Deep Learning

more at http://ml.memect.com

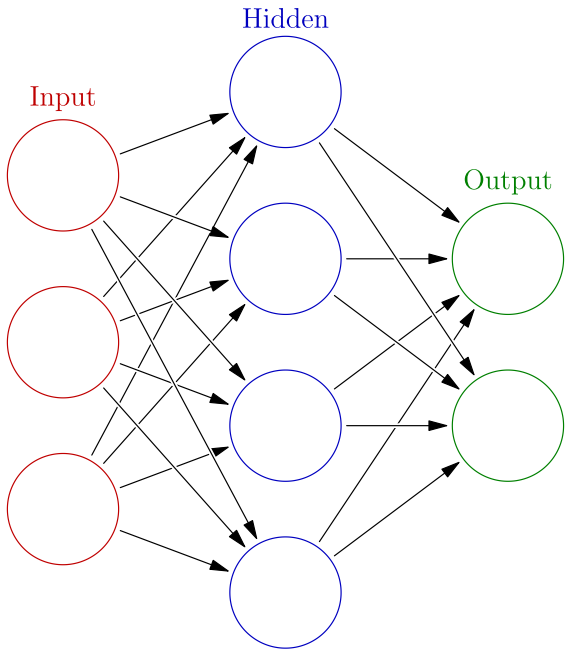
Contents

Chapter 1

Artificial neural network

"Neural network" redirects here. For networks of living neurons, see Biological neural network. For the journal, see Neural Networks (journal). For the evolutionary concept, see Neutral network (evolution).

In machine learning and cognitive science, **artificial**



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 (**ANNs**) are a family of statistical learning models inspired by biological neural networks (the central nervous systems of animals, in particular the brain) and are used to estimate or approximate functions that can depend on a large number of inputs and are generally unknown. Artificial neural networks are generally presented as systems of interconnected "neurons" which send messages to each other. The connections have numeric weights that can be tuned based on experience, making neural nets adaptive to inputs and capable of learning.

For example, a neural network for handwriting recognition is defined by a set of input neurons which may be activated by the pixels of an input image. After being weighted and transformed by a function (determined by the network's designer), the activations of these neurons are then passed on to other neurons. This process is repeated until finally, an output neuron is activated. This determines which character was read.

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

1.1 Background

Examinations of the human's central nervous system inspired the concept of neural networks. In an Artificial Neural Network, simple artificial nodes, known as "neurons", "neurodes", "processing elements" or "units", are connected together to form a network which mimics a biological neural network.

There is no single formal definition of what an artificial neural network is. However, a class of statistical models may commonly be called "Neural" if they possess the following characteristics:

1. consist of sets of adaptive weights, i.e. numerical parameters that are tuned by a learning algorithm, and
2. are capable of approximating non-linear functions of their inputs.

The adaptive weights are conceptually connection strengths between neurons, which are activated during training and prediction.

Neural networks are similar to biological neural networks in performing functions collectively and in parallel by the units, rather than there being a clear delineation of subtasks to which various units are assigned. The term "neural network" usually refers to models employed in statistics, cognitive psychology and artificial intelligence.

Neural network models which emulate the central nervous system are part of theoretical neuroscience and computational neuroscience.

In modern software implementations of artificial neural networks, the approach inspired by biology has been largely abandoned for a more practical approach based on statistics and signal processing. In some of these systems, neural networks or parts of neural networks (like artificial neurons) form components in larger systems that combine both adaptive and non-adaptive elements. While the more general approach of such systems is more suitable for real-world problem solving, it has little to do with the traditional artificial intelligence connectionist models. What they do have in common, however, is the principle of non-linear, distributed, parallel and local processing and adaptation. Historically, the use of neural networks models marked a paradigm shift in the late eighties from high-level (symbolic) AI, 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.

1.2 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.

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. These ideas started being applied 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 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 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 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 later advances was the backpropagation algorithm which effectively solved the exclusive-or problem (Werbos 1975).*[6]

The parallel distributed processing of the mid-1980s became popular under the name connectionism. The text by David E. Rumelhart and James McClelland*[8] (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.*[9]

Neural networks were gradually overtaken in popularity in machine learning by support vector machines and other, much simpler methods such as linear classifiers. Renewed interest in neural nets was sparked in the late 2000s by the advent of deep learning.

1.2.1 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, these efforts could usher in a new era of neural computing*[11] 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.*[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]

1.3 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 $f : X \rightarrow Y$ 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 really means 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).

1.3.1 Network function

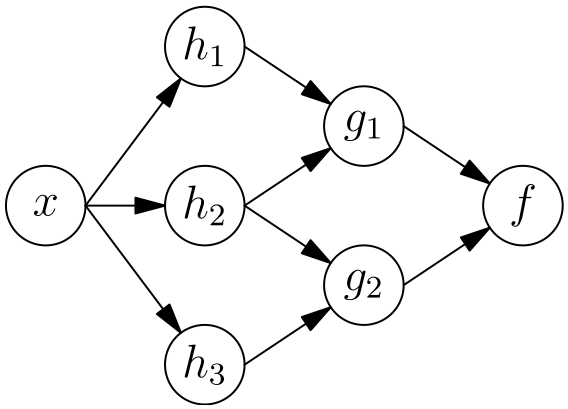
See also: Graphical models

The word *network* in the term 'artificial neural network' refers to the inter–connections 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 with 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

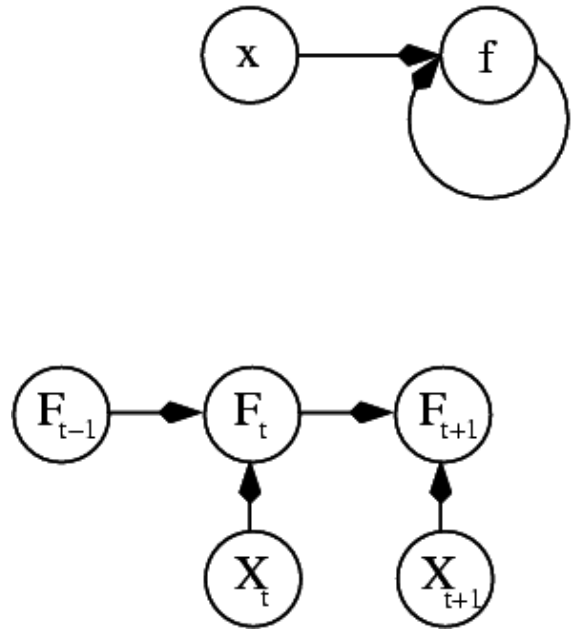
This figure depicts such a decomposition of f , with dependencies between variables indicated by arrows. These can be interpreted in two ways.

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.

Networks such as the previous one are commonly called feedforward, because their graph is a directed acyclic graph. Networks with cycles are commonly called



Two separate depictions of the recurrent ANN dependency graph

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.

1.3.2 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 data set.

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.

See also: Mathematical optimization, Estimation theory and Machine learning

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:

1.3.3 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, 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 x are usually not given, but generated by an agent's interactions with the environment. At each point in time t , the agent performs an action y_t and the environment generates an observation x_t and an instantaneous cost 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; i.e., 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 modelled as a Markov decision process (MDP) with states $s_1, ..., s_n \in S$ and actions $a_1, ..., a_m \in A$ with the following probability distributions: the instantaneous cost distribution $P(c_t|s_t)$, the observation distribution $P(x_t|s_t)$ and the transition $P(s_{t+1}|s_t, a_t)$, while a policy is defined as conditional distribution over actions given the observations. Taken together, the two then define a Markov chain (MC). The aim is to discover the policy that minimizes the cost; i.e., the MC for which the cost is minimal.

ANNs are frequently used in reinforcement learning as part of the overall algorithm.[*][30][*][31] Dynamic programming has been coupled with ANNs (Neuro dynamic 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.

See also: dynamic programming and stochastic control

1.3.4 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.

Evolutionary methods,[*][37] gene expression programming,[*][38] simulated annealing,[*][39] expectation-maximization, non-parametric methods and particle swarm optimization[*][40] are some commonly used methods for training neural networks.

See also: machine learning

1.4 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. 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 problems with 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 un-

seen data requires 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.

1.5 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.

1.5.1 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, process control, natural resources management), quantum chemistry,[*][41] gameplaying 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.[*][42] 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.[*][43]

1.5.2 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, 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.

1.6 Neural network software

Main article: 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.

1.7 Types of artificial neural networks

Main article: 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.

1.8 Theoretical properties

1.8.1 Computational power

The multi-layer perceptron (MLP) is a universal function approximator, as proven by the universal approximation theorem. However, the proof is not constructive regarding the number of neurons required or the settings of the weights.

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* [44] 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.* [45]

1.8.2 Capacity

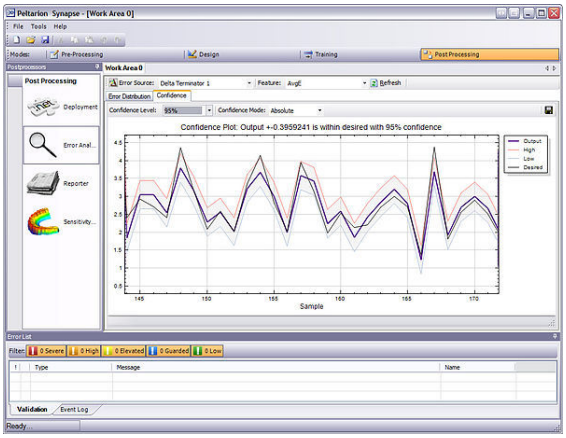
Artificial neural network models have a property called 'capacity', which roughly corresponds to their ability to model any given function. It is related to the amount of information that can be stored in the network and to the notion of complexity.

1.8.3 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.

1.8.4 Generalization and statistics

In applications where the goal is to create a system that generalizes well in unseen examples, the problem of overtraining has emerged. This arises in convoluted or overspecified 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.



Confidence analysis of a neural network

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.

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:

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

1.9 Controversies

1.9.1 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 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)

1.9.2 Hardware issues

To implement large and effective software neural networks, considerable 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).

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.

1.9.3 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*[46] 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.*[47]

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.*[48]

1.9.4 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.*[49]*[50]

1.10 Gallery

- A single-layer feedforward artificial neural network. Arrows originating from are omitted for clarity. There are p inputs to this network and q outputs. In this system, the value of the qth output, would be calculated as
- A two-layer feedforward artificial neural network.
-
-

1.11 See also

- 20Q
- ADALINE
- Adaptive resonance theory
- Artificial life
- Associative memory
- Autoencoder
- Backpropagation
- BEAM robotics
- Biological cybernetics
- Biologically inspired computing
- Blue brain
- 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
- Group method of data handling
- Habituation
- In Situ Adaptive Tabulation
- Models of neural computation
- Multilinear subspace learning
- Neuroevolution
- Neural coding
- Neural gas
- 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)

1.12 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, and 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). *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.

[10] Yang, J. J.; Pickett, M. D.; Li, X. M.; Ohlberg, D. A. A.; Stewart, D. R.; Williams, R. S. Nat. Nanotechnol. 2008, 3, 429–433.

[11] Strukov, D. B.; Snider, G. S.; Stewart, D. R.; Williams, R. S. *Nature* 2008, 453, 80–83.

[12] 2012 Kurzweil AI Interview 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*, 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] A. Graves, M. Liwicki, S. Fernandez, R. Bertolami, H. Bunke, J. Schmidhuber. A Novel Connectionist System for Improved Unconstrained Handwriting Recognition. IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 31, no. 5, 2009.

[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] A. Graves, M. Liwicki, S. Fernandez, R. Bertolami, H. Bunke, J. Schmidhuber. A Novel Connectionist System for Improved Unconstrained Handwriting Recognition. IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 31, no. 5, 2009.

[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 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_networks /

[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] 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.

[38] 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.

[39] 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.

[40] Wu, J., Chen, E. (May 2009). Wang, H., Shen, Y., Huang, T., Zeng, Z., ed. *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.

[41] 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.

[42] Ganesan, N. "Application of Neural Networks in Diagnosing Cancer Disease Using Demographic Data" (PDF). International Journal of Computer Applications.

[43] Bottaci, Leonardo. "Artificial Neural Networks Applied to Outcome Prediction for Colorectal Cancer Patients in Separate Institutions" (PDF). The Lancet.

[44] 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.

[45] 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.

[46] NASA - Dryden Flight Research Center - News Room: News Releases: NASA NEURAL NETWORK PROJECT PASSES MILESTONE. Nasa.gov. Retrieved on 2013-11-20.

[47] Roger Bridgman's defence of neural networks

[48] http://www.iro.umontreal.ca/~{ }lisa/publications2/index.php/publications/show/4

[49] Sun and Bookman (1990)

[50] 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.

1.13 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
- 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

- 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
- 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
- Sergios Theodoridis, Konstantinos Koutroumbas (2009) "Pattern Recognition", 4th Edition, Academic Press, ISBN 978-1-59749-272-0.
- 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

1.14 External links

- Neural Networks at DMOZ
- A brief introduction to Neural Networks (PDF), illustrated 250p textbook covering the common kinds of neural networks (CC license).

Chapter 2

Deep learning

For deep versus shallow learning in educational psychology, see Student approaches to learning

Deep learning (*deep machine learning*, or *deep structured learning*, or *hierarchical learning*, or sometimes *DL*) is a branch of machine learning based on a set of algorithms that attempt to model high-level abstractions in data by using model architectures, with complex structures or otherwise, composed of multiple non-linear transformations.[*][1][*](p198)[*][2][*][3][*][4]

Deep learning is part of a broader family of machine learning methods based on learning representations of data. An observation (e.g., an image) can be represented in many ways such as a vector of intensity values per pixel, or in a more abstract way as a set of edges, regions of particular shape, etc. Some representations make it easier to learn tasks (e.g., face recognition or facial expression recognition[*][5]) from examples. One of the promises of deep learning is replacing handcrafted features with efficient algorithms for unsupervised or semi-supervised feature learning and hierarchical feature extraction.[*][6]

Research in this area attempts to make better representations and create models to learn these representations from large-scale unlabeled data. Some of the representations are inspired by advances in neuroscience and are loosely based on interpretation of information processing and communication patterns in a nervous system, such as neural coding which attempts to define a relationship between the stimulus and the neuronal responses and the relationship among the electrical activity of the neurons in the brain.[*][7]

Various deep learning architectures such as deep neural networks, convolutional deep neural networks, and deep belief networks have been applied to fields like computer vision, automatic speech recognition, natural language processing, audio recognition and bioinformatics where they have been shown to produce state-of-the-art results on various tasks.

Alternatively, *deep learning* has been characterized as a buzzword, or a rebranding of neural networks.[*][8][*][9]

2.1 Introduction

2.1.1 Definitions

There are a number of ways that the field of deep learning has been characterized. Deep learning is a class of machine learning training algorithms that[*][1][*](pp199–200)

- use a cascade of many layers of nonlinear processing units for feature extraction and transformation. The next layer uses the output from the previous layer as input. The algorithms may be supervised or unsupervised and applications include pattern recognition and statistical classification.
- are based on the (unsupervised) learning of multiple levels of features or representations of the data. Higher level features are derived from lower level features to form a hierarchical representation.
- are part of the broader machine learning field of learning representations of data.
- learn multiple levels of representations that correspond to different levels of abstraction; the levels form a hierarchy of concepts.

These definitions have in common (1) multiple layers of nonlinear processing units and (2) the supervised or unsupervised learning of feature representations in each layer, with the layers forming a hierarchy from low-level to high-level features.[*][1][*](p200) The composition of a layer of nonlinear processing units used in a deep belief algorithm depends on the problem to be solved. Layers that have been used in deep learning include hidden layers of an artificial neural network, restricted Boltzmann machines, and sets of complicated propositional formulas.[*][2]

Deep learning algorithms are contrasted with shallow learning algorithms by the number of parameterized transformations a signal encounters as it propagates from the input layer to the output layer, where a parameterized transformation is a processing unit that has trainable parameters, such as weights and thresholds.[*][4][*](p6) A

chain of transformations from input to output is a *credit assignment path* (CAP). CAPs describe potentially causal connections between input and output and may vary in length. For a feedforward neural network, the depth of the CAPs, and thus the depth of the network, is the number of hidden layers plus one (the output layer is also parameterized). For recurrent neural networks, in which a signal may propagate through a layer more than once, the CAP is potentially unlimited in length. There is no universally agreed upon threshold of depth dividing shallow learning from deep learning, but most researchers in the field agree that deep learning has multiple nonlinear layers (CAP > 2) and Schmidhuber considers CAP > 10 to be very deep learning.*[4]*(p7)

2.1.2 Fundamental concepts

Deep learning algorithms are based on distributed representations. The underlying assumption behind distributed representations is that observed data is generated by the interactions of many different factors on different levels. Deep learning adds the assumption that these factors are organized into multiple levels, corresponding to different levels of abstraction or composition. Varying numbers of layers and layer sizes can be used to provide different amounts of abstraction.*[3]

Deep learning algorithms in particular exploit this idea of hierarchical explanatory factors. Different concepts are learned from other concepts, with the more abstract, higher level concepts being learned from the lower level ones. These architectures are often constructed with a greedy layer-by-layer method that models this idea. Deep learning helps to disentangle these abstractions and pick out which features are useful for learning.*[3]

For supervised learning tasks where label information is readily available in training, deep learning promotes a principle which is very different than traditional methods of machine learning. That is, rather than focusing on feature engineering which is often labor-intensive and varies from one task to another, deep learning methods is focused on end-to-end learning based on raw features. In other words, deep learning moves away from feature engineering to a maximal extent possible. To accomplish end-to-end optimization starting with raw features and ending in labels, layered structures are often necessary. From this perspective, we can regard the use of layered structures to derive intermediate representations in deep learning as a natural consequence of raw-feature-based end-to-end learning.*[1] Understanding the connection between the above two aspects of deep learning is important to appreciate its use in several application areas, all involving supervised learning tasks (e.g., supervised speech and image recognition), as to be discussed in a later part of this article.

