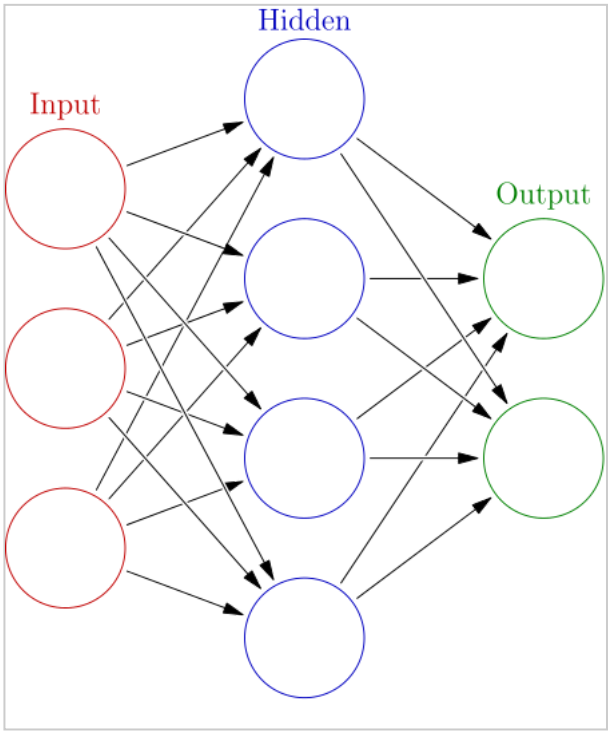
Artificial neural network

From Wikipedia, the free encyclopedia

Neural Networks (also referred to as **connectionist systems**) are a computational approach which is based on a large collection of neural units loosely modeling the way the brain solves problems with large clusters of biological neurons connected by axons. Each neural unit is connected with many others, and links can be enforcing or inhibitory in their effect on the activation state of connected neural units. Each individual neural unit may have a summation function which combines the values of all its inputs together. There may be a threshold function or limiting function on each connection and on the unit itself such that it must surpass it before it can propagate to other neurons. These systems are self-learning and trained rather than explicitly programmed and excel in areas where the solution or feature detection is difficult to express in a traditional computer program.



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 neuron to the input of another.

Neural networks typically consist of multiple layers or a cube design, and the signal path traverses from front to back. Back propagation is where the forward stimulation is used to reset weights on the "front" neural units and this is sometimes done in combination with training where the correct result is known. More modern networks are a bit more free flowing in terms of stimulation and inhibition with connections interacting in a much more chaotic and complex fashion. Dynamic neural networks are the most advanced in that they dynamically can, based on rules, form new connections and even new neural units while disabling others.

The goal of the neural network is to solve problems in the same way that the human brain would, although several neural networks are much more abstract. Modern neural network projects typically work with a few thousand to a few million neural units and millions of connections, which is still several orders of magnitude less complex than the human brain and closer to the computing power of a worm.

New brain research often stimulates new patterns in neural networks. One new approach is using connections which span much further and link processing layers rather than always being localized to adjacent neurons. Other research being explored with the different types of signal over time that axons propagate which is more complex than simply on or off.

Neural networks are based on real numbers, with the value of the core and of the axon typically being a representation between 0.0 and 1.

An interesting facet of these systems is that they are unpredictable in their success with self learning. After training some become great problem solvers and others don't perform as well. In order to train them several thousand cycles of interaction typically occur.

Like other machine learning methods – systems that learn from data – neural networks have been used to solve a wide variety of tasks, like computer vision and speech recognition, that are hard to solve using ordinary rule-based programming.

Historically, the use of neural network models marked a directional shift in the late eighties from high-level (symbolic) artificial intelligence, characterized by expert systems with knowledge embodied in *if-then* rules, to low-level (sub-symbolic) machine learning, characterized by knowledge embodied in the parameters of a dynamical system.

Contents

History

Warren McCulloch and Walter Pitts[1] (1943) created a computational model for neural networks based on mathematics and algorithms called threshold logic. This 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.

Hebbian learning

In the late 1940s psychologist Donald Hebb[2] 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. Researchers started applying these ideas to computational models in 1948 with Turing's B-type machines.

Farley and Wesley A. Clark[3] (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[4] (1956).

Frank Rosenblatt[5] (1958) created the perceptron, an algorithm for pattern recognition based on a two-layer computer learning 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 which could not be processed by neural networks until after the backpropagation algorithm was created by Paul Werbos[6] (1975).

Neural network research stagnated after the publication of machine learning research by Marvin Minsky and Seymour Papert[7] (1969), who discovered two key issues with the computational machines that processed neural networks. The first was that basic perceptrons were incapable of processing the exclusive-or circuit. The second significant issue was that computers didn't have enough processing power to effectively handle the long run time required by large neural networks. Neural network research slowed until computers achieved greater processing power.

Backpropagation and resurgence

A key advance that came later was the backpropagation algorithm which effectively solved the exclusive-or problem, and more generally the problem of quickly training multi-layer neural networks (Werbos 1975).[6]

In the mid-1980s, parallel distributed processing became popular under the name connectionism. The textbook by David E. Rumelhart and James McClelland[8] (1986) provided a full exposition of 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 the biological architecture of the brain is debated; it's not clear to what degree artificial neural networks mirror brain function.[9]

Support vector machines and other, much simpler methods such as linear classifiers gradually overtook neural networks in machine learning popularity. As computing power increased through the use of GPUs and distributed computing, image and visual recognition problems came to the forefront, neural networks were deployed again, on larger scales. This became called "deep learning" which is simply a re-branding of neural networks, though emphasizing the use of modern parallel hardware implementations.

Improvements since 2006

Computational devices have been created in CMOS, for both biophysical simulation and neuromorphic computing. More recent efforts show promise for creating nanodevices[10] for very large scale principal components analyses and convolution. If successful, it would create a new class of neural computing[11] 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.[12][13] For example, the bi-directional and multi-dimensional long short-term memory (LSTM)[14][15][16][17] of Alex Graves et al. 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.

Fast GPU-based implementations of this approach by Dan Ciresan and colleagues at IDSIA have won several pattern recognition contests, including the IJCNN 2011 Traffic Sign Recognition Competition,[18][19] the ISBI 2012 Segmentation of Neuronal Structures in Electron Microscopy Stacks challenge,[20] and others. Their neural networks also were the first artificial pattern recognizers to achieve human-competitive or even superhuman performance[21] on important benchmarks such as traffic sign recognition (IJCNN 2012), or the MNIST handwritten digits problem of Yann LeCun at NYU.

