the data set. Supervised anomaly detection techniques require a data set that has been labeled as "normal" and "abnormal" and involves training a classifier (the key difference to many other statistical classification problems is the inherently unbalanced nature of outlier detection). Semi-supervised anomaly detection techniques construct a model representing normal behavior from a given normal training data set and then test the likelihood of a test instance to be generated by the model.

Robot learning

In developmental robotics, robot learning algorithms generate their own sequences of learning experiences, also known as a curriculum, to cumulatively acquire new skills through self-guided exploration and social interaction with humans. These robots use guidance mechanisms such as active learning, maturation, motor synergies and imitation.

Association rules

Association rule learning is a rule-based machine learning method for discovering relationships between variables in large databases. It is intended to identify strong rules discovered in databases using some measure of "interestingness".

Rule-based machine learning is a general term for any machine learning method that identifies, learns, or evolves "rules" to store, manipulate or apply knowledge. The defining characteristic of a rule-based machine learning algorithm is the identification and utilization of a set of relational rules that collectively represent the knowledge captured by the system. This is in contrast to other machine learning algorithms that commonly identify a singular model that can be universally applied to any instance in order to make a prediction. Rule-based machine learning approaches include learning classifier systems, association rule learning, and artificial immune systems.

Based on the concept of strong rules, Rakesh Agrawal, Tomasz Imieliński and Arun Swami introduced association rules for discovering regularities between products in large-scale

transaction data recorded by point-of-sale (POS) systems in supermarkets. For example, the rule found in the sales data of a supermarket would indicate that if a customer buys onions and potatoes together, they are likely to also buy hamburger meat. Such information can be used as the basis for decisions about marketing activities such as promotional pricing or product placements. In addition to market basket analysis, association rules are employed today in application areas including Web usage mining, intrusion detection, continuous production, and bioinformatics. In contrast with sequence mining, association rule learning typically does not consider the order of items either within a transaction or across transactions.

Learning classifier systems (LCS) are a family of rule-based machine learning algorithms that combine a discovery component, typically a genetic algorithm, with a learning component, performing either supervised learning, reinforcement learning, or unsupervised learning. They seek to identify a set of context-dependent rules that collectively store and apply knowledge in a piecewise manner in order to make predictions.

Inductive logic programming (ILP) is an approach to rule-learning using logic programming as a uniform representation for input examples, background knowledge, and hypotheses. Given an encoding of the known background knowledge and a set of examples represented as a logical database of facts, an ILP system will derive a hypothesized logic program that entails all positive and no negative examples. Inductive programming is a related field that considers any kind of

programming language for representing hypotheses (and not only logic programming), such as functional programs.

Inductive logic programming is particularly useful in bioinformatics and natural language processing. Gordon Plotkin and Ehud Shapiro laid the initial theoretical foundation for inductive machine learning in a logical setting. Shapiro built their first implementation (Model Inference System) in 1981: a Prolog program that inductively inferred logic programs from positive and negative examples. The term *inductive* here refers to philosophical induction, suggesting a theory to explain observed facts, rather than mathematical induction, proving a property for all members of a well-ordered set.

Models

Performing machine learning involves creating a model, which is trained on some training data and then can process additional data to make predictions. Various types of models have been used and researched for machine learning systems.

Artificial neural networks

Artificial neural networks (ANNs), or connectionist systems, are computing systems vaguely inspired by the biological neural networks that constitute animal brains. Such systems "learn" to perform tasks by considering examples, generally without being programmed with any task-specific rules.

An ANN is a model based on a collection of connected units or nodes called "artificial neurons", which loosely model the neurons in a biological brain. Each connection, like the

synapses in a biological brain, can transmit information, a "signal", from one artificial neuron to another. An artificial neuron that receives a signal can process it and then signal additional artificial neurons connected to it. In common ANN implementations, the signal at a connection between artificial neurons is a real number, and the output of each artificial neuron is computed by some non-linear function of the sum of its inputs. The connections between artificial neurons are called "edges". Artificial neurons and edges typically have a weight that adjusts as learning proceeds. The weight increases or decreases the strength of the signal at a connection. Artificial neurons may have a threshold such that the signal is only sent if the aggregate signal crosses that threshold. Typically, artificial neurons are aggregated into layers. Different layers may perform different kinds of transformations on their inputs. Signals travel from the first layer (the input layer) to the last layer (the output layer), possibly after traversing the layers multiple times.

The original goal of the ANN approach was to solve problems in the same way that a human brain would. However, over time, attention moved to performing specific tasks, leading to deviations from biology. Artificial neural networks have been used on a variety of tasks, including computer vision, speech recognition, machine translation, social network filtering, playing board and video games and medical diagnosis.

Deep learning consists of multiple hidden layers in an artificial neural network. This approach tries to model the way the human brain processes light and sound into vision and hearing. Some successful applications of deep learning are computer vision and speech recognition.

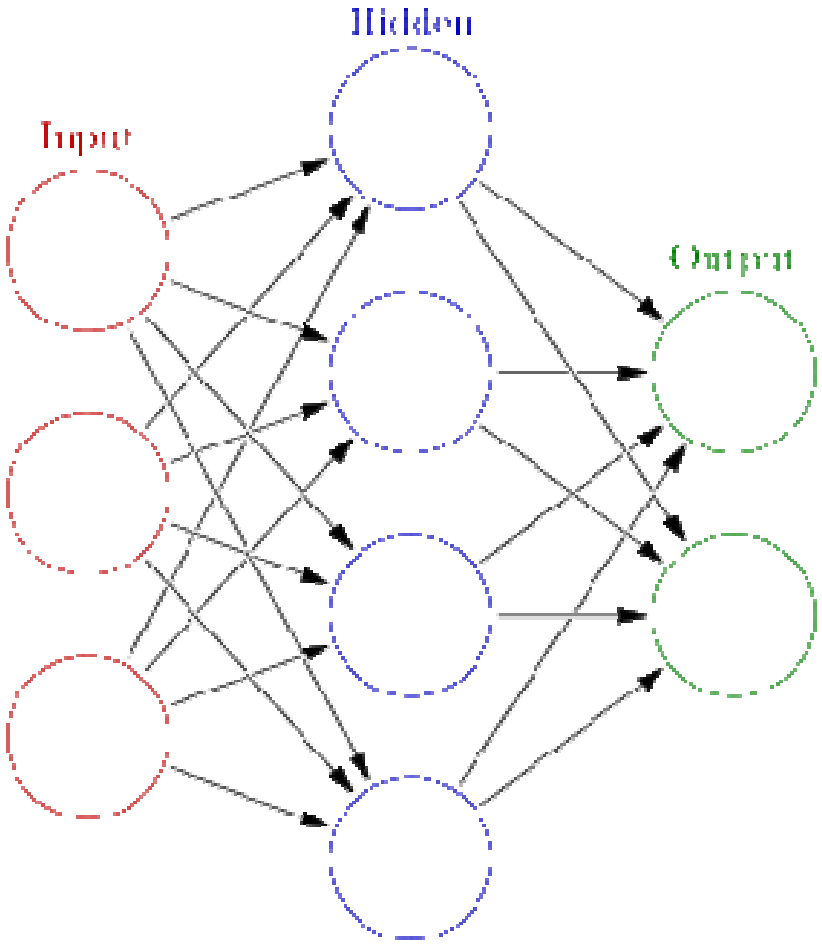


Fig: An artificial neural network is an interconnected group of nodes, akin to the vast network of neurons in a brain. Here, each circular node represents an artificial neuron and an arrow represents a connection from the output of one artificial neuron to the input of another.

Decision trees

Decision tree learning uses a decision tree as a predictive model to go from observations about an item (represented in the branches) to conclusions about the item's target value (represented in the leaves). It is one of the predictive modeling approaches used in statistics, data mining, and machine learning. Tree models where the target variable can take a

discrete set of values are called classification trees; in these tree structures, leaves represent class labels and branches represent conjunctions of features that lead to those class labels. Decision trees where the target variable can take continuous values (typically real numbers) are called regression trees. In decision analysis, a decision tree can be used to visually and explicitly represent decisions and decision making. In data mining, a decision tree describes data, but the resulting classification tree can be an input for decision making.

Support vector machines

Support vector machines (SVMs), also known as support vector networks, are a set of related supervised learning methods used for classification and regression. Given a set of training examples, each marked as belonging to one of two categories, an SVM training algorithm builds a model that predicts whether a new example falls into one category or the other. An SVM training algorithm is a non-probabilistic, binary, linear classifier, although methods such as Platt scaling exist to use SVM in a probabilistic classification setting. In addition to performing linear classification, SVMs can efficiently perform a non-linear classification using what is called the kernel trick, implicitly mapping their inputs into high-dimensional feature spaces.

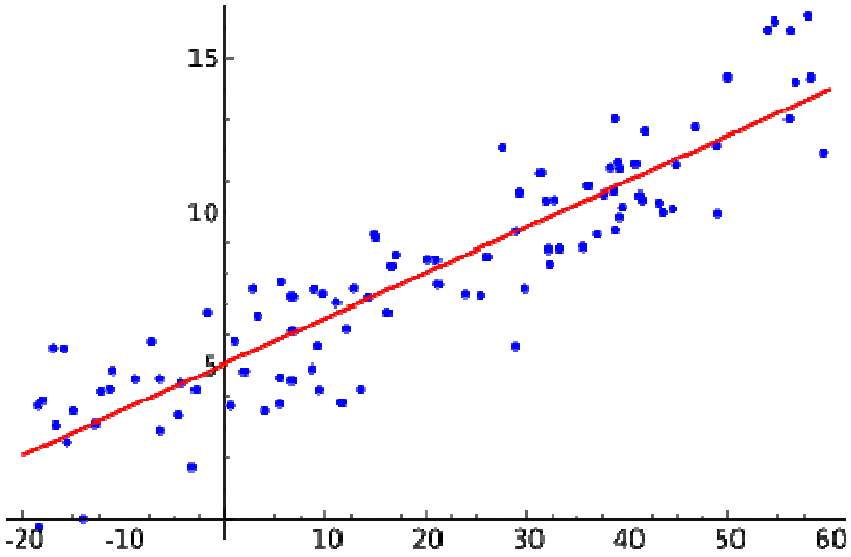


Fig: Illustration of linear regression on a data set.

Regression analysis

Regression analysis encompasses a large variety of statistical methods to estimate the relationship between input variables and their associated features. Its most common form is linear regression, where a single line is drawn to best fit the given data according to a mathematical criterion such as ordinary least squares. The latter is often extended by regularization (mathematics) methods to mitigate overfitting and bias, as in ridge regression. When dealing with non-linear problems, go-to models include polynomial regression (for example, used for trendline fitting in Microsoft Excel), logistic regression (often used in statistical classification) or even kernel regression, which introduces non-linearity by taking advantage of the kernel trick to implicitly map input variables to higher-dimensional space.

Bayesian networks

A Bayesian network, belief network, or directed acyclic graphical model is a probabilistic graphical model that represents a set of random variables and their conditional independence with a directed acyclic graph (DAG). For example, a Bayesian network could represent the probabilistic relationships between diseases and symptoms. Given symptoms, the network can be used to compute the probabilities of the presence of various diseases. Efficient algorithms exist that perform inference and learning. Bayesian networks that model sequences of variables, like speech signals or protein sequences, are called dynamic Bayesian networks. Generalizations of Bayesian networks that can represent and solve decision problems under uncertainty are called influence diagrams.

Genetic algorithms

A genetic algorithm (GA) is a search algorithm and heuristic technique that mimics the process of natural selection, using methods such as mutation and crossover to generate new genotypes in the hope of finding good solutions to a given problem. In machine learning, genetic algorithms were used in the 1980s and 1990s. Conversely, machine learning techniques have been used to improve the performance of genetic and evolutionary algorithms.

Training models

Usually, machine learning models require a lot of data in order for them to perform well. Usually, when training a machine

learning model, one needs to collect a large, representative sample of data from a training set. Data from the training set can be as varied as a corpus of text, a collection of images, and data collected from individual users of a service. Overfitting is something to watch out for when training a machine learning model. Trained models derived from biased data can result in skewed or undesired predictions. Algorithmic bias is a potential result from data not fully prepared for training.

Federated learning

Federated learning is an adapted form of distributed artificial intelligence to training machine learning models that decentralizes the training process, allowing for users' privacy to be maintained by not needing to send their data to a centralized server. This also increases efficiency by decentralizing the training process to many devices. For example, Gboard uses federated machine learning to train search query prediction models on users' mobile phones without having to send individual searches back to Google.

Applications

There are many applications for machine learning, including:

- Agriculture
- Anatomy
- Adaptive websites
- Affective computing
- Banking

- Bioinformatics
- Brain–machine interfaces
- Cheminformatics
- Citizen science
- Computer networks
- Computer vision
- Credit-card fraud detection
- Data quality
- DNA sequence classification
- Economics
- Financial market analysis
- General game playing
- Handwriting recognition
- Information retrieval
- Insurance
- Internet fraud detection
- Linguistics
- Machine learning control
- Machine perception
- Machine translation
- Marketing
- Medical diagnosis
- Natural language processing
- Natural language understanding
- Online advertising
- Optimization
- Recommender systems
- Robot locomotion
- Search engines
- Sentiment analysis

- Sequence mining
- Software engineering
- Speech recognition
- Structural health monitoring
- Syntactic pattern recognition
- Telecommunication
- Theorem proving
- Time series forecasting
- User behavior analytics

In 2006, the media-services provider Netflix held the first "Netflix Prize" competition to find a program to better predict user preferences and improve the accuracy of its existing Cinematch movie recommendation algorithm by at least 10%. A joint team made up of researchers from AT&T Labs-Research in collaboration with the teams Big Chaos and Pragmatic Theory built an ensemble model to win the Grand Prize in 2009 for $1 million. Shortly after the prize was awarded, Netflix realized that viewers' ratings were not the best indicators of their viewing patterns ("everything is a recommendation") and they changed their recommendation engine accordingly. In 2010 The Wall Street Journal wrote about the firm Rebellion Research and their use of machine learning to predict the financial crisis. In 2012, co-founder of Sun Microsystems, Vinod Khosla, predicted that 80% of medical doctors' jobs would be lost in the next two decades to automated machine learning medical diagnostic software. In 2014, it was reported that a machine learning algorithm had been applied in the field of art history to study fine art paintings and that it may have revealed previously unrecognized influences among artists. In 2019 Springer Nature published the first research book created using machine learning.

Machine Learning based Mobile Applications

Mobile applications based on machine learning are reshaping and affecting many aspects of our lives.

- Challenges
  - computational power
  - energy
  - latency
  - low memory
  - privacy risks
- Application Architectures
  - *Cloud inference without training* The mobile application sends a request to the cloud through an application programming interface (API) together with the new data, and the service returns a prediction.
  - *Both inference and training in the cloud*
  - *On-device inference with pre-trained models*
  - *Both inference and training on device*
  - *Hybrid Architecture*

Limitations

Although machine learning has been transformative in some fields, machine-learning programs often fail to deliver expected results. Reasons for this are numerous: lack of (suitable) data, lack of access to the data, data bias, privacy problems, badly chosen tasks and algorithms, wrong tools and people, lack of resources, and evaluation problems.

In 2018, a self-driving car from Uber failed to detect a pedestrian, who was killed after a collision. Attempts to use machine learning in healthcare with the IBM Watson system failed to deliver even after years of time and billions of dollars invested.

Bias

Machine learning approaches in particular can suffer from different data biases. A machine learning system trained on current customers only may not be able to predict the needs of new customer groups that are not represented in the training data. When trained on man-made data, machine learning is likely to pick up the same constitutional and unconscious biases already present in society. Language models learned from data have been shown to contain human-like biases. Machine learning systems used for criminal risk assessment have been found to be biased against black people. In 2015, Google photos would often tag black people as gorillas, and in 2018 this still was not well resolved, but Google reportedly was still using the workaround to remove all gorillas from the training data, and thus was not able to recognize real gorillas at all. Similar issues with recognizing non-white people have been found in many other systems. In 2016, Microsoft tested a chatbot that learned from Twitter, and it quickly picked up racist and sexist language. Because of such challenges, the effective use of machine learning may take longer to be adopted in other domains. Concern for fairness in machine learning, that is, reducing bias in machine learning and propelling its use for human good is increasingly expressed by artificial intelligence scientists, including Fei-Fei Li, who reminds engineers that "There’s nothing artificial about AI...It’s

inspired by people, it's created by people, and—most importantly—it impacts people. It is a powerful tool we are only just beginning to understand, and that is a profound responsibility."