Many deep learning algorithms are framed as unsupervised learning problems. Because of this, these algorithms can make use of the unlabeled data that supervised algorithms cannot. Unlabeled data is usually more abundant than labeled data, making this an important benefit of these algorithms. The deep belief network is an example of a deep structure that can be trained in an unsupervised manner.*[3]

2.2 History

Deep learning architectures, specifically those built from artificial neural networks (ANN), date back at least to the Neocognitron introduced by Kunihiko Fukushima in 1980.*[10] The ANNs themselves date back even further. In 1989, Yann LeCun et al. were able to apply the standard backpropagation algorithm, which had been around since 1974,*[11] to a deep neural network with the purpose of recognizing handwritten ZIP codes on mail. Despite the success of applying the algorithm, the time to train the network on this dataset was approximately 3 days, making it impractical for general use.*[12] Many factors contribute to the slow speed, one being due to the so-called vanishing gradient problem analyzed in 1991 by Sepp Hochreiter.*[13]*[14]

While such neural networks by 1991 were used for recognizing isolated 2-D hand-written digits, 3-D object recognition by 1991 used a 3-D model-based approach – matching 2-D images with a handcrafted 3-D object model. Juyang Weng *et al.*. proposed that a human brain does not use a monolithic 3-D object model and in 1992 they published Cresceptron,*[15]*[16]*[17] a method for performing 3-D object recognition directly from cluttered scenes. Cresceptron is a cascade of many layers similar to Neocognitron. But unlike Neocognitron which required the human programmer to hand-merge features, Cresceptron fully *automatically* learned an open number of unsupervised features in each layer of the cascade where each feature is represented by a convolution kernel. In addition, Cresceptron also segmented each learned object from a cluttered scene through back-analysis through the network. Max-pooling, now often adopted by deep neural networks (e.g., ImageNet tests), was first used in Cresceptron to reduce the position resolution by a factor of (2x2) to 1 through the cascade for better generalization. Because of a great lack of understanding how the brain autonomously wire its biological networks and the computational cost by ANNs then, simpler models that use task-specific handcrafted features such as Gabor filter and support vector machines (SVMs) were of popular choice of the field in the 1990s and 2000s.

In the long history of speech recognition, both shallow form and deep form (e.g., recurrent nets) of artificial neural networks had been explored for many years.*[18]*[19]*[20] But these methods never won over the non-uniform internal-handcrafting Gaussian mixture model/Hidden Markov model (GMM-HMM) technology based on generative models of speech trained discrim-

inatively.*[21] A number of key difficulties had been methodologically analyzed, including gradient diminishing and weak temporal correlation structure in the neural predictive models.*[22]*[23] All these difficulties were in addition to the lack of big training data and big computing power in these early days. Most speech recognition researchers who understood such barriers hence subsequently moved away from neural nets to pursue generative modeling approaches until the recent resurgence of deep learning that has overcome all these difficulties. Hinton et al. and Deng et al. reviewed part of this recent history about how their collaboration with each other and then with cross-group colleagues ignited the renaissance of neural networks and initiated deep learning research and applications in speech recognition.*[24]*[25]*[26]*[27]

The term "deep learning" gained traction in the mid-2000s after a publication by Geoffrey Hinton and Ruslan Salakhutdinov showed how a many-layered feedforward neural network could be effectively pre-trained one layer at a time, treating each layer in turn as an unsupervised restricted Boltzmann machine, then using supervised backpropagation for fine-tuning.*[28] In 1992, Schmidhuber had already implemented a very similar idea for the more general case of unsupervised deep hierarchies of recurrent neural networks, and also experimentally shown its benefits for speeding up supervised learning *[29]*[30]

Since the resurgence of deep learning, it has become part of many state-of-the-art systems in different disciplines, particularly that of computer vision and automatic speech recognition (ASR). Results on commonly used evaluation sets such as TIMIT (ASR) and MNIST (image classification) as well as a range of large vocabulary speech recognition tasks are constantly being improved with new applications of deep learning.*[24]*[31]*[32] Currently, it has been shown that deep learning architectures in the form of convolutional neural networks have been nearly best performing;*[33]*[34] however, these are more widely used in computer vision than in ASR.

The real impact of deep learning in industry started in large-scale speech recognition around 2010. In late 2009, Geoff Hinton was invited by Li Deng to work with him and colleagues at Microsoft Research in Redmond to apply deep learning to speech recognition. They co-organized the 2009 NIPS Workshop on Deep Learning for Speech Recognition. The workshop was motivated by the limitations of deep generative models of speech, and the possibility that the big-compute, big-data era warranted a serious try of the deep neural net (DNN) approach. It was then (incorrectly) believed that pre-training of DNNs using generative models of deep belief net (DBN) would be the cure for the main difficulties of neural nets encountered during 1990's.*[26] However, soon after the research along this direction started at Microsoft Research, it was discovered that when large amounts of training data are used and especially when DNNs are designed correspondingly with large, context-dependent output layers, dramatic error reduction occurred over the then-state-of-the-art GMM-HMM and more advanced generative model-based speech recognition systems without the need for generative DBN pre-training, the finding verified subsequently by several other major speech recognition research groups *[24]*[35] Further, the nature of recognition errors produced by the two types of systems was found to be characteristically different,*[25]*[36] offering technical insights into how to artfully integrate deep learning into the existing highly efficient, run-time speech decoding system deployed by all major players in speech recognition industry. The history of this significant development in deep learning has been described and analyzed in recent books.*[1]*[37]

Advances in hardware have also been an important enabling factor for the renewed interest of deep learning. In particular, powerful graphics processing units (GPUs) are highly suited for the kind of number crunching, matrix/vector math involved in machine learning. GPUs have been shown to speed up training algorithms by orders of magnitude, bringing running times of weeks back to days.*[38]*[39]

2.3 Deep learning in artificial neural networks

Some of the most successful deep learning methods involve artificial neural networks. Artificial neural networks are inspired by the 1959 biological model proposed by Nobel laureates David H. Hubel & Torsten Wiesel, who found two types of cells in the primary visual cortex: simple cells and complex cells. Many artificial neural networks can be viewed as cascading models *[15]*[16]*[17]*[40] of cell types inspired by these biological observations.

Fukushima's Neocognitron introduced convolutional neural networks partially trained by unsupervised learning while humans directed features in the neural plane. Yann LeCun et al. (1989) applied supervised backpropagation to such architectures.*[41] Weng et al. (1992) published convolutional neural networks Cresceptron*[15]*[16]*[17] for 3-D object recognition from images of cluttered scenes and segmentation of such objects from images.

An obvious need for recognizing general 3-D objects is least shift invariance and tolerance to deformation. Max-pooling appeared to be first proposed by Cresceptron*[15]*[16] to enable the network to tolerate small-to-large deformation in a hierarchical way while using convolution. Max-pooling helps, but still does not fully guarantee, shift-invariance at the pixel level.*[17]

With the advent of the back-propagation algorithm in the 1970s, many researchers tried to train supervised deep artificial neural networks from scratch, initially with little success. Sepp Hochreiter's diploma thesis of

1991[*][42][*][43] formally identified the reason for this failure in the "vanishing gradient problem," which not only affects many-layered feedforward networks, but also recurrent neural networks. The latter are trained by unfolding them into very deep feedforward networks, where a new layer is created for each time step of an input sequence processed by the network. As errors propagate from layer to layer, they shrink exponentially with the number of layers.

To overcome this problem, several methods were proposed. One is Jürgen Schmidhuber's multi-level hierarchy of networks (1992) pre-trained one level at a time through unsupervised learning, fine-tuned through backpropagation.[*][29] Here each level learns a compressed representation of the observations that is fed to the next level.

Another method is the long short term memory (LSTM) network of 1997 by Hochreiter & Schmidhuber.[*][44] In 2009, deep multidimensional LSTM networks won three ICDAR 2009 competitions in connected handwriting recognition, without any prior knowledge about the three different languages to be learned.[*][45][*][46]

Sven Behnke relied only on the sign of the gradient (Rprop) when training his Neural Abstraction Pyramid[*][47] to solve problems like image reconstruction and face localization.

Other methods also use unsupervised pre-training to structure a neural network, making it first learn generally useful feature detectors. Then the network is trained further by supervised back-propagation to classify labeled data. The deep model of Hinton et al. (2006) involves learning the distribution of a high level representation using successive layers of binary or real-valued latent variables. It uses a restricted Boltzmann machine (Smolensky, 1986[*][48]) to model each new layer of higher level features. Each new layer guarantees an increase on the lower-bound of the log likelihood of the data, thus improving the model, if trained properly. Once sufficiently many layers have been learned the deep architecture may be used as a generative model by reproducing the data when sampling down the model (an "ancestral pass") from the top level feature activations.[*][49] Hinton reports that his models are effective feature extractors over high-dimensional, structured data.[*][50]

The Google Brain team led by Andrew Ng and Jeff Dean created a neural network that learned to recognize higher-level concepts, such as cats, only from watching unlabeled images taken from YouTube videos.[*][51] [*][52]

Other methods rely on the sheer processing power of modern computers, in particular, GPUs. In 2010 it was shown by Dan Ciresan and colleagues[*][38] in Jürgen Schmidhuber's group at the Swiss AI Lab IDSIA that despite the above-mentioned "vanishing gradient problem," the superior processing power of GPUs makes plain back-propagation feasible for deep feedforward neural networks with many layers. The method outperformed all other machine learning techniques on the old, famous MNIST handwritten digits problem of Yann LeCun and colleagues at NYU.

As of 2011, the state of the art in deep learning feedforward networks alternates convolutional layers and max-pooling layers,[*][53][*][54] topped by several pure classification layers. Training is usually done without any unsupervised pre-training. Since 2011, GPU-based implementations[*][53] of this approach won many pattern recognition contests, including the IJCNN 2011 Traffic Sign Recognition Competition,[*][55] the ISBI 2012 Segmentation of neuronal structures in EM stacks challenge,[*][56] and others.

Such supervised deep learning methods also were the first artificial pattern recognizers to achieve human-competitive performance on certain tasks.[*][57]

To break the barriers of weak AI represented by deep learning, it is necessary to go beyond the deep learning architectures because biological brains use both shallow and deep circuits as reported by brain anatomy[*][58] in order to deal with the wide variety of invariance that the brain displays. Weng[*][59] argued that the brain self-wires largely according to signal statistics and, therefore, a serial cascade cannot catch all major statistical dependencies. Fully guaranteed shift invariance for ANNs to deal with small and large natural objects in large cluttered scenes became true when the invariance went beyond shift, to extend to all ANN-learned concepts, such as location, type (object class label), scale, lighting, in the Developmental Networks (DNs)[*][60] whose embodiments are Where-What Networks, WWN-1 (2008)[*][61] through WWN-7 (2013).[*][62]

2.4 Deep learning architectures

There are huge number of different variants of deep architectures; however, most of them are branched from some original parent architectures. It is not always possible to compare the performance of multiple architectures all together, since they are not all implemented on the same data set. Deep learning is a fast-growing field so new architectures, variants, or algorithms may appear every few weeks.

2.4.1 Deep neural networks

A deep neural network (DNN) is an artificial neural network with multiple hidden layers of units between the input and output layers.[*][2][*][4] Similar to shallow ANNs, DNNs can model complex non-linear relationships. DNN architectures, e.g., for object detection and parsing generate compositional models where the object is expressed as layered composition of image primitives.[*][63] The extra layers enable composition of features from lower layers, giving the potential of modeling

complex data with fewer units than a similarly performing shallow network.*[2]

DNNs are typically designed as feedforward networks, but recent research has successfully applied the deep learning architecture to recurrent neural networks for applications such as language modeling.*[64] Convolutional deep neural networks (CNNs) are used in computer vision where their success is well-documented.*[65] More recently, CNNs have been applied to acoustic modeling for automatic speech recognition (ASR), where they have shown success over previous models.*[34] For simplicity, a look at training DNNs is given here.

A DNN can be discriminatively trained with the standard backpropagation algorithm. The weight updates can be done via stochastic gradient descent using the following equation:

$$\Delta w_{ij}(t+1) = \Delta w_{ij}(t) + \eta \frac{\partial C}{\partial w_{ij}}$$

Here, η is the learning rate, and C is the cost function. The choice of the cost function depends on factors such as the learning type (supervised, unsupervised, reinforcement, etc.) and the activation function. For example, when performing supervised learning on a multiclass classification problem, common choices for the activation function and cost function are the softmax function and cross entropy function, respectively. The softmax function is defined as $p_j = \frac{\exp(x_j)}{\sum_k \exp(x_k)}$ where p_j represents the class probability and x_j and x_k represent the total input to units j and k respectively. Cross entropy is defined as $C = -\sum_j d_j \log(p_j)$ where d_j represents the target probability for output unit j and p_j is the probability output for j after applying the activation function.*[66]

2.4.2 Issues with deep neural networks

As with ANNs, many issues can arise with DNNs if they are naively trained. Two common issues are overfitting and computation time.

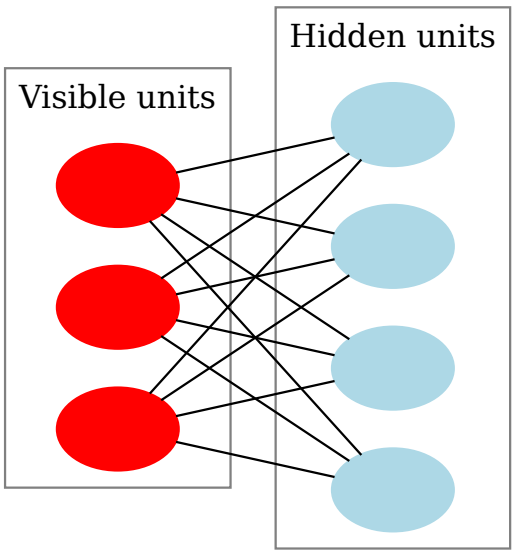
DNNs are prone to overfitting because of the added layers of abstraction, which allow them to model rare dependencies in the training data. Regularization methods such as weight decay (ℓ_2 -regularization) or sparsity (ℓ_1 -regularization) can be applied during training to help combat overfitting.*[67] A more recent regularization method applied to DNNs is *dropout* regularization. In dropout, some number of units are randomly omitted from the hidden layers during training. This helps to break the rare dependencies that can occur in the training data *[68]

Backpropagation and gradient descent have been the preferred method for training these structures due to the ease of implementation and their tendency to converge to better local optima in comparison with other training methods. However, these methods can be computationally expensive, especially when being used to train DNNs. There are many training parameters to be considered with a DNN, such as the size (number of layers and number of units per layer), the learning rate and initial weights. Sweeping through the parameter space for optimal parameters may not be feasible due to the cost in time and computational resources. Various 'tricks' such as using mini-batching (computing the gradient on several training examples at once rather than individual examples) *[69] have been shown to speed up computation. The large processing throughput of GPUs has produced significant speedups in training, due to the matrix and vector computations required being well suited for GPUs.*[4]

2.4.3 Deep belief networks

Main article: Deep belief network

A deep belief network (DBN) is a probabilistic,



A restricted Boltzmann machine (RBM) with fully connected visible and hidden units. Note there are no hidden-hidden or visible-visible connections.

generative model made up of multiple layers of hidden units. It can be looked at as a composition of simple learning modules that make up each layer.*[70]

A DBN can be used for generatively pre-training a DNN by using the learned weights as the initial weights. Backpropagation or other discriminative algorithms can then be applied for fine-tuning of these weights. This is particularly helpful in situations where limited training data is available, as poorly initialized weights can have significant impact on the performance of the final model. These pre-trained weights are in a region of the weight space that is closer to the optimal weights (as compared to just random initialization). This allows for both improved mod-

eling capability and faster convergence of the fine-tuning phase.*[71]

A DBN can be efficiently trained in an unsupervised, layer-by-layer manner where the layers are typically made of restricted Boltzmann machines (RBM). A description of training a DBN via RBMs is provided below. An RBM is an undirected, generative energy-based model with an input layer and single hidden layer. Connections only exist between the visible units of the input layer and the hidden units of the hidden layer; there are no visible-visible or hidden-hidden connections.