Deep, highly nonlinear neural architectures similar to the 1980 neocognitron by Kunihiko Fukushima[22] and the "standard architecture of vision",[23] inspired by the simple and complex cells identified by David H. Hubel and Torsten Wiesel in the primary visual cortex, can also be pre-trained by unsupervised methods[24][25] of Geoff Hinton's lab at University of Toronto.[26][27] A team from this lab won a 2012 contest sponsored by Merck to design software to help find molecules that might lead to new drugs.[28]

Models

Neural network models in artificial intelligence are usually referred to as artificial neural networks (ANNs); these are essentially simple mathematical models defining a function $f : X \to Y$ or a distribution over X or both X and Y, but sometimes models are also intimately associated with a particular learning algorithm or learning rule. A common use of the phrase "ANN model" is really the definition of a *class* of such functions (where members of the class are obtained by varying parameters, connection weights, or specifics of the architecture such as the number of neurons or their connectivity).

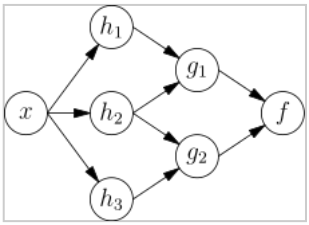
Network function

The word *network* in the term 'artificial neural network' refers to the interconnections between the neurons in the different layers of each system. An example system has three layers. The first layer has input neurons which send data via synapses to the second layer of neurons, and then via more synapses to the third layer of output neurons. More complex systems will have more layers of neurons, some having increased layers of input neurons and output neurons. The synapses store parameters called "weights" that manipulate the data in the calculations.

An ANN is typically defined by three types of parameters:

1. The interconnection pattern between the different layers of neurons
2. The learning process for updating the weights of the interconnections
3. The activation function that converts a neuron's weighted input to its output activation.

Mathematically, a neuron's network function $f(x)$ is defined as a composition of other functions $g_i(x)$, which can further be defined as a composition of other functions. This can be conveniently represented as a network structure, with arrows depicting the dependencies between variables. A widely used type of composition is the *nonlinear weighted sum*, where $f(x) = K\left(\sum_i w_i g_i(x)\right)$, where K (commonly referred to as the activation function[29]) is some predefined function, such as the hyperbolic tangent. It will be convenient for the following to refer to a collection of functions g_i as simply a vector $g = (g_1, g_2, \ldots, g_n)$.



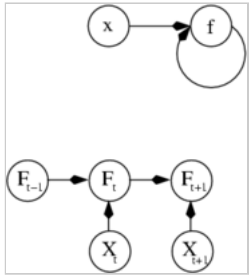
ANN dependency graph

This figure depicts such a decomposition of f, with dependencies between variables indicated by arrows. These can be interpreted in two ways.

The first view is the functional view: the input x is transformed into a 3-dimensional vector h, which is then transformed into a 2-dimensional vector g, which is finally transformed into f. This view is most commonly encountered in the context of optimization.

The second view is the probabilistic view: the random variable $F = f(G)$ depends upon the random variable $G = g(H)$, which depends upon $H = h(X)$, which depends upon the random variable X. This view is most commonly encountered in the context of graphical models.

The two views are largely equivalent. In either case, for this particular network architecture, the components of individual layers are independent of each other (e.g., the components of g are independent of each other given their input h). This naturally enables a degree of parallelism in the implementation.



Two separate depictions of the recurrent ANN dependency graph

Networks such as the previous one are commonly called feedforward, because their graph is a directed acyclic graph. Networks with cycles are commonly called recurrent. Such networks are commonly depicted in the manner shown at the top of the figure, where f is shown as being dependent upon itself. However, an implied temporal dependence is not shown.

Learning

What has attracted the most interest in neural networks is the possibility of *learning*. Given a specific *task* to solve, and a *class* of functions F, learning means using a set of *observations* to find $f^* \in F$ which solves the task in some *optimal* sense.

This entails defining a cost function $C : F \to \mathbb{R}$ such that, for the optimal solution f^*, $C(f^*) \leq C(f)\ \forall f \in F$ – i.e., no solution has a cost less than the cost of the optimal solution (see mathematical optimization).

The cost function C is an important concept in learning, as it is a measure of how far away a particular solution is from an optimal solution to the problem to be solved. Learning algorithms search through the solution space to find a function that has the smallest possible cost.

For applications where the solution is dependent on some data, the cost must necessarily be a *function of the observations*, otherwise we would not be modelling anything related to the data. It is frequently defined as a statistic to which only approximations can be made. As a simple example, consider the problem of finding the model f, which minimizes $C = E\left[(f(x) - y)^2\right]$, for data pairs (x, y) drawn from some distribution $\mathcal{D}$. In practical situations we would only have N samples from $\mathcal{D}$ and thus, for the above example, we would only minimize $\hat{C} = \frac{1}{N}\sum_{i=1}^{N}(f(x_i) - y_i)^2$. Thus, the cost is minimized over a sample of the data rather than the entire distribution generating the data.

When $N \to \infty$ some form of online machine learning must be used, where the cost is partially minimized as each new example is seen. While online machine learning is often used when $\mathcal{D}$ is fixed, it is most useful in the case where the distribution changes slowly over time. In neural network methods, some form of online machine learning is frequently used for finite datasets.

Choosing a cost function

While it is possible to define some arbitrary ad hoc cost function, frequently a particular cost will be used, either because it has desirable properties (such as convexity) or because it arises naturally from a particular formulation of the problem (e.g., in a probabilistic formulation the posterior probability of the model can be used as an inverse cost). Ultimately, the cost function will depend on the desired task. An overview of the three main categories of learning tasks is provided below:

Learning paradigms

There are three major learning paradigms, each corresponding to a particular abstract learning task. These are supervised learning, unsupervised learning and reinforcement learning.

Supervised learning

In supervised learning, we are given a set of example pairs $(x, y), x \in X, y \in Y$ and the aim is to find a function $f : X \to Y$ in the allowed class of functions that matches the examples. In other words, we wish to *infer* the mapping implied by the data; the cost function is related to the mismatch between our mapping and the data and it implicitly contains prior knowledge about the problem domain.

A commonly used cost is the mean-squared error, which tries to minimize the average squared error between the network's output, $f(x)$, and the target value y over all the example pairs. When one tries to minimize this cost using gradient descent for the class of neural networks called multilayer perceptrons (MLP), one obtains the common and well-known backpropagation algorithm for training neural networks.

Tasks that fall within the paradigm of supervised learning are pattern recognition (also known as classification) and regression (also known as function approximation). The supervised learning paradigm is also applicable to sequential data (e.g., for speech and gesture recognition). This can be thought of as learning with a "teacher", in the form of a function that provides continuous feedback on the quality of solutions obtained thus far.

Unsupervised learning

In unsupervised learning, some data x is given and the cost function to be minimized, that can be any function of the data x and the network's output, f.

The cost function is dependent on the task (what we are trying to model) and our *a priori* assumptions (the implicit properties of our model, its parameters and the observed variables).