Model assessments

Classification of machine learning models can be validated by accuracy estimation techniques like the holdout method, which splits the data in a training and test set (conventionally 2/3 training set and 1/3 test set designation) and evaluates the performance of the training model on the test set. In comparison, the K-fold-cross-validation method randomly partitions the data into K subsets and then K experiments are performed each respectively considering 1 subset for evaluation and the remaining K-1 subsets for training the model. In addition to the holdout and cross-validation methods, bootstrap, which samples n instances with replacement from the dataset, can be used to assess model accuracy.

In addition to overall accuracy, investigators frequently report sensitivity and specificity meaning True Positive Rate (TPR) and True Negative Rate (TNR) respectively. Similarly, investigators sometimes report the false positive rate (FPR) as well as the false negative rate (FNR). However, these rates are ratios that fail to reveal their numerators and denominators. The total operating characteristic (TOC) is an effective method to express a model's diagnostic ability. TOC shows the numerators and denominators of the previously mentioned rates, thus TOC provides more information than the commonly

used receiver operating characteristic (ROC) and ROC's associated area under the curve (AUC).

Ethics

Machine learning poses a host of ethical questions. Systems which are trained on datasets collected with biases may exhibit these biases upon use (algorithmic bias), thus digitizing cultural prejudices. For example, using job hiring data from a firm with racist hiring policies may lead to a machine learning system duplicating the bias by scoring job applicants against similarity to previous successful applicants. Responsible collection of data and documentation of algorithmic rules used by a system thus is a critical part of machine learning.

The evolvement of AI systems raises a lot questions in the realm of ethics and morality. AI can be well equipped in making decisions in certain fields such technical and scientific which rely heavily on data and historical information. These decisions rely on objectivity and logical reasoning. Because human languages contain biases, machines trained on language *corpora* will necessarily also learn these biases.

Other forms of ethical challenges, not related to personal biases, are more seen in health care. There are concerns among health care professionals that these systems might not be designed in the public's interest but as income-generating machines. This is especially true in the United States where there is a long-standing ethical dilemma of improving health care, but also increasing profits. For example, the algorithms could be designed to provide patients with unnecessary tests

or medication in which the algorithm's proprietary owners hold stakes. There is huge potential for machine learning in health care to provide professionals a great tool to diagnose, medicate, and even plan recovery paths for patients, but this will not happen until the personal biases mentioned previously, and these "greed" biases are addressed.

Hardware

Since the 2010s, advances in both machine learning algorithms and computer hardware have led to more efficient methods for training deep neural networks (a particular narrow subdomain of machine learning) that contain many layers of non-linear hidden units. By 2019, graphic processing units (GPUs), often with AI-specific enhancements, had displaced CPUs as the dominant method of training large-scale commercial cloud AI. OpenAI estimated the hardware compute used in the largest deep learning projects from AlexNet (2012) to AlphaZero (2017), and found a 300,000-fold increase in the amount of compute required, with a doubling-time trendline of 3.4 months.

Software

Software suites containing a variety of machine learning algorithms include the following:

Free and open-source software

- Caffe

- CNTK
- Deeplearning4j
- DeepSpeed
- ELKI
- Infer.NET
- Keras
- Mahout
- Mallet
- ML.NET
- XGBoost
- LightGBM
- mlpack
- MXNet
- Neural Lab
- Octave
- OpenNN
- Orange
- ROOT (TMVA with ROOT)
- scikit-learn
- Shogun
- Spark MLlib
- SystemML
- TensorFlow
- Torch / PyTorch
- Weka / MOA
- Yooreeka
- pandas (software)

Proprietary software with free and open-source editions

- KNIME
- RapidMiner

Proprietary software

- Amazon Machine Learning
- Angoss KnowledgeSTUDIO
- Azure Machine Learning
- Ayasdi
- IBM Watson Studio
- Google Prediction API
- IBM SPSS Modeler
- KXEN Modeler
- LIONsolver
- Mathematica
- MATLAB
- Neural Designer
- NeuroSolutions
- Oracle Data Mining
- Oracle AI Platform Cloud Service
- RCASE
- SAS Enterprise Miner
- SequenceL
- Splunk
- STATISTICA Data Miner

Neural network

A **neural network** is a network or circuit of neurons, or in a modern sense, an artificial neural network, composed of artificial neurons or nodes. Thus a neural network is either a biological neural network, made up of real biological neurons, or an artificial neural network, for solving artificial intelligence (AI) problems. The connections of the biological neuron are modeled as weights. A positive weight reflects an excitatory connection, while negative values mean inhibitory connections. All inputs are modified by a weight and summed. This activity is referred to as a linear combination. Finally, an activation function controls the amplitude of the output. For example, an acceptable range of output is usually between 0 and 1, or it could be −1 and 1.

These artificial networks may be used for predictive modeling, adaptive control and applications where they can be trained via a dataset. Self-learning resulting from experience can occur within networks, which can derive conclusions from a complex and seemingly unrelated set of information.

A biological neural network is composed of a groups of chemically connected or functionally associated neurons. A single neuron may be connected to many other neurons and the total number of neurons and connections in a network may be extensive. Connections, called synapses, are usually formed from axons to dendrites, though dendrodendritic synapses and other connections are possible. Apart from the electrical signaling, there are other forms of signaling that arise from neurotransmitter diffusion.

Artificial intelligence, cognitive modeling, and neural networks are information processing paradigms inspired by the way biological neural systems process data. Artificial intelligence and cognitive modeling try to simulate some properties of biological neural networks. In the artificial intelligence field, artificial neural networks have been applied successfully to speech recognition, image analysis and adaptive control, in order to construct software agents (in computer and video games) or autonomous robots.

Historically, digital computers evolved from the von Neumann model, and operate via the execution of explicit instructions via access to memory by a number of processors. On the other hand, the origins of neural networks are based on efforts to model information processing in biological systems. Unlike the von Neumann model, neural network computing does not separate memory and processing.

Neural network theory has served both to better identify how the neurons in the brain function and to provide the basis for efforts to create artificial intelligence.

History

The preliminary theoretical base for contemporary neural networks was independently proposed by Alexander Bain (1873) and William James (1890). In their work, both thoughts and body activity resulted from interactions among neurons within the brain.

For Bain, every activity led to the firing of a certain set of neurons. When activities were repeated, the connections between those neurons strengthened. According to his theory, this repetition was what led to the formation of memory. The general scientific community at the time was skeptical of Bain's theory because it required what appeared to be an inordinate number of neural connections within the brain. It is now apparent that the brain is exceedingly complex and that the same brain “wiring” can handle multiple problems and inputs.

James's theory was similar to Bain's, however, he suggested that memories and actions resulted from electrical currents flowing among the neurons in the brain. His model, by focusing on the flow of electrical currents, did not require individual neural connections for each memory or action.

C. S. Sherrington (1898) conducted experiments to test James's theory. He ran electrical currents down the spinal cords of rats. However, instead of demonstrating an increase in electrical current as projected by James, Sherrington found that the electrical current strength decreased as the testing continued over time. Importantly, this work led to the discovery of the concept of habituation.

McCulloch and Pitts (1943) created a computational model for neural networks based on mathematics and algorithms. They called this model threshold logic. The model paved the way for neural network research to split into two distinct approaches. One approach focused on biological processes in the brain and the other focused on the application of neural networks to artificial intelligence.

In the late 1940s psychologist Donald Hebb created a hypothesis of learning based on the mechanism of neural plasticity that is now known as Hebbian learning. Hebbian learning is considered to be a 'typical' unsupervised learning rule and its later variants were early models for long term potentiation. These ideas started being applied to computational models in 1948 with Turing's B-type machines.

Farley and Clark (1954) first used computational machines, then called calculators, to simulate a Hebbian network at MIT. Other neural network computational machines were created by Rochester, Holland, Habit, and Duda (1956).

Rosenblatt (1958) created the perceptron, an algorithm for pattern recognition based on a two-layer learning computer network using simple addition and subtraction. With mathematical notation, Rosenblatt also described circuitry not in the basic perceptron, such as the exclusive-or circuit, a circuit whose mathematical computation could not be processed until after the backpropagation algorithm was created by Werbos (1975).

Neural network research stagnated after the publication of machine learning research by Marvin Minsky and Seymour Papert (1969). They discovered two key issues with the computational machines that processed neural networks. The first issue was that single-layer neural networks were incapable of processing the exclusive-or circuit. The second significant issue was that computers were not sophisticated enough to effectively handle the long run time required by large neural networks. Neural network research slowed until computers achieved greater processing power. Also key in later

advances was the backpropagation algorithm which effectively solved the exclusive-or problem (Werbos 1975).

The parallel distributed processing of the mid-1980s became popular under the name connectionism. The text by Rumelhart and McClelland (1986) provided a full exposition on the use of connectionism in computers to simulate neural processes.

Neural networks, as used in artificial intelligence, have traditionally been viewed as simplified models of neural processing in the brain, even though the relation between this model and brain biological architecture is debated, as it is not clear to what degree artificial neural networks mirror brain function.

Artificial intelligence

A *neural network* (NN), in the case of artificial neurons called *artificial neural network* (ANN) or *simulated neural network* (SNN), is an interconnected group of natural or artificial neurons that uses a mathematical or computational model for information processing based on a connectionistic approach to computation. In most cases an ANN is an adaptive system that changes its structure based on external or internal information that flows through the network.

In more practical terms neural networks are non-linear statistical data modeling or decision making tools. They can be used to model complex relationships between inputs and outputs or to find patterns in data.

An artificial neural network involves a network of simple processing elements (artificial neurons) which can exhibit complex global behavior, determined by the connections between the processing elements and element parameters. Artificial neurons were first proposed in 1943 by Warren McCulloch, a neurophysiologist, and Walter Pitts, a logician, who first collaborated at the University of Chicago.

One classical type of artificial neural network is the recurrent Hopfield network.

The concept of a neural network appears to have first been proposed by Alan Turing in his 1948 paper *Intelligent Machinery* in which he called them "B-type unorganised machines".

The utility of artificial neural network models lies in the fact that they can be used to infer a function from observations and also to use it. Unsupervised neural networks can also be used to learn representations of the input that capture the salient characteristics of the input distribution, e.g., see the Boltzmann machine (1983), and more recently, deep learning algorithms, which can implicitly learn the distribution function of the observed data. Learning in neural networks is particularly useful in applications where the complexity of the data or task makes the design of such functions by hand impractical.

Applications

Neural networks can be used in different fields. The tasks to which artificial neural networks are applied tend to fall within the following broad categories:

Function approximation, or regression analysis, including time series prediction and modeling.

Classification, including pattern and sequence recognition, novelty detection and sequential decision making.

Data processing, including filtering, clustering, blind signal separation and compression.

Application areas of ANNs include nonlinear system identification and control (vehicle control, process control), game-playing and decision making (backgammon, chess, racing), pattern recognition (radar systems, face identification, object recognition), sequence recognition (gesture, speech, handwritten text recognition), medical diagnosis, financial applications, data mining (or knowledge discovery in databases, "KDD"), visualization and e-mail spam filtering. For example, it is possible to create a semantic profile of user's interests emerging from pictures trained for object recognition.

Neuroscience

Theoretical and computational neuroscience is the field concerned with the analysis and computational modeling of

biological neural systems. Since neural systems are intimately related to cognitive processes and behaviour, the field is closely related to cognitive and behavioural modeling.

The aim of the field is to create models of biological neural systems in order to understand how biological systems work. To gain this understanding, neuroscientists strive to make a link between observed biological processes (data), biologically plausible mechanisms for neural processing and learning (biological neural network models) and theory (statistical learning theory and information theory).

Types of models

Many models are used; defined at different levels of abstraction, and modeling different aspects of neural systems. They range from models of the short-term behaviour of individual neurons, through models of the dynamics of neural circuitry arising from interactions between individual neurons, to models of behaviour arising from abstract neural modules that represent complete subsystems. These include models of the long-term and short-term plasticity of neural systems and its relation to learning and memory, from the individual neuron to the system level.

Connectivity

In August 2020 scientists reported that bi-directional connections, or added appropriate feedback connections, can accelerate and improve communication between and in modular neural networks of the brain's cerebral cortex and lower the threshold for their successful communication. They showed that adding feedback connections between a resonance

pair can support successful propagation of a single pulse packet throughout the entire network.

Criticism

A common criticism of neural networks, particularly in robotics, is that they require a large diversity of training for real-world operation. This is not surprising, since any learning machine needs sufficient representative examples in order to capture the underlying structure that allows it to generalize to new cases. Dean Pomerleau, in his research presented in the paper "Knowledge-based Training of Artificial Neural Networks for Autonomous Robot Driving," uses a neural network to train a robotic vehicle to drive on multiple types of roads (single lane, multi-lane, dirt, etc.). A large amount of his research is devoted to (1) extrapolating multiple training scenarios from a single training experience, and (2) preserving past training diversity so that the system does not become overtrained (if, for example, it is presented with a series of right turns—it should not learn to always turn right). These issues are common in neural networks that must decide from amongst a wide variety of responses, but can be dealt with in several ways, for example by randomly shuffling the training examples, by using a numerical optimization algorithm that does not take too large steps when changing the network connections following an example, or by grouping examples in so-called mini-batches.

A. K. Dewdney, a former *Scientific American* columnist, wrote in 1997, "Although neural nets do solve a few toy problems, their powers of computation are so limited that I am surprised

anyone takes them seriously as a general problem-solving tool" (Dewdney, p. 82).

Arguments for Dewdney's position are that to implement large and effective software neural networks, much processing and storage resources need to be committed. While the brain has hardware tailored to the task of processing signals through a graph of neurons, simulating even a most simplified form on Von Neumann technology may compel a neural network designer to fill many millions of database rows for its connections—which can consume vast amounts of computer memory and hard disk space. Furthermore, the designer of neural network systems will often need to simulate the transmission of signals through many of these connections and their associated neurons—which must often be matched with incredible amounts of CPU processing power and time. While neural networks often yield *effective* programs, they too often do so at the cost of *efficiency* (they tend to consume considerable amounts of time and money).

Arguments against Dewdney's position are that neural nets have been successfully used to solve many complex and diverse tasks, such as autonomously flying aircraft.

Technology writer Roger Bridgman commented on Dewdney's statements about neural nets:

Neural networks, for instance, are in the dock not only because they have been hyped to high heaven, (what hasn't?) but also because you could create a successful net without understanding how it worked: the bunch of numbers that captures its behaviour would in all probability be "an opaque, unreadable table...valueless as a scientific resource".

In spite of his emphatic declaration that science is not technology, Dewdney seems here to pillory neural nets as bad science when most of those devising them are just trying to be good engineers. An unreadable table that a useful machine could read would still be well worth having.

Although it is true that analyzing what has been learned by an artificial neural network is difficult, it is much easier to do so than to analyze what has been learned by a biological neural network. Moreover, recent emphasis on the explainability of AI has contributed towards the development of methods, notably those based on attention mechanisms, for visualizing and explaining learned neural networks. Furthermore, researchers involved in exploring learning algorithms for neural networks are gradually uncovering generic principles that allow a learning machine to be successful. For example, Bengio and LeCun (2007) wrote an article regarding local vs non-local learning, as well as shallow vs deep architecture.

Some other criticisms came from believers of hybrid models (combining neural networks and symbolic approaches). They advocate the intermix of these two approaches and believe that hybrid models can better capture the mechanisms of the human mind (Sun and Bookman, 1990).

Recent improvements

While initially research had been concerned mostly with the electrical characteristics of neurons, a particularly important part of the investigation in recent years has been the

exploration of the role of neuromodulators such as dopamine, acetylcholine, and serotonin on behaviour and learning.

Biophysical models, such as BCM theory, have been important in understanding mechanisms for synaptic plasticity, and have had applications in both computer science and neuroscience. Research is ongoing in understanding the computational algorithms used in the brain, with some recent biological evidence for radial basis networks and neural backpropagation as mechanisms for processing data.

Computational devices have been created in CMOS for both biophysical simulation and neuromorphic computing. More recent efforts show promise for creating nanodevices for very large scale principal components analyses and convolution. If successful, these efforts could usher in a new era of neural computing that is a step beyond digital computing, because it depends on learning rather than programming and because it is fundamentally analog rather than digital even though the first instantiations may in fact be with CMOS digital devices.

Between 2009 and 2012, the recurrent neural networks and deep feedforward neural networks developed in the research group of Jürgen Schmidhuber at the Swiss AI Lab IDSIA have won eight international competitions in pattern recognition and machine learning. For example, multi-dimensional long short term memory (LSTM) won three competitions in connected handwriting recognition at the 2009 International Conference on Document Analysis and Recognition (ICDAR), without any prior knowledge about the three different languages to be learned.

Variants of the back-propagation algorithm as well as unsupervised methods by Geoff Hinton and colleagues at the University of Toronto can be used to train deep, highly nonlinear neural architectures, similar to the 1980 Neocognitron by Kunihiko Fukushima, and the "standard architecture of vision", inspired by the simple and complex cells identified by David H. Hubel and Torsten Wiesel in the primary visual cortex.