The training method for RBMs was initially proposed by Geoffrey Hinton for use with training "Product of Expert" models and is known as contrastive divergence (CD).*[72] CD provides an approximation to the maximum likelihood method that would ideally be applied for learning the weights of the RBM.*[69]*[73]

In training a single RBM, weight updates are performed with gradient ascent via the following equation: $\Delta w_{ij}(t+1) = w_{ij}(t) + \eta \frac{\partial \log(p(v))}{\partial w_{ij}}$. Here, $p(v)$ is the probability of a visible vector, which is given by $p(v) = \frac{1}{Z}\sum_h e^{-E(v,h)}$. Z is the partition function (used for normalizing) and $E(v,h)$ is the energy function assigned to the state of the network. A lower energy indicates the network is in a more "desirable" configuration. The gradient $\frac{\partial \log(p(v))}{\partial w_{ij}}$ has the simple form $\langle v_i h_j \rangle_{\text{data}} - \langle v_i h_j \rangle_{\text{model}}$ where $\langle \cdots \rangle_p$ represent averages with respect to distribution p . The issue arises in sampling $\langle v_i h_j \rangle_{\text{model}}$ as this requires running alternating Gibbs sampling for a long time. CD replaces this step by running alternating Gibbs sampling for n steps (values of $n = 1$ have empirically been shown to perform well). After n steps, the data is sampled and that sample is used in place of $\langle v_i h_j \rangle_{\text{model}}$. The CD procedure works as follows:*[69]

1. Initialize the visible units to a training vector.
2. Update the hidden units in parallel given the visible units: $p(h_j = 1 \mid \mathbf{V}) = \sigma(b_j + \sum_i v_i w_{ij})$. σ represents the sigmoid function and b_j is the bias of h_j .
3. Update the visible units in parallel given the hidden units: $p(v_i = 1 \mid \mathbf{H}) = \sigma(a_i + \sum_j h_j w_{ij})$. a_i is the bias of v_i . This is called the "reconstruction" step.
4. Reupdate the hidden units in parallel given the reconstructed visible units using the same equation as in step 2.
5. Perform the weight update: $\Delta w_{ij} \propto \langle v_i h_j \rangle_{\text{data}} - \langle v_i h_j \rangle_{\text{reconstruction}}$.

Once an RBM is trained, another RBM can be "stacked" atop of it to create a multilayer model. Each time another RBM is stacked, the input visible layer is initialized to a training vector and values for the units in the already-trained RBM layers are assigned using the current weights and biases. The final layer of the already-trained layers is used as input to the new RBM. The new RBM is then trained with the procedure above, and then this whole process can be repeated until some desired stopping criterion is met.*[2]

Despite the approximation of CD to maximum likelihood being very crude (CD has been shown to not follow the gradient of any function), empirical results have shown it to be an effective method for use with training deep architectures.*[69]

2.4.4 Convolutional neural networks

Main article: Convolutional neural network

A CNN is composed of one or more convolutional layers with fully connected layers (matching those in typical artificial neural networks) on top. It also uses tied weights and pooling layers. This architecture allows CNNs to take advantage of the 2D structure of input data. In comparison with other deep architectures, convolutional neural networks are starting to show superior results in both image and speech applications. They can also be trained with standard backpropagation. CNNs are easier to train than other regular, deep, feed-forward neural networks and have many fewer parameters to estimate, making them a highly attractive architecture to use.*[74]

2.4.5 Convolutional Deep Belief Networks

A recent achievement in deep learning is from the use of convolutional deep belief networks (CDBN). A CDBN is very similar to normal Convolutional neural network in terms of its structure. Therefore, like CNNs they are also able to exploit the 2D structure of images combined with the advantage gained by pre-training in Deep belief network. They provide a generic structure which can be used in many image and signal processing tasks and can be trained in a way similar to that for Deep Belief Networks. Recently, many benchmark results on standard image datasets like CIFAR *[75] have been obtained using CDBNs.*[76]

2.4.6 Deep Boltzmann Machines

A *Deep Boltzmann Machine* (DBM) is a type of binary pairwise Markov random field (undirected probabilistic graphical models) with multiple layers of hidden random variables. It is a network of symmetrically coupled stochastic binary units. It comprises a set of visible units $\boldsymbol{\nu} \in \{0,1\}^D$, and a series of layers of hidden units $\boldsymbol{h}^{(1)} \in \{0,1\}^{F_1}, \boldsymbol{h}^{(2)} \in \{0,1\}^{F_2}, \ldots, \boldsymbol{h}^{(L)} \in \{0,1\}^{F_L}$. There is no connection between the units of the same

layer (like RBM). For the DBM, we can write the probability which is assigned to vector ν as:

$$p(\boldsymbol{\nu}) = \frac{1}{Z} \sum_h e^{\sum_{ij} W_{ij}^{(1)} \nu_i h_j^{(1)} + \sum_{jl} W_{jl}^{(2)} h_j^{(1)} h_l^{(2)} + \sum_{lm} W_{lm}^{(3)} h_l^{(2)} h_m^{(3)}}$$

where $\boldsymbol{h} = \{\boldsymbol{h}^{(1)}, \boldsymbol{h}^{(2)}, \boldsymbol{h}^{(3)}\}$ are the set of hidden units, and $\theta = \{\boldsymbol{W}^{(1)}, \boldsymbol{W}^{(2)}, \boldsymbol{W}^{(3)}\}$ are the model parameters, representing visible-hidden and hidden-hidden *symmetric interaction*, since they are undirected links. As it is clear by setting $\boldsymbol{W}^{(2)} = 0$ and $\boldsymbol{W}^{(3)} = 0$ the network becomes the well-known Restricted Boltzmann machine.*[77]

There are several reasons which motivate us to take advantage of deep Boltzmann machine architectures. Like DBNs, they benefit from the ability of learning complex and abstract internal representations of the input in tasks such as object or speech recognition, with the use of *limited number* of *labeled* data to fine-tune the representations built based on a large *supply* of *unlabeled* sensory input data. However, unlike DBNs and *deep convolutional* neural networks, they adopt the inference and training procedure in both directions, bottom-up and top-down pass, which enable the DBMs to better unveil the representations of the ambiguous and complex input structures,*[78] .*[79]

Since the *exact maximum likelihood* learning is intractable for the DBMs, we may perform the *approximate maximum likelihood* learning. There is another possibility, to use *mean-field* inference to estimate data-dependent expectations, incorporation with a *Markov chain Monte Carlo (MCMC)* based stochastic approximation technique to approximate the expected *sufficient statistics* of the model.*[77]

We can see the difference between DBNs and DBM. In DBNs, the top two layers form a restricted Boltzmann machine which is an undirected graphical model, but the lower layers form a directed generative model.

Apart from all the advantages of DBMs discussed so far, they have a crucial disadvantage which limits the performance and functionality of this kind of architecture. The approximate inference, which is based on mean-field method, is about 25 to 50 times slower than a single bottom-up pass in DBNs. This time consuming task make the joint optimization, quite impractical for large data sets, and seriously restricts the use of DBMs in tasks such as feature representations (the mean-field inference have to be performed for each new test input).*[80]

2.4.7 Stacked (Denoising) Auto-Encoders

The auto encoder idea is motivated by the concept of *good* representation. For instance for the case of classifier it is possible to define that a *good representation is one that will yield a better performing classifier*.

An *encoder* is referred to a deterministic mapping f_θ that transforms an input vector $\boldsymbol{x}$ into hidden representation $\boldsymbol{y}$, where $\theta = \{\boldsymbol{W}, b\}$, $\boldsymbol{W}$ is the weight matrix and $\mathbf{b}$ is an offset vector (bias). On the contrary a *decoder* maps back the hidden representation $\mathbf{y}$ to the reconstructed input $\boldsymbol{z}$ via g_θ . The whole process of auto encoding is to compare this reconstructed input to the original and try to minimize this error to make the reconstructed value as close as possible to the original.

In *stacked denoising auto encoders*, the partially corrupted output is cleaned (*denoised*). This fact has been introduced in *[81] with a specific approach to *good* representation, a *good representation is one that can be obtained robustly from a corrupted input and that will be useful for recovering the corresponding clean input.* Implicit in this definition are the ideas of

- The higher level representations are relatively stable and robust to the corruption of the input;
- It is required to extract features that are useful for representation of the input distribution.

The algorithm consists of multiple steps; starts by a stochastic mapping of $\boldsymbol{x}$ to $\tilde{\boldsymbol{x}}$ through $q_D(\tilde{\boldsymbol{x}}|\boldsymbol{x})$, this is the corrupting step. Then the corrupted input $\tilde{\boldsymbol{x}}$ passes through a basic auto encoder process and is mapped to a hidden representation $\boldsymbol{y} = f_\theta(\tilde{\boldsymbol{x}}) = s(\boldsymbol{W}\tilde{\boldsymbol{x}} + b)$. From this hidden representation we can reconstruct $\boldsymbol{z} = g_\theta(\boldsymbol{y})$. In the last stage a minimization algorithm is done in order to have a $\boldsymbol{z}$ as close as possible to uncorrupted input $\boldsymbol{x}$. The reconstruction error $L_H(\boldsymbol{x}, \boldsymbol{z})$ might be either the cross-entropy loss with an affine-sigmoid decoder, or the squared error loss with an affine decoder.*[81]

In order to make a deep architecture, auto encoders stack one on top of another. Once the encoding function f_θ of the first denoising auto encoder is learned and used to uncorrupt the input (corrupted input), we can train the second level.*[81]

Once the stacked auto encoder is trained, its output might be used as the input to a supervised learning algorithm such as support vector machine classifier or a multiclass logistic regression.*[81]

2.4.8 Deep Stacking Networks

One of the deep architectures recently introduced in*[82] which is based on building hierarchies with blocks of simplified neural network modules, is called *deep convex network*. They are called "convex" because of the formulation of the weights learning problem, which is a convex optimization problem with a closed-form solution. The network is also called the *deep stacking network (DSN)*,*[83] emphasizing on this fact that a similar mechanism as the *stacked generalization* is used.*[84]

The DSN blocks, each consisting of a simple, easy-to-learn module, are stacked to form the overall deep network. It can be trained block-wise in a supervised fash-

ion without the need for back-propagation for the entire blocks.*[85]

As designed in *[82] each block consists of a simplified MLP with a single hidden layer. It comprises a weight matrix $\boldsymbol{U}$ as the connection between the logistic sigmoidal units of the hidden layer $\boldsymbol{h}$ to the linear output layer $\boldsymbol{y}$, and a weight matrix $\boldsymbol{W}$ which connects each input of the blocks to their respective hidden layers. If we assume that the target vectors $\boldsymbol{t}$ be arranged to form the columns of $\boldsymbol{T}$ (the target matrix), let the input data vectors $\boldsymbol{x}$ be arranged to form the columns of $\boldsymbol{X}$, let $\boldsymbol{H} = \sigma(\boldsymbol{W}^T \boldsymbol{X})$ denote the matrix of hidden units, and assume the lower-layer weights $\boldsymbol{W}$ are known (training layer-by-layer). The function performs the element-wise logistic sigmoid operation. Then learning the upper-layer weight matrix $\boldsymbol{U}$ given other weights in the network can be formulated as a convex optimization problem:

$$\min_{U^T} f = ||\boldsymbol{U}^T \boldsymbol{H} - \boldsymbol{T}||_F^2,$$

which has a closed-form solution. The input to the first block $\boldsymbol{X}$ only contains the original data, however in the upper blocks in addition to this original (raw) data there is a copy of the lower-block(s) output y.

In each block an estimate of the same final label class y is produced, then this estimated label concatenated with original input to form the *expanded input* for the upper block. In contrast with other deep architectures, such as DBNs, the goal is not to discover the transformed feature representation. Regarding the structure of the hierarchy of this kind of architecture, it makes the parallel training straightforward as the problem is naturally a batch-mode optimization one. In purely discriminative tasks DSN performance is better than the conventional DBN.*[83]

2.4.9 Tensor Deep Stacking Networks (T-DSN)

This architecture is an extension of the DSN. It improves the DSN in two important ways, using the higher order information by means of covariance statistics and transforming the non-convex problem of the lower-layer to a convex sub-problem of the upper-layer.*[86]

Unlike the DSN, the covariance statistics of the data is employed using a bilinear mapping from two distinct sets of hidden units in the same layer to predictions via a third-order tensor.

The scalability and parallelization are the two important factors in the learning algorithms which are not considered seriously in the conventional DNNs.*[87]*[88]*[89] All the learning process for the DSN (and TDSN as well) is done on a batch-mode basis so as to make the parallelization possible on a cluster of CPU or GPU nodes.*[82]*[83] Parallelization gives the opportunity to scale up the design to larger (deeper) architectures and data sets.

The basic architecture is suitable for diverse tasks such as classification and regression.

2.4.10 Spike-and-Slab RBMs (ssRBMs)

The need for real-valued inputs which are employed in Gaussian RBMs (GRBMs), motivates scientists seeking new methods. One of these methods is the *spike and slab* RBM (*ss*RBMs), which models continuous-valued inputs with strictly binary latent variables.*[90]

Similar to basic RBMs and its variants, the spike and slab RBM is a bipartite graph. Like GRBM, the visible units (input) are real-valued. The difference arises in the hidden layer, where each hidden unit come along with a binary spike variable and real-valued slab variable. These terms (spike and slab) come from the statistics literature,*[91] and refer to a prior including a mixture of two components. One is a discrete probability mass at zero called spike, and the other is a density over continuous domain.*[92]*[92]

There is also an extension of the ssRBM model, which is called μ-ssRBM. This variant provides extra modeling capacity to the architecture using additional terms in the energy function. One of these terms enable model to form a conditional distribution of the spike variables by means of marginalizing out the slab variables given an observation.

2.4.11 Compound Hierarchical-Deep Models

The class architectures called *compound HD models*, where HD stands for *Hierarchical-Deep* are structured as a composition of non-parametric Bayesian models with deep networks. The features, learned by deep architectures such as DBNs,*[93] DBMs,*[78] deep auto encoders,*[94] convolutional variants,*[95]*[96] ssRBMs,*[92] deep coding network,*[97] DBNs with sparse feature learning,*[98] recursive neural networks,*[99] conditional DBNs,*[100] denoising auto encoders,*[101] are able to provide better representation for more rapid and accurate classification tasks with high-dimensional training data sets. However, they are not quite powerful in learning novel classes with few examples, themselves. In these architectures, all units through the network are involved in the representation of the input (*distributed representations*), and they have to be adjusted together (high degree of freedom). However, if we limit the degree of freedom, we make it easier for the model to learn new classes out of few training samples (less parameters to learn). *Hierarchical Bayesian (HB)* models, provide learning from few examples, for example *[102]*[103]*[104]*[105]*[106] for computer

vision, statistics, and cognitive science.

Compound HD architectures try to integrate both characteristics of HB and deep networks. The compound HDP-DBM architecture, a *hierarchical Dirichlet process (HDP)* as a hierarchical model, incorporated with DBM architecture. It is a full generative model, generalized from abstract concepts flowing through the layers of the model, which is able to synthesize new examples in novel classes that look *reasonably natural*. Note that all the levels are learned jointly by maximizing a joint log-probability score.*[107]

Consider a DBM with three hidden layers, the probability of a visible input $\boldsymbol{\nu}$ is:

$$p(\boldsymbol{\nu}, \psi) = \frac{1}{Z} \sum_h e^{\sum_{ij} W_{ij}^{(1)} \nu_i h_j^1 + \sum_{jl} W_{jl}^{(2)} h_j^1 h_l^2 + \sum_{lm} W_{lm}^{(3)} h_l^2 h_m^3}$$

where $\boldsymbol{h} = \{\boldsymbol{h}^{(1)}, \boldsymbol{h}^{(2)}, \boldsymbol{h}^{(3)}\}$ are the set of hidden units, and $\psi = \{\boldsymbol{W}^{(1)}, \boldsymbol{W}^{(2)}, \boldsymbol{W}^{(3)}\}$ are the model parameters, representing visible-hidden and hidden-hidden symmetric interaction terms.

After a DBM model has been learned, we have an undirected model that defines the joint distribution $P(\nu, h^1, h^2, h^3)$. One way to express what has been learned is the conditional model $P(\nu, h^1, h^2|h^3)$ and a prior term $P(h^3)$.

The part $P(\nu, h^1, h^2|h^3)$, represents a *conditional* DBM model, which can be viewed as a two-layer DBM but with bias terms given by the states of h^3 :

$$P(\nu, h^1, h^2|h^3) = \frac{1}{Z(\psi, h^3)} e^{\sum_{ij} W_{ij}^{(1)} \nu_i h_j^1 + \sum_{jl} W_{jl}^{(2)} h_j^1 h_l^2 + \sum_{lm} W_{lm}^{(3)} h_l^2 h_m^3}$$

2.4.12 Deep Coding Networks

There are several advantages to having a model which can *actively* update itself to the context in data. One of these methods arises from the idea to have a model which is able to adjust its prior knowledge dynamically according to the context of the data. Deep coding network (DPCN) is a predictive coding scheme where top-down information is used to empirically adjust the priors needed for the bottom-up inference procedure by means of a deep locally-connected generative model. This is based on extracting sparse features out of time-varying observations using a linear dynamical model. Then, a pooling strategy is employed in order to learn invariant feature representations. Similar to other deep architectures, these blocks are the building elements of a deeper architecture where greedy layer-wise unsupervised learning are used. Note that the layers constitute a kind of Markov chain such that the states at any layer are only dependent on the succeeding and preceding layers.