As a trivial example, consider the model $f(x) = a$ where a is a constant and the cost $C = E[(x - f(x))^2]$. Minimizing this cost will give us a value of a that is equal to the mean of the data. The cost function can be much more complicated. Its form depends on the application: for example, in compression it could be related to the mutual information between x and $f(x)$, whereas in statistical modeling, it could be related to the posterior probability of the model given the data (note that in both of those examples those quantities would be maximized rather than minimized).

Tasks that fall within the paradigm of unsupervised learning are in general estimation problems; the applications include clustering, the estimation of statistical distributions, compression and filtering.

Reinforcement learning

In reinforcement learning, data $\boldsymbol{x}$ are usually not given, but generated by an agent's interactions with the environment. At each point in time $\boldsymbol{t}$, the agent performs an action $\boldsymbol{y_t}$ and the environment generates an observation $\boldsymbol{x_t}$ and an instantaneous cost $\boldsymbol{c_t}$, according to some (usually unknown) dynamics. The aim is to discover a *policy* for selecting actions that minimizes some measure of a long-term cost, e.g., the expected cumulative cost. The environment's dynamics and the long-term cost for each policy are usually unknown, but can be estimated.

More formally the environment is modeled as a Markov decision process (MDP) with states $\boldsymbol{s_1, \ldots, s_n \in S}$ and actions $\boldsymbol{a_1, \ldots, a_m \in A}$ with the following probability distributions: the instantaneous cost distribution $\boldsymbol{P(c_t|s_t)}$, the observation distribution $\boldsymbol{P(x_t|s_t)}$ and the transition $\boldsymbol{P(s_{t+1}|s_t, a_t)}$, while a policy is defined as the conditional distribution over actions given the observations. Taken together, the two then define a Markov chain (MC). The aim is to discover the policy (i.e., the MC) that minimizes the cost.

ANNs are frequently used in reinforcement learning as part of the overall algorithm.[30][31] Dynamic programming has been coupled with ANNs (giving neurodynamic programming) by Bertsekas and Tsitsiklis[32] and applied to multi-dimensional nonlinear problems such as those involved in vehicle routing,[33] natural resources management[34][35] or medicine[36] because of the ability of ANNs to mitigate losses of accuracy even when reducing the discretization grid density for numerically approximating the solution of the original control problems.

Tasks that fall within the paradigm of reinforcement learning are control problems, games and other sequential decision making tasks.

Learning algorithms

Training a neural network model essentially means selecting one model from the set of allowed models (or, in a Bayesian framework, determining a distribution over the set of allowed models) that minimizes the cost criterion. There are numerous algorithms available for training neural network models; most of them can be viewed as a straightforward application of optimization theory and statistical estimation.

Most of the algorithms used in training artificial neural networks employ some form of gradient descent, using backpropagation to compute the actual gradients. This is done by simply taking the derivative of the cost function with respect to the network parameters and then changing those parameters in a gradient-related direction. The backpropagation training algorithms are usually classified into three categories:

1. steepest descent (with variable learning rate, with variable learning rate and momentum, resilient backpropagation);
2. quasi-Newton (Broyden-Fletcher-Goldfarb-Shanno, one step secant);
3. Levenberg-Marquardt and conjugate gradient (Fletcher-Reeves update, Polak-Ribiére update, Powell-Beale restart, scaled conjugate gradient).[37]

Evolutionary methods,[38] gene expression programming,[39] simulated annealing,[40] expectation-maximization, non-parametric methods and particle swarm optimization[41] are some other methods for training neural networks.

Employing artificial neural networks

Perhaps the greatest advantage of ANNs is their ability to be used as an arbitrary function approximation mechanism that 'learns' from observed data.[42] However, using them is not so straightforward, and a relatively good understanding of the underlying theory is essential.

- Choice of model: This will depend on the data representation and the application. Overly complex models tend to lead to challenges in learning.
- Learning algorithm: There are numerous trade-offs between learning algorithms. Almost any algorithm will work well with the *correct hyperparameters* for training on a particular fixed data set. However, selecting and tuning an algorithm for training on unseen data require a significant amount of experimentation.
- Robustness: If the model, cost function and learning algorithm are selected appropriately, the resulting ANN can be extremely robust.

With the correct implementation, ANNs can be used naturally in online learning and large data set applications. Their simple implementation and the existence of mostly local dependencies exhibited in the structure allows for fast, parallel implementations in hardware.

Applications

The utility of artificial neural network models lies in the fact that they can be used to infer a function from observations. This is particularly useful in applications where the complexity of the data or task makes the design of such a function by hand impractical.

Real-life applications

The tasks artificial neural networks are applied to tend to fall within the following broad categories:

- Function approximation, or regression analysis, including time series prediction, fitness approximation and modeling
- Classification, including pattern and sequence recognition, novelty detection and sequential decision making
- Data processing, including filtering, clustering, blind source separation and compression
- Robotics, including directing manipulators, prosthesis.
- Control, including Computer numerical control

Application areas include the system identification and control (vehicle control, trajectory prediction,[43] process control, natural resources management), quantum chemistry,[44] game-playing and decision making (backgammon, chess, poker), pattern recognition (radar systems, face identification, object recognition and more), sequence recognition (gesture, speech, handwritten text recognition), medical diagnosis, financial applications (e.g. automated trading systems), data mining (or knowledge discovery in databases, "KDD"), visualization and e-mail spam filtering.

Artificial neural networks have also been used to diagnose several cancers. An ANN based hybrid lung cancer detection system named HLND improves the accuracy of diagnosis and the speed of lung cancer radiology.[45] These networks have also been used to diagnose prostate cancer. The diagnoses can be used to make specific models taken from a large group of patients compared to information of one given patient. The models do not depend on assumptions about correlations of different variables. Colorectal cancer has also been predicted using the neural networks. Neural networks could predict the outcome for a patient with colorectal cancer with more accuracy than the current clinical methods. After training, the networks could predict multiple patient outcomes from unrelated institutions.[46]

Neural networks and neuroscience

Theoretical and computational neuroscience is the field concerned with the theoretical analysis and the computational modeling of biological neural systems. Since neural systems are intimately related to cognitive processes and behavior, the field is closely related to cognitive and behavioral 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 in the field, defined at different levels of abstraction and modeling different aspects of neural systems. They range from models of the short-term behavior of individual neurons (e.g.[47]), models of how the dynamics of neural circuitry arise from interactions between individual neurons and finally to models of how behavior can arise from abstract neural modules that represent complete subsystems. These include models of the long-term, and short-term plasticity, of neural systems and their relations to learning and memory from the individual neuron to the system level.

Networks with memory

Integrating external memory components with artificial neural networks has a long history dating back to early research in distributed representations[48] and self-organizing maps. E.g. in sparse distributed memory the patterns encoded by neural networks are used as memory addresses for content-addressable memory, with "neurons" essentially serving as address encoders and decoders.

More recently deep learning was shown to be useful in semantic hashing[49] where a deep graphical model of the word-count vectors[50] is obtained from a large set of documents. Documents are mapped to memory addresses in such a way that semantically similar documents are located at nearby addresses. Documents similar to a query document can then be found by simply accessing all the addresses that differ by only a few bits from the address of the query document.