Radial basis function and wavelet networks have also been introduced. These can be shown to offer best approximation properties and have been applied in nonlinear system identification and classification applications.

Deep learning feedforward networks alternate convolutional layers and max-pooling layers, topped by several pure classification layers. Fast GPU-based implementations of this approach have won several pattern recognition contests, including the IJCNN 2011 Traffic Sign Recognition Competition and the ISBI 2012 Segmentation of Neuronal Structures in Electron Microscopy Stacks challenge. Such neural networks also were the first artificial pattern recognizers to achieve human-competitive or even superhuman performance on benchmarks such as traffic sign recognition (IJCNN 2012), or the MNIST handwritten digits problem of Yann LeCun and colleagues at NYU.

Hybrid neural network

The term **hybrid neural network** can have two meanings:

biological neural networks interacting with artificial neuronal models, and

Artificial neural networks with a symbolic part (or, conversely, symbolic computations with a connectionist part).

As for the first meaning, the artificial neurons and synapses in hybrid networks can be digital or analog. For the digital variant voltage clamps are used to monitor the membrane potential of neurons, to computationally simulate artificial neurons and synapses and to stimulate biological neurons by inducing synaptic. For the analog variant, specially designed electronic circuits connect to a network of living neurons through electrodes.

As for the second meaning, incorporating elements of symbolic computation and artificial neural networks into one model was an attempt to combine the advantages of both paradigms while avoiding the shortcomings. Symbolic representations have advantages with respect to explicit, direct control, fast initial coding, dynamic variable binding and knowledge abstraction. Representations of artificial neural networks, on the other hand, show advantages for biological plausibility, learning, robustness (fault-tolerant processing and graceful decay), and generalization to similar input. Since the early 1990s many attempts have been made to reconcile the two approaches.

Recurrent neural network

A **recurrent neural network** (**RNN**) is a class of artificial neural networks where connections between nodes form a

directed graph along a temporal sequence. This allows it to exhibit temporal dynamic behavior. Derived from feedforward neural networks, RNNs can use their internal state (memory) to process variable length sequences of inputs. This makes them applicable to tasks such as unsegmented, connected handwriting recognition or speech recognition.

The term “recurrent neural network” is used indiscriminately to refer to two broad classes of networks with a similar general structure, where one is finite impulse and the other is infinite impulse. Both classes of networks exhibit temporal dynamic behavior. A finite impulse recurrent network is a directed acyclic graph that can be unrolled and replaced with a strictly feedforward neural network, while an infinite impulse recurrent network is a directed cyclic graph that can not be unrolled.

Both finite impulse and infinite impulse recurrent networks can have additional stored states, and the storage can be under direct control by the neural network. The storage can also be replaced by another network or graph, if that incorporates time delays or has feedback loops. Such controlled states are referred to as gated state or gated memory, and are part of long short-term memory networks (LSTMs) and gated recurrent units. This is also called Feedback Neural Network (FNN).

History

Recurrent neural networks were based on David Rumelhart's work in 1986. Hopfield networks - a special kind of RNN - were

discovered by John Hopfield in 1982. In 1993, a neural history compressor system solved a “Very Deep Learning” task that required more than 1000 subsequent layers in an RNN unfolded in time.

LSTM

Long short-term memory (LSTM) networks were invented by Hochreiter and Schmidhuber in 1997 and set accuracy records in multiple applications domains.

Around 2007, LSTM started to revolutionize speech recognition, outperforming traditional models in certain speech applications. In 2009, a Connectionist Temporal Classification (CTC)-trained LSTM network was the first RNN to win pattern recognition contests when it won several competitions in connected handwriting recognition. In 2014, the Chinese search giant Baidu used CTC-trained RNNs to break the Switchboard Hub5'00 speech recognition dataset benchmark without using any traditional speech processing methods.

LSTM also improved large-vocabulary speech recognition and text-to-speech synthesis and was used in Google Android. In 2015, Google's speech recognition reportedly experienced a dramatic performance jump of 49% through CTC-trained LSTM.

LSTM broke records for improved machine translation, Language Modeling and Multilingual Language Processing. LSTM combined with convolutional neural networks (CNNs) improved automatic image captioning. Given the computation and memory overheads of running LSTMs, there have been efforts on accelerating LSTM using hardware accelerators.

Architectures

RNNs come in many variants.

Fully recurrent

Basic RNNs are a network of neuron-like nodes organized into successive layers. Each node in a given layer is connected with a directed (one-way) connection to every other node in the next successive layer. Each node (neuron) has a time-varying real-valued activation. Each connection (synapse) has a modifiable real-valued weight. Nodes are either input nodes (receiving data from outside of the network), output nodes (yielding results), or hidden nodes (that modify the data *en route* from input to output).

For supervised learning in discrete time settings, sequences of real-valued input vectors arrive at the input nodes, one vector at a time. At any given time step, each non-input unit computes its current activation (result) as a nonlinear function of the weighted sum of the activations of all units that connect to it. Supervisor-given target activations can be supplied for some output units at certain time steps. For example, if the input sequence is a speech signal corresponding to a spoken digit, the final target output at the end of the sequence may be a label classifying the digit.

In reinforcement learning settings, no teacher provides target signals. Instead, a fitness function or reward function is occasionally used to evaluate the RNN's performance, which influences its input stream through output units connected to

actuators that affect the environment. This might be used to play a game in which progress is measured with the number of points won.

Each sequence produces an error as the sum of the deviations of all target signals from the corresponding activations computed by the network. For a training set of numerous sequences, the total error is the sum of the errors of all individual sequences.

Elman networks and Jordan networks

An Elman network is a three-layer network (arranged horizontally as x, y, and z in the illustration) with the addition of a set of context units (u in the illustration). The middle (hidden) layer is connected to these context units fixed with a weight of one. At each time step, the input is fed forward and a learning rule is applied. The fixed back-connections save a copy of the previous values of the hidden units in the context units (since they propagate over the connections before the learning rule is applied). Thus the network can maintain a sort of state, allowing it to perform such tasks as sequence-prediction that are beyond the power of a standard multilayer perceptron.

Jordan networks are similar to Elman networks. The context units are fed from the output layer instead of the hidden layer. The context units in a Jordan network are also referred to as the state layer. They have a recurrent connection to themselves.

Hopfield

The Hopfield network is an RNN in which all connections are symmetric. It requires stationary inputs and is thus not a general RNN, as it does not process sequences of patterns. It guarantees that it will converge. If the connections are trained using Hebbian learning then the Hopfield network can perform as robust content-addressable memory, resistant to connection alteration.

Bidirectional associative memory

Introduced by Bart Kosko, a bidirectional associative memory (BAM) network is a variant of a Hopfield network that stores associative data as a vector. The bi-directionality comes from passing information through a matrix and its transpose. Typically, bipolar encoding is preferred to binary encoding of the associative pairs. Recently, stochastic BAM models using Markov stepping were optimized for increased network stability and relevance to real-world applications.

A BAM network has two layers, either of which can be driven as an input to recall an association and produce an output on the other layer.

Echo state

The echo state network (ESN) has a sparsely connected random hidden layer. The weights of output neurons are the only part of the network that can change (be trained). ESNs are good at reproducing certain time series. A variant for spiking neurons is known as a liquid state machine.

Independently RNN (IndRNN)

The Independently recurrent neural network (IndRNN) addresses the gradient vanishing and exploding problems in the traditional fully connected RNN. Each neuron in one layer only receives its own past state as context information (instead of full connectivity to all other neurons in this layer) and thus neurons are independent of each other's history. The gradient backpropagation can be regulated to avoid gradient vanishing and exploding in order to keep long or short-term memory. The cross-neuron information is explored in the next layers. IndRNN can be robustly trained with the non-saturated nonlinear functions such as ReLU. Using skip connections, deep networks can be trained.

Recursive

A recursive neural network is created by applying the same set of weights recursively over a differentiable graph-like structure by traversing the structure in topological order. Such networks are typically also trained by the reverse mode of automatic differentiation. They can process distributed representations of structure, such as logical terms. A special case of recursive neural networks is the RNN whose structure corresponds to a linear chain. Recursive neural networks have been applied to natural language processing. The Recursive Neural Tensor Network uses a tensor-based composition function for all nodes in the tree.

Neural history compressor

The neural history compressor is an unsupervised stack of RNNs. At the input level, it learns to predict its next input

from the previous inputs. Only unpredictable inputs of some RNN in the hierarchy become inputs to the next higher level RNN, which therefore recomputes its internal state only rarely. Each higher level RNN thus studies a compressed representation of the information in the RNN below. This is done such that the input sequence can be precisely reconstructed from the representation at the highest level.

The system effectively minimises the description length or the negative logarithm of the probability of the data. Given a lot of learnable predictability in the incoming data sequence, the highest level RNN can use supervised learning to easily classify even deep sequences with long intervals between important events.

It is possible to distill the RNN hierarchy into two RNNs: the "conscious" chunker (higher level) and the "subconscious" automatizer (lower level). Once the chunker has learned to predict and compress inputs that are unpredictable by the automatizer, then the automatizer can be forced in the next learning phase to predict or imitate through additional units the hidden units of the more slowly changing chunker. This makes it easy for the automatizer to learn appropriate, rarely changing memories across long intervals. In turn this helps the automatizer to make many of its once unpredictable inputs predictable, such that the chunker can focus on the remaining unpredictable events.

A generative model partially overcame the vanishing gradient problem of automatic differentiation or backpropagation in neural networks in 1992. In 1993, such a system solved a

"Very Deep Learning" task that required more than 1000 subsequent layers in an RNN unfolded in time.

Second order RNNs

Second order RNNs use higher order weights instead of the standard weights, and states can be a product. This allows a direct mapping to a finite state machine both in training, stability, and representation. Long short-term memory is an example of this but has no such formal mappings or proof of stability.

Long short-term memory

Long short-term memory (LSTM) is a deep learning system that avoids the vanishing gradient problem. LSTM is normally augmented by recurrent gates called "forget gates". LSTM prevents backpropagated errors from vanishing or exploding. Instead, errors can flow backwards through unlimited numbers of virtual layers unfolded in space. That is, LSTM can learn tasks that require memories of events that happened thousands or even millions of discrete time steps earlier. Problem-specific LSTM-like topologies can be evolved. LSTM works even given long delays between significant events and can handle signals that mix low and high frequency components.

Many applications use stacks of LSTM RNNs and train them by Connectionist Temporal Classification (CTC) to find an RNN weight matrix that maximizes the probability of the label sequences in a training set, given the corresponding input sequences. CTC achieves both alignment and recognition.

LSTM can learn to recognize context-sensitive languages unlike previous models based on hidden Markov models (HMM) and similar concepts.

Gated recurrent unit

Gated recurrent units (GRUs) are a gating mechanism in recurrent neural networks introduced in 2014. They are used in the full form and several simplified variants. Their performance on polyphonic music modeling and speech signal modeling was found to be similar to that of long short-term memory. They have fewer parameters than LSTM, as they lack an output gate.

Bi-directional

Bi-directional RNNs use a finite sequence to predict or label each element of the sequence based on the element's past and future contexts. This is done by concatenating the outputs of two RNNs, one processing the sequence from left to right, the other one from right to left. The combined outputs are the predictions of the teacher-given target signals. This technique has been proven to be especially useful when combined with LSTM RNNs.

Hierarchical

Hierarchical RNNs connect their neurons in various ways to decompose hierarchical behavior into useful subprograms.

Recurrent multilayer perceptron network

Generally, a recurrent multilayer perceptron network (RMLP) network consists of cascaded subnetworks, each of which contains multiple layers of nodes. Each of these subnetworks is feed-forward except for the last layer, which can have feedback connections. Each of these subnets is connected only by feed forward connections.

Multiple timescales model

A multiple timescales recurrent neural network (MTRNN) is a neural-based computational model that can simulate the functional hierarchy of the brain through self-organization that depends on spatial connection between neurons and on distinct types of neuron activities, each with distinct time properties. With such varied neuronal activities, continuous sequences of any set of behaviors are segmented into reusable primitives, which in turn are flexibly integrated into diverse sequential behaviors. The biological approval of such a type of hierarchy was discussed in the memory-prediction theory of brain function by Hawkins in his book *On Intelligence*.

Neural Turing machines

Neural Turing machines (NTMs) are a method of extending recurrent neural networks by coupling them to external memory resources which they can interact with by attentional processes. The combined system is analogous to a Turing machine or Von Neumann architecture but is differentiable end-to-end, allowing it to be efficiently trained with gradient descent.

Differentiable neural computer

Differentiable neural computers (DNCs) are an extension of Neural Turing machines, allowing for usage of fuzzy amounts of each memory address and a record of chronology.

Neural network pushdown automata

Neural network pushdown automata (NNPDA) are similar to NTMs, but tapes are replaced by analogue stacks that are differentiable and that are trained. In this way, they are similar in complexity to recognizers of context free grammars (CFGs).

Memristive Networks

Greg Snider of HP Labs describes a system of cortical computing with memristive nanodevices. The memristors (memory resistors) are implemented by thin film materials in which the resistance is electrically tuned via the transport of ions or oxygen vacancies within the film. DARPA's SyNAPSE project has funded IBM Research and HP Labs, in collaboration with the Boston University Department of Cognitive and Neural Systems (CNS), to develop neuromorphic architectures which may be based on memristive systems. Memristive networks are a particular type of physical neural network that have very similar properties to (Little-)Hopfield networks, as they have a continuous dynamics, have a limited memory capacity and they natural relax via the minimization of a function which is asymptotic to the Ising model. In this sense, the dynamics of a memristive circuit has the advantage compared to a Resistor-Capacitor network to have a more interesting non-linear behavior. From this point of view, engineering an analog

memristive networks accounts to a peculiar type of neuromorphic engineering in which the device behavior depends on the circuit wiring, or topology.

Training

Gradient descent

Gradient descent is a first-order iterative optimization algorithm for finding the minimum of a function. In neural networks, it can be used to minimize the error term by changing each weight in proportion to the derivative of the error with respect to that weight, provided the non-linear activation functions are differentiable. Various methods for doing so were developed in the 1980s and early 1990s by Werbos, Williams, Robinson, Schmidhuber, Hochreiter, Pearlmutter and others.

The standard method is called “backpropagation through time” or BPTT, and is a generalization of back-propagation for feed-forward networks. Like that method, it is an instance of automatic differentiation in the reverse accumulation mode of Pontryagin's minimum principle. A more computationally expensive online variant is called “Real-Time Recurrent Learning” or RTRL, which is an instance of automatic differentiation in the forward accumulation mode with stacked tangent vectors. Unlike BPTT, this algorithm is local in time but not local in space.

In this context, local in space means that a unit's weight vector can be updated using only information stored in the

connected units and the unit itself such that update complexity of a single unit is linear in the dimensionality of the weight vector. Local in time means that the updates take place continually (on-line) and depend only on the most recent time step rather than on multiple time steps within a given time horizon as in BPTT. Biological neural networks appear to be local with respect to both time and space.

For recursively computing the partial derivatives, RTRL has a time-complexity of O(number of hidden x number of weights) per time step for computing the Jacobian matrices, while BPTT only takes O(number of weights) per time step, at the cost of storing all forward activations within the given time horizon. An online hybrid between BPTT and RTRL with intermediate complexity exists, along with variants for continuous time.

A major problem with gradient descent for standard RNN architectures is that error gradients vanish exponentially quickly with the size of the time lag between important events. LSTM combined with a BPTT/RTRL hybrid learning method attempts to overcome these problems. This problem is also solved in the independently recurrent neural network (IndRNN) by reducing the context of a neuron to its own past state and the cross-neuron information can then be explored in the following layers. Memories of different range including long-term memory can be learned without the gradient vanishing and exploding problem.

The on-line algorithm called causal recursive backpropagation (CRBP), implements and combines BPTT and RTRL paradigms for locally recurrent networks. It works with the most general locally recurrent networks. The CRBP algorithm can minimize

the global error term. This fact improves stability of the algorithm, providing a unifying view on gradient calculation techniques for recurrent networks with local feedback.

One approach to the computation of gradient information in RNNs with arbitrary architectures is based on signal-flow graphs diagrammatic derivation. It uses the BPTT batch algorithm, based on Lee's theorem for network sensitivity calculations. It was proposed by Wan and Beaufays, while its fast online version was proposed by Campolucci, Uncini and Piazza.

Global optimization methods

Training the weights in a neural network can be modeled as a non-linear global optimization problem. A target function can be formed to evaluate the fitness or error of a particular weight vector as follows: First, the weights in the network are set according to the weight vector. Next, the network is evaluated against the training sequence. Typically, the sum-squared-difference between the predictions and the target values specified in the training sequence is used to represent the error of the current weight vector. Arbitrary global optimization techniques may then be used to minimize this target function.