Deep predictive coding network (DPCN)*[108] predicts the representation of the layer, by means of a top-down approach using the information in upper layer and also temporal dependencies from the previous states, it is called

It is also possible to extend the DPCN to form a convolutional network.*[108]

2.4.13 Deep Kernel Machines

The *Multilayer Kernel Machine (MKM)* as introduced in *[109] is a way of learning highly nonlinear functions with the iterative applications of weakly nonlinear kernels. They use the *kernel principal component analysis (KPCA)*, in,*[110] as method for unsupervised greedy layer-wise pre-training step of the deep learning architecture.

Layer $l + 1$ -th learns the representation of the previous layer l , extracting the n_l principal component (PC) of the projection layer l output in the feature domain induced by the kernel. For the sake of dimensionality reduction of the updated representation in each layer, a supervised strategy is proposed to select the best informative features among the ones extracted by KPCA. The process is:

- ranking the n_l features according to their mutual information with the class labels;
- for different values of K and $m_l \in \{1, \ldots, n_l\}$, compute the classification error rate of a *K-nearest neighbor (K-NN)* classifier using only the m_l most informative features on a validation set;
- the value of m_l with which the classifier has reached the lowest error rate determines the number of features to retain.

There are some drawbacks in using the KPCA method as the building cells of an MKM.

Another, more straightforward method of integrating kernel machine into the deep learning architecture was developed by Microsoft researchers for spoken language understanding applications.*[111] The main idea is to use a kernel machine to approximate a shallow neural net with an infinite number of hidden units, and then to use the stacking technique to splice the output of the kernel machine and the raw input in building the next, higher level of the kernel machine. The number of the levels in this kernel version of the deep convex network is a hyper-parameter of the overall system determined by cross validation.

2.4.14 Deep Q-Networks

This is the latest class of deep learning models targeted for reinforcement learning, published in February 2015 in Nature*[112]

2.5 Applications

2.5.1 Automatic speech recognition

The results shown in the table below are for automatic speech recognition on the popular TIMIT data set. This is a common data set used for initial evaluations of deep learning architectures. The entire set contains 630 speakers from eight major dialects of American English, with each speaker reading 10 different sentences.[*][113] Its small size allows many different configurations to be tried effectively with it. More importantly, the TIMIT task concerns phone-sequence recognition, which, unlike word-sequence recognition, permits very weak "language models" and thus the weaknesses in acoustic modeling aspects of speech recognition can be more easily analyzed. It was such analysis on TIMIT contrasting the GMM (and other generative models of speech) vs. DNN models carried out by Li Deng and collaborators around 2009-2010 that stimulated early industrial investment on deep learning technology for speech recognition from small to large scales,[*][25][*][36] eventually leading to pervasive and dominant uses of deep learning in speech recognition industry. That analysis was carried out with comparable performance (less than 1.5% in error rate) between discriminative DNNs and generative models. The error rates presented below, including these early results and measured as percent phone error rates (PER), have been summarized over a time span of the past 20 years:

Extension of the success of deep learning from TIMIT to large vocabulary speech recognition occurred in 2010 by industrial researchers, where large output layers of the DNN based on context dependent HMM states constructed by decision trees were adopted.[*][116][*][117] See comprehensive reviews of this development and of the state of the art as of October 2014 in the recent Springer book from Microsoft Research.[*][37] See also the related background of automatic speech recognition and the impact of various machine learning paradigms including notably deep learning in a recent overview article.[*][118]

One fundamental principle of deep learning is to do away with hand-crafted feature engineering and to use raw features. This principle was first explored successfully in the architecture of deep autoencoder on the "raw" spectrogram or linear filter-bank features,[*][119] showing its superiority over the Mel-Cepstral features which contain a few stages of fixed transformation from spectrograms. The true "raw" features of speech, waveforms, have more recently been shown to produce excellent larger-scale speech recognition results.[*][120]

Since the initial successful debut of DNNs for speech recognition around 2009-2011, there has been huge progress made. This progress (as well as future directions) has been summarized into the following eight major areas:[*][1][*][27][*][37] 1) Scaling up/out and speedup DNN training and decoding; 2) Sequence discriminative training of DNNs; 3) Feature processing by deep models with solid understanding of the underlying mechanisms; 4) Adaptation of DNNs and of related deep models; 5) Multi-task and transfer learning by DNNs and related deep models; 6) Convolution neural networks and how to design them to best exploit domain knowledge of speech; 7) Recurrent neural network and its rich LSTM variants; 8) Other types of deep models including tensor-based models and integrated deep generative/discriminative models.

Large-scale automatic speech recognition is the first and the most convincing successful case of deep learning in the recent history, embraced by both industry and academic across the board. Between 2010 and 2014, the two major conferences on signal processing and speech recognition, IEEE-ICASSP and Interspeech, have seen near exponential growth in the numbers of accepted papers in their respective annual conference papers on the topic of deep learning for speech recognition. More importantly, all major commercial speech recognition systems (e.g., Microsoft Cortana, Xbox, Skype Translator, Google Now, Apple Siri, Baidu and iFlyTek voice search, and a range of Nuance speech products, etc.) nowadays are based on deep learning methods.[*][1][*][121][*][122] See also the recent media interview with the CTO of Nuance Communications.[*][123]

The wide-spreading success in speech recognition achieved by 2011 was followed shortly by large-scale image recognition described next.

2.5.2 Image recognition

A common evaluation set for image classification is the MNIST database data set. MNIST is composed of handwritten digits and includes 60000 training examples and 10000 test examples. Similar to TIMIT, its small size allows multiple configurations to be tested. A comprehensive list of results on this set can be found in.[*][124] The current best result on MNIST is an error rate of 0.23%, achieved by Ciresan et al. in 2012.[*][125]

The real impact of deep learning in image or object recognition, one major branch of computer vision, was felt in the fall of 2012 after the team of Geoff Hinton and his students won the large-scale ImageNet competition by a significant margin over the then-state-of-the-art shallow machine learning methods. The technology is based on 20-year-old deep convolutional nets, but with much larger scale on a much larger task, since it had been learned that deep learning works quite well on large-scale speech recognition. In 2013 and 2014, the error rate on the ImageNet task using deep learning was further reduced at a rapid pace, following a similar trend in large-scale speech recognition.

As in the ambitious moves from automatic speech recognition toward automatic speech translation and understanding, image classification has recently been extended to the more ambitious and challenging task of automatic image captioning, in which deep learning is the essential underlying technology. [*][126] [*][127] [*][128] [*][129]

One example application is a car computer said to be trained with deep learning, which may be able to let cars interpret 360° camera views.*[130]

2.5.3 Natural language processing

Neural networks have been used for implementing language models since the early 2000s.*[131] Key techniques in this field are negative sampling*[132] and word embedding. A word embedding, such as *word2vec*, can be thought of as a representational layer in a deep learning architecture transforming an atomic word into a positional representation of the word relative to other words in the dataset; the position is represented as a point in a vector space. Using a word embedding as an input layer to a recursive neural network (RNN) allows for the training of the network to parse sentences and phrases using an effective *compositional vector grammar*. A compositional vector grammar can be thought of as probabilistic context free grammar (PCFG) implemented by a recursive neural network.*[133] Recursive autoencoders built atop word embeddings have been trained to assess sentence similarity and detect paraphrasing.*[133] Deep neural architectures have achieved state-of-the-art results in many tasks in natural language processing, such as constituency parsing,*[134] sentiment analysis,*[135] information retrieval,*[136] *[137] machine translation, *[138] *[139] contextual entity linking, *[140] and other areas of NLP. *[141]

2.5.4 Drug discovery and toxicology

The pharmaceutical industry faces the problem that a large percentage of candidate drugs fail to reach the market. These failures of chemical compounds are caused by insufficient efficacy on the biomolecular target (on-target effect), undetected and undesired interactions with other biomolecules (off-target effects), or unanticipated toxic effects.*[142]*[143] In 2012 a team led by George Dahl won the "Merck Molecular Activity Challenge" using multi-task deep neural networks to predict the biomolecular target of a compound.*[144]*[145] In 2014 Sepp Hochreiter's group used Deep Learning to detect off-target and toxic effects of environmental chemicals in nutrients, household products and drugs and won the "Tox21 Data Challenge" of NIH, FDA and NCATS.*[146]*[147] These impressive successes show Deep Learning may be superior to other virtual screening methods.*[148]*[149] Researchers from Google and Stanford enhanced Deep Learning for drug discovery by combining data from a variety of sources.*[150]

2.5.5 Customer relationship management

Recently success has been reported with application of deep reinforcement learning in direct marketing settings, illustrating suitability of the method for CRM automation. A neural network was used to approximate the value of possible direct marketing actions over the customer state space, defined in terms of RFM variables. The estimated value function was shown to have a natural interpretation as CLV (customer lifetime value).*[151]

2.6 Deep learning in the human brain

Computational deep learning is closely related to a class of theories of brain development (specifically, neocortical development) proposed by cognitive neuroscientists in the early 1990s.*[152] An approachable summary of this work is Elman, et al.'s 1996 book "Rethinking Innateness" *[153] (see also: Shrager and Johnson;*[154] Quartz and Sejnowski *[155]). As these developmental theories were also instantiated in computational models, they are technical predecessors of purely computationally-motivated deep learning models. These developmental models share the interesting property that various proposed learning dynamics in the brain (e.g., a wave of nerve growth factor) conspire to support the self-organization of just the sort of inter-related neural networks utilized in the later, purely computational deep learning models; and such computational neural networks seem analogous to a view of the brain's neocortex as a hierarchy of filters in which each layer captures some of the information in the operating environment, and then passes the remainder, as well as modified base signal, to other layers further up the hierarchy. This process yields a self-organizing stack of transducers, well-tuned to their operating environment. As described in The New York Times in 1995: "...the infant's brain seems to organize itself under the influence of waves of so-called trophic-factors ... different regions of the brain become connected sequentially, with one layer of tissue maturing before another and so on until the whole brain is mature." *[156]

The importance of deep learning with respect to the evolution and development of human cognition did not escape the attention of these researchers. One aspect of human development that distinguishes us from our nearest primate neighbors may be changes in the timing of development.*[157] Among primates, the human brain remains relatively plastic until late in the post-natal period, whereas the brains of our closest relatives are more completely formed by birth. Thus, humans have greater access to the complex experiences afforded by being out in the world during the most formative period of brain development. This may enable us to "tune in" to rapidly changing features of the environment that other animals, more constrained by evolutionary structuring of their brains, are unable to take account of. To the extent that these changes are reflected in similar timing changes in hypothesized wave of cortical development,

they may also lead to changes in the extraction of information from the stimulus environment during the early self-organization of the brain. Of course, along with this flexibility comes an extended period of immaturity, during which we are dependent upon our caretakers and our community for both support and training. The theory of deep learning therefore sees the coevolution of culture and cognition as a fundamental condition of human evolution.*[158]

2.7 Commercial activity

Deep learning is often presented as a step towards realising strong AI*[159] and thus many organizations have become interested in its use for particular applications. Most recently, in December 2013, Facebook announced that it hired Yann LeCun to head its new artificial intelligence (AI) lab that will have operations in California, London, and New York. The AI lab will be used for developing deep learning techniques that will help Facebook do tasks such as automatically tagging uploaded pictures with the names of the people in them.*[160]

In March 2013, Geoffrey Hinton and two of his graduate students, Alex Krizhevsky and Ilya Sutskever, were hired by Google. Their work will be focused on both improving existing machine learning products at Google and also help deal with the growing amount of data Google has. Google also purchased Hinton's company, DNNresearch.

In 2014 Google also acquired DeepMind Technologies, a British start-up that developed a system capable of learning how to play Atari video games using only raw pixels as data input.

Baidu hired Andrew Ng to head their new Silicon Valley based research lab focusing on deep learning.

2.8 Criticism and comment

Given the far-reaching implications of artificial intelligence coupled with the realization that deep learning is emerging as one of its most powerful techniques, the subject is understandably attracting both criticism and comment, and in some cases from outside the field of computer science itself.

A main criticism of deep learning concerns the lack of theory surrounding many of the methods. Most of the learning in deep architectures is just some form of gradient descent. While gradient descent has been understood for a while now, the theory surrounding other algorithms, such as contrastive divergence is less clear (i.e., Does it converge? If so, how fast? What is it approximating?). Deep learning methods are often looked at as a black box, with most confirmations done empirically, rather than theoretically.

Others point out that deep learning should be looked at as a step towards realizing strong AI, not as an all-encompassing solution. Despite the power of deep learning methods, they still lack much of the functionality needed for realizing this goal entirely. Research psychologist Gary Marcus has noted that:

"Realistically, deep learning is only part of the larger challenge of building intelligent machines. Such techniques lack ways of representing causal relationships (...) have no obvious ways of performing logical inferences, and they are also still a long way from integrating abstract knowledge, such as information about what objects are, what they are for, and how they are typically used. The most powerful A.I. systems, like Watson (...) use techniques like deep learning as just one element in a very complicated ensemble of techniques, ranging from the statistical technique of Bayesian inference to deductive reasoning." *[161]

To the extent that such a viewpoint implies, without intending to, that deep learning will ultimately constitute nothing more than the primitive discriminatory levels of a comprehensive future machine intelligence, a recent pair of speculations regarding art and artificial intelligence*[162] offers an alternative and more expansive outlook. The first such speculation is that it might be possible to train a machine vision stack to perform the sophisticated task of discriminating between "old master" and amateur figure drawings; and the second is that such a sensitivity might in fact represent the rudiments of a non-trivial machine empathy. It is suggested, moreover, that such an eventuality would be in line with both anthropology, which identifies a concern with aesthetics as a key element of behavioral modernity, and also with a current school of thought which suspects that the allied phenomenon of consciousness, formerly thought of as a purely high-order phenomenon, may in fact have roots deep within the structure of the universe itself.

Some currently popular and successful deep learning architectures display certain problematical behaviors*[163] (e.g. confidently classifying random data as belonging to a familiar category of nonrandom images;*[164] and misclassifying miniscule perturbations of correctly classified images *[165]). The creator of OpenCog, Ben Goertzel hypothesized *[163] that these behaviors are tied with limitations in the internal representations learned by these architectures, and that these same limitations would inhibit integration of these architectures into heterogeneous multi-component AGI architectures. It is suggested that these issues can be worked around by developing deep learning architectures that internally form states homologous to image-grammar *[166] decompositions of observed entities and events.*[163] Learning a grammar (visual or linguistic) from training data would be equivalent to restricting the system to commonsense reasoning which operates on concepts in terms of production rules of the grammar, and is a basic goal of both human language acquisition *[167] and A.I. (Also see Grammar in-

duction *[168])

2.9 Deep learning software libraries

- Torch
- Theano
- Deeplearning4j, distributed deep learning for the JVM. Parallel GPUs.
- ND4J
- NVIDIA cuDNN library of accelerated primitives for deep neural networks.
- DeepLearnToolbox, Matlab/Octave toolbox for deep learning
- convnetjs, deep learning library in Javascript. Contains online demos.
- Gensim a toolkit for natural language processing; includes word2vec
- Caffe

2.10 See also

- Unsupervised learning
- Graphical model
- Feature learning
- Sparse coding
- Compressed Sensing
- Connectionism
- Self-organizing map
- Principal component analysis
- Applications of artificial intelligence
- List of artificial intelligence projects
- Extreme Learning Machines
- Reservoir computing
- Liquid state machine
- Echo state network

2.11 References

[1] L. Deng and D. Yu (2014) "Deep Learning: Methods and Applications" http://research.microsoft.com/pubs/209355/DeepLearning-NowPublishing-Vol7-SIG-039.pdf

[2] Bengio, Yoshua (2009). "Learning Deep Architectures for AI" (PDF). *Foundations and Trends in Machine Learning* **2** (1).

[3] Y. Bengio, A. Courville, and P. Vincent., "Representation Learning: A Review and New Perspectives," *IEEE Trans. PAMI, special issue Learning Deep Architectures*, 2013

[4] J. Schmidhuber, "Deep Learning in Neural Networks: An Overview" http://arxiv.org/abs/1404.7828, 2014

[5] Patrick Glauner (2015), *Comparison of Training Methods for Deep Neural Networks*, arXiv:1504.06825

[6] Song, Hyun Ah, and Soo-Young Lee. "Hierarchical Representation Using NMF." Neural Information Processing. Springer Berlin Heidelberg, 2013.

[7] Olshausen, Bruno A. "Emergence of simple-cell receptive field properties by learning a sparse code for natural images." Nature 381.6583 (1996): 607-609.