Memory Networks[51] is another extension to neural networks incorporating long-term memory which was developed by Facebook research. The long-term memory can be read and written to, with the goal of using it for prediction. These models have been applied in the context of question answering (QA) where the long-term memory effectively acts as a (dynamic) knowledge base, and the output is a textual response.

Neural Turing Machines[52] developed by Google DeepMind extend the capabilities of deep 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 but is differentiable end-to-end, allowing it to be efficiently trained with gradient descent. Preliminary results demonstrate that Neural Turing Machines can infer simple algorithms such as copying, sorting, and associative recall from input and output examples.

Differentiable neural computers (DNC) are an extension of Neural Turing Machines, also from DeepMind. They have out-performed Neural turing machines, Long short-term memory systems and memory networks on sequence-processing tasks.[53][54][55][56][57]

Neural network software

Neural network software is used to simulate, research, develop and apply artificial neural networks, biological neural networks and, in some cases, a wider array of adaptive systems.

Types of artificial neural networks

Artificial neural network types vary from those with only one or two layers of single direction logic, to complicated multi–input many directional feedback loops and layers. On the whole, these systems use algorithms in their programming to determine control and organization of their functions. Most systems use "weights" to change the parameters of the throughput and the varying connections to the neurons. Artificial neural networks can be autonomous and learn by input from outside "teachers" or even self-teaching from written-in rules. Neural Cube style neural networks first pioneered by Gianna Giavelli provide a dynamic space in which networks dynamically recombine information and links across billions of self adapting nodes utilizing Neural Darwinism,[58] a technique developed by Gerald Edelman which allows for more biologically modeled systems.

Theoretical properties

Computational power

The multilayer perceptron is a universal function approximator, as proven by the universal approximation theorem. However, the proof is not constructive regarding the number of neurons required, the network topology, the settings of the weights and the learning parameters.

Work by Hava Siegelmann and Eduardo D. Sontag has provided a proof that a specific recurrent architecture with rational valued weights (as opposed to full precision real number-valued weights) has the full power of a Universal Turing Machine[59] using a finite number of neurons and standard linear connections. Further, it has been shown that the use of irrational values for weights results in a machine with super-Turing power.[60]

Capacity

Artificial neural network models have a property called 'capacity', which roughly corresponds to their ability to model any given function. It is related to the amount of information that can be stored in the network and to the notion of complexity.

Convergence

Nothing can be said in general about convergence since it depends on a number of factors. Firstly, there may exist many local minima. This depends on the cost function and the model. Secondly, the optimization method used might not be guaranteed to converge when far away from a local minimum. Thirdly, for a very large amount of data or parameters, some methods become impractical. In general, it has been found that theoretical guarantees regarding convergence are an unreliable guide to practical application.

Generalization and statistics

In applications where the goal is to create a system that generalizes well in unseen examples, the problem of over-training has emerged. This arises in convoluted or over-specified systems when the capacity of the network significantly exceeds the needed free parameters. There are two schools of thought for avoiding this problem: The first is to use cross-validation and similar techniques to check for the presence of overtraining and optimally select hyperparameters such as to minimize the generalization error. The second is to use some form of *regularization*. This is a concept that emerges naturally in a probabilistic (Bayesian) framework, where the regularization can be

performed by selecting a larger prior probability over simpler models; but also in statistical learning theory, where the goal is to minimize over two quantities: the 'empirical risk' and the 'structural risk', which roughly corresponds to the error over the training set and the predicted error in unseen data due to overfitting.

Supervised neural networks that use a mean squared error (MSE) cost function can use formal statistical methods to determine the confidence of the trained model. The MSE on a validation set can be used as an estimate for variance. This value can then be used to calculate the confidence interval of the output of the network, assuming a normal distribution. A confidence analysis made this way is statistically valid as long as the output probability distribution stays the same and the network is not modified.

Confidence analysis of a neural network

By assigning a softmax activation function, a generalization of the logistic function, on the output layer of the neural network (or a softmax component in a component-based neural network) for categorical target variables, the outputs can be interpreted as posterior probabilities. This is very useful in classification as it gives a certainty measure on classifications.

The softmax activation function is:

$$y_i = \frac{e^{x_i}}{\sum_{j=1}^{c} e^{x_j}}$$

Criticism

Training issues

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 A. 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.

Theoretical issues

A. K. Dewdney, a mathematician and computer scientist at University of Western Ontario and 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".[61] No neural network has ever been shown that solves computationally difficult problems such as the n-Queens problem, the travelling salesman problem, or the problem of factoring large integers.

Aside from their utility, a fundamental objection to artificial neural networks is that they fail to reflect how real neurons function. Back propagation is at the heart of most artificial neural networks and not only is there no evidence of any such mechanism in natural neural networks, it seems to contradict the fundamental principle of real neurons that information can only flow forward along the axon.[62] How information is coded by real neurons is not yet known. What is known is that sensor neurons fire action potentials more frequently with sensor activation and muscle cells pull more strongly when their associated motor neurons receive action potentials more frequently.[63] Other than the simplest case of just relaying information from a sensor neuron to a motor neuron almost nothing of the underlying general principles of how information is handled by real neural networks is known.

The motivation behind artificial neural networks is not necessarily to replicate real neural function but to use natural neural networks as an inspiration for an approach to computing that is inherently parallel and which provides solutions to problems that have up until now been considered intractable. A central claim of artificial neural networks is therefore that it embodies some new and powerful general principle for processing information. Unfortunately, these general principles are ill defined and it is often claimed that they are *emergent* from the neural network itself. This allows simple statistical association (the basic function of artificial neural networks) to be described as *learning* or *recognition*. As a result, artificial neural networks have a "something-for-nothing quality, one that imparts a peculiar aura of laziness and a distinct lack of curiosity about just how good these computing systems are. No human hand (or mind) intervenes; solutions are found as if by magic; and no one, it seems, has learned anything".[61]

Hardware issues

To implement large and effective software neural networks, considerable processing and storage resources need to be committed.[64] 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 architecture 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.

Jürgen Schmidhuber notes that the resurgence of neural networks in the twenty-first century, and their renewed success at image recognition tasks is largely attributable to advances in hardware: from 1991 to 2015, computing power, especially as delivered by GPGPUs (on GPUs), has increased around a million-fold, making the standard backpropagation algorithm feasible for training networks that are several layers deeper than before (but adds that this doesn't overcome algorithmic problems such as vanishing gradients "in a fundamental way").[65] The use of GPUs instead of ordinary CPUs can bring training times for some networks down from months to mere days.[64]

Computing power continues to grow roughly according to Moore's Law, which may provide sufficient resources to accomplish new tasks. Neuromorphic engineering addresses the hardware difficulty directly, by constructing non-von-Neumann chips with circuits designed to implement neural nets from the ground up. Google has also designed a chip optimized for neural network processing called a Tensor Processing Unit, or TPU.[66]

Practical counterexamples to criticisms

Arguments against Dewdney's position are that neural networks have been successfully used to solve many complex and diverse tasks, ranging from autonomously flying aircraft[67] to detecting credit card fraud.