The most common global optimization method for training RNNs is genetic algorithms, especially in unstructured networks.

Initially, the genetic algorithm is encoded with the neural network weights in a predefined manner where one gene in the chromosome represents one weight link. The whole network is

represented as a single chromosome. The fitness function is evaluated as follows:

- Each weight encoded in the chromosome is assigned to the respective weight link of the network.
- The training set is presented to the network which propagates the input signals forward.
- The mean-squared-error is returned to the fitness function.
- This function drives the genetic selection process.

Many chromosomes make up the population; therefore, many different neural networks are evolved until a stopping criterion is satisfied. A common stopping scheme is:

- When the neural network has learnt a certain percentage of the training data or
- When the minimum value of the mean-squared-error is satisfied or
- When the maximum number of training generations has been reached.

The stopping criterion is evaluated by the fitness function as it gets the reciprocal of the mean-squared-error from each network during training. Therefore, the goal of the genetic algorithm is to maximize the fitness function, reducing the mean-squared-error.

Other global (and/or evolutionary) optimization techniques may be used to seek a good set of weights, such as simulated annealing or particle swarm optimization.

Related fields and models

RNNs may behave chaotically. In such cases, dynamical systems theory may be used for analysis.

They are in fact recursive neural networks with a particular structure: that of a linear chain. Whereas recursive neural networks operate on any hierarchical structure, combining child representations into parent representations, recurrent neural networks operate on the linear progression of time, combining the previous time step and a hidden representation into the representation for the current time step.

In particular, RNNs can appear as nonlinear versions of finite impulse response and infinite impulse response filters and also as a nonlinear autoregressive exogenous model (NARX).

Libraries

- Apache Singa
- Caffe: Created by the Berkeley Vision and Learning Center (BVLC). It supports both CPU and GPU. Developed in C++, and has Python and MATLAB wrappers.
- Chainer: The first stable deep learning library that supports dynamic, define-by-run neural networks. Fully in Python, production support for CPU, GPU, distributed training.

- Deeplearning4j: Deep learning in Java and Scala on multi-GPU-enabled Spark. A general-purpose deep learning library for the JVM production stack running on a C++ scientific computing engine. Allows the creation of custom layers. Integrates with Hadoop and Kafka.
- Dynet: The Dynamic Neural Networks toolkit.
- Flux: includes interfaces for RNNs, including GRUs and LSTMs, written in Julia.
- Keras: High-level, easy to use API, providing a wrapper to many other deep learning libraries.
- Microsoft Cognitive Toolkit
- MXNet: a modern open-source deep learning framework used to train and deploy deep neural networks.
- Paddle Paddle (https://github.com/paddlepaddle/paddle): PaddlePaddle (PArallel Distributed Deep LEarning) is a deep learning platform, which is originally developed by Baidu scientists and engineers for the purpose of applying deep learning to many products at Baidu.
- PyTorch: Tensors and Dynamic neural networks in Python with strong GPU acceleration.
- TensorFlow: Apache 2.0-licensed Theano-like library with support for CPU, GPU and Google's proprietary TPU, mobile
- Theano: The reference deep-learning library for Python with an API largely compatible with the popular NumPy library. Allows user to write symbolic mathematical expressions, then

automatically generates their derivatives, saving the user from having to code gradients or backpropagation. These symbolic expressions are automatically compiled to CUDA code for a fast, on-the-GPU implementation.

- Torch (www.torch.ch): A scientific computing framework with wide support for machine learning algorithms, written in C and lua. The main author is Ronan Collobert, and it is now used at Facebook AI Research and Twitter.

Applications

Applications of Recurrent Neural Networks include:

- Machine Translation
- Robot control
- Time series prediction
- Speech recognition
- Speech synthesis
- Time series anomaly detection
- Rhythm learning
- Music composition
- Grammar learning
- Handwriting recognition
- Human action recognition
- Protein Homology Detection
- Predicting subcellular localization of proteins
- Several prediction tasks in the area of business process management

- Prediction in medical care pathways

Perceptron

In machine learning, the **perceptron** is an algorithm for supervised learning of binary classifiers. A binary classifier is a function which can decide whether or not an input, represented by a vector of numbers, belongs to some specific class. It is a type of linear classifier, i.e. a classification algorithm that makes its predictions based on a linear predictor function combining a set of weights with the feature vector.

History

The perceptron algorithm was invented in 1958 at the Cornell Aeronautical Laboratory by Frank Rosenblatt, funded by the United States Office of Naval Research.

The perceptron was intended to be a machine, rather than a program, and while its first implementation was in software for the IBM 704, it was subsequently implemented in custom-built hardware as the "Mark 1 perceptron". This machine was designed for image recognition: it had an array of 400 photocells, randomly connected to the "neurons". Weights were encoded in potentiometers, and weight updates during learning were performed by electric motors.

In a 1958 press conference organized by the US Navy, Rosenblatt made statements about the perceptron that caused

a heated controversy among the fledgling AI community; based on Rosenblatt's statements, *The New York Times* reported the perceptron to be "the embryo of an electronic computer that [the Navy] expects will be able to walk, talk, see, write, reproduce itself and be conscious of its existence."

Although the perceptron initially seemed promising, it was quickly proved that perceptrons could not be trained to recognise many classes of patterns. This caused the field of neural network research to stagnate for many years, before it was recognised that a feedforward neural network with two or more layers (also called a multilayer perceptron) had greater processing power than perceptrons with one layer (also called a single layer perceptron).

Single layer perceptrons are only capable of learning linearly separable patterns. For a classification task with some step activation function a single node will have a single line dividing the data points forming the patterns. More nodes can create more dividing lines, but those lines must somehow be combined to form more complex classifications. A second layer of perceptrons, or even linear nodes, are sufficient to solve a lot of otherwise non-separable problems.

In 1969 a famous book entitled *Perceptrons* by Marvin Minsky and Seymour Papert showed that it was impossible for these classes of network to learn an XOR function. It is often believed (incorrectly) that they also conjectured that a similar result would hold for a multi-layer perceptron network. However, this is not true, as both Minsky and Papert already knew that multi-layer perceptrons were capable of producing an XOR function. (See the page on *Perceptrons (book)* for more

information.) Nevertheless, the often-miscited Minsky/Papert text caused a significant decline in interest and funding of neural network research. It took ten more years until neural network research experienced a resurgence in the 1980s. This text was reprinted in 1987 as "Perceptrons - Expanded Edition" where some errors in the original text are shown and corrected.

The kernel perceptron algorithm was already introduced in 1964 by Aizerman et al. Margin bounds guarantees were given for the Perceptron algorithm in the general non-separable case first by Freund and Schapire (1998), and more recently by Mohri and Rostamizadeh (2013) who extend previous results and give new L1 bounds.

The perceptron is a simplified model of a biological neuron. While the complexity of biological neuron models is often required to fully understand neural behavior, research suggests a perceptron-like linear model can produce some behavior seen in real neurons.

Variants

The pocket algorithm with ratchet (Gallant, 1990) solves the stability problem of perceptron learning by keeping the best solution seen so far "in its pocket". The pocket algorithm then returns the solution in the pocket, rather than the last solution. It can be used also for non-separable data sets, where the aim is to find a perceptron with a small number of misclassifications. However, these solutions appear purely stochastically and hence the pocket algorithm neither approaches them gradually in the course of learning, nor are

they guaranteed to show up within a given number of learning steps.

The Maxover algorithm (Wendemuth, 1995) is "robust" in the sense that it will converge regardless of (prior) knowledge of linear separability of the data set. In the linearly separable case, it will solve the training problem – if desired, even with optimal stability (maximum margin between the classes). For non-separable data sets, it will return a solution with a small number of misclassifications. In all cases, the algorithm gradually approaches the solution in the course of learning, without memorizing previous states and without stochastic jumps. Convergence is to global optimality for separable data sets and to local optimality for non-separable data sets.

The Voted Perceptron (Freund and Schapire, 1999), is a variant using multiple weighted perceptrons. The algorithm starts a new perceptron every time an example is wrongly classified, initializing the weights vector with the final weights of the last perceptron. Each perceptron will also be given another weight corresponding to how many examples do they correctly classify before wrongly classifying one, and at the end the output will be a weighted vote on all perceptrons.

In separable problems, perceptron training can also aim at finding the largest separating margin between the classes. The so-called perceptron of optimal stability can be determined by means of iterative training and optimization schemes, such as the Min-Over algorithm (Krauth and Mezard, 1987) or the AdaTron (Anlauf and Biehl, 1989)). AdaTron uses the fact that the corresponding quadratic optimization problem is convex. The perceptron of optimal stability, together with the kernel

trick, are the conceptual foundations of the support vector machine.

The *K*-perceptron further used a pre-processing layer of fixed random weights, with thresholded output units. This enabled the perceptron to classify analogue patterns, by projecting them into a binary space. In fact, for a projection space of sufficiently high dimension, patterns can become linearly separable.

Another way to solve nonlinear problems without using multiple layers is to use higher order networks (sigma-pi unit). In this type of network, each element in the input vector is extended with each pairwise combination of multiplied inputs (second order). This can be extended to an *n*-order network.

It should be kept in mind, however, that the best classifier is not necessarily that which classifies all the training data perfectly. Indeed, if we had the prior constraint that the data come from equi-variant Gaussian distributions, the linear separation in the input space is optimal, and the nonlinear solution is overfitted.

Other linear classification algorithms include Winnow, support vector machine and logistic regression.

Support vector machine

In machine learning, **support-vector machines** (**SVMs**, also **support-vector networks**) are supervised learning models with associated learning algorithms that analyze data used for

classification and regression analysis. Developed at AT&T Bell Laboratories by Vapnik with colleagues (Boser et al., 1992, Guyon et al., 1993, Vapnik et al., 1997), it presents one of the most robust prediction methods, based on the statistical learning framework or VC theory proposed by Vapnik and Chervonenkis (1974) and Vapnik (1982, 1995). Given a set of training examples, each marked as belonging to one or the other of two categories, an SVM training algorithm builds a model that assigns new examples to one category or the other, making it a non-probabilistic binary linear classifier (although methods such as Platt scaling exist to use SVM in a probabilistic classification setting). An SVM model is a representation of the examples as points in space, mapped so that the examples of the separate categories are divided by a clear gap that is as wide as possible. New examples are then mapped into that same space and predicted to belong to a category based on the side of the gap on which they fall.

In addition to performing linear classification, SVMs can efficiently perform a non-linear classification using what is called the kernel trick, implicitly mapping their inputs into high-dimensional feature spaces.

When data are unlabelled, supervised learning is not possible, and an unsupervised learning approach is required, which attempts to find natural clustering of the data to groups, and then map new data to these formed groups. The **support-vector clustering** algorithm, created by Hava Siegelmann and Vladimir Vapnik, applies the statistics of support vectors, developed in the support vector machines algorithm, to categorize unlabeled data, and is one of the most widely used clustering algorithms in industrial applications.

Applications

SVMs can be used to solve various real-world problems:

- SVMs are helpful in text and hypertext categorization, as their application can significantly reduce the need for labeled training instances in both the standard inductive and transductive settings. Some methods for shallow semantic parsing are based on support vector machines.
- Classification of images can also be performed using SVMs. Experimental results show that SVMs achieve significantly higher search accuracy than traditional query refinement schemes after just three to four rounds of relevance feedback. This is also true for image segmentation systems, including those using a modified version SVM that uses the privileged approach as suggested by Vapnik.
- Classification of satellite data like SAR data using supervised SVM.
- Hand-written characters can be recognized using SVM.
- The SVM algorithm has been widely applied in the biological and other sciences. They have been used to classify proteins with up to 90% of the compounds classified correctly. Permutation tests based on SVM weights have been suggested as a mechanism for interpretation of SVM models.

Support-vector machine weights have also been used to interpret SVM models in the past. Posthoc interpretation of support-vector machine models in order to identify features used by the model to make predictions is a relatively new area of research with special significance in the biological sciences.

History

The original SVM algorithm was invented by Vladimir N. Vapnik and Alexey Ya. Chervonenkis in 1963. In 1992, Bernhard Boser, Isabelle Guyon and Vladimir Vapnik suggested a way to create nonlinear classifiers by applying the kernel trick to maximum-margin hyperplanes. The current standard incarnation (soft margin) was proposed by Corinna Cortes and Vapnik in 1993 and published in 1995.

Implementation

The parameters of the maximum-margin hyperplane are derived by solving the optimization. There exist several specialized algorithms for quickly solving the quadratic programming (QP) problem that arises from SVMs, mostly relying on heuristics for breaking the problem down into smaller, more manageable chunks.

Another approach is to use an interior-point method that uses Newton-like iterations to find a solution of the Karush–Kuhn–

Tucker conditions of the primal and dual problems. Instead of solving a sequence of broken-down problems, this approach directly solves the problem altogether. To avoid solving a linear system involving the large kernel matrix, a low-rank approximation to the matrix is often used in the kernel trick.

Another common method is Platt's sequential minimal optimization (SMO) algorithm, which breaks the problem down into 2-dimensional sub-problems that are solved analytically, eliminating the need for a numerical optimization algorithm and matrix storage. This algorithm is conceptually simple, easy to implement, generally faster, and has better scaling properties for difficult SVM problems.

The special case of linear support-vector machines can be solved more efficiently by the same kind of algorithms used to optimize its close cousin, logistic regression; this class of algorithms includes sub-gradient descent (e.g., PEGASOS) and coordinate descent (e.g., LIBLINEAR). LIBLINEAR has some attractive training-time properties. Each convergence iteration takes time linear in the time taken to read the train data, and the iterations also have a Q-linear convergence property, making the algorithm extremely fast.

The general kernel SVMs can also be solved more efficiently using sub-gradient descent (e.g. P-packSVM), especially when parallelization is allowed.

Kernel SVMs are available in many machine-learning toolkits, including LIBSVM, MATLAB, SAS, SVMlight, kernlab, scikit-learn, Shogun, Weka, Shark, JKernelMachines, OpenCV and others.

Chapter 4

Evolutionary Computation

In computer science, **evolutionary computation** is a family of algorithms for global optimization inspired by biological evolution, and the subfield of artificial intelligence and soft computing studying these algorithms. In technical terms, they are a family of population-based trial and error problem solvers with a metaheuristic or stochastic optimization character.

In evolutionary computation, an initial set of candidate solutions is generated and iteratively updated. Each new generation is produced by stochastically removing less desired solutions, and introducing small random changes. In biological terminology, a population of solutions is subjected to natural selection (or artificial selection) and mutation. As a result, the population will gradually evolve to increase in fitness, in this case the chosen fitness function of the algorithm.

Evolutionary computation techniques can produce highly optimized solutions in a wide range of problem settings, making them popular in computer science. Many variants and extensions exist, suited to more specific families of problems and data structures. Evolutionary computation is also sometimes used in evolutionary biology as an *in silico* experimental procedure to study common aspects of general evolutionary processes.

History

The use of evolutionary principles for automated problem solving originated in the 1950s. It was not until the 1960s that three distinct interpretations of this idea started to be developed in three different places.

Evolutionary programming was introduced by Lawrence J. Fogel in the US, while John Henry Holland called his method a *genetic algorithm*. In Germany Ingo Rechenberg and Hans-Paul Schwefel introduced *evolution strategies*. These areas developed separately for about 15 years. From the early nineties on they are unified as different representatives ("dialects") of one technology, called *evolutionary computing*. Also in the early nineties, a fourth stream following the general ideas had emerged – *genetic programming*. Since the 1990s, nature-inspired algorithms are becoming an increasingly significant part of the evolutionary computation.

These terminologies denote the field of evolutionary computing and consider evolutionary programming, evolution strategies, genetic algorithms, and genetic programming as sub-areas.

The earliest computational simulations of evolution using evolutionary algorithms and artificial life techniques were performed by Nils Aall Barricelli in 1953, with first results published in 1954. Another pioneer in the 1950s was Alex Fraser, who published a series of papers on simulation of artificial selection. Artificial evolution became a widely recognised optimisation method as a result of the work of Ingo Rechenberg in the 1960s and early 1970s, who used evolution

strategies to solve complex engineering problems. Genetic algorithms in particular became popular through the writing of John Holland. As academic interest grew, dramatic increases in the power of computers allowed practical applications, including the automatic evolution of computer programs. Evolutionary algorithms are now used to solve multi-dimensional problems more efficiently than software produced by human designers, and also to optimise the design of systems.