[8] Ronan Collobert (May 6, 2011). "Deep Learning for Efficient Discriminative Parsing" . *videolectures.net*. Ca. 7:45.

[9] Gomes, Lee (20 October 2014). "Machine-Learning Maestro Michael Jordan on the Delusions of Big Data and Other Huge Engineering Efforts" . *IEEE Spectrum*.

[10] Fukushima, K. (1980). "Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position" . *Biol. Cybern* **36**: 193–202. doi:10.1007/bf00344251.

[11] P. Werbos., "Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences," *PhD thesis, Harvard University*, 1974.

[12] LeCun *et al.*, "Backpropagation Applied to Handwritten Zip Code Recognition," *Neural Computation*, 1, pp. 541–551, 1989.

[13] S. Hochreiter., "Untersuchungen zu dynamischen neuronalen Netzen," *Diploma thesis. Institut f. Informatik, Technische Univ. Munich. Advisor: J. Schmidhuber*, 1991.

[14] S. Hochreiter *et al.*, "Gradient flow in recurrent nets: the difficulty of learning long-term dependencies," *In S. C. Kremer and J. F. Kolen, editors, A Field Guide to Dynamical Recurrent Neural Networks. IEEE Press*, 2001.

[15] J. Weng, N. Ahuja and T. S. Huang, "Cresceptron: a self-organizing neural network which grows adaptively," *Proc. International Joint Conference on Neural Networks*, Baltimore, Maryland, vol I, pp. 576-581, June, 1992.

[16] J. Weng, N. Ahuja and T. S. Huang, "Learning recognition and segmentation of 3-D objects from 2-D images," *Proc. 4th International Conf. Computer Vision*, Berlin, Germany, pp. 121-128, May, 1993.

[17] J. Weng, N. Ahuja and T. S. Huang, "Learning recognition and segmentation using the Cresceptron," *International Journal of Computer Vision*, vol. 25, no. 2, pp. 105-139, Nov. 1997.

[18] Morgan, Bourlard, Renals, Cohen, Franco (1993) "Hybrid neural network/hidden Markov model systems for continuous speech recognition. ICASSP/IJPRAI"

[19] T. Robinson. (1992) A real-time recurrent error propagation network word recognition system, ICASSP.

[20] Waibel, Hanazawa, Hinton, Shikano, Lang. (1989) "Phoneme recognition using time-delay neural networks. IEEE Transactions on Acoustics, Speech and Signal Processing."

[21] J. Baker, Li Deng, Jim Glass, S. Khudanpur, C.-H. Lee, N. Morgan, and D. O'Shaughnessy (2009). "Research Developments and Directions in Speech Recognition and Understanding, Part 1," *IEEE Signal Processing Magazine*, vol. 26, no. 3, pp. 75-80, 2009.

[22] Y. Bengio (1991). "Artificial Neural Networks and their Application to Speech/Sequence Recognition," Ph.D. thesis, McGill University, Canada.

[23] L. Deng, K. Hassanein, M. Elmasry. (1994) "Analysis of correlation structure for a neural predictive model with applications to speech recognition," *Neural Networks*, vol. 7, No. 2., pp. 331-339.

[24] Hinton, G.; Deng, L.; Yu, D.; Dahl, G.; Mohamed, A.; Jaitly, N.; Senior, A.; Vanhoucke, V.; Nguyen, P.; Sainath, T.; Kingsbury, B. (2012). "Deep Neural Networks for Acoustic Modeling in Speech Recognition --- The shared views of four research groups" . *IEEE Signal Processing Magazine* **29** (6): 82–97. doi:10.1109/msp.2012.2205597.

[25] Deng, L.; Hinton, G.; Kingsbury, B. (2013). "New types of deep neural network learning for speech recognition and related applications: An overview (ICASSP)".

[26] Keynote talk: Recent Developments in Deep Neural Networks. ICASSP, 2013 (by Geoff Hinton).

[27] Keynote talk: "Achievements and Challenges of Deep Learning - From Speech Analysis and Recognition To Language and Multimodal Processing," Interspeech, September 2014.

[28] G. E. Hinton., "Learning multiple layers of representation," *Trends in Cognitive Sciences*, 11, pp. 428–434, 2007.

[29] J. Schmidhuber., "Learning complex, extended sequences using the principle of history compression," *Neural Computation*, 4, pp. 234–242, 1992.

[30] J. Schmidhuber., "My First Deep Learning System of 1991 + Deep Learning Timeline 1962–2013."

[31] http://research.microsoft.com/apps/pubs/default.aspx?id=189004

[32] L. Deng et al. Recent Advances in Deep Learning for Speech Research at Microsoft, ICASSP, 2013.

[33] L. Deng, O. Abdel-Hamid, and D. Yu, A deep convolutional neural network using heterogeneous pooling for trading acoustic invariance with phonetic confusion, ICASSP, 2013.

[34] T. Sainath *et al.*, "Convolutional neural networks for LVCSR," *ICASSP*, 2013.

[35] D. Yu, L. Deng, G. Li, and F. Seide (2011). "Discriminative pretraining of deep neural networks," U.S. Patent Filing.

[36] NIPS Workshop: Deep Learning for Speech Recognition and Related Applications, Whistler, BC, Canada, Dec. 2009 (Organizers: Li Deng, Geoff Hinton, D. Yu).

[37] Yu, D.; Deng, L. (2014). "Automatic Speech Recognition: A Deep Learning Approach (Publisher: Springer)".

[38] D. C. Ciresan *et al.*, "Deep Big Simple Neural Nets for Handwritten Digit Recognition," *Neural Computation*, 22, pp. 3207–3220, 2010.

[39] R. Raina, A. Madhavan, A. Ng., "Large-scale Deep Unsupervised Learning using Graphics Processors," *Proc. 26th Int. Conf. on Machine Learning*, 2009.

[40] Riesenhuber, M; Poggio, T. "Hierarchical models of object recognition in cortex" . *Nature Neuroscience* **1999** (11): 1019–1025.

[41] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, L. D. Jackel. *Backpropagation Applied to Handwritten Zip Code Recognition.* Neural Computation, 1(4):541–551, 1989.

[42] S. Hochreiter. Untersuchungen zu dynamischen neuronalen Netzen. Diploma thesis, Institut f. Informatik, Technische Univ. Munich, 1991. Advisor: J. Schmidhuber

[43] S. Hochreiter, Y. Bengio, P. Frasconi, and J. Schmidhuber. Gradient flow in recurrent nets: the difficulty of learning long-term dependencies. In S. C. Kremer and J. F. Kolen, editors, A Field Guide to Dynamical Recurrent Neural Networks. IEEE Press, 2001.

[44] Hochreiter, Sepp; and Schmidhuber, Jürgen; *Long Short-Term Memory*, Neural Computation, 9(8):1735–1780, 1997

[45] 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

[46] Graves, A.; Liwicki, M.; Fernandez, S.; Bertolami, R.; Bunke, H.; Schmidhuber, J. "A Novel Connectionist System for Improved Unconstrained Handwriting Recognition" . *IEEE Transactions on Pattern Analysis and Machine Intelligence* **31** (5): 2009.

[47] Sven Behnke (2003). *Hierarchical Neural Networks for Image Interpretation.* (PDF). Lecture Notes in Computer Science **2766**. Springer.

[48] Smolensky, P. (1986). *Information processing in dynamical systems: Foundations of harmony theory. In D. E. Rumelhart, J. L. McClelland, & the PDP Research Group, Parallel Distributed Processing: Explorations in the Microstructure of Cognition.* **1**. pp. 194–281.

[49] 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.

[50] Hinton, G. (2009). "Deep belief networks". *Scholarpedia* **4** (5): 5947. doi:10.4249/scholarpedia.5947.

[51] John Markoff (25 June 2012). "How Many Computers to Identify a Cat? 16,000." . *New York Times*.

[52] Ng, Andrew; Dean, Jeff (2012). "Building High-level Features Using Large Scale Unsupervised Learning" (PDF).

[53] D. C. Ciresan, U. Meier, J. Masci, L. M. Gambardella, J. Schmidhuber. Flexible, High Performance Convolutional Neural Networks for Image Classification. International Joint Conference on Artificial Intelligence (IJCAI-2011, Barcelona), 2011.

[54] Martines, H.; Bengio, Y.; Yannakakis, G. N. (2013). "Learning Deep Physiological Models of Affect" . *IEEE Computational Intelligence* **8** (2): 20.

[55] D. C. Ciresan, U. Meier, J. Masci, J. Schmidhuber. Multi-Column Deep Neural Network for Traffic Sign Classification. Neural Networks, 2012.

[56] 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.

[57] 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.

[58] D. J. Felleman and D. C. Van Essen, "Distributed hierarchical processing in the primate cerebral cortex," *Cerebral Cortex*, 1, pp. 1-47, 1991.

[59] J. Weng, "Natural and Artificial Intelligence: Introduction to Computational Brain-Mind," BMI Press, ISBN 978-0985875725, 2012.

[60] J. Weng, "Why Have We Passed `Neural Networks Do not Abstract Well'?," *Natural Intelligence: the INNS Magazine*, vol. 1, no.1, pp. 13-22, 2011.

[61] Z. Ji, J. Weng, and D. Prokhorov, "Where-What Network 1: Where and What Assist Each Other Through Top-down Connections," *Proc. 7th International Conference on Development and Learning (ICDL'08)*, Monterey, CA, Aug. 9-12, pp. 1-6, 2008.

[62] X. Wu, G. Guo, and J. Weng, "Skull-closed Autonomous Development: WWN-7 Dealing with Scales," *Proc. International Conference on Brain-Mind*, July 27–28, East Lansing, Michigan, pp. +1-9, 2013.

[63] Szegedy, Christian, Alexander Toshev, and Dumitru Erhan. "Deep neural networks for object detection." Advances in Neural Information Processing Systems. 2013.

[64] T. Mikolov *et al.*, "Recurrent neural network based language model," *Interspeech*, 2010.

[65] LeCun, Y. et al. "Gradient-based learning applied to document recognition" . *Proceedings of the IEEE* **86** (11): 2278–2324. doi:10.1109/5.726791.

[66] G. E. Hinton *et al.*., "Deep Neural Networks for Acoustic Modeling in Speech Recognition: The shared views of four research groups," *IEEE Signal Processing Magazine*, pp. 82–97, November 2012.

[67] Y. Bengio *et al.*., "Advances in optimizing recurrent networks," *ICASSP*, 2013.

[68] G. Dahl *et al.*., "Improving DNNs for LVCSR using rectified linear units and dropout," *ICASSP*, 2013.

[69] G. E. Hinton., "A Practical Guide to Training Restricted Boltzmann Machines," *Tech. Rep. UTML TR 2010-003, Dept. CS., Univ. of Toronto*, 2010.

[70] Hinton, G.E. "Deep belief networks" . *Scholarpedia* **4** (5): 5947. doi:10.4249/scholarpedia.5947.

[71] H. Larochelle *et al.*., "An empirical evaluation of deep architectures on problems with many factors of variation," *in Proc. 24th Int. Conf. Machine Learning*, pp. 473–480, 2007.

[72] G. E. Hinton., "Training Product of Experts by Minimizing Contrastive Divergence," *Neural Computation*, 14, pp. 1771–1800, 2002.

[73] A. Fischer and C. Igel. Training Restricted Boltzmann Machines: An Introduction. Pattern Recognition 47, pp. 25-39, 2014

[74] http://ufldl.stanford.edu/tutorial/index.php/Convolutional_Neural_Network

[75]

[76]

[77] Hinton, Geoffrey; Salakhutdinov, Ruslan (2012). "A better way to pretrain deep Boltzmann machines" (PDF). *Advances in Neural* **3**: 1–9.

[78] Hinton, Geoffrey; Salakhutdinov, Ruslan (2009). "Efficient Learning of Deep Boltzmann Machines" (PDF) **3**. pp. 448–455.

[79] Bengio, Yoshua; LeCun, Yann (2007). "Scaling Learning Algorithms towards AI" (PDF) **1**. pp. 1–41.

[80] Larochelle, Hugo; Salakhutdinov, Ruslan (2010). "Efficient Learning of Deep Boltzmann Machines" (PDF). pp. 693–700.

[81] Vincent, Pascal; Larochelle, Hugo; Lajoie, Isabelle; Bengio, Yoshua; Manzagol, Pierre-Antoine (2010). "Stacked Denoising Autoencoders: Learning Useful Representations in a Deep Network with a Local Denoising Criterion" . *The Journal of Machine Learning Research* **11**: 3371–3408.

[82] Deng, Li; Yu, Dong (2011). "Deep Convex Net: A Scalable Architecture for Speech Pattern Classification" (PDF). *Proceedings of the Interspeech*: 2285–2288.

[83] Deng, Li; Yu, Dong; Platt, John (2012). "Scalable stacking and learning for building deep architectures" . *2012 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*: 2133–2136.

[84] David, Wolpert (1992). "Stacked generalization" . *Neural networks* **5(2)**: 241–259. doi:10.1016/S0893-6080(05)80023-1.

[85] Bengio, Yoshua (2009). "Learning deep architectures for AI" . *Foundations and trends in Machine Learning* **2(1)**: 1–127.

[86] Hutchinson, Brian; Deng, Li; Yu, Dong (2012). "Tensor deep stacking networks" . *IEEE transactions on pattern analysis and machine intelligence* **1–15**.

[87] Hinton, Geoffrey; Salakhutdinov, Ruslan (2006). "Reducing the Dimensionality of Data with Neural Networks" . *Science* **313**: 504–507. doi:10.1126/science.1127647. PMID 16873662.

[88] Dahl, G.; Yu, D.; Deng, L.; Acero, A. (2012). "Context-Dependent Pre-Trained Deep Neural Networks for Large-Vocabulary Speech Recognition" . *Audio, Speech, and ...* 20(1): 30–42.

[89] Mohamed, Abdel-rahman; Dahl, George; Hinton, Geoffrey (2012). "Acoustic Modeling Using Deep Belief Networks" . *IEEE Transactions on Audio, Speech, and Language Processing*. 20(1): 14–22.

[90] Courville, Aaron; Bergstra, James; Bengio, Yoshua (2011). "A Spike and Slab Restricted Boltzmann Machine" (PDF). *International . . .* **15**: 233–241.

[91] Mitchell, T; Beauchamp, J (1988). "Bayesian Variable Selection in Linear Regression" . *Journal of the American Statistical Association*. 83 (404): 1023–1032. doi:10.1080/01621459.1988.10478694.

[92] Courville, Aaron; Bergstra, James; Bengio, Yoshua (2011). "Unsupervised Models of Images by Spike-and-Slab RBMs" (PDF). *Proceedings of the . . .* **10**: 1–8.

[93] Hinton, Geoffrey; Osindero, Simon; Teh, Yee-Whye (2006). "A Fast Learning Algorithm for Deep Belief Nets" . *Neural Computation* **1554**: 1527–1554.

[94] Larochelle, Hugo; Bengio, Yoshua; Louradour, Jerdme; Lamblin, Pascal (2009). "Exploring Strategies for Training Deep Neural Networks" . *The Journal of Machine Learning Research* **10**: 1–40.

[95] Coates, Adam; Carpenter, Blake (2011). "Text Detection and Character Recognition in Scene Images with Unsupervised Feature Learning" . pp. 440–445.

[96] Lee, Honglak; Grosse, Roger (2009). "Convolutional deep belief networks for scalable unsupervised learning of hierarchical representations" . *Proceedings of the 26th Annual International Conference on Machine Learning - ICML '09*: 1–8.

[97] Lin, Yuanqing; Zhang, Tong (2010). "Deep Coding Network" (PDF). *Advances in Neural . . .*: 1–9.

[98] Ranzato, Marc Aurelio; Boureau, Y-Lan (2007). "Sparse Feature Learning for Deep Belief Networks" (PDF). *Advances in neural information . . .*: 1–8.

[99] Socher, Richard; Lin, Clif (2011). "Parsing Natural Scenes and Natural Language with Recursive Neural Networks" (PDF). *Proceedings of the . . .*

[100] Taylor, Graham; Hinton, Geoffrey (2006). "Modeling Human Motion Using Binary Latent Variables" (PDF). *Advances in neural . . .*

[101] Vincent, Pascal; Larochelle, Hugo (2008). "Extracting and composing robust features with denoising autoencoders" . *Proceedings of the 25th international conference on Machine learning - ICML '08*: 1096–1103.

[102] Kemp, Charles; Perfors, Amy; Tenenbaum, Joshua (2007). "Learning overhypotheses with hierarchical Bayesian models" . *Developmental science*. 10(3): 307–21. doi:10.1111/j.1467-7687.2007.00585.x. PMID 17444972.

[103] Xu, Fei; Tenenbaum, Joshua (2007). "Word learning as Bayesian inference" . *Psychol Rev*. 114(2): 245–72. doi:10.1037/0033-295X.114.2.245. PMID 17500627.