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.[68]

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. Furthermore, researchers involved in exploring learning algorithms for neural networks are gradually uncovering generic principles which 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.[69]

Hybrid approaches

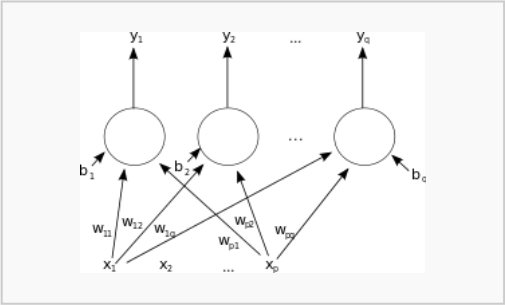
Some other criticisms come from advocates of hybrid models (combining neural networks and symbolic approaches), who believe that the intermix of these two approaches can better capture the mechanisms of the human mind.[70][71]

Classes and types of ANNs

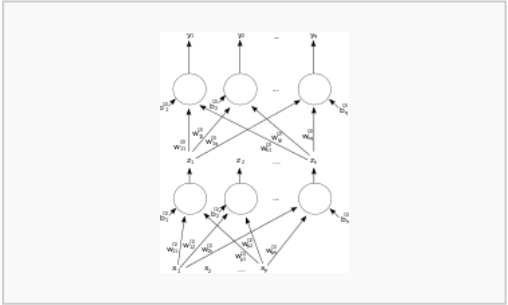
- Dynamic Neural Network
  - Feedforward neural network (FNN)
  - Recurrent neural network (RNN)
    - Hopfield network
    - Boltzmann machine
    - Simple recurrent networks
    - Echo state network
    - Long short-term memory
    - Bi-directional RNN
    - Hierarchical RNN
    - Stochastic neural networks
  - Kohonen Self-Organizing Maps
  - Autoencoder
  - Probabilistic neural network (PNN)
  - Time delay neural network (TDNN)
  - Regulatory feedback network (RFNN)
- Static Neural Network
  - Neocognitron
  - McCulloch-Pitts cell
  - Radial basis function network (RBF)
  - Learning vector quantization
  - Perceptron
    - Adaline model
    - Convolutional neural network (CNN)
  - Modular neural networks
    - Committee of machines (COM)
    - Associative neural network (ASNN)

- Memory Network [1] (https://www.facebook.com/FBAIResearch/posts/362517620591864)
  - Google / Deep Mind
  - facebook / MemNN (https://github.com/facebook/MemNN)
  - Holographic associative memory
  - One-shot associative memory
  - Neural Turing Machine
  - Adaptive resonance theory
  - Hierarchical temporal memory
- Other types of networks
  - Instantaneously trained neural networks (ITNN)
  - Spiking neural network (SNN)
    - Pulse Coded Neural Networks (PCNN)
  - Cascading neural networks
  - Neuro-fuzzy networks
  - Growing Neural Gas (GNG)
  - Compositional pattern-producing networks
  - Counterpropagation network
  - Oscillating neural network
  - Hybridization neural network
  - Physical neural network
    - Optical neural network

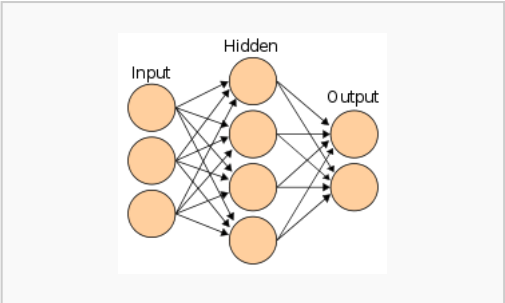
Gallery



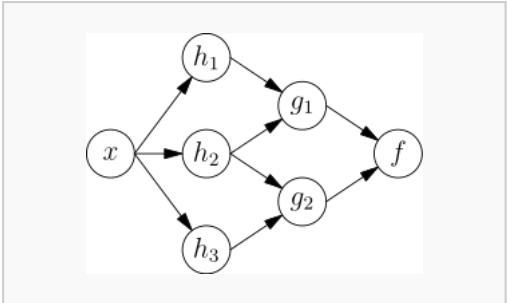
A single-layer feedforward artificial neural network. Arrows originating from x_2 are omitted for clarity. There are p inputs to this network and q outputs. In this system, the value of the qth output, y_q would be calculated as $y_q = K*(\sum(x_i * w_{iq}) - b_q)$



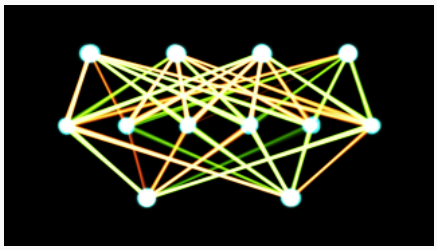
A two-layer feedforward artificial neural network.



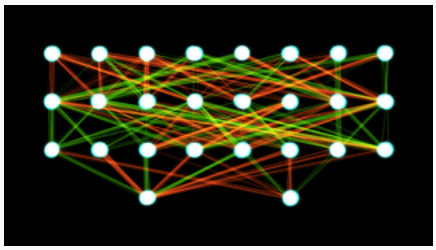
undefined



undefined



A single-layer feedforward artificial neural network with 4 inputs, 6 hidden and 2 outputs. Given position state and direction outputs wheel based control values.



A two-layer feedforward artificial neural network with 8 inputs, 2x8 hidden and 2 outputs. Given position state, direction and other environment values outputs thruster based control values.

See also

- 20Q
- ADALINE
- Adaptive resonance theory
- Artificial life
- Associative memory
- Autoencoder
- BEAM robotics
- Biological cybernetics
- Biologically inspired computing
- Blue Brain Project
- Catastrophic interference
- Cerebellar Model Articulation Controller
- Cognitive architecture
- Cognitive science
- Convolutional neural network (CNN)
- Connectionist expert system
- Connectomics
- Cultured neuronal networks
- Deep learning
- Digital morphogenesis
- Encog
- Fuzzy logic
- Gene expression programming
- Genetic algorithm
- Genetic programming
- Group method of data handling
- Habituation
- In Situ Adaptive Tabulation
- Models of neural computation
- Multilinear subspace learning
- Neuroevolution
- Neural coding
- Neural gas
- Neural machine translation
- Neural network software

- Neuroscience
- Ni1000 chip
- Nonlinear system identification
- Optical neural network
- Parallel Constraint Satisfaction Processes
- Parallel distributed processing
- Radial basis function network
- Recurrent neural networks
- Self-organizing map
- Spiking neural network
- Systolic array
- Tensor product network
- Time delay neural network (TDNN)