Techniques

Evolutionary computing techniques mostly involve metaheuristic optimization algorithms. Broadly speaking, the field includes:

- Ant colony optimization
- Artificial immune systems
- Artificial life (also see digital organism)
- Cultural algorithms
- Differential evolution
- Dual-phase evolution
- Estimation of distribution algorithms
- Evolutionary algorithms
- Evolutionary programming
- Evolution strategy
- Gene expression programming
- Genetic algorithm
- Genetic programming
- Grammatical evolution

- Learnable evolution model
- Learning classifier systems
- Memetic algorithms
- Neuroevolution
- Particle swarm optimization
- Self-organization such as self-organizing maps, competitive learning
- Swarm intelligence

Evolutionary algorithms

Evolutionary algorithms form a subset of evolutionary computation in that they generally only involve techniques implementing mechanisms inspired by biological evolution such as reproduction, mutation, recombination, natural selection and survival of the fittest. Candidate solutions to the optimization problem play the role of individuals in a population, and the cost function determines the environment within which the solutions "live" (see also fitness function). Evolution of the population then takes place after the repeated application of the above operators.

In this process, there are two main forces that form the basis of evolutionary systems: **Recombinationmutation** and **crossover** create the necessary diversity and thereby facilitate novelty, while **selection** acts as a force increasing quality.

Many aspects of such an evolutionary process are stochastic. Changed pieces of information due to recombination and mutation are randomly chosen. On the other hand, selection operators can be either deterministic, or stochastic. In the

latter case, individuals with a higher fitness have a higher chance to be selected than individuals with a lower fitness, but typically even the weak individuals have a chance to become a parent or to survive.

Evolutionary algorithms and biology

Genetic algorithms deliver methods to model biological systems and systems biology that are linked to the theory of dynamical systems, since they are used to predict the future states of the system. This is just a vivid (but perhaps misleading) way of drawing attention to the orderly, well-controlled and highly structured character of development in biology.

However, the use of algorithms and informatics, in particular of computational theory, beyond the analogy to dynamical systems, is also relevant to understand evolution itself.

This view has the merit of recognizing that there is no central control of development; organisms develop as a result of local interactions within and between cells. The most promising ideas about program-development parallels seem to us to be ones that point to an apparently close analogy between processes within cells, and the low-level operation of modern computers. Thus, biological systems are like computational machines that process input information to compute next states, such that biological systems are closer to a computation than classical dynamical system.

Furthermore, following concepts from computational theory, micro processes in biological organisms are fundamentally incomplete and undecidable (completeness (logic)), implying that "there is more than a crude metaphor behind the analogy between cells and computers.

The analogy to computation extends also to the relationship between inheritance systems and biological structure, which is often thought to reveal one of the most pressing problems in explaining the origins of life.

Evolutionary automata, a generalization of *Evolutionary Turing machines*, have been introduced in order to investigate more precisely properties of biological and evolutionary computation. In particular, they allow to obtain new results on expressiveness of evolutionary computation. This confirms the initial result about undecidability of natural evolution and evolutionary algorithms and processes. *Evolutionary finite automata*, the simplest subclass of Evolutionary automata working in *terminal mode* can accept arbitrary languages over a given alphabet, including non-recursively enumerable (e.g., diagonalization language) and recursively enumerable but not recursive languages (e.g., language of the universal Turing machine).

Notable practitioners

The list of active researchers is naturally dynamic and non-exhaustive. A network analysis of the community was published in 2007.

- Kalyanmoy Deb
- Kenneth A De Jong
- Peter J. Fleming
- David B. Fogel
- Stephanie Forrest
- David E. Goldberg
- John Henry Holland
- Theo Jansen
- John Koza
- Zbigniew Michalewicz
- Melanie Mitchell
- Peter Nordin
- Riccardo Poli
- Ingo Rechenberg
- Hans-Paul Schwefel

Conferences

The main conferences in the evolutionary computation area include

- ACM Genetic and Evolutionary Computation Conference (GECCO),
- IEEE Congress on Evolutionary Computation (CEC),
- EvoStar, which comprises four conferences: EuroGP, EvoApplications, EvoCOP and EvoMUSART,
- Parallel Problem Solving from Nature (PPSN).

Evolutionary algorithm

In computational intelligence (CI), an **evolutionary algorithm** (**EA**) is a subset of evolutionary computation, a generic population-based metaheuristic optimization algorithm. An EA uses mechanisms inspired by biological evolution, such as reproduction, mutation, recombination, and selection.

Candidate solutions to the optimization problem play the role of individuals in a population, and the fitness function determines the quality of the solutions (see also loss function). Evolution of the population then takes place after the repeated application of the above operators.

Evolutionary algorithms often perform well approximating solutions to all types of problems because they ideally do not make any assumption about the underlying fitness landscape. Techniques from evolutionary algorithms applied to the modeling of biological evolution are generally limited to explorations of microevolutionary processes and planning models based upon cellular processes. In most real applications of EAs, computational complexity is a prohibiting factor. In fact, this computational complexity is due to fitness function evaluation.

Fitness approximation is one of the solutions to overcome this difficulty. However, seemingly simple EA can solve often complex problems; therefore, there may be no direct link between algorithm complexity and problem complexity.

Implementation

The following is an example of a generic single-objective genetic algorithm.

Step One: Generate the initial population of individuals randomly. (First generation)

Step Two: Repeat the following regenerational steps until termination:

- Evaluate the fitness of each individual in the population (time limit, sufficient fitness achieved, etc.)
- Select the fittest individuals for reproduction. (Parents)
- Breed new individuals through crossover and mutation operations to give birth to offspring.
- Replace the least-fit individuals of the population with new individuals.

Types

Similar techniques differ in genetic representation and other implementation details, and the nature of the particular applied problem.

- Genetic algorithm – This is the most popular type of EA. One seeks the solution of a problem in the form of strings of numbers (traditionally binary,

although the best representations are usually those that reflect something about the problem being solved), by applying operators such as recombination and mutation (sometimes one, sometimes both). This type of EA is often used in optimization problems.

- Genetic programming – Here the solutions are in the form of computer programs, and their fitness is determined by their ability to solve a computational problem.
- Evolutionary programming – Similar to genetic programming, but the structure of the program is fixed and its numerical parameters are allowed to evolve.
- Gene expression programming – Like genetic programming, GEP also evolves computer programs but it explores a genotype-phenotype system, where computer programs of different sizes are encoded in linear chromosomes of fixed length.
- Evolution strategy – Works with vectors of real numbers as representations of solutions, and typically uses self-adaptive mutation rates.
- Differential evolution – Based on vector differences and is therefore primarily suited for numerical optimization problems.
- Neuroevolution – Similar to genetic programming but the genomes represent artificial neural networks by describing structure and connection weights. The genome encoding can be direct or indirect.

- Learning classifier system – Here the solution is a set of classifiers (rules or conditions). A Michigan-LCS evolves at the level of individual classifiers whereas a Pittsburgh-LCS uses populations of classifier-sets. Initially, classifiers were only binary, but now include real, neural net, or S-expression types. Fitness is typically determined with either a strength or accuracy based reinforcement learning or supervised learning approach.

Comparison to biological processes

A possible limitation of many evolutionary algorithms is their lack of a clear genotype-phenotype distinction. In nature, the fertilized egg cell undergoes a complex process known as embryogenesis to become a mature phenotype. This indirect encoding is believed to make the genetic search more robust (i.e. reduce the probability of fatal mutations), and also may improve the evolvability of the organism. Such indirect (a.k.a. generative or developmental) encodings also enable evolution to exploit the regularity in the environment. Recent work in the field of artificial embryogeny, or artificial developmental systems, seeks to address these concerns. And gene expression programming successfully explores a genotype-phenotype system, where the genotype consists of linear multigenic chromosomes of fixed length and the phenotype consists of

multiple expression trees or computer programs of different sizes and shapes.

Related techniques

Swarm algorithms include

- Ant colony optimization – Based on the ideas of ant foraging by pheromone communication to form paths. Primarily suited for combinatorial optimization and graph problems.
- The runner-root algorithm (RRA) is inspired by the function of runners and roots of plants in nature
- Artificial bee colony algorithm – Based on the honey bee foraging behaviour. Primarily proposed for numerical optimization and extended to solve combinatorial, constrained and multi-objective optimization problems.
- Bees algorithm is based on the foraging behaviour of honey bees. It has been applied in many applications such as routing and scheduling.
- Cuckoo search is inspired by the brooding parasitism of the cuckoo species. It also uses Lévy flights, and thus it suits for global optimization problems.
- Electimize optimization - Based on the behavior of electron flow through electric circuit branches with the least electric resistance.

- Particle swarm optimization – Based on the ideas of animal flocking behaviour. Also primarily suited for numerical optimization problems.

Other population-based metaheuristic methods

- Hunting Search – A method inspired by the group hunting of some animals such as wolves that organize their position to surround the prey, each of them relative to the position of the others and especially that of their leader. It is a continuous optimization method adapted as a combinatorial optimization method.
- Adaptive dimensional search – Unlike nature-inspired metaheuristic techniques, an adaptive dimensional search algorithm does not implement any metaphor as an underlying principle. Rather it uses a simple performance-oriented method, based on the update of the search dimensionality ratio (SDR) parameter at each iteration.
- Firefly algorithm is inspired by the behavior of fireflies, attracting each other by flashing light. This is especially useful for multimodal optimization.
- Harmony search – Based on the ideas of musicians' behavior in searching for better harmonies. This algorithm is suitable for

combinatorial optimization as well as parameter optimization.

- Gaussian adaptation – Based on information theory. Used for maximization of manufacturing yield, mean fitness or average information. See for instance Entropy in thermodynamics and information theory.
- Memetic algorithm – A hybrid method, inspired by Richard Dawkins's notion of a meme, it commonly takes the form of a population-based algorithm coupled with individual learning procedures capable of performing local refinements. Emphasizes the exploitation of problem-specific knowledge, and tries to orchestrate local and global search in a synergistic way.

Examples

In 2020, Google stated their AutoML-Zero can successfully rediscover classic algorithms such as the concept of neural networks. The computer simulations *Tierra* and *Avida* attempt to model macroevolutionary dynamics.

Genetic algorithm

In computer science and operations research, a **genetic algorithm** (**GA**) is a metaheuristic inspired by the process of natural selection that belongs to the larger class of evolutionary algorithms (EA). Genetic algorithms are commonly

used to generate high-quality solutions to optimization and search problems by relying on biologically inspired operators such as mutation, crossover and selection.

Methodology

Optimization problems

In a genetic algorithm, a population of candidate solutions (called individuals, creatures, or phenotypes) to an optimization problem is evolved toward better solutions. Each candidate solution has a set of properties (its chromosomes or genotype) which can be mutated and altered; traditionally, solutions are represented in binary as strings of 0s and 1s, but other encodings are also possible.

The evolution usually starts from a population of randomly generated individuals, and is an iterative process, with the population in each iteration called a *generation*. In each generation, the fitness of every individual in the population is evaluated; the fitness is usually the value of the objective function in the optimization problem being solved. The more fit individuals are stochastically selected from the current population, and each individual's genome is modified (recombined and possibly randomly mutated) to form a new generation. The new generation of candidate solutions is then used in the next iteration of the algorithm. Commonly, the algorithm terminates when either a maximum number of generations has been produced, or a satisfactory fitness level has been reached for the population.

A typical genetic algorithm requires:

- a genetic representation of the solution domain,
- a fitness function to evaluate the solution domain.

A standard representation of each candidate solution is as an array of bits. Arrays of other types and structures can be used in essentially the same way. The main property that makes these genetic representations convenient is that their parts are easily aligned due to their fixed size, which facilitates simple crossover operations. Variable length representations may also be used, but crossover implementation is more complex in this case. Tree-like representations are explored in genetic programming and graph-form representations are explored in evolutionary programming; a mix of both linear chromosomes and trees is explored in gene expression programming.

Once the genetic representation and the fitness function are defined, a GA proceeds to initialize a population of solutions and then to improve it through repetitive application of the mutation, crossover, inversion and selection operators.

Initialization

The population size depends on the nature of the problem, but typically contains several hundreds or thousands of possible solutions. Often, the initial population is generated randomly, allowing the entire range of possible solutions (the search space). Occasionally, the solutions may be "seeded" in areas where optimal solutions are likely to be found.

Selection

During each successive generation, a portion of the existing population is selected to breed a new generation. Individual solutions are selected through a *fitness-based* process, where fitter solutions (as measured by a fitness function) are typically more likely to be selected. Certain selection methods rate the fitness of each solution and preferentially select the best solutions. Other methods rate only a random sample of the population, as the former process may be very time-consuming.

The fitness function is defined over the genetic representation and measures the *quality* of the represented solution. The fitness function is always problem dependent. For instance, in the knapsack problem one wants to maximize the total value of objects that can be put in a knapsack of some fixed capacity. A representation of a solution might be an array of bits, where each bit represents a different object, and the value of the bit (0 or 1) represents whether or not the object is in the knapsack. Not every such representation is valid, as the size of objects may exceed the capacity of the knapsack. The *fitness* of the solution is the sum of values of all objects in the knapsack if the representation is valid, or 0 otherwise.

In some problems, it is hard or even impossible to define the fitness expression; in these cases, a simulation may be used to determine the fitness function value of a phenotype (e.g. computational fluid dynamics is used to determine the air resistance of a vehicle whose shape is encoded as the phenotype), or even interactive genetic algorithms are used.

Genetic operators

The next step is to generate a second generation population of solutions from those selected through a combination of genetic operators: crossover (also called recombination), and mutation.

For each new solution to be produced, a pair of "parent" solutions is selected for breeding from the pool selected previously. By producing a "child" solution using the above methods of crossover and mutation, a new solution is created which typically shares many of the characteristics of its "parents". New parents are selected for each new child, and the process continues until a new population of solutions of appropriate size is generated. Although reproduction methods that are based on the use of two parents are more "biology inspired", some research suggests that more than two "parents" generate higher quality chromosomes.

These processes ultimately result in the next generation population of chromosomes that is different from the initial generation. Generally, the average fitness will have increased by this procedure for the population, since only the best organisms from the first generation are selected for breeding, along with a small proportion of less fit solutions. These less fit solutions ensure genetic diversity within the genetic pool of the parents and therefore ensure the genetic diversity of the subsequent generation of children.

Opinion is divided over the importance of crossover versus mutation. There are many references in Fogel (2006) that support the importance of mutation-based search.

Although crossover and mutation are known as the main genetic operators, it is possible to use other operators such as regrouping, colonization-extinction, or migration in genetic algorithms.

It is worth tuning parameters such as the mutation probability, crossover probability and population size to find reasonable settings for the problem class being worked on. A very small mutation rate may lead to genetic drift (which is non-ergodic in nature). A recombination rate that is too high may lead to premature convergence of the genetic algorithm. A mutation rate that is too high may lead to loss of good solutions, unless elitist selection is employed. An adequate population size ensures sufficient genetic diversity for the problem at hand, but can lead to a waste of computational resources if set to a value larger than required.

Heuristics

In addition to the main operators above, other heuristics may be employed to make the calculation faster or more robust. The *speciation* heuristic penalizes crossover between candidate solutions that are too similar; this encourages population diversity and helps prevent premature convergence to a less optimal solution.

Termination

This generational process is repeated until a termination condition has been reached. Common terminating conditions are:

- A solution is found that satisfies minimum criteria
- Fixed number of generations reached
- Allocated budget (computation time/money) reached
- The highest ranking solution's fitness is reaching or has reached a plateau such that successive iterations no longer produce better results
- Manual inspection
- Combinations of the above

The building block hypothesis

Genetic algorithms are simple to implement, but their behavior is difficult to understand. In particular, it is difficult to understand why these algorithms frequently succeed at generating solutions of high fitness when applied to practical problems. The building block hypothesis (BBH) consists of:

- A description of a heuristic that performs adaptation by identifying and recombining "building blocks", i.e. low order, low defining-length schemata with above average fitness.
- A hypothesis that a genetic algorithm performs adaptation by implicitly and efficiently implementing this heuristic.

Goldberg describes the heuristic as follows:

- "Short, low order, and highly fit schemata are sampled, recombined [crossed over], and resampled to form strings of potentially higher

fitness. In a way, by working with these particular schemata [the building blocks], we have reduced the complexity of our problem; instead of building high-performance strings by trying every conceivable combination, we construct better and better strings from the best partial solutions of past samplings.

- "Because highly fit schemata of low defining length and low order play such an important role in the action of genetic algorithms, we have already given them a special name: building blocks. Just as a child creates magnificent fortresses through the arrangement of simple blocks of wood, so does a genetic algorithm seek near optimal performance through the juxtaposition of short, low-order, high-performance schemata, or building blocks."

Despite the lack of consensus regarding the validity of the building-block hypothesis, it has been consistently evaluated and used as reference throughout the years. Many estimation of distribution algorithms, for example, have been proposed in an attempt to provide an environment in which the hypothesis would hold. Although good results have been reported for some classes of problems, skepticism concerning the generality and/or practicality of the building-block hypothesis as an explanation for GAs efficiency still remains. Indeed, there is a reasonable amount of work that attempts to understand its limitations from the perspective of estimation of distribution algorithms.