[104] Chen, Bo; Polatkan, Gungor (2011). "The Hierarchical Beta Process for Convolutional Factor Analysis and Deep Learning" (PDF). *Machine Learning . . .*

[105] Fei-Fei, Li; Fergus, Rob (2006). "One-shot learning of object categories" . *IEEE Trans Pattern Anal Mach Intell*. 28(4): 594–611. doi:10.1109/TPAMI.2006.79. PMID 16566508.

[106] Rodriguez, Abel; Dunson, David (2008). "The Nested Dirichlet Process" . *Journal of the American Statistical Association*. 103(483): 1131–1154. doi:10.1198/016214508000000553.

[107] Ruslan, Salakhutdinov; Joshua, Tenenbaum (2012). "Learning with Hierarchical-Deep Models" . *IEEE transactions on pattern analysis and machine intelligence*: 1–14. PMID 23267196.

[108] Chalasani, Rakesh; Principe, Jose (2013). "Deep Predictive Coding Networks" . *arXiv preprint arXiv*: 1–13.

[109] Cho, Youngmin (2012). "Kernel Methods for Deep Learning" (PDF). pp. 1–9.

[110] Scholkopf, B; Smola, Alexander (1998). "Nonlinear component analysis as a kernel eigenvalue problem" . *Neural computation* **(44)**.

[111] L. Deng, G. Tur, X. He, and D. Hakkani-Tur. "Use of Kernel Deep Convex Networks and End-To-End Learning for Spoken Language Understanding," *Proc. IEEE Workshop on Spoken Language Technologies*, 2012

[112] Mnih, Volodymyr et al. (2015). "Human-level control through deep reinforcement learning" (PDF) **518**. pp. 529–533.

[113] *TIMIT Acoustic-Phonetic Continuous Speech Corpus* Linguistic Data Consortium, Philadelphia.

[114] Abdel-Hamid, O. et al. (2014). "Convolutional Neural Networks for Speech Recognition". *IEEE/ACM Transactions on Audio, Speech, and Language Processing* **22** (10): 1533–1545. doi:10.1109/taslp.2014.2339736.

[115] Deng, L.; Platt, J. (2014). "Ensemble Deep Learning for Speech Recognition". *Proc. Interspeech*.

[116] Yu, D.; Deng, L. (2010). "Roles of Pre-Training and Fine-Tuning in Context-Dependent DBN-HMMs for Real-World Speech Recognition". *NIPS Workshop on Deep Learning and Unsupervised Feature Learning*.

[117] Deng L., Li, J., Huang, J., Yao, K., Yu, D., Seide, F. et al. Recent Advances in Deep Learning for Speech Research at Microsoft. ICASSP, 2013.

[118] Deng, L.; Li, Xiao (2013). "Machine Learning Paradigms for Speech Recognition: An Overview". *IEEE Transactions on Audio, Speech, and Language Processing*.

[119] L. Deng, M. Seltzer, D. Yu, A. Acero, A. Mohamed, and G. Hinton (2010) Binary Coding of Speech Spectrograms Using a Deep Auto-encoder. Interspeech.

[120] Z. Tuske, P. Golik, R. Schlüter and H. Ney (2014). Acoustic Modeling with Deep Neural Networks Using Raw Time Signal for LVCSR. Interspeech.

[121] McMillan, R. "How Skype Used AI to Build Its Amazing New Language Translator", Wire, Dec. 2014.

[122] Hannun et al. (2014) "Deep Speech: Scaling up end-to-end speech recognition", arXiv:1412.5567.

[123] Ron Schneiderman (2015) "Accuracy, Apps Advance Speech Recognition --- Interview with Vlad Sejnoha and Li Deng", IEEE Signal Processing Magazine, Jan, 2015.

[124] http://yann.lecun.com/exdb/mnist/.

[125] D. Ciresan, U. Meier, J. Schmidhuber., "Multi-column Deep Neural Networks for Image Classification," *Technical Report No. IDSIA-04-12', 2012.*

[126] Vinyals et al. (2014)."Show and Tell: A Neural Image Caption Generator," arXiv:1411.4555.

[127] Fang et al. (2014)."From Captions to Visual Concepts and Back," arXiv:1411.4952.

[128] Kiros et al. (2014)."Unifying Visual-Semantic Embeddings with Multimodal Neural Language Models," arXiv: 1411.2539.

[129] Zhong, S.; Liu, Y.; Liu, Y. "Bilinear Deep Learning for Image Classification". *Proceedings of the 19th ACM International Conference on Multimedia* **11**: 343–352.

[130] Nvidia Demos a Car Computer Trained with "Deep Learning" (2015-01-06), David Talbot, *MIT Technology Review*

[131] Y. Bengio, R. Ducharme, P. Vincent, C. Jauvin., "A Neural Probabilistic Language Model," *Journal of Machine Learning Research 3 (2003) 1137–1155', 2003.*

[132] Goldberg, Yoav; Levy, Omar. "word2vec Explained: Deriving Mikolov et al.' s Negative-Sampling Word-Embedding Method" (PDF). *Arxiv*. Retrieved 26 October 2014.

[133] Socher, Richard; Manning, Christopher. "Deep Learning for NLP" (PDF). Retrieved 26 October 2014.

[134] Socher, Richard; Bauer, John; Manning, Christopher; Ng, Andrew (2013). "Parsing With Compositional Vector Grammars" (PDF). *Proceedings of the ACL 2013 conference*.

[135] Socher, Richard (2013). "Recursive Deep Models for Semantic Compositionality Over a Sentiment Treebank" (PDF). *EMNLP 2013*.

[136] Y. Shen, X. He, J. Gao, L. Deng, and G. Mesnil (2014) " A Latent Semantic Model with Convolutional-Pooling Structure for Information Retrieval," Proc. CIKM.

[137] P. Huang, X. He, J. Gao, L. Deng, A. Acero, and L. Heck (2013) "Learning Deep Structured Semantic Models for Web Search using Clickthrough Data," Proc. CIKM.

[138] I. Sutskever, O. Vinyals, Q. Le (2014) "Sequence to Sequence Learning with Neural Networks," Proc. NIPS.

[139] J. Gao, X. He, W. Yih, and L. Deng(2014) "Learning Continuous Phrase Representations for Translation Modeling," Proc. ACL.

[140] J. Gao, P. Pantel, M. Gamon, X. He, L. Deng (2014) "Modeling Interestingness with Deep Neural Networks," Proc. EMNLP.

[141] J. Gao, X. He, L. Deng (2014) "Deep Learning for Natural Language Processing: Theory and Practice (Tutorial)," CIKM.

[142] Arrowsmith, J; Miller, P (2013). "Trial watch: Phase II and phase III attrition rates 2011-2012". *Nature Reviews Drug Discovery* **12** (8): 569. doi:10.1038/nrd4090. PMID 23903212.

[143] Verbist, B; Klambauer, G; Vervoort, L; Talloen, W; The Qstar, Consortium; Shkedy, Z; Thas, O; Bender, A; Göhlmann, H. W.; Hochreiter, S (2015). "Using transcriptomics to guide lead optimization in drug discovery projects: Lessons learned from the QSTAR project". *Drug Discovery Today*. doi:10.1016/j.drudis.2014.12.014. PMID 25582842.

[144] "Announcement of the winners of the Merck Molecular Activity Challenge" https://www.kaggle.com/c/MerckActivity/details/winners.

[145] Dahl, G. E.; Jaitly, N.; & Salakhutdinov, R. (2014) "Multi-task Neural Networks for QSAR Predictions," ArXiv, 2014.

[146] "Toxicology in the 21st century Data Challenge" https://tripod.nih.gov/tox21/challenge/leaderboard.jsp

[147] "NCATS Announces Tox21 Data Challenge Winners" http://www.ncats.nih.gov/news-and-events/features/tox21-challenge-winners.html

[148] Unterthiner, T.; Mayr, A.; Klambauer, G.; Steijaert, M.; Ceulemans, H.; Wegner, J. K.; & Hochreiter, S. (2014) "Deep Learning as an Opportunity in Virtual Screening" . Workshop on Deep Learning and Representation Learning (NIPS2014).

[149] Unterthiner, T.; Mayr, A.; Klambauer, G.; & Hochreiter, S. (2015) "Toxicity Prediction using Deep Learning" . ArXiv, 2015.

[150] Ramsundar, B.; Kearnes, S.; Riley, P.; Webster, D.; Konerding, D.;& Pande, V. (2015) "Massively Multitask Networks for Drug Discovery" . ArXiv, 2015.

[151] Tkachenko, Yegor. Autonomous CRM Control via CLV Approximation with Deep Reinforcement Learning in Discrete and Continuous Action Space. (April 8, 2015). arXiv.org: http://arxiv.org/abs/1504.01840

[152] Utgoff, P. E.; Stracuzzi, D. J. (2002). "Many-layered learning" . *Neural Computation* **14**: 2497–2529. doi:10.1162/08997660260293319.

[153] J. Elman, *et al.*, "Rethinking Innateness," 1996.

[154] Shrager, J.; Johnson, MH (1996). "Dynamic plasticity influences the emergence of function in a simple cortical array" . *Neural Networks* **9** (7): 1119–1129. doi:10.1016/0893-6080(96)00033-0.

[155] Quartz, SR; Sejnowski, TJ (1997). "The neural basis of cognitive development: A constructivist manifesto" . *Behavioral and Brain Sciences* **20** (4): 537–556. doi:10.1017/s0140525x97001581.

[156] S. Blakeslee., "In brain's early growth, timetable may be critical," *The New York Times, Science Section*, pp. B5–B6, 1995.

[157] {BUFILL} E. Bufill, J. Agusti, R. Blesa., "Human neoteny revisited: The case of synaptic plasticity," *American Journal of Human Biology*, 23 (6), pp. 729–739, 2011.

[158] J. Shrager and M. H. Johnson., "Timing in the development of cortical function: A computational approach," *In B. Julesz and I. Kovacs (Eds.), Maturational windows and adult cortical plasticity*, 1995.

[159] D. Hernandez., "The Man Behind the Google Brain: Andrew Ng and the Quest for the New AI," *http://www.wired.com/wiredenterprise/2013/05/neuro-artificial-intelligence/all/. Wired*, 10 May 2013.

[160] C. Metz., "Facebook's 'Deep Learning' Guru Reveals the Future of AI," *http://www.wired.com/wiredenterprise/2013/12/facebook-yann-lecun-qa/. Wired*, 12 December 2013.

[161] G. Marcus., "Is "Deep Learning"a Revolution in Artificial Intelligence?" *The New Yorker*, 25 November 2012.

[162] Smith, G. W. (March 27, 2015). "Art and Artificial Intelligence" . ArtEnt. Retrieved March 27, 2015.

[163] Ben Goertzel. Are there Deep Reasons Underlying the Pathologies of Today's Deep Learning Algorithms? (2015) Url: http://goertzel.org/DeepLearning_v1.pdf

[164] Nguyen, Anh, Jason Yosinski, and Jeff Clune. "Deep Neural Networks are Easily Fooled: High Confidence Predictions for Unrecognizable Images." arXiv preprint arXiv:1412.1897 (2014).

[165] Szegedy, Christian, et al. "Intriguing properties of neural networks." arXiv preprint arXiv:1312.6199 (2013).

[166] Zhu, S.C.; Mumford, D. "A stochastic grammar of images" . *Found. Trends. Comput. Graph. Vis.* **2** (4): 259–362. doi:10.1561/0600000018.

[167] Miller, G. A., and N. Chomsky. "Pattern conception." Paper for Conference on pattern detection, University of Michigan. 1957.

[168] Jason Eisner, Deep Learning of Recursive Structure: Grammar Induction, http://techtalks.tv/talks/deep-learning-of-recursive-structure-grammar-induction/58089/

2.12 External links

- TED talk on the applications of deep learning and future consequences by Jeremy Howard
- Deep learning information from the University of Montreal
- Deep learning information from Stanford University
- Deep Learning Resources, NVIDIA Developer Zone
- Geoffrey Hinton's webpage
- Hinton deep learning tutorial
- Yann LeCun's webpage
- The Center for Biological and Computational Learning (CBCL)
- Stanford tutorial on unsupervised feature learning and deep learning
- Google's DistBelief Framework
- NIPS 2013 Conference (talks on deep learning related material)
- Mnih, Volodymyr; Kavukcuoglu, Koray; Silver, David; Graves, Alex; Antonoglou, Ioannis; Wierstra, Daan; Riedmiller, Martin (2013), *Playing Atari with Deep Reinforcement Learning* (PDF), arXiv:1312.5602

- 100 Best GitHub: Deep Learning
- Silicon Chips That See Are Going to Make Your Smartphone Brilliant. "Many of the devices around us may soon acquire powerful new abilities to understand images and video, thanks to hardware designed for the machine-learning technique called deep learning." Tom Simonite (May 2015), *MIT Technology Review*

Chapter 3

Feature learning

Feature learning or **representation learning**[*][1] is a set of techniques that learn a transformation of raw data input to a representation that can be effectively exploited in machine learning tasks.

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 sensor measurement is usually complex, redundant, and highly variable. Thus, it is necessary to discover useful features or representations from raw data. Traditional hand-crafted features often require expensive human labor and often rely on expert knowledge. Also, they normally do not generalize well. This motivates the design of efficient feature learning techniques.

Feature learning can be divided into two categories: supervised and unsupervised feature learning.

- In supervised feature learning, features are learned with labeled input data. Examples include neural networks, multilayer perceptron, 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,[*][2] and various forms of clustering.[*][3][*][4][*][5]

3.1 Supervised feature learning

Supervised feature learning is to learn features from labeled data. Several approaches are introduced in the following.

3.1.1 Supervised dictionary learning

Dictionary learning is to learn a set (dictionary) of representative elements from the input data such that each data point can be represented as a weighted sum of the representative elements. The dictionary elements and the weights may be found by minimizing the average representation error (over the input data), together with a *L1* regularization on the weights to enable sparsity (i.e., the representation of each data point has only a few nonzero weights).

Supervised dictionary learning exploits both the structure underlying the input data and the labels for optimizing the dictionary elements. For example, a supervised dictionary learning technique was proposed by Mairal et al. in 2009.[*][6] The authors apply dictionary learning on classification problems by jointly optimizing the dictionary elements, weights for representing data points, and parameters of the classifier based on the input data. In particular, a minimization problem is formulated, where the objective function consists of the classification error, the representation error, an *L1* regularization on the representing weights for each data point (to enable sparse representation of data), and an *L2* regularization on the parameters of the classifier.

3.1.2 Neural networks

Neural networks are used to illustrate a family of learning algorithms via a "network" consisting of multiple layers of inter-connected nodes. It is inspired by the nervous system, where the nodes are viewed as neurons and edges are viewed as synapse. Each edge has an associated weight, and the network defines computational rules that passes input data from the input layer to the output layer. A network function associated with a neural network characterizes the relationship between input and output layers, which is parameterized by the weights. With appropriately defined network functions, various learning tasks can be performed by minimizing a cost function over the network function (weights).

Multilayer neural networks can be used to perform feature learning, since they learn a representation of their input at the hidden layer(s) which is subsequently used for classification or regression at the output layer.

3.2 Unsupervised feature learning

Unsupervised feature learning is to learn features from unlabeled data. The goal of unsupervised feature learning is often to discover low-dimensional features that captures some structure underlying the high-dimensional input data. When the feature learning is performed in an unsupervised way, it enables a form of semisupervised learning where first, features are learned from an unlabeled dataset, which are then employed to improve performance in a supervised setting with labeled data.*[7]*[8] Several approaches are introduced in the following.

3.2.1 *K*-means clustering

K-means clustering is an approach for vector quantization. In particular, given a set of n vectors, k-means clustering groups them into k clusters (i.e., subsets) in such a way that each vector belongs to the cluster with the closest mean. The problem is computationally NP-hard, and suboptimal greedy algorithms have been developed for k-means clustering.

In feature learning, k-means clustering can be used to group an unlabeled set of inputs into k clusters, and then use the centroids of these clusters to produce features. These features can be produced in several ways. The simplest way is to add k binary features to each sample, where each feature j has value one iff the jth centroid learned by k-means is the closest to the sample under consideration.*[3] It is also possible to use the distances to the clusters as features, perhaps after transforming them through a radial basis function (a technique that has used to train RBF networks*[9]). Coates and Ng note that certain variants of k-means behave similarly to sparse coding algorithms.*[10]

In a comparative evaluation of unsupervised feature learning methods, Coates, Lee and Ng found that k-means clustering with an appropriate transformation outperforms the more recently invented auto-encoders and RBMs on an image classification task.*[3] *K*-means has also been shown to improve performance in the domain of NLP, specifically for named-entity recognition;*[11] there, it competes with Brown clustering, as well as with distributed word representations (also known as neural word embeddings).*[8]

3.2.2 Principal component analysis

Principal component analysis (PCA) is often used for dimension reduction. Given a unlabeled set of n input data vectors, PCA generates p (which is much smaller than the dimension of the input data) right singular vectors corresponding to the p largest singular values of the data matrix, where the kth row of the data matrix is the kth input data vector shifted by the sample mean of the input (i.e., subtracting the sample mean from the data vector). Equivalently, these singular vectors are the eigenvectors corresponding to the p largest eigenvalues of the sample covariance matrix of the input vectors. These p singular vectors are the feature vectors learned from the input data, and they represent directions along which the data has the largest variations.