References

1. McCulloch, Warren; Walter Pitts (1943). "A Logical Calculus of Ideas Immanent in Nervous Activity". *Bulletin of Mathematical Biophysics*. **5** (4): 115–133. doi:10.1007/BF02478259.
2. Hebb, Donald (1949). *The Organization of Behavior*. New York: Wiley.
3. Farley, B.G.; W.A. Clark (1954). "Simulation of Self-Organizing Systems by Digital Computer". *IRE Transactions on Information Theory*. **4** (4): 76–84. doi:10.1109/TIT.1954.1057468.
4. Rochester, N.; J.H. Holland; L.H. Habit; W.L. Duda (1956). "Tests on a cell assembly theory of the action of the brain, using a large digital computer". *IRE Transactions on Information Theory*. **2** (3): 80–93. doi:10.1109/TIT.1956.1056810.
5. Rosenblatt, F. (1958). "The Perceptron: A Probabilistic Model For Information Storage And Organization In The Brain". *Psychological Review*. **65** (6): 386–408. doi:10.1037/h0042519. PMID 13602029.
6. Werbos, P.J. (1975). *Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences*.
7. Minsky, M.; S. Papert (1969). *Perceptrons: An Introduction to Computational Geometry*. MIT Press. ISBN 0-262-63022-2.
8. Rumelhart, D.E; James McClelland (1986). *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*. Cambridge: MIT Press.
9. Russell, Ingrid. "Neural Networks Module". Retrieved 2012. Check date values in: `|access-date=` (help)
10. Yang, J. J.; Pickett, M. D.; Li, X. M.; Ohlberg, D. A. A.; Stewart, D. R.; Williams, R. S. (2008). "Memristive switching mechanism for metal/oxide/metal nanodevices". *Nat. Nanotechnol.* **3**: 429–433. doi:10.1038/nnano.2008.160.
11. Strukov, D. B.; Snider, G. S.; Stewart, D. R.; Williams, R. S. (2008). "The missing memristor found". *Nature*. **453**: 80–83. doi:10.1038/nature06932. PMID 18451858.
12. 2012 Kurzweil AI Interview (http://www.kurzweilai.net/how-bio-inspired-deep-learning-keeps-winning-competitions) with Jürgen Schmidhuber on the eight competitions won by his Deep Learning team 2009–2012
13. http://www.kurzweilai.net/how-bio-inspired-deep-learning-keeps-winning-competitions 2012 Kurzweil AI Interview with Jürgen Schmidhuber on the eight competitions won by his Deep Learning team 2009–2012
14. Graves, Alex; and Schmidhuber, Jürgen; *Offline Handwriting Recognition with Multidimensional Recurrent Neural Networks (http://www.idsia.ch/~juergen/nips2009.pdf)*, in Bengio, Yoshua; Schuurmans, Dale; Lafferty, John; Williams, Chris K. I.; and Culotta, Aron (eds.), *Advances in Neural Information Processing Systems 22 (NIPS'22), 7–10 December 2009, Vancouver, BC*, Neural Information Processing Systems (NIPS) Foundation, 2009, pp. 545–552.
15. Graves, A.; Liwicki, M.; Fernandez, S.; Bertolami, R.; Bunke, H.; Schmidhuber, J. (2009). "A Novel Connectionist System for Improved Unconstrained Handwriting Recognition" (PDF). *IEEE Transactions on Pattern Analysis and Machine Intelligence*. **31** (5).
16. Graves, Alex; and Schmidhuber, Jürgen; *Offline Handwriting Recognition with Multidimensional Recurrent Neural Networks*, in Bengio, Yoshua; Schuurmans, Dale; Lafferty, John; Williams, Chris K. I.; and Culotta, Aron (eds.), *Advances in Neural Information Processing Systems 22 (NIPS'22), December 7th–10th, 2009, Vancouver, BC*, Neural Information Processing Systems (NIPS) Foundation, 2009, pp. 545–552
17. Graves, A.; Liwicki, M.; Fernandez, S.; Bertolami, R.; Bunke, H.; Schmidhuber, J. (2009). "A Novel Connectionist System for Improved Unconstrained Handwriting Recognition". *IEEE Transactions on Pattern Analysis and Machine Intelligence*. **31** (5).