Limitations

There are limitations of the use of a genetic algorithm compared to alternative optimization algorithms:

- Repeated fitness function evaluation for complex problems is often the most prohibitive and limiting segment of artificial evolutionary algorithms. Finding the optimal solution to complex high-dimensional, multimodal problems often requires very expensive fitness function evaluations. In real world problems such as structural optimization problems, a single function evaluation may require several hours to several days of complete simulation. Typical optimization methods cannot deal with such types of problem. In this case, it may be necessary to forgo an exact evaluation and use an approximated fitness that is computationally efficient. It is apparent that amalgamation of approximate models may be one of the most promising approaches to convincingly use GA to solve complex real life problems.
- Genetic algorithms do not scale well with complexity. That is, where the number of elements which are exposed to mutation is large there is often an exponential increase in search space size. This makes it extremely difficult to use the technique on problems such as designing an engine, a house or a plane. In order to make such problems tractable to evolutionary search, they must be broken down into the simplest

representation possible. Hence we typically see evolutionary algorithms encoding designs for fan blades instead of engines, building shapes instead of detailed construction plans, and airfoils instead of whole aircraft designs. The second problem of complexity is the issue of how to protect parts that have evolved to represent good solutions from further destructive mutation, particularly when their fitness assessment requires them to combine well with other parts.

- The "better" solution is only in comparison to other solutions. As a result, the stop criterion is not clear in every problem.
- In many problems, GAs have a tendency to converge towards local optima or even arbitrary points rather than the global optimum of the problem. This means that it does not "know how" to sacrifice short-term fitness to gain longer-term fitness. The likelihood of this occurring depends on the shape of the fitness landscape: certain problems may provide an easy ascent towards a global optimum, others may make it easier for the function to find the local optima. This problem may be alleviated by using a different fitness function, increasing the rate of mutation, or by using selection techniques that maintain a diverse population of solutions, although the No Free Lunch theorem proves that there is no general solution to this problem. A common technique to maintain diversity is to impose a "niche penalty", wherein, any group of individuals of sufficient similarity (niche radius) have a penalty added,

which will reduce the representation of that group in subsequent generations, permitting other (less similar) individuals to be maintained in the population. This trick, however, may not be effective, depending on the landscape of the problem. Another possible technique would be to simply replace part of the population with randomly generated individuals, when most of the population is too similar to each other. Diversity is important in genetic algorithms (and genetic programming) because crossing over a homogeneous population does not yield new solutions. In evolution strategies and evolutionary programming, diversity is not essential because of a greater reliance on mutation.

- Operating on dynamic data sets is difficult, as genomes begin to converge early on towards solutions which may no longer be valid for later data. Several methods have been proposed to remedy this by increasing genetic diversity somehow and preventing early convergence, either by increasing the probability of mutation when the solution quality drops (called *triggered hypermutation*), or by occasionally introducing entirely new, randomly generated elements into the gene pool (called *random immigrants*). Again, evolution strategies and evolutionary programming can be implemented with a so-called "comma strategy" in which parents are not maintained and new parents are selected only from offspring. This can be more effective on dynamic problems.

- GAs cannot effectively solve problems in which the only fitness measure is a single right/wrong measure (like decision problems), as there is no way to converge on the solution (no hill to climb). In these cases, a random search may find a solution as quickly as a GA. However, if the situation allows the success/failure trial to be repeated giving (possibly) different results, then the ratio of successes to failures provides a suitable fitness measure.
- For specific optimization problems and problem instances, other optimization algorithms may be more efficient than genetic algorithms in terms of speed of convergence. Alternative and complementary algorithms include evolution strategies, evolutionary programming, simulated annealing, Gaussian adaptation, hill climbing, and swarm intelligence (e.g.: ant colony optimization, particle swarm optimization) and methods based on integer linear programming. The suitability of genetic algorithms is dependent on the amount of knowledge of the problem; well known problems often have better, more specialized approaches.

Variants

Chromosome representation

The simplest algorithm represents each chromosome as a bit string. Typically, numeric parameters can be represented by integers, though it is possible to use floating point

representations. The floating point representation is natural to evolution strategies and evolutionary programming. The notion of real-valued genetic algorithms has been offered but is really a misnomer because it does not really represent the building block theory that was proposed by John Henry Holland in the 1970s. This theory is not without support though, based on theoretical and experimental results (see below). The basic algorithm performs crossover and mutation at the bit level. Other variants treat the chromosome as a list of numbers which are indexes into an instruction table, nodes in a linked list, hashes, objects, or any other imaginable data structure. Crossover and mutation are performed so as to respect data element boundaries. For most data types, specific variation operators can be designed. Different chromosomal data types seem to work better or worse for different specific problem domains.

When bit-string representations of integers are used, Gray coding is often employed. In this way, small changes in the integer can be readily affected through mutations or crossovers. This has been found to help prevent premature convergence at so-called *Hamming walls*, in which too many simultaneous mutations (or crossover events) must occur in order to change the chromosome to a better solution.

Other approaches involve using arrays of real-valued numbers instead of bit strings to represent chromosomes. Results from the theory of schemata suggest that in general the smaller the alphabet, the better the performance, but it was initially surprising to researchers that good results were obtained from using real-valued chromosomes. This was explained as the set of real values in a finite population of chromosomes as forming

a *virtual alphabet* (when selection and recombination are dominant) with a much lower cardinality than would be expected from a floating point representation.

An expansion of the Genetic Algorithm accessible problem domain can be obtained through more complex encoding of the solution pools by concatenating several types of heterogenously encoded genes into one chromosome. This particular approach allows for solving optimization problems that require vastly disparate definition domains for the problem parameters. For instance, in problems of cascaded controller tuning, the internal loop controller structure can belong to a conventional regulator of three parameters, whereas the external loop could implement a linguistic controller (such as a fuzzy system) which has an inherently different description. This particular form of encoding requires a specialized crossover mechanism that recombines the chromosome by section, and it is a useful tool for the modelling and simulation of complex adaptive systems, especially evolution processes.

Elitism

A practical variant of the general process of constructing a new population is to allow the best organism(s) from the current generation to carry over to the next, unaltered. This strategy is known as *elitist selection* and guarantees that the solution quality obtained by the GA will not decrease from one generation to the next.

Parallel implementations

Parallel implementations of genetic algorithms come in two flavors. Coarse-grained parallel genetic algorithms assume a

population on each of the computer nodes and migration of individuals among the nodes. Fine-grained parallel genetic algorithms assume an individual on each processor node which acts with neighboring individuals for selection and reproduction. Other variants, like genetic algorithms for online optimization problems, introduce time-dependence or noise in the fitness function.

Adaptive GAs

Genetic algorithms with adaptive parameters (adaptive genetic algorithms, AGAs) is another significant and promising variant of genetic algorithms. The probabilities of crossover (pc) and mutation (pm) greatly determine the degree of solution accuracy and the convergence speed that genetic algorithms can obtain. Instead of using fixed values of *pc* and *pm*, AGAs utilize the population information in each generation and adaptively adjust the *pc* and *pm* in order to maintain the population diversity as well as to sustain the convergence capacity. In AGA (adaptive genetic algorithm), the adjustment of *pc* and *pm* depends on the fitness values of the solutions. In *CAGA* (clustering-based adaptive genetic algorithm), through the use of clustering analysis to judge the optimization states of the population, the adjustment of *pc* and *pm* depends on these optimization states. It can be quite effective to combine GA with other optimization methods. GA tends to be quite good at finding generally good global solutions, but quite inefficient at finding the last few mutations to find the absolute optimum. Other techniques (such as simple hill climbing) are quite efficient at finding absolute optimum in a limited region. Alternating GA and hill climbing can improve the efficiency of GA while overcoming the lack of robustness of hill climbing.

This means that the rules of genetic variation may have a different meaning in the natural case. For instance – provided that steps are stored in consecutive order – crossing over may sum a number of steps from maternal DNA adding a number of steps from paternal DNA and so on. This is like adding vectors that more probably may follow a ridge in the phenotypic landscape. Thus, the efficiency of the process may be increased by many orders of magnitude. Moreover, the inversion operator has the opportunity to place steps in consecutive order or any other suitable order in favour of survival or efficiency.

A variation, where the population as a whole is evolved rather than its individual members, is known as gene pool recombination.

A number of variations have been developed to attempt to improve performance of GAs on problems with a high degree of fitness epistasis, i.e. where the fitness of a solution consists of interacting subsets of its variables. Such algorithms aim to learn (before exploiting) these beneficial phenotypic interactions. As such, they are aligned with the Building Block Hypothesis in adaptively reducing disruptive recombination. Prominent examples of this approach include the mGA, GEMGA and LLGA.

Problem domains

Problems which appear to be particularly appropriate for solution by genetic algorithms include timetabling and scheduling problems, and many scheduling software packages are based on GAs. GAs have also been applied to engineering

and to reducing the length of psychological questionnaires. Genetic algorithms are often applied as an approach to solve global optimization problems.

As a general rule of thumb genetic algorithms might be useful in problem domains that have a complex fitness landscape as mixing, i.e., mutation in combination with crossover, is designed to move the population away from local optima that a traditional hill climbing algorithm might get stuck in. Observe that commonly used crossover operators cannot change any uniform population. Mutation alone can provide ergodicity of the overall genetic algorithm process (seen as a Markov chain).

Examples of problems solved by genetic algorithms include: mirrors designed to funnel sunlight to a solar collector, antennae designed to pick up radio signals in space, walking methods for computer figures, optimal design of aerodynamic bodies in complex flowfields

In his *Algorithm Design Manual*, Skiena advises against genetic algorithms for any task:

[I]t is quite unnatural to model applications in terms of genetic operators like mutation and crossover on bit strings. The pseudobiology adds another level of complexity between you and your problem. Second, genetic algorithms take a very long time on nontrivial problems. [...] [T]he analogy with evolution—where significant progress require [sic] millions of years—can be quite appropriate.

[...]

I have never encountered any problem where genetic algorithms seemed to me the right way to attack it. Further, I have never seen any computational results reported using genetic algorithms that have favorably impressed me. Stick to simulated annealing for your heuristic search voodoo needs.

— *Steven Skiena*

History

In 1950, Alan Turing proposed a "learning machine" which would parallel the principles of evolution. Computer simulation of evolution started as early as in 1954 with the work of Nils Aall Barricelli, who was using the computer at the Institute for Advanced Study in Princeton, New Jersey. His 1954 publication was not widely noticed. Starting in 1957, the Australian quantitative geneticist Alex Fraser published a series of papers on simulation of artificial selection of organisms with multiple loci controlling a measurable trait. From these beginnings, computer simulation of evolution by biologists became more common in the early 1960s, and the methods were described in books by Fraser and Burnell (1970) and Crosby (1973). Fraser's simulations included all of the essential elements of modern genetic algorithms. In addition, Hans-Joachim Bremermann published a series of papers in the 1960s that also adopted a population of solution to optimization problems, undergoing recombination, mutation, and selection. Bremermann's research also included the elements of modern genetic algorithms. Other noteworthy early pioneers include Richard Friedberg, George Friedman, and Michael Conrad. Many early papers are reprinted by Fogel (1998).

Although Barricelli, in work he reported in 1963, had simulated the evolution of ability to play a simple game, artificial evolution only became a widely recognized optimization method as a result of the work of Ingo Rechenberg and Hans-Paul Schwefel in the 1960s and early 1970s – Rechenberg's group was able to solve complex engineering problems through evolution strategies. Another approach was the evolutionary programming technique of Lawrence J. Fogel, which was proposed for generating artificial intelligence. Evolutionary programming originally used finite state machines for predicting environments, and used variation and selection to optimize the predictive logics. Genetic algorithms in particular became popular through the work of John Holland in the early 1970s, and particularly his book *Adaptation in Natural and Artificial Systems* (1975). His work originated with studies of cellular automata, conducted by Holland and his students at the University of Michigan. Holland introduced a formalized framework for predicting the quality of the next generation, known as Holland's Schema Theorem. Research in GAs remained largely theoretical until the mid-1980s, when The First International Conference on Genetic Algorithms was held in Pittsburgh, Pennsylvania.

Commercial products

In the late 1980s, General Electric started selling the world's first genetic algorithm product, a mainframe-based toolkit designed for industrial processes. In 1989, Axcelis, Inc. released Evolver, the world's first commercial GA product for desktop computers. The New York Times technology writer John Markoff wrote about Evolver in 1990, and it remained the only interactive commercial genetic algorithm until 1995.

Evolver was sold to Palisade in 1997, translated into several languages, and is currently in its 6th version. Since the 1990s, MATLAB has built in three derivative-free optimization heuristic algorithms (simulated annealing, particle swarm optimization, genetic algorithm) and two direct search algorithms (simplex search, pattern search).

Related techniques

Parent fields

Genetic algorithms are a sub-field:

- Evolutionary algorithms
- Evolutionary computing
- Metaheuristics
- Stochastic optimization
- Optimization

Related fields

Evolutionary algorithms

Evolutionary algorithms is a sub-field of evolutionary computing.

- Evolution strategies (ES, see Rechenberg, 1994) evolve individuals by means of mutation and intermediate or discrete recombination. ES

algorithms are designed particularly to solve problems in the real-value domain. They use self-adaptation to adjust control parameters of the search. De-randomization of self-adaptation has led to the contemporary Covariance Matrix Adaptation Evolution Strategy (CMA-ES).

- Evolutionary programming (EP) involves populations of solutions with primarily mutation and selection and arbitrary representations. They use self-adaptation to adjust parameters, and can include other variation operations such as combining information from multiple parents.
- Estimation of Distribution Algorithm (EDA) substitutes traditional reproduction operators by model-guided operators. Such models are learned from the population by employing machine learning techniques and represented as Probabilistic Graphical Models, from which new solutions can be sampled or generated from guided-crossover.
- Gene expression programming (GEP) also uses populations of computer programs. These complex computer programs are encoded in simpler linear chromosomes of fixed length, which are afterwards expressed as expression trees. Expression trees or computer programs evolve because the chromosomes undergo mutation and recombination in a manner similar to the canonical GA. But thanks to the special organization of GEP chromosomes, these genetic modifications always result in valid computer programs.

- Genetic programming (GP) is a related technique popularized by John Koza in which computer programs, rather than function parameters, are optimized. Genetic programming often uses tree-based internal data structures to represent the computer programs for adaptation instead of the list structures typical of genetic algorithms.
- Grouping genetic algorithm (GGA) is an evolution of the GA where the focus is shifted from individual items, like in classical GAs, to groups or subset of items. The idea behind this GA evolution proposed by Emanuel Falkenauer is that solving some complex problems, a.k.a. *clustering* or *partitioning* problems where a set of items must be split into disjoint group of items in an optimal way, would better be achieved by making characteristics of the groups of items equivalent to genes. These kind of problems include bin packing, line balancing, clustering with respect to a distance measure, equal piles, etc., on which classic GAs proved to perform poorly. Making genes equivalent to groups implies chromosomes that are in general of variable length, and special genetic operators that manipulate whole groups of items. For bin packing in particular, a GGA hybridized with the Dominance Criterion of Martello and Toth, is arguably the best technique to date.
- Interactive evolutionary algorithms are evolutionary algorithms that use human evaluation. They are usually applied to domains where it is hard to design a computational fitness

function, for example, evolving images, music, artistic designs and forms to fit users' aesthetic preference.

Swarm intelligence

Swarm intelligence is a sub-field of evolutionary computing.

- Ant colony optimization (**ACO**) uses many ants (or agents) equipped with a pheromone model to traverse the solution space and find locally productive areas. Although considered an Estimation of distribution algorithm,
- Particle swarm optimization (PSO) is a computational method for multi-parameter optimization which also uses population-based approach. A population (swarm) of candidate solutions (particles) moves in the search space, and the movement of the particles is influenced both by their own best known position and swarm's global best known position. Like genetic algorithms, the PSO method depends on information sharing among population members. In some problems the PSO is often more computationally efficient than the GAs, especially in unconstrained problems with continuous variables.

Other evolutionary computing algorithms

Evolutionary computation is a sub-field of the metaheuristic methods.