PCA is a linear feature learning approach since the p singular vectors are linear functions of the data matrix. The singular vectors can be generated via a simple algorithm with p iterations. In the ith iteration, the projection of the data matrix on the *(i-1)*th eigenvector is subtracted, and the ith singular vector is found as the right singular vector corresponding to the largest singular of the residual data matrix.

PCA has several limitations. First, it assumes that the directions with large variance are of most interest, which may not be the case in many applications. PCA only relies on orthogonal transformations of the original data, and it only exploits the first- and second-order moments of the data, which may not well characterize the distribution of the data. Furthermore, PCA can effectively reduce dimension only when the input data vectors are correlated (which results in a few dominant eigenvalues).

3.2.3 Local linear embedding

Local linear embedding (LLE) is a nonlinear unsupervised learning approach for generating low-dimensional neighbor-preserving representations from (unlabeled) high-dimension input. The approach was proposed by Sam T. Roweis and Lawrence K. Saul in 2000.*[12]*[13]

The general idea of LLE is to reconstruct the original high-dimensional data using lower-dimensional points while maintaining some geometric properties of the neighborhoods in the original data set. LLE consists of two major steps. The first step is for "neighbor-preserving," where each input data point Xi is reconstructed as a weighted sum of K nearest neighboring data points, and the optimal weights are found by minimizing the average squared reconstruction error (i.e., difference between a point and its reconstruction) under the constraint that the weights associated to each point sum up to one. The second step is for "dimension reduction," by looking for vectors in a lower-dimensional space that minimizes the representation error using the optimized weights in the first step. Note that in the first step, the weights are optimized with data being fixed, which can be solved as a least squares problem; while in the second step, lower-dimensional points are optimized with the weights being fixed, which can be solved via sparse eigenvalue decomposition.

The reconstruction weights obtained in the first step captures the "intrinsic geometric properties" of a neighborhood in the input data.*[13] It is assumed that original data lie on a smooth lower-dimensional manifold, and the

"intrinsic geometric properties" captured by the weights of the original data are expected also on the manifold. This is why the same weights are used in the second step of LLE. Compared with PCA, LLE is more powerful in exploiting the underlying structure of data.

3.2.4 Independent component analysis

Independent component analysis (ICA) is technique for learning a representation of data using a weighted sum of independent non-Gaussian components.*[14] The assumption of non-Gaussian is imposed since the weights cannot be uniquely determined when all the components follow Gaussian distribution.

3.2.5 Unsupervised dictionary learning

Different from supervised dictionary learning, unsupervised dictionary learning does not utilize the labels of the data and only exploits the structure underlying the data for optimizing the dictionary elements. An example of unsupervised dictionary learning is sparse coding, which aims to learn basis functions (dictionary elements) for data representation from unlabeled input data. Sparse coding can be applied to learn overcomplete dictionary, where the number of dictionary elements is larger than the dimension of the input data.*[15] Aharon et al. proposed an algorithm known as K-SVD for learning from unlabeled input data a dictionary of elements that enables sparse representation of the data.*[16]

3.3 Multilayer/Deep architectures

The hierarchical architecture of the neural system inspires deep learning architectures for feature learning by stacking multiple layers of simple learning blocks.*[17] These architectures are often designed based on the assumption of distributed representation: observed data is generated by the interactions of many different factors on multiple levels. In a deep learning architecture, the output of each intermediate layer can be viewed as a representation of the original input data. Each level uses the representation produced by previous level as input, and produces new representations as output, which is then fed to higher levels. The input of bottom layer is the raw data, and the output of the final layer is the final low-dimensional feature or representation.

3.3.1 Restricted Boltzmann machine

Restricted Boltzmann machines (RBMs) are often used as a building block for multilayer learning architectures.*[3]*[18] An RBM can be represented by an undirected bipartite graph consisting of a group of binary hidden variables, a group of visible variables, and edges connecting the hidden and visible nodes. It is a special case of the more general Boltzmann machines with the constraint of no intra-node connections. Each edge in an RBM is associated with a weight. The weights together with the connections define an energy function, based on which a joint distribution of visible and hidden nodes can be devised. Based on the topology of the RBM, the hidden (visible) variables are independent conditioned on the visible (hidden) variables. Such conditional independence facilitates computations on RBM.

An RBM can be viewed as a single layer architecture for unsupervised feature learning. In particular, the visible variables correspond to input data, and the hidden variables correspond to feature detectors. The weights can be trained by maximizing the probability of visible variables using the contrastive divergence (CD) algorithm by Geoffrey Hinton.*[18]

In general, the training of RBM by solving the above maximization problem tends to result in non-sparse representations. The sparse RBM, *[19] a modification of the RBM, was proposed to enable sparse representations. The idea is to add a regularization term in the objective function of data likelihood, which penalizes the deviation of the expected hidden variables from a small constant p .

3.3.2 Autoencoder

An autoencoder consisting of encoder and decoder is a paradigm for deep learning architectures. An example is provided by Hinton and Salakhutdinov*[18] where the encoder uses raw data (e.g., image) as input and produces feature or representation as output, and the decoder uses the extracted feature from the encoder as input and reconstructs the original input raw data as output. The encoder and decoder are constructed by stacking multiple layers of RBMs. The parameters involved in the architecture are trained in a greedy layer-by-layer manner: after one layer of feature detectors is learned, they are fed to upper layers as visible variables for training the corresponding RBM. The process can be repeated until some stopping criteria is satisfied.

3.4 See also

- Basis function
- Deep learning
- Feature detection (computer vision)
- Feature extraction
- Kernel trick
- Vector quantization

3.5 References

[1] Y. Bengio; A. Courville; P. Vincent (2013). "Representation Learning: A Review and New Perspectives". *IEEE Trans. PAMI, special issue Learning Deep Architectures*.

[2] Nathan Srebro; Jason D. M. Rennie; Tommi S. Jaakkola (2004). *Maximum-Margin Matrix Factorization*. NIPS.

[3] Coates, Adam; Lee, Honglak; Ng, Andrew Y. (2011). *An analysis of single-layer networks in unsupervised feature learning* (PDF). Int'l Conf. on AI and Statistics (AISTATS).

[4] Csurka, Gabriella; Dance, Christopher C.; Fan, Lixin; Willamowski, Jutta; Bray, Cédric (2004). *Visual categorization with bags of keypoints* (PDF). ECCV Workshop on Statistical Learning in Computer Vision.

[5] Daniel Jurafsky; James H. Martin (2009). *Speech and Language Processing*. Pearson Education International. pp. 145–146.

[6] Mairal, Julien; Bach, Francis; Ponce, Jean; Sapiro, Guillermo; Zisserman, Andrew (2009). "Supervised Dictionary Learning". *Advances in neural information processing systems*.

[7] Percy Liang (2005). *Semi-Supervised Learning for Natural Language* (PDF) (M. Eng.). MIT. pp. 44–52.

[8] Joseph Turian; Lev Ratinov; Yoshua Bengio (2010). *Word representations: a simple and general method for semi-supervised learning* (PDF). Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics.

[9] Schwenker, Friedhelm; Kestler, Hans A.; Palm, Günther (2001). "Three learning phases for radial-basis-function networks". *Neural Networks* **14**: 439–458. doi:10.1016/s0893-6080(01)00027-2. CiteSeerX: 10.1.1.109.312.

[10] Coates, Adam; Ng, Andrew Y. (2012). "Learning feature representations with k-means". In G. Montavon, G. B. Orr and K.-R. Müller. *Neural Networks: Tricks of the Trade*. Springer.

[11] Dekang Lin; Xiaoyun Wu (2009). *Phrase clustering for discriminative learning* (PDF). Proc. J. Conf. of the ACL and 4th Int'l J. Conf. on Natural Language Processing of the AFNLP. pp. 1030–1038.

[12] Roweis, Sam T; Saul, Lawrence K (2000). "Nonlinear Dimensionality Reduction by Locally Linear Embedding". *Science, New Series* **290** (5500): 2323–2326. doi:10.1126/science.290.5500.2323.

[13] Saul, Lawrence K; Roweis, Sam T (2000). "An Introduction to Locally Linear Embedding".

[14] Hyvärinen, Aapo; Oja, Erkki (2000). "Independent Component Analysis: Algorithms and Applications". *Neural networks* (4): 411–430.

[15] Lee, Honglak; Battle, Alexis; Raina, Rajat; Ng, Andrew Y (2007). "Efficient sparse coding algorithms". *Advances in neural information processing systems*.

[16] Aharon, Michal; Elad, Michael; Bruckstein, Alfred (2006). "K-SVD: An Algorithm for Designing Overcomplete Dictionaries for Sparse Representation". *IEEE Trans. Signal Process.* **54** (11): 4311–4322. doi:10.1109/TSP.2006.881199.

[17] Bengio, Yoshua (2009). "Learning Deep Architectures for AI". *Foundations and Trends® in Machine Learning* **2** (1): 1–127. doi:10.1561/2200000006.

[18] Hinton, G. E.; Salakhutdinov, R. R. (2006). "Reducing the Dimensionality of Data with Neural Networks" (PDF). *Science* **313** (5786): 504–507. doi:10.1126/science.1127647. PMID 16873662.

[19] Lee, Honglak; Ekanadham, Chaitanya; Andrew, Ng (2008). "Sparse deep belief net model for visual area V2". *Advances in neural information processing systems*.

Chapter 4

Unsupervised learning

In machine learning, the problem of **unsupervised learning** is that of trying to find hidden structure in unlabeled data. Since the examples given to the learner are unlabeled, there is no error or reward signal to evaluate a potential solution. This distinguishes unsupervised learning from supervised learning and reinforcement learning.

Unsupervised learning is closely related to the problem of density estimation in statistics.*[1] However unsupervised learning also encompasses many other techniques that seek to summarize and explain key features of the data. Many methods employed in unsupervised learning are based on data mining methods used to preprocess data.

Approaches to unsupervised learning include:

- clustering (e.g., k-means, mixture models, hierarchical clustering),*[2]
- Approaches for learning latent variable models such as
  - Expectation–maximization algorithm (EM)
  - Method of moments
  - Blind signal separation techniques, e.g.,
    - Principal component analysis,
    - Independent component analysis,
    - Non-negative matrix factorization,
    - Singular value decomposition.*[3]

Among neural network models, the self-organizing map (SOM) and adaptive resonance theory (ART) are commonly used unsupervised learning algorithms. The SOM is a topographic organization in which nearby locations in the map represent inputs with similar properties. The ART model allows the number of clusters to vary with problem size and lets the user control the degree of similarity between members of the same clusters by means of a user-defined constant called the vigilance parameter. ART networks are also used for many pattern recognition tasks, such as automatic target recognition and seismic signal processing. The first version of ART was "ART1", developed by Carpenter and Grossberg (1988).*[4]

4.1 Method of moments

One of the approaches in unsupervised learning is the method of moments. In the method of moments, the unknown parameters (of interest) in the model are related to the moments of one or more random variables, and thus, these unknown parameters can be estimated given the moments. The moments are usually estimated from samples in an empirical way. The basic moments are first and second order moments. For a random vector, the first order moment is the mean vector, and the second order moment is the covariance matrix (when the mean is zero). Higher order moments are usually represented using tensors which are the generalization of matrices to higher orders as multi-dimensional arrays.

In particular, the method of moments is shown to be effective in learning the parameters of latent variable models.*[5] Latent variable models are statistical models where in addition to the observed variables, a set of latent variables also exists which is not observed. A highly practical example of latent variable models in machine learning is the topic modeling which is a statistical model for generating the words (observed variables) in the document based on the topic (latent variable) of the document. In the topic modeling, the words in the document are generated according to different statistical parameters when the topic of the document is changed. It is shown that method of moments (tensor decomposition techniques) consistently recover the parameters of a large class of latent variable models under some assumptions.*[5]

Expectation–maximization algorithm (EM) is also one of the most practical methods for learning latent variable models. But, it can be stuck in local optima, and the global convergence of the algorithm to the true unknown parameters of the model is not guaranteed. While, for the method of moments, the global convergence is guaranteed under some conditions.*[5]

4.2 See also

- Cluster analysis
- Expectation–maximization algorithm

- Generative topographic map
- Multilinear subspace learning
- Multivariate analysis
- Radial basis function network

4.3 Notes

[1] Jordan, Michael I.; Bishop, Christopher M. (2004). "Neural Networks" . In Allen B. Tucker. *Computer Science Handbook, Second Edition (Section VII: Intelligent Systems)*. Boca Raton, FL: Chapman & Hall/CRC Press LLC. ISBN 1-58488-360-X.

[2] Hastie,Trevor,Robert Tibshirani, Friedman,Jerome (2009). *The Elements of Statistical Learning: Data mining,Inference,and Prediction*. New York: Springer. pp. 485–586. ISBN 978-0-387-84857-0.

[3] Acharyya, Ranjan (2008); *A New Approach for Blind Source Separation of Convolutive Sources*, ISBN 978-3-639-07797-1 (this book focuses on unsupervised learning with Blind Source Separation)

[4] Carpenter, G.A. and Grossberg, S. (1988). "The ART of adaptive pattern recognition by a self-organizing neural network" (PDF). *Computer* **21**: 77–88. doi:10.1109/2.33.

[5] Anandkumar, Animashree; Ge, Rong; Hsu, Daniel; Kakade, Sham; Telgarsky, Matus (2014). "Tensor Decompositions for Learning Latent Variable Models" (PDF). *Journal of Machine Learning Research (JMLR)* **15**: 2773–2832.

4.4 Further reading

- Bousquet, O.; von Luxburg, U.; Raetsch, G., eds. (2004). *Advanced Lectures on Machine Learning*. Springer-Verlag. ISBN 978-3540231226.
- Duda, Richard O.; Hart, Peter E.; Stork, David G. (2001). "Unsupervised Learning and Clustering" . *Pattern classification* (2nd ed.). Wiley. ISBN 0-471-05669-3.
- Hastie, Trevor; Tibshirani, Robert (2009). *The Elements of Statistical Learning: Data mining,Inference,and Prediction*. New York: Springer. pp. 485–586. doi:10.1007/978-0-387-84858-7_14. ISBN 978-0-387-84857-0.
- Hinton, Geoffrey; Sejnowski, Terrence J., eds. (1999). *Unsupervised Learning: Foundations of Neural Computation*. MIT Press. ISBN 0-262-58168-X. (This book focuses on unsupervised learning in neural networks)

Chapter 5

Generative model

In probability and statistics, a **generative model** is a model for randomly generating observable-data values, typically given some hidden parameters. It specifies a joint probability distribution over observation and label sequences. Generative models are used in machine learning for either modeling data directly (i.e., modeling observations drawn from a probability density function), or as an intermediate step to forming a conditional probability density function. A conditional distribution can be formed from a generative model through Bayes' rule.

Shannon (1948) gives an example in which a table of frequencies of English word pairs is used to generate a sentence beginning with "representing and speedily is an good"; which is not proper English but which will increasingly approximate it as the table is moved from word pairs to word triplets etc.

Generative models contrast with discriminative models, in that a generative model is a full probabilistic model of all variables, whereas a discriminative model provides a model only for the target variable(s) conditional on the observed variables. Thus a generative model can be used, for example, to simulate (i.e. *generate*) values of any variable in the model, whereas a discriminative model allows only sampling of the target variables conditional on the observed quantities. Despite the fact that discriminative models do not need to model the distribution of the observed variables, they cannot generally express more complex relationships between the observed and target variables. They don't necessarily perform better than generative models at classification and regression tasks. In modern applications the two classes are seen as complementary or as different views of the same procedure.[*] [1]

Examples of generative models include:

- Gaussian mixture model and other types of mixture model
- Hidden Markov model
- Probabilistic context-free grammar
- Naive Bayes
- Averaged one-dependence estimators
- Latent Dirichlet allocation
- Restricted Boltzmann machine

If the observed data are truly sampled from the generative model, then fitting the parameters of the generative model to maximize the data likelihood is a common method. However, since most statistical models are only approximations to the *true* distribution, if the model's application is to infer about a subset of variables conditional on known values of others, then it can be argued that the approximation makes more assumptions than are necessary to solve the problem at hand. In such cases, it can be more accurate to model the conditional density functions directly using a discriminative model (see above), although application-specific details will ultimately dictate which approach is most suitable in any particular case.

5.1 See also

- Discriminative model
- Graphical model

5.2 References

[1] C. M. Bishop and J. Lasserre, Generative or Discriminative? getting the best of both worlds. In Bayesian Statistics 8, Bernardo, J. M. et al. (Eds), Oxford University Press. 3–23, 2007.