18. D. C. Ciresan, U. Meier, J. Masci, J. Schmidhuber. Multi-Column Deep Neural Network for Traffic Sign Classification. Neural Networks, 2012.
19. D. C. Ciresan, U. Meier, J. Masci, J. Schmidhuber. Multi-Column Deep Neural Network for Traffic Sign Classification. Neural Networks, 2012.
20. D. Ciresan, A. Giusti, L. Gambardella, J. Schmidhuber. Deep Neural Networks Segment Neuronal Membranes in Electron Microscopy Images. In Advances in Neural Information Processing Systems (NIPS 2012), Lake Tahoe, 2012.
21. D. C. Ciresan, U. Meier, J. Schmidhuber. Multi-column Deep Neural Networks for Image Classification. IEEE Conf. on Computer Vision and Pattern Recognition CVPR 2012.
22. Fukushima, K. (1980). "Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position". *Biological Cybernetics*. **36** (4): 93–202. doi:10.1007/BF00344251. PMID 7370364.
23. M Riesenhuber, T Poggio. Hierarchical models of object recognition in cortex. Nature neuroscience, 1999.
24. Deep belief networks (http://www.scholarpedia.org/article/Deep_belief_networks) at Scholarpedia.
25. Hinton, G. E.; Osindero, S.; Teh, Y. W. (2006). "A Fast Learning Algorithm for Deep Belief Nets" (PDF). *Neural Computation*. **18** (7): 1527–1554. doi:10.1162/neco.2006.18.7.1527. PMID 16764513.
26. http://www.scholarpedia.org/article/Deep_belief_network/
27. Hinton, G. E.; Osindero, S.; Teh, Y. (2006). "A fast learning algorithm for deep belief nets" (PDF). *Neural Computation*. **18** (7): 1527–1554. doi:10.1162/neco.2006.18.7.1527. PMID 16764513.
28. John Markoff (November 23, 2012). "Scientists See Promise in Deep-Learning Programs". *New York Times*.
29. "The Machine Learning Dictionary".
30. Dominic, S.; Das, R.; Whitley, D.; Anderson, C. (July 1991). "Genetic reinforcement learning for neural networks". *IJCNN-91-Seattle International Joint Conference on Neural Networks*. IJCNN-91-Seattle International Joint Conference on Neural Networks. Seattle, Washington, USA: IEEE. doi:10.1109/IJCNN.1991.155315. ISBN 0-7803-0164-1. Retrieved 29 July 2012.
31. Hoskins, J.C.; Himmelblau, D.M. (1992). "Process control via artificial neural networks and reinforcement learning". *Computers & Chemical Engineering*. **16** (4): 241–251. doi:10.1016/0098-1354(92)80045-B.
32. Bertsekas, D.P.; Tsitsiklis, J.N. (1996). *Neuro-dynamic programming*. Athena Scientific. p. 512. ISBN 1-886529-10-8.
33. Secomandi, Nicola (2000). "Comparing neuro-dynamic programming algorithms for the vehicle routing problem with stochastic demands". *Computers & Operations Research*. **27** (11–12): 1201–1225. doi:10.1016/S0305-0548(99)00146-X.
34. de Rigo, D.; Rizzoli, A. E.; Soncini-Sessa, R.; Weber, E.; Zenesi, P. (2001). "Neuro-dynamic programming for the efficient management of reservoir networks" (PDF). *Proceedings of MODSIM 2001, International Congress on Modelling and Simulation*. MODSIM 2001, International Congress on Modelling and Simulation. Canberra, Australia: Modelling and Simulation Society of Australia and New Zealand. doi:10.5281/zenodo.7481. ISBN 0-867405252. Retrieved 29 July 2012.
35. Damas, M.; Salmeron, M.; Diaz, A.; Ortega, J.; Prieto, A.; Olivares, G. (2000). "Genetic algorithms and neuro-dynamic programming: application to water supply networks". *Proceedings of 2000 Congress on Evolutionary Computation*. 2000 Congress on Evolutionary Computation. La Jolla, California, USA: IEEE. doi:10.1109/CEC.2000.870269. ISBN 0-7803-6375-2. Retrieved 29 July 2012.
36. Deng, Geng; Ferris, M.C. (2008). "Neuro-dynamic programming for fractionated radiotherapy planning". *Springer Optimization and Its Applications*. **12**: 47–70. doi:10.1007/978-0-387-73299-2_3.
37. M. Forouzanfar; H. R. Dajani; V. Z. Groza; M. Bolic & S. Rajan (July 2010). *Comparison of Feed-Forward Neural Network Training Algorithms for Oscillometric Blood Pressure Estimation* (PDF). 4th Int. Workshop Soft Computing Applications. Arad, Romania: IEEE.
38. de Rigo, D., Castelletti, A., Rizzoli, A.E., Soncini-Sessa, R., Weber, E. (January 2005). "A selective improvement technique for fastening Neuro-Dynamic Programming in Water Resources Network Management". In Pavel Zítek. *Proceedings of the 16th IFAC World Congress – IFAC-PapersOnLine*. 16th IFAC World Congress. **16**. Prague, Czech Republic: IFAC. doi:10.3182/20050703-6-CZ-1902.02172. ISBN 978-3-902661-75-3. Retrieved 30 December 2011.
39. Ferreira, C. (2006). "Designing Neural Networks Using Gene Expression Programming" (PDF). In A. Abraham, B. de Baets, M. Köppen, and B. Nickolay, eds., Applied Soft Computing Technologies: The Challenge of Complexity, pages 517–536, Springer-Verlag.
40. Da, Y.; Xiurun, G. (July 2005). T. Villmann, ed. *An improved PSO-based ANN with simulated annealing technique*. New Aspects in Neurocomputing: 11th European Symposium on Artificial Neural Networks. Elsevier. doi:10.1016/j.neucom.2004.07.002.
41. Wu, J.; Chen, E. (May 2009). Wang, H., Shen, Y., Huang, T., Zeng, Z., eds. *A Novel Nonparametric Regression Ensemble for Rainfall Forecasting Using Particle Swarm Optimization Technique Coupled with Artificial Neural Network*. 6th International Symposium on Neural Networks, ISNN 2009. Springer. doi:10.1007/978-3-642-01513-7-6. ISBN 978-3-642-01215-0.

42. "A Survey Of Techniques for Approximate Computing (https://www.academia.edu/20201007/A_Survey_Of_Techniques_for_Approximate_Computing)", S. Mittal, ACM Computing Surveys, 2016
43. Zissis, Dimitrios (October 2015). "A cloud based architecture capable of perceiving and predicting multiple vessel behaviour". *Applied Soft Computing*. **35**: 652–661. doi:10.1016/j.asoc.2015.07.002.
44. Roman M. Balabin; Ekaterina I. Lomakina (2009). "Neural network approach to quantum-chemistry data: Accurate prediction of density functional theory energies". *J. Chem. Phys.* **131** (7): 074104. doi:10.1063/1.3206326. PMID 19708729.
45. Ganesan, N. "Application of Neural Networks in Diagnosing Cancer Disease Using Demographic Data" (PDF). International Journal of Computer Applications.
46. Bottaci, Leonardo. "Artificial Neural Networks Applied to Outcome Prediction for Colorectal Cancer Patients in Separate Institutions" (PDF). The Lancet.
47. Forrest MD (April 2015). "Simulation of alcohol action upon a detailed Purkinje neuron model and a simpler surrogate model that runs >400 times faster". *BMC Neuroscience*. **16** (27). doi:10.1186/s12868-015-0162-6.
48. Hinton, Geoffrey E. "Distributed representations." (1984)
49. Salakhutdinov, Ruslan, and Geoffrey Hinton. "Semantic hashing." International Journal of Approximate Reasoning 50.7 (2009): 969-978.
50. Le, Quoc V., and Tomas Mikolov. "Distributed representations of sentences and documents." arXiv preprint arXiv:1405.4053 (2014).
51. Weston, Jason, Sumit Chopra, and Antoine Bordes. "Memory networks." arXiv preprint arXiv:1410.3916 (2014).
52. Graves, Alex, Greg Wayne, and Ivo Danihelka. "Neural Turing Machines." arXiv preprint arXiv:1410.5401 (2014).
53. Burgess, Matt. "DeepMind's AI learned to ride the London Underground using human-like reason and memory". *WIRED UK*. Retrieved 2016-10-19.
54. "DeepMind AI 'Learns' to Navigate London Tube". *PCMAG*. Retrieved 2016-10-19.
55. Mannes, John. "DeepMind's differentiable neural computer helps you navigate the subway with its memory". *TechCrunch*. Retrieved 2016-10-19.
56. Graves, Alex; Wayne, Greg; Reynolds, Malcolm; Harley, Tim; Danihelka, Ivo; Grabska-Barwińska, Agnieszka; Colmenarejo, Sergio Gómez; Grefenstette, Edward; Ramalho, Tiago (2016-10-12). "Hybrid computing using a neural network with dynamic external memory". *Nature*. advance online publication. doi:10.1038/nature20101. ISSN 1476-4687.
57. "Differentiable neural computers | DeepMind". *DeepMind*. Retrieved 2016-10-19.
58. Neural Darwinism, the theory of neuronal group selection, basic books, 1987
59. Siegelmann, H.T.; Sontag, E.D. (1991). "Turing computability with neural nets" (PDF). *Appl. Math. Lett.* **4** (6): 77–80. doi:10.1016/0893-9659(91)90080-F.
60. Balcázar, José (Jul 1997). "Computational Power of Neural Networks: A Kolmogorov Complexity Characterization". *Information Theory, IEEE Transactions on*. **43** (4): 1175–1183. doi:10.1109/18.605580. Retrieved 3 November 2014.
61. Dewdney, A.K. (1997). *Yes, We Have No Neutrons: An Eye-Opening Tour through the Twists and Turns of Bad Science*. New York: Wiley. p. 82.
62. Crick, Francis (1989). "The recent excitement about neural networks". *Nature*. **337** (6203): 129–132. doi:10.1038/337129a0.
63. Adrian, Edward D. (1926). "The impulses produced by sensory nerve endings". *The Journal of Physiology*. **61** (1): 49–72. doi:10.1113/jphysiol.1926.sp002273.
64. Edwards, Chris (25 June 2015). "Growing pains for deep learning". *Communications of the ACM*. **58** (7): 14–16. doi:10.1145/2771283.
65. Schmidhuber, Jürgen (2015). "Deep learning in neural networks: An overview". *Neural Networks*. **61**: 85–117. arXiv:1404.7828. doi:10.1016/j.neunet.2014.09.003.
66. Cade Metz (May 18, 2016). "Google Built Its Very Own Chips to Power Its AI Bots". *Wired*.
67. NASA - Dryden Flight Research Center - News Room: News Releases: NASA NEURAL NETWORK PROJECT PASSES MILESTONE (http://www.nasa.gov/centers/dryden/news/NewsReleases/2003/03-49.html). Nasa.gov. Retrieved on 2013-11-20.
68. Roger Bridgman's defence of neural networks (http://members.fortunecity.com/templarseries/popper.html)
69. "Scaling Learning Algorithms towards {AI} - LISA - Publications - Aigaion 2.0".
70. Sun and Bookman (1990)
71. Tahmasebi; Hezarkhani (2012). "A hybrid neural networks-fuzzy logic-genetic algorithm for grade estimation". *Computers & Geosciences*. **42**: 18–27. doi:10.1016/j.cageo.2012.02.004.