- Electimize algorithm is an evolutionary algorithm that simulates the phenomenon of electron flow and electrical conductivity. Some current research showed Electimize to be more efficient in solving NP-hard optimization problems than traditional evolutionary algorithms. The algorithm provides higher capacity in searching the solution space extensively, and identifying global optimal alternatives. Unlike other evolutionary algorithms, Electimize evaluates the quality of the values in the solution string independently.
- Memetic algorithm (MA), often called *hybrid genetic algorithm* among others, is a population-based method in which solutions are also subject to local improvement phases. The idea of memetic algorithms comes from memes, which unlike genes, can adapt themselves. In some problem areas they are shown to be more efficient than traditional evolutionary algorithms.
- Bacteriologic algorithms (BA) inspired by evolutionary ecology and, more particularly, bacteriologic adaptation. Evolutionary ecology is the study of living organisms in the context of their environment, with the aim of discovering how they adapt. Its basic concept is that in a heterogeneous environment, there is not one individual that fits the whole environment. So, one needs to reason at the population level. It is also believed BAs could be successfully applied to complex positioning problems (antennas for cell phones, urban planning, and so on) or data mining.

- Cultural algorithm (CA) consists of the population component almost identical to that of the genetic algorithm and, in addition, a knowledge component called the belief space.
- Differential evolution (DS) inspired by migration of superorganisms.
- Gaussian adaptation (normal or natural adaptation, abbreviated NA to avoid confusion with GA) is intended for the maximisation of manufacturing yield of signal processing systems. It may also be used for ordinary parametric optimisation. It relies on a certain theorem valid for all regions of acceptability and all Gaussian distributions. The efficiency of NA relies on information theory and a certain theorem of efficiency. Its efficiency is defined as information divided by the work needed to get the information. Because NA maximises mean fitness rather than the fitness of the individual, the landscape is smoothed such that valleys between peaks may disappear. Therefore it has a certain "ambition" to avoid local peaks in the fitness landscape. NA is also good at climbing sharp crests by adaptation of the moment matrix, because NA may maximise the disorder (average information) of the Gaussian simultaneously keeping the mean fitness constant.

Other metaheuristic methods

Metaheuristic methods broadly fall within stochastic optimisation methods.

- Simulated annealing (SA) is a related global optimization technique that traverses the search space by testing random mutations on an individual solution. A mutation that increases fitness is always accepted. A mutation that lowers fitness is accepted probabilistically based on the difference in fitness and a decreasing temperature parameter. In SA parlance, one speaks of seeking the lowest energy instead of the maximum fitness. SA can also be used within a standard GA algorithm by starting with a relatively high rate of mutation and decreasing it over time along a given schedule.
- Tabu search (TS) is similar to simulated annealing in that both traverse the solution space by testing mutations of an individual solution. While simulated annealing generates only one mutated solution, tabu search generates many mutated solutions and moves to the solution with the lowest energy of those generated. In order to prevent cycling and encourage greater movement through the solution space, a tabu list is maintained of partial or complete solutions. It is forbidden to move to a solution that contains elements of the tabu list, which is updated as the solution traverses the solution space.
- Extremal optimization (EO) Unlike GAs, which work with a population of candidate solutions, EO evolves a single solution and makes local modifications to the worst components. This requires that a suitable representation be selected which permits individual solution components to

be assigned a quality measure ("fitness"). The governing principle behind this algorithm is that of *emergent* improvement through selectively removing low-quality components and replacing them with a randomly selected component. This is decidedly at odds with a GA that selects good solutions in an attempt to make better solutions.

Other stochastic optimisation methods

- The cross-entropy (CE) method generates candidate solutions via a parameterized probability distribution. The parameters are updated via cross-entropy minimization, so as to generate better samples in the next iteration.
- Reactive search optimization (RSO) advocates the integration of sub-symbolic machine learning techniques into search heuristics for solving complex optimization problems. The word reactive hints at a ready response to events during the search through an internal online feedback loop for the self-tuning of critical parameters. Methodologies of interest for Reactive Search include machine learning and statistics, in particular reinforcement learning, active or query learning, neural networks, and metaheuristics.

Differential evolution

In evolutionary computation, **differential evolution** (**DE**) is a method that optimizes a problem by iteratively trying to improve a candidate solution with regard to a given measure of quality. Such methods are commonly known as metaheuristics as they make few or no assumptions about the problem being optimized and can search very large spaces of candidate solutions. However, metaheuristics such as DE do not guarantee an optimal solution is ever found.

DE is used for multidimensional real-valued functions but does not use the gradient of the problem being optimized, which means DE does not require the optimization problem to be differentiable, as is required by classic optimization methods such as gradient descent and quasi-newton methods. DE can therefore also be used on optimization problems that are not even continuous, are noisy, change over time, etc.

DE optimizes a problem by maintaining a population of candidate solutions and creating new candidate solutions by combining existing ones according to its simple formulae, and then keeping whichever candidate solution has the best score or fitness on the optimization problem at hand. In this way the optimization problem is treated as a black box that merely provides a measure of quality given a candidate solution and the gradient is therefore not needed.

DE was introduced by Storn and Price in the 1990s. Books have been published on theoretical and practical aspects of using DE in parallel computing, multiobjective optimization,

constrained optimization, and the books also contain surveys of application areas. Surveys on the multi-faceted research aspects of DE can be found in journal articles .

Chapter 5

Metaheuristic and Swarm Intelligence

Metaheuristic

In computer science and mathematical optimization, a **metaheuristic** is a higher-level procedure or heuristic designed to find, generate, or select a heuristic (partial search algorithm) that MAY provide a sufficiently good solution to an optimization problem, especially with incomplete or imperfect information or limited computation capacity. Metaheuristics sample a subset of solutions which is otherwise too large to be completely enumerated or otherwise explored. Metaheuristics may make relatively few assumptions about the optimization problem being solved and so may be usable for a variety of problems.

Compared to optimization algorithms and iterative methods, metaheuristics do not guarantee that a globally optimal solution can be found on some class of problems. Many metaheuristics implement some form of stochastic optimization, so that the solution found is dependent on the set of random variables generated. In combinatorial optimization, by searching over a large set of feasible solutions, metaheuristics can often find good solutions with

less computational effort than optimization algorithms, iterative methods, or simple heuristics. As such, they are useful approaches for optimization problems. Several books and survey papers have been published on the subject.

Most literature on metaheuristics is experimental in nature, describing empirical results based on computer experiments with the algorithms. But some formal theoretical results are also available, often on convergence and the possibility of finding the global optimum. Many metaheuristic methods have been published with claims of novelty and practical efficacy. While the field also features high-quality research, many of the publications have been of poor quality; flaws include vagueness, lack of conceptual elaboration, poor experiments, and ignorance of previous literature.

Properties

These are properties that characterize most metaheuristics:

- Metaheuristics are strategies that guide the search process.
- The goal is to efficiently explore the search space in order to find near–optimal solutions.
- Techniques which constitute metaheuristic algorithms range from simple local search procedures to complex learning processes.
- Metaheuristic algorithms are approximate and usually non-deterministic.
- Metaheuristics are not problem-specific.

Classification

There are a wide variety of metaheuristics and a number of properties with respect to which to classify them.

Local search vs. global search

One approach is to characterize the type of search strategy. One type of search strategy is an improvement on simple local search algorithms. A well known local search algorithm is the hill climbing method which is used to find local optimums. However, hill climbing does not guarantee finding global optimum solutions.

Many metaheuristic ideas were proposed to improve local search heuristic in order to find better solutions. Such metaheuristics include simulated annealing, tabu search, iterated local search, variable neighborhood search, and GRASP. These metaheuristics can both be classified as local search-based or global search metaheuristics.

Other global search metaheuristic that are not local search-based are usually population-based metaheuristics. Such metaheuristics include ant colony optimization, evolutionary computation, particle swarm optimization, genetic algorithm, and rider optimization algorithm

Single-solution vs. population-based

Another classification dimension is single solution vs population-based searches. Single solution approaches focus on modifying and improving a single candidate solution; single

solution metaheuristics include simulated annealing, iterated local search, variable neighborhood search, and guided local search. Population-based approaches maintain and improve multiple candidate solutions, often using population characteristics to guide the search; population based metaheuristics include evolutionary computation, genetic algorithms, and particle swarm optimization. Another category of metaheuristics is Swarm intelligence which is a collective behavior of decentralized, self-organized agents in a population or swarm. Ant colony optimization, particle swarm optimization, social cognitive optimization are examples of this category.

Hybridization and memetic algorithms

A hybrid metaheuristic is one that combines a metaheuristic with other optimization approaches, such as algorithms from mathematical programming, constraint programming, and machine learning. Both components of a hybrid metaheuristic may run concurrently and exchange information to guide the search.

On the other hand, Memetic algorithms represent the synergy of evolutionary or any population-based approach with separate individual learning or local improvement procedures for problem search. An example of memetic algorithm is the use of a local search algorithm instead of a basic mutation operator in evolutionary algorithms.

Parallel metaheuristics

A parallel metaheuristic is one that uses the techniques of parallel programming to run multiple metaheuristic searches

in parallel; these may range from simple distributed schemes to concurrent search runs that interact to improve the overall solution.

Nature-inspired and metaphor-based metaheuristics

A very active area of research is the design of nature-inspired metaheuristics. Many recent metaheuristics, especially evolutionary computation-based algorithms, are inspired by natural systems. Nature acts as a source of concepts, mechanisms and principles for designing of artificial computing systems to deal with complex computational problems. Such metaheuristics include simulated annealing, evolutionary algorithms, ant colony optimization and particle swarm optimization. A large number of more recent metaphor-inspired metaheuristics have started to attract criticism in the research community for hiding their lack of novelty behind an elaborate metaphor.

Ancient-inspired metaheuristics

We seem to be facing a novel source of inspiration. This will make way towards many ancient-inspired algorithms. There have been numerous limitations in the ancient era, but various man-made structures indicate that limitations and lack of facilities have led to some sort of optimization. A closer look at these ancient relics shows that the methods, strategies, and technologies used in antiquity are far more advanced and optimized than we would have imagined. Ancient-inspired ideology observes and reflects on the features and seeks to understand ways of managing the project at that time.

Applications

Metaheuristics are used for combinatorial optimization in which an optimal solution is sought over a discrete search-space. An example problem is the travelling salesman problem where the search-space of candidate solutions grows faster than exponentially as the size of the problem increases, which makes an exhaustive search for the optimal solution infeasible. Additionally, multidimensional combinatorial problems, including most design problems in engineering such as form-finding and behavior-finding, suffer from the curse of dimensionality, which also makes them infeasible for exhaustive search or analytical methods. Metaheuristics are also widely used for jobshop scheduling and job selection problems. Popular metaheuristics for combinatorial problems include simulated annealing by Kirkpatrick et al., genetic algorithms by Holland et al., scatter search and tabu search by Glover. Literature review on metaheuristic optimization, suggested that it was Fred Glover who coined the word metaheuristics.

Metaheuristic Optimization Frameworks (MOFs)

A MOF can be defined as ‘‘a set of software tools that provide a correct and reusable implementation of a set of metaheuristics, and the basic mechanisms to accelerate the implementation of

its partner subordinate heuristics (possibly including solution encodings and technique-specific operators), which are necessary to solve a particular problem instance using techniques provided''.

There are many candidate optimization tools which can be considered as a MOF of varying feature: Comet, EvA2, evolvica, Evolutionary::Algorithm, GAPlayground, jaga, JCLEC, JGAP, jMetal, n-genes, Open Beagle, Opt4j, ParadisEO/EO, Pisa, Watchmaker, FOM, Hypercube, HotFrame, Templar, EasyLocal, iOpt, OptQuest, JDEAL, Optimization Algorithm Toolkit, HeuristicLab, MAFRA, Localizer, GALIB, DREAM, Discropt, MALLBA, MAGMA and UOF.

Contributions

Many different metaheuristics are in existence and new variants are continually being proposed. Some of the most significant contributions to the field are:

- 1952: Robbins and Monro work on stochastic optimization methods.
- 1954: Barricelli carry out the first simulations of the evolution process and use them on general optimization problems.
- 1963: Rastrigin proposes random search.
- 1965: Matyas proposes random optimization.
- 1965: Nelder and Mead propose a simplex heuristic, which was shown by Powell to converge to non-stationary points on some problems.

- 1965: Ingo Rechenberg discovers the first Evolution Strategies algorithm.
- 1966: Fogel et al. propose evolutionary programming.
- 1970: Hastings proposes the Metropolis–Hastings algorithm.
- 1970: Cavicchio proposes adaptation of control parameters for an optimizer.
- 1970: Kernighan and Lin propose a graph partitioning method, related to variable-depth search and prohibition-based (tabu) search.
- 1975: Holland proposes the genetic algorithm.
- 1977: Glover proposes scatter search.
- 1978: Mercer and Sampson propose a metaplan for tuning an optimizer's parameters by using another optimizer.
- 1980: Smith describes genetic programming.
- 1983: Kirkpatrick et al. propose simulated annealing.
- 1986: Glover proposes tabu search, first mention of the term *metaheuristic*.
- 1989: Moscato proposes memetic algorithms.
- 1990: Moscato and Fontanari , and Dueck and Scheuer , independently proposed a deterministic update rule for simulated annealing which accelerated the search. This led to the threshold accepting metaheuristic.
- 1992: Dorigo introduces ant colony optimization in his PhD thesis.
- 1995: Wolpert and Macready prove the no free lunch theorems.

Swarm intelligence

Swarm intelligence (**SI**) is the collective behavior of decentralized, self-organized systems, natural or artificial. The concept is employed in work on artificial intelligence. The expression was introduced by Gerardo Beni and Jing Wang in 1989, in the context of cellular robotic systems.

SI systems consist typically of a population of simple agents or boids interacting locally with one another and with their environment.

The inspiration often comes from nature, especially biological systems. The agents follow very simple rules, and although there is no centralized control structure dictating how individual agents should behave, local, and to a certain degree random, interactions between such agents lead to the emergence of "intelligent" global behavior, unknown to the individual agents. Examples of swarm intelligence in natural systems include ant colonies, bird flocking, hawks hunting, animal herding, bacterial growth, fish schooling and microbial intelligence.

The application of swarm principles to robots is called *swarm robotics* while *swarm intelligence* refers to the more general set of algorithms. *Swarm prediction* has been used in the context of forecasting problems.

Similar approaches to those proposed for swarm robotics are considered for genetically modified organisms in synthetic collective intelligence.

Models of swarm behavior

Boids (Reynolds 1987)

Boids is an artificial life program, developed by Craig Reynolds in 1986, which simulates the flocking behaviour of birds. His paper on this topic was published in 1987 in the proceedings of the ACM SIGGRAPH conference. The name "boid" corresponds to a shortened version of "bird-oid object", which refers to a bird-like object.

As with most artificial life simulations, Boids is an example of emergent behavior; that is, the complexity of Boids arises from the interaction of individual agents (the boids, in this case) adhering to a set of simple rules. The rules applied in the simplest Boids world are as follows:

- **separation**: steer to avoid crowding local flockmates
- **alignment**: steer towards the average heading of local flockmates
- **cohesion**: steer to move toward the average position (center of mass) of local flockmates

More complex rules can be added, such as obstacle avoidance and goal seeking.

Self-propelled particles (Vicsek et al. 1995)

Self-propelled particles (SPP), also referred to as the *Vicsek model*, was introduced in 1995 by Vicsek *et al.* as a special

case of the boids model introduced in 1986 by Reynolds. A swarm is modelled in SPP by a collection of particles that move with a constant speed but respond to a random perturbation by adopting at each time increment the average direction of motion of the other particles in their local neighbourhood. SPP models predict that swarming animals share certain properties at the group level, regardless of the type of animals in the swarm. Swarming systems give rise to emergent behaviours which occur at many different scales, some of which are turning out to be both universal and robust. It has become a challenge in theoretical physics to find minimal statistical models that capture these behaviours.

Metaheuristics

Evolutionary algorithms (EA), particle swarm optimization (PSO), differential evolution (DE), ant colony optimization (ACO) and their variants dominate the field of nature-inspired metaheuristics. This list includes algorithms published up to circa the year 2000. A large number of more recent metaphor-inspired metaheuristics have started to attract criticism in the research community for hiding their lack of novelty behind an elaborate metaphor. For algorithms published since that time, see List of metaphor-based metaheuristics.

It should also be noted that metaheuristics, as good as they are, lack a confidence in a solution. When appropriate parameters are determined, and when sufficient convergence stage is achieved, they often find a solution that is optimal, or near close to optimum – nevertheless, if one does not know optimal solution in advance, a quality of a solution is not

known. In spite of this obvious drawback it has been shown that these types of algorithms work well in practice, and have been extensively researched, and developed. On the other hand, it is possible to avoid this drawback by calculating solution quality for a special case where such calculation is possible, and after such run it is known that every solution that is at least as good as the solution a special case had, has at least a solution confidence a special case had. One such instance is Ant inspired Monte Carlo algorithm for Minimum Feedback Arc Set where this has been achieved probabilistically via hybridization of Monte Carlo algorithm with Ant Colony Optimization technique.

Stochastic diffusion search (Bishop 1989)

First published in 1989 Stochastic diffusion search (SDS) was the first Swarm Intelligence metaheuristic. SDS is an agent-based probabilistic global search and optimization technique best suited to problems where the objective function can be decomposed into multiple independent partial-functions. Each agent maintains a hypothesis that is iteratively tested by evaluating a randomly selected partial objective function parameterised by the agent's current hypothesis. In the standard version of SDS such partial function evaluations are binary, resulting in each agent becoming active or inactive. Information on hypotheses is diffused across the population via inter-agent communication. Unlike the stigmergic communication used in ACO, in SDS agents communicate hypotheses via a one-to-one communication strategy analogous to the tandem running procedure observed in Leptothorax acervorum. A positive feedback mechanism ensures that, over time, a population of agents stabilise around the global-best

solution. SDS is both an efficient and robust global search and optimisation algorithm, which has been extensively mathematically described. Recent work has involved merging the global search properties of SDS with other swarm intelligence algorithms.

Ant colony optimization (Dorigo 1992)

Ant colony optimization (ACO), introduced by Dorigo in his doctoral dissertation, is a class of optimization algorithms modeled on the actions of an ant colony. ACO is a probabilistic technique useful in problems that deal with finding better paths through graphs. Artificial 'ants'—simulation agents—locate optimal solutions by moving through a parameter space representing all possible solutions. Natural ants lay down pheromones directing each other to resources while exploring their environment. The simulated 'ants' similarly record their positions and the quality of their solutions, so that in later simulation iterations more ants locate for better solutions.

Particle swarm optimization (Kennedy, Eberhart & Shi 1995)

Particle swarm optimization (PSO) is a global optimization algorithm for dealing with problems in which a best solution can be represented as a point or surface in an n-dimensional space. Hypotheses are plotted in this space and seeded with an initial velocity, as well as a communication channel between the particles. Particles then move through the solution space, and are evaluated according to some fitness criterion after each timestep. Over time, particles are accelerated towards those particles within their communication grouping which have better fitness values. The main advantage of such an

approach over other global minimization strategies such as simulated annealing is that the large number of members that make up the particle swarm make the technique impressively resilient to the problem of local minima.

Artificial Swarm Intelligence (2015)

Artificial Swarm Intelligence (ASI) is method of amplifying the collective intelligence of networked human groups using control algorithms modeled after natural swarms. Sometimes referred to as Human Swarming or Swarm AI, the technology connects groups of human participants into real-time systems that deliberate and converge on solutions as dynamic swarms when simultaneously presented with a question ASI has been used for a wide range of applications, from enabling business teams to generate highly accurate financial forecasts to enabling sports fans to outperform Vegas betting markets. ASI has also been used to enable groups of doctors to generate diagnoses with significantly higher accuracy than traditional methods.

Applications

Swarm Intelligence-based techniques can be used in a number of applications. The U.S. military is investigating swarm techniques for controlling unmanned vehicles. The European Space Agency is thinking about an orbital swarm for self-assembly and interferometry. NASA is investigating the use of swarm technology for planetary mapping. A 1992 paper by M. Anthony Lewis and George A. Bekey discusses the possibility of using swarm intelligence to control nanobots within the body

for the purpose of killing cancer tumors. Conversely al-Rifaie and Aber have used stochastic diffusion search to help locate tumours. Swarm intelligence has also been applied for data mining. Ant based models are further subject of modern management theory.

Ant-based routing

The use of swarm intelligence in telecommunication networks has also been researched, in the form of ant-based routing. This was pioneered separately by Dorigo et al. and Hewlett Packard in the mid-1990s, with a number of variants existing. Basically, this uses a probabilistic routing table rewarding/reinforcing the route successfully traversed by each "ant" (a small control packet) which flood the network. Reinforcement of the route in the forwards, reverse direction and both simultaneously have been researched: backwards reinforcement requires a symmetric network and couples the two directions together; forwards reinforcement rewards a route before the outcome is known (but then one would pay for the cinema before one knows how good the film is). As the system behaves stochastically and is therefore lacking repeatability, there are large hurdles to commercial deployment. Mobile media and new technologies have the potential to change the threshold for collective action due to swarm intelligence (Rheingold: 2002, P175).

The location of transmission infrastructure for wireless communication networks is an important engineering problem involving competing objectives. A minimal selection of locations (or sites) are required subject to providing adequate area coverage for users. A very different-ant inspired swarm

intelligence algorithm, stochastic diffusion search (SDS), has been successfully used to provide a general model for this problem, related to circle packing and set covering. It has been shown that the SDS can be applied to identify suitable solutions even for large problem instances.

Airlines have also used ant-based routing in assigning aircraft arrivals to airport gates. At Southwest Airlines a software program uses swarm theory, or swarm intelligence—the idea that a colony of ants works better than one alone. Each pilot acts like an ant searching for the best airport gate. "The pilot learns from his experience what's the best for him, and it turns out that that's the best solution for the airline," Douglas A. Lawson explains. As a result, the "colony" of pilots always go to gates they can arrive at and depart from quickly. The program can even alert a pilot of plane back-ups before they happen. "We can anticipate that it's going to happen, so we'll have a gate available," Lawson says.

Crowd simulation

Artists are using swarm technology as a means of creating complex interactive systems or simulating crowds.

Stanley and Stella in: Breaking the Ice was the first movie to make use of swarm technology for rendering, realistically depicting the movements of groups of fish and birds using the Boids system. Tim Burton's *Batman Returns* also made use of swarm technology for showing the movements of a group of bats. *The Lord of the Rings* film trilogy made use of similar technology, known as Massive, during battle scenes. Swarm technology is particularly attractive because it is cheap, robust, and simple.

Airlines have used swarm theory to simulate passengers boarding a plane. Southwest Airlines researcher Douglas A. Lawson used an ant-based computer simulation employing only six interaction rules to evaluate boarding times using various boarding methods.(Miller, 2010, xii-xviii).

Human swarming

Enabled by mediating software such as the SWARM platform (formally unu) from Unanimous A.I., networks of distributed users can be organized into "human swarms" through the implementation of real-time closed-loop control systems. As published by Rosenberg (2015), such real-time systems enable groups of human participants to behave as a unified collective intelligence that works as a single entity to make predictions, answer questions, and evoke opinions. Such systems, also referred to as "Artificial Swarm Intelligence" (or the brand name Swarm AI) have been shown to significantly amplify human intelligence, resulting in a string of high-profile predictions of extreme accuracy. Academic testing shows that human swarms can out-predict individuals across a variety of real-world projections. Famously, human swarming was used to correctly predict the Kentucky Derby Superfecta, against 541 to 1 odds, in response to a challenge from reporters.

Medical Use of Human Swarming—in 2018, Stanford University School of Medicine and Unanimous AI published studies showing that groups of human doctors, when connected together by real-time swarming algorithms, could diagnose medical conditions with substantially higher accuracy than individual doctors or groups of doctors working together using traditional crowd-sourcing methods. In one such study,

swarms of human radiologists connected together using the SWARM platform were tasked with diagnosing chest x-rays and demonstrated a 33% reduction in diagnostic errors as compared to the traditional human methods, and a 22% improvement over traditional machine-learning.

Swarm grammars

Swarm grammars are swarms of stochastic grammars that can be evolved to describe complex properties such as found in art and architecture. These grammars interact as agents behaving according to rules of swarm intelligence. Such behavior can also suggest deep learning algorithms, in particular when mapping of such swarms to neural circuits is considered.

Swarmic art

In a series of works, al-Rifaie et al. have successfully used two swarm intelligence algorithms—one mimicking the behaviour of one species of ants (*Leptothorax acervorum*) foraging (stochastic diffusion search, SDS) and the other algorithm mimicking the behaviour of birds flocking (particle swarm optimization, PSO)—to describe a novel integration strategy exploiting the local search properties of the PSO with global SDS behaviour. The resulting hybrid algorithm is used to sketch novel drawings of an input image, exploiting an artistic tension between the local behaviour of the 'birds flocking'—as they seek to follow the input sketch—and the global behaviour of the "ants foraging"—as they seek to encourage the flock to explore novel regions of the canvas. The "creativity" of this hybrid swarm system has been analysed under the philosophical light of the "rhizome" in the context of Deleuze's "Orchid and Wasp" metaphor.

A more recent work of al-Rifaie et al., "Swarmic Sketches and Attention Mechanism", introduces a novel approach deploying the mechanism of 'attention' by adapting SDS to selectively attend to detailed areas of a digital canvas. Once the attention of the swarm is drawn to a certain line within the canvas, the capability of PSO is used to produce a 'swarmic sketch' of the attended line. The swarms move throughout the digital canvas in an attempt to satisfy their dynamic roles—attention to areas with more details—associated with them via their fitness function. Having associated the rendering process with the concepts of attention, the performance of the participating swarms creates a unique, non-identical sketch each time the 'artist' swarms embark on interpreting the input line drawings. In other works, while PSO is responsible for the sketching process, SDS controls the attention of the swarm.

In a similar work, "Swarmic Paintings and Colour Attention", non-photorealistic images are produced using SDS algorithm which, in the context of this work, is responsible for colour attention.

The "computational creativity" of the above-mentioned systems are discussed in through the two prerequisites of creativity (i.e. freedom and constraints) within the swarm intelligence's two infamous phases of exploration and exploitation.

Michael Theodore and Nikolaus Correll use swarm intelligent art installation to explore what it takes to have engineered systems to appear lifelike.

Nikolaus Correll

Nikolaus Correll (born 1977 in Munich, Germany) is a roboticist and an associate professor at the University of Colorado at Boulder in the Department of Computer Science with courtesy appointments in the departments of Aerospace, Electrical and Materials Engineering. Nikolaus is the faculty director of the Interdisciplinary Research Theme on Multi-functional Materials at the College of Engineering and Applied Science, and the founder and CTO of Robotic Materials Inc..

Correll obtained a Diploma (Masters) in Electrical Engineering from the Eidgenössische Technische Hochschule Zürich. He spent the first two years of his studies at the Technische Hochschule München, participated in the Erasmus Programme to spend a semester at Lunds Tekniska Hogsköla working with Rolf Johansson and wrote his Diploma thesis at Caltech working with Alcherio Martinoli and Joel Burdick.

Correll received his Dr. és science (Phd) degree in Computer Science in 2007 at the École polytechnique fédérale de Lausanne (EPFL) working under Alcherio Martinoli. Correll did a post-doc with Prof. Daniela Rus at Massachusetts Institute of Technology's Computer Science and Artificial Intelligence Laboratory. He became an assistant professor at the University of Colorado at Boulder in 2009, and was promoted to Associate Professor with tenure in 2017.

Correll is the recipient of a 2012 NSF *CAREER award*, the 2012 NASA *Early Career Faculty Fellowship*, and a 2016 *Provost*

Faculty Achievement Award. He is a senior member of the IEEE.

Correll's research is on Swarm Robotics, Swarm Intelligence, and Self-organization. He is using these concepts to equip composite materials with intelligence and enabling robots with autonomy, for which he coined the term Robotic Materials and founded a company of the same name, which markets robotic manipulation systems based on smart tactile sensors, skins and hands.

Correll is also an active researcher in robotics education and is the author of an open-source, collaborative textbook "Introduction to Autonomous Robots".

Correll's work on robotic materials has received worldwide media attention including the Associated Press, Neue Zuercher Zeitung, the IEEE Signal Processing Magazine, and Popular Science, among others.

In addition to serving as CEO of Robotic Materials Inc., Correll has been an advisor to TempAlert (acquired, NASDAQ: DGII), TABS, and Occam Robotics, and currently to Robotise SE.

Since starting at the University of Colorado in 2009, two of Correll's former PhD students have taken up faculty jobs at PhD-granting institutions:

- Erik Komendera, PhD 2014, Assistant Professor in Mechanical Engineering at Virginia Tech
- Michael Otte, PhD 2011, Assistant Professor in Aerospace Engineering at the University of Maryland, College Park

Honors:

- Keynote Speaker at the Int. Conf. on Intelligent Robots and Systems (IROS), 2020
- Provost Faculty Achievement Award, University of Colorado at Boulder, 2016
- Plenary speaker at the Int. Symposium on Distributed Autonomous Robotic Systems (DARS), 2016
- Best Paper Award at the 3rd Conference on System-integrated Intelligence (SysInt), 2016
- Best Paper Award at the Int. Symposium on Distributed Autonomous Robotic Systems (DARS) 2006 and 2014
- NSF CAREER award 2012
- NASA Early Career Faculty Fellowship 2012
- Best Paper Award at the Int. Conference on the Simulation of Adaptive Behavior (SAB), 2008

Marco Dorigo

Marco Dorigo (born 26 August 1961, in Milan, Italy) is a research director for the Belgian Funds for Scientific Research and a co-director of IRIDIA, the artificial intelligence lab of the Université Libre de Bruxelles. He is the proponent of the ant colony optimization metaheuristic (see his book published by MIT Press in 2004), and one of the founders of the swarm intelligence research field. Recently he got involved with research in swarm robotics: he is the coordinator of Swarm-bots: Swarms of self-assembling artefacts and of Swarmanoid:

Towards humanoid robotic swarms two swarm robotics projects funded by the Future and Emerging Technologies Program of the European Commission. He is also the founding editor and editor in chief of *Swarm Intelligence*, the principal peer reviewed publication dedicated to reporting research and new developments in this multidisciplinary field.

For these contributions, in 2003 he was awarded the Marie Curie Research Excellence Award by the European Commission; on 22 November 2005, he was presented the Dr A. De Leeuw-Damry-Bourlart award in Applied Sciences by the King of Belgium, Albert II; in 2007 he received the Cajastur International Prize for Soft Computing, awarded by the European Centre for Soft Computing; and in 2015 he received the IEEE Frank Rosenblatt Award. He is the recipient of an ERC Advanced Grant (2010).

Russell C. Eberhart

Russell C. Eberhart, an American electrical engineer, best known as the co-developer of particle swarm optimization concept (with James Kennedy (social psychologist)). He is professor of Electrical and Computer Engineering, and adjunct professor of Biomedical Engineering at the Purdue School of Engineering and Technology, Indiana University Purdue University Indianapolis (IUPUI). Fellow of the IEEE. Fellow of the American Institute for Medical and Biological Engineering.

He earned a Ph.D. in electrical engineering from Kansas State University in 1972. And he was Associate Editor of IEEE

Transactions on Evolutionary Computation and Past president of IEEE Neural Networks Council.

Luca Maria Gambardella

Luca Maria Gambardella (born 4 January 1962) is an Italian computer scientist and author. He is director of the Dalle Molle Institute for Artificial Intelligence Research in Manno, in the Ticino canton of Switzerland.

With Marco Dorigo and others, he has published papers on the application of ant colony optimization theory to the traveling salesman problem and similar questions. Several of his papers have been extensively cited.

Beside working in research, Gambardella is also a novelist. The genres he approached broad from Bildungsroman of his first book "Sei vite" ("Six lives"), to romance of his second book "Il suono dell'alba" ("The sound of sunrise").

James Kennedy (social psychologist)

James Kennedy (born November 5, 1950) is an American social psychologist, best known as an originator and researcher of particle swarm optimization. The first papers on the topic, by Kennedy and Russell C. Eberhart, were presented in 1995; since then tens of thousands of papers have been

published on particle swarms. The Academic Press / Morgan Kaufmann book, *Swarm Intelligence*, by Kennedy and Eberhart with Yuhui Shi, was published in 2001.

The particle swarm paradigm draws on social-psychological simulation research in which Kennedy had participated at the University of North Carolina, integrated with evolutionary computation methods that Eberhart had been working with in the 1990s. The result was a problem-solving or optimization algorithm based on the principles of human social interaction. Individuals begin the program with random guesses at the problem solution. As the program runs, the "particles" share their successes with their topological neighbors; each particle is both teacher and learner. Over time, the population converges reliably on optimal vectors.

While there has been a trend in the research literature toward a "Gbest" or centralized particle network, Blackwell and Kennedy (2018) demonstrated the importance of a distributed population topology in solving more complex problems.

A recent paper discusses the possible contribution of human female orgasm to the species' prosociality.

Kennedy has been an active combatant in the controversy over sex education in Montgomery County, Maryland, supporting the public schools' efforts to develop a comprehensive and inclusive program. He also worked to support a gender identity nondiscrimination law in Montgomery County that came under attack from conservatives, and has maintained an online progressive presence.

He also worked as a professional musician for fifty years and currently plays in a rockabilly band called The Colliders, which released albums in 2011 and 2015. In 2018 Kennedy released a DIY album, The Life of Mischief, and is currently organizing live performance of that material.

Kennedy worked in survey methods for the US government, and has conducted basic and applied research into social effects on cognition and attitude. He served as Director of the Office of Analysis and Research Services at the US International Trade Commission until his retirement in 2017. He has worked with particle swarms since 1994, with research publications in fields related and unrelated to swarms and surveys.