5.3 Sources

- Shannon, C.E. (1948) "A Mathematical Theory of Communication", *Bell System Technical Journal*, vol. 27, pp. 379–423, 623–656, July, October, 1948

Chapter 6

Neural coding

Neural coding is a neuroscience-related field concerned with characterizing the relationship between the stimulus and the individual or ensemble neuronal responses and the relationship among the electrical activity of the neurons in the ensemble.[*][1] Based on the theory that sensory and other information is represented in the brain by networks of neurons, it is thought that neurons can encode both digital and analog information.[*][2]

6.1 Overview

Neurons are remarkable among the cells of the body in their ability to propagate signals rapidly over large distances. They do this by generating characteristic electrical pulses called action potentials: voltage spikes that can travel down nerve fibers. Sensory neurons change their activities by firing sequences of action potentials in various temporal patterns, with the presence of external sensory stimuli, such as light, sound, taste, smell and touch. It is known that information about the stimulus is encoded in this pattern of action potentials and transmitted into and around the brain.

Although action potentials can vary somewhat in duration, amplitude and shape, they are typically treated as identical stereotyped events in neural coding studies. If the brief duration of an action potential (about 1ms) is ignored, an action potential sequence, or spike train, can be characterized simply by a series of all-or-none point events in time.[*][3] The lengths of interspike intervals (ISIs) between two successive spikes in a spike train often vary, apparently randomly.[*][4] The study of neural coding involves measuring and characterizing how stimulus attributes, such as light or sound intensity, or motor actions, such as the direction of an arm movement, are represented by neuron action potentials or spikes. In order to describe and analyze neuronal firing, statistical methods and methods of probability theory and stochastic point processes have been widely applied.

With the development of large-scale neural recording and decoding technologies, researchers have begun to crack the neural code and already provided the first glimpse into the real-time neural code as memory is formed and recalled in the hippocampus, a brain region known to be central for memory formation.[*][5][*][6][*][7] Neuroscientists have initiated several large-scale brain decoding projects.[*][8][*][9]

6.2 Encoding and decoding

The link between stimulus and response can be studied from two opposite points of view. Neural encoding refers to the map from stimulus to response. The main focus is to understand how neurons respond to a wide variety of stimuli, and to construct models that attempt to predict responses to other stimuli. Neural decoding refers to the reverse map, from response to stimulus, and the challenge is to reconstruct a stimulus, or certain aspects of that stimulus, from the spike sequences it evokes.

6.3 Coding schemes

A sequence, or 'train', of spikes may contain information based on different coding schemes. In motor neurons, for example, the strength at which an innervated muscle is flexed depends solely on the 'firing rate', the average number of spikes per unit time (a 'rate code'). At the other end, a complex 'temporal code' is based on the precise timing of single spikes. They may be locked to an external stimulus such as in the visual[*][10] and auditory system or be generated intrinsically by the neural circuitry.[*][11]

Whether neurons use rate coding or temporal coding is a topic of intense debate within the neuroscience community, even though there is no clear definition of what these terms mean. In one theory, termed “neuroelectrodynamics”, the following coding schemes are all considered to be epiphenomena, replaced instead by molecular changes reflecting the spatial distribution of electric fields within neurons as a result of the broad electromagnetic spectrum of action potentials, and manifested in information as spike directivity.[*][12][*][13][*][14][*][15][*][16]

6.3.1 Rate coding

The rate coding model of neuronal firing communication states that as the intensity of a stimulus increases, the frequency or rate of action potentials, or "spike firing", increases. Rate coding is sometimes called frequency coding.

Rate coding is a traditional coding scheme, assuming that most, if not all, information about the stimulus is contained in the firing rate of the neuron. Because the sequence of action potentials generated by a given stimulus varies from trial to trial, neuronal responses are typically treated statistically or probabilistically. They may be characterized by firing rates, rather than as specific spike sequences. In most sensory systems, the firing rate increases, generally non-linearly, with increasing stimulus intensity.*[17] Any information possibly encoded in the temporal structure of the spike train is ignored. Consequently, rate coding is inefficient but highly robust with respect to the ISI 'noise'.*[4]

During rate coding, precisely calculating firing rate is very important. In fact, the term "firing rate" has a few different definitions, which refer to different averaging procedures, such as an average over time or an average over several repetitions of experiment.

In rate coding, learning is based on activity-dependent synaptic weight modifications.

Rate coding was originally shown by ED Adrian and Y Zotterman in 1926.*[18] In this simple experiment different weights were hung from a muscle. As the weight of the stimulus increased, the number of spikes recorded from sensory nerves innervating the muscle also increased. From these original experiments, Adrian and Zotterman concluded that action potentials were unitary events, and that the frequency of events, and not individual event magnitude, was the basis for most interneuronal communication.

In the following decades, measurement of firing rates became a standard tool for describing the properties of all types of sensory or cortical neurons, partly due to the relative ease of measuring rates experimentally. However, this approach neglects all the information possibly contained in the exact timing of the spikes. During recent years, more and more experimental evidence has suggested that a straightforward firing rate concept based on temporal averaging may be too simplistic to describe brain activity.*[4]

Spike-count rate

The Spike-count rate, also referred to as temporal average, is obtained by counting the number of spikes that appear during a trial and dividing by the duration of trial. The length T of the time window is set by experimenter and depends on the type of neuron recorded from and the stimulus. In practice, to get sensible averages, several spikes should occur within the time window. Typical values are T = 100 ms or T = 500 ms, but the duration may also be longer or shorter.*[19]

The spike-count rate can be determined from a single trial, but at the expense of losing all temporal resolution about variations in neural response during the course of the trial. Temporal averaging can work well in cases where the stimulus is constant or slowly varying and does not require a fast reaction of the organism —and this is the situation usually encountered in experimental protocols. Real-world input, however, is hardly stationary, but often changing on a fast time scale. For example, even when viewing a static image, humans perform saccades, rapid changes of the direction of gaze. The image projected onto the retinal photoreceptors changes therefore every few hundred milliseconds.*[19]

Despite its shortcomings, the concept of a spike-count rate code is widely used not only in experiments, but also in models of neural networks. It has led to the idea that a neuron transforms information about a single input variable (the stimulus strength) into a single continuous output variable (the firing rate).

Time-dependent firing rate

The time-dependent firing rate is defined as the average number of spikes (averaged over trials) appearing during a short interval between times t and t+Δt, divided by the duration of the interval. It works for stationary as well as for time-dependent stimuli. To experimentally measure the time-dependent firing rate, the experimenter records from a neuron while stimulating with some input sequence. The same stimulation sequence is repeated several times and the neuronal response is reported in a Peri-Stimulus-Time Histogram (PSTH). The time t is measured with respect to the start of the stimulation sequence. The Δt must be large enough (typically in the range of one or a few milliseconds) so there are sufficient number of spikes within the interval to obtain a reliable estimate of the average. The number of occurrences of spikes $n_K(t;t+\Delta t)$ summed over all repetitions of the experiment divided by the number K of repetitions is a measure of the typical activity of the neuron between time t and t+Δt. A further division by the interval length Δt yields time-dependent firing rate r(t) of the neuron, which is equivalent to the spike density of PSTH.

For sufficiently small Δt, r(t)Δt is the average number of spikes occurring between times t and t+Δt over multiple trials. If Δt is small, there will never be more than one spike within the interval between t and t+Δt on any given trial. This means that r(t)Δt is also the fraction of trials on which a spike occurred between those times. Equivalently, r(t)Δt is the probability that a spike occurs during this time interval.

As an experimental procedure, the time-dependent firing

rate measure is a useful method to evaluate neuronal activity, in particular in the case of time-dependent stimuli. The obvious problem with this approach is that it can not be the coding scheme used by neurons in the brain. Neurons can not wait for the stimuli to repeatedly present in an exactly same manner before generating response.

Nevertheless, the experimental time-dependent firing rate measure can make sense, if there are large populations of independent neurons that receive the same stimulus. Instead of recording from a population of N neurons in a single run, it is experimentally easier to record from a single neuron and average over N repeated runs. Thus, the time-dependent firing rate coding relies on the implicit assumption that there are always populations of neurons.

6.3.2 Temporal coding

When precise spike timing or high-frequency firing-rate fluctuations are found to carry information, the neural code is often identified as a temporal code.*[20] A number of studies have found that the temporal resolution of the neural code is on a millisecond time scale, indicating that precise spike timing is a significant element in neural coding.*[2]*[21]

Neurons exhibit high-frequency fluctuations of firing-rates which could be noise or could carry information. Rate coding models suggest that these irregularities are noise, while temporal coding models suggest that they encode information. If the nervous system only used rate codes to convey information, a more consistent, regular firing rate would have been evolutionarily advantageous, and neurons would have utilized this code over other less robust options.*[22] Temporal coding supplies an alternate explanation for the "noise," suggesting that it actually encodes information and affects neural processing. To model this idea, binary symbols can be used to mark the spikes: 1 for a spike, 0 for no spike. Temporal coding allows the sequence 000111000111 to mean something different from 001100110011, even though the mean firing rate is the same for both sequences, at 6 spikes/10 ms.*[23] Until recently, scientists had put the most emphasis on rate encoding as an explanation for post-synaptic potential patterns. However, functions of the brain are more temporally precise than the use of only rate encoding seems to allow. In other words, essential information could be lost due to the inability of the rate code to capture all the available information of the spike train. In addition, responses are different enough between similar (but not identical) stimuli to suggest that the distinct patterns of spikes contain a higher volume of information than is possible to include in a rate code.*[24]

Temporal codes employ those features of the spiking activity that cannot be described by the firing rate. For example, time to first spike after the stimulus onset, characteristics based on the second and higher statistical moments of the ISI probability distribution, spike randomness, or precisely timed groups of spikes (temporal patterns) are candidates for temporal codes.*[25] As there is no absolute time reference in the nervous system, the information is carried either in terms of the relative timing of spikes in a population of neurons or with respect to an ongoing brain oscillation.*[2]*[4]

The temporal structure of a spike train or firing rate evoked by a stimulus is determined both by the dynamics of the stimulus and by the nature of the neural encoding process. Stimuli that change rapidly tend to generate precisely timed spikes and rapidly changing firing rates no matter what neural coding strategy is being used. Temporal coding refers to temporal precision in the response that does not arise solely from the dynamics of the stimulus, but that nevertheless relates to properties of the stimulus. The interplay between stimulus and encoding dynamics makes the identification of a temporal code difficult.

In temporal coding, learning can be explained by activity-dependent synaptic delay modifications.*[26] The modifications can themselves depend not only on spike rates (rate coding) but also on spike timing patterns (temporal coding), i.e., can be a special case of spike-timing-dependent plasticity.

The issue of temporal coding is distinct and independent from the issue of independent-spike coding. If each spike is independent of all the other spikes in the train, the temporal character of the neural code is determined by the behavior of time-dependent firing rate r(t). If r(t) varies slowly with time, the code is typically called a rate code, and if it varies rapidly, the code is called temporal.

Temporal coding in sensory systems

For very brief stimuli, a neuron's maximum firing rate may not be fast enough to produce more than a single spike. Due to the density of information about the abbreviated stimulus contained in this single spike, it would seem that the timing of the spike itself would have to convey more information than simply the average frequency of action potentials over a given period of time. This model is especially important for sound localization, which occurs within the brain on the order of milliseconds. The brain must obtain a large quantity of information based on a relatively short neural response. Additionally, if low firing rates on the order of ten spikes per second must be distinguished from arbitrarily close rate coding for different stimuli, then a neuron trying to discriminate these two stimuli may need to wait for a second or more to accumulate enough information. This is not consistent with numerous organisms which are able to discriminate between stimuli in the time frame of milliseconds, suggesting that a rate code is not the only model at work.*[23]

To account for the fast encoding of visual stimuli, it has been suggested that neurons of the retina encode visual information in the latency time between stimulus onset and

first action potential, also called latency to first spike.*[27] This type of temporal coding has been shown also in the auditory and somato-sensory system. The main drawback of such a coding scheme is its sensitivity to intrinsic neuronal fluctuations.*[28] In the primary visual cortex of macaques, the timing of the first spike relative to the start of the stimulus was found to provide more information than the interval between spikes. However, the interspike interval could be used to encode additional information, which is especially important when the spike rate reaches its limit, as in high-contrast situations. For this reason, temporal coding may play a part in coding defined edges rather than gradual transitions.*[29]

The mammalian gustatory system is useful for studying temporal coding because of its fairly distinct stimuli and the easily discernible responses of the organism.*[30] Temporally encoded information may help an organism discriminate between different tastants of the same category (sweet, bitter, sour, salty, umami) that elicit very similar responses in terms of spike count. The temporal component of the pattern elicited by each tastant may be used to determine its identity (e.g., the difference between two bitter tastants, such as quinine and denatonium). In this way, both rate coding and temporal coding may be used in the gustatory system – rate for basic tastant type, temporal for more specific differentiation.*[31] Research on mammalian gustatory system has shown that there is an abundance of information present in temporal patterns across populations of neurons, and this information is different from that which is determined by rate coding schemes. Groups of neurons may synchronize in response to a stimulus. In studies dealing with the front cortical portion of the brain in primates, precise patterns with short time scales only a few milliseconds in length were found across small populations of neurons which correlated with certain information processing behaviors. However, little information could be determined from the patterns; one possible theory is they represented the higher-order processing taking place in the brain.*[24]

As with the visual system, in mitral/tufted cells in the olfactory bulb of mice, first-spike latency relative to the start of a sniffing action seemed to encode much of the information about an odor. This strategy of using spike latency allows for rapid identification of and reaction to an odorant. In addition, some mitral/tufted cells have specific firing patterns for given odorants. This type of extra information could help in recognizing a certain odor, but is not completely necessary, as average spike count over the course of the animal's sniffing was also a good identifier.*[32] Along the same lines, experiments done with the olfactory system of rabbits showed distinct patterns which correlated with different subsets of odorants, and a similar result was obtained in experiments with the locust olfactory system.*[23]

Temporal coding applications

The specificity of temporal coding requires highly refined technology to measure informative, reliable, experimental data. Advances made in optogenetics allow neurologists to control spikes in individual neurons, offering electrical and spatial single-cell resolution. For example, blue light causes the light-gated ion channel channelrhodopsin to open, depolarizing the cell and producing a spike. When blue light is not sensed by the cell, the channel closes, and the neuron ceases to spike. The pattern of the spikes matches the pattern of the blue light stimuli. By inserting channelrhodopsin gene sequences into mouse DNA, researchers can control spikes and therefore certain behaviors of the mouse (e.g., making the mouse turn left).*[33] Researchers, through optogenetics, have the tools to effect different temporal codes in a neuron while maintaining the same mean firing rate, and thereby can test whether or not temporal coding occurs in specific neural circuits.*[34]

Optogenetic technology also has the potential to enable the correction of spike abnormalities at the root of several neurological and psychological disorders.*[34] If neurons do encode information in individual spike timing patterns, key signals could be missed by attempting to crack the code while looking only at mean firing rates.*[23] Understanding any temporally encoded aspects of the neural code and replicating these sequences in neurons could allow for greater control and treatment of neurological disorders such as depression, schizophrenia, and Parkinson's disease. Regulation of spike intervals in single cells more precisely controls brain activity than the addition of pharmacological agents intravenously.*[33]

Phase-of-firing code

Phase-of-firing code is a neural coding scheme that combines the spike count code with a time reference based on oscillations. This type of code takes into account a time label for each spike according to a time reference based on phase of local ongoing oscillations at low*[35] or high frequencies.*[36] A feature of this code is that neurons adhere to a preferred order of spiking, resulting in firing sequence.*[37]

It has been shown that neurons in some cortical sensory areas encode rich naturalistic stimuli in terms of their spike times relative to the phase of ongoing network fluctuations, rather than only in terms of their spike count.*[35]*[38] Oscillations reflect local field potential signals. It is often categorized as a temporal code although the time label used for spikes is coarse grained. That is, four discrete values for phase are enough to represent all the information content in this kind of code with respect to the phase of oscillations in low frequencies. Phase-of-firing code is loosely based on the phase precession phenomena observed in place cells of the hippocampus.

Phase code has been shown in visual cortex to involve also high-frequency oscillations.*[37] Within a cycle of gamma oscillation, each neuron has it own preferred relative firing time. As a result, an entire population of neurons generates a firing sequence that has a duration of up to about 15 ms.*[37]

6.3.3 Population coding

Population coding is a method to represent stimuli by using the joint activities of a number of neurons. In population coding, each neuron has a distribution of responses over some set of inputs, and the responses of many neurons may be combined to determine some value about the inputs.

From the theoretical point of view, population coding is one of a few mathematically well-formulated problems in neuroscience. It grasps the essential features of neural coding and yet, is simple enough for theoretic analysis.*[39] Experimental studies have revealed that this coding paradigm is widely used in the sensor and motor areas of the brain. For example, in the visual area medial temporal (MT), neurons are tuned to the moving direction.*[40] In response to an object moving in a particular direction, many neurons in MT fire, with a noise-corrupted and bell-shaped activity pattern across the population. The moving direction of the object is retrieved from the population activity, to be immune from the fluctuation existing in a single neuron's signal. In one classic example in the primary motor cortex, Apostolos Georgopoulos