Bibliography

- Bhadeshia H. K. D. H. (1999). "Neural Networks in Materials Science" (PDF). *ISIJ International*. **39** (10): 966–979. doi:10.2355/isijinternational.39.966.

- Bishop, C.M. (1995) *Neural Networks for Pattern Recognition*, Oxford: Oxford University Press. ISBN 0-19-853849-9 (hardback) or ISBN 0-19-853864-2 (paperback)
- Cybenko, G.V. (1989). Approximation by Superpositions of a Sigmoidal function, *Mathematics of Control, Signals, and Systems*, Vol. 2 pp. 303–314. electronic version (http://actcomm.dartmouth.edu/gvc/papers/approx_by_superposition.pdf)
- Dewdney, A.K. (1997). *Yes, We Have No Neutrons: An Eye-Opening Tour through the Twists and Turns of Bad Science*. New York: Wiley. ISBN 978-0-471-10806-1.
- Duda, R.O., Hart, P.E., Stork, D.G. (2001) *Pattern classification (2nd edition)*, Wiley, ISBN 0-471-05669-3
- Egmont-Petersen, M.; de Ridder, D.; Handels, H. (2002). "Image processing with neural networks – a review". *Pattern Recognition*. **35** (10): 2279–2301. doi:10.1016/S0031-3203(01)00178-9.
- Gurney, K. (1997) *An Introduction to Neural Networks* London: Routledge. ISBN 1-85728-673-1 (hardback) or ISBN 1-85728-503-4 (paperback)
- Haykin, S. (1999) *Neural Networks: A Comprehensive Foundation*, Prentice Hall, ISBN 0-13-273350-1
- Fahlman, S, Lebiere, C (1991). *The Cascade-Correlation Learning Architecture*, created for National Science Foundation, Contract Number EET-8716324, and Defense Advanced Research Projects Agency (DOD), ARPA Order No. 4976 under Contract F33615-87-C-1499. electronic version (http://www.cs.iastate.edu/~honavar/fahlman.pdf)
- Hertz, J., Palmer, R.G., Krogh. A.S. (1990) *Introduction to the theory of neural computation*, Perseus Books. ISBN 0-201-51560-1
- Lawrence, Jeanette (1994) *Introduction to Neural Networks*, California Scientific Software Press. ISBN 1-883157-00-5
- MacKay, David, J.C. (2003). *Information Theory, Inference, and Learning Algorithms* (PDF). Cambridge University Press. ISBN 9780521642989.
- Masters, Timothy (1994) *Signal and Image Processing with Neural Networks*, John Wiley & Sons, Inc. ISBN 0-471-04963-8
- Ripley, Brian D. (1996) *Pattern Recognition and Neural Networks*, Cambridge
- Siegelmann, H.T. and Sontag, E.D. (1994). Analog computation via neural networks, *Theoretical Computer Science*, v. 131, no. 2, pp. 331–360. electronic version (http://www.math.rutgers.edu/~sontag/FTP_DIR/nets-real.pdf)
- Smith, Murray (1993) *Neural Networks for Statistical Modeling*, Van Nostrand Reinhold, ISBN 0-442-01310-8
- Wasserman, Philip (1993) *Advanced Methods in Neural Computing*, Van Nostrand Reinhold, ISBN 0-442-00461-3
- *Computational Intelligence: A Methodological Introduction* by Kruse, Borgelt, Klawonn, Moewes, Steinbrecher, Held, 2013, Springer, ISBN 9781447150121
- *Neuro-Fuzzy-Systeme* (3rd edition) by Borgelt, Klawonn, Kruse, Nauck, 2003, Vieweg, ISBN 9783528252656

External links

Wikibooks has a book on the topic of: ***Artificial Neural Networks***

- Neural Networks (https://www.dmoz.org/Computers/Artificial_Intelligence/Neural_Networks) at DMOZ
- A brief introduction to Neural Networks (http://www.dkriesel.com/en/science/neural_networks) (PDF), illustrated 250p textbook covering the common kinds of neural networks (CC license).
- An Introduction to Deep Neural Networks (http://deeplearning4j.org/neuralnet-overview.html).
- MIT course on Neural Networks (video) (https://www.youtube.com/watch?v=q0pm3BrIUFo)
- Neural-network (https://github.com/A1essandro/neural-network) github-repository (PHP)
- A Concise Introduction to Machine Learning with Artificial Neural Networks (https://www.academia.edu/25708860/A_Concise_Introduction_to_Machine_Learning_with_Artificial_Neural_Networks)
- Neural Networks for Machine Learning - a course by Geoffrey Hinton (https://www.coursera.org/course/neuralnets)
- Deep Learning - An MIT Press Book (http://www.deeplearningbook.org/)

Categories: Computational statistics | Artificial neural networks | Classification algorithms | Computational neuroscience | Mathematical psychology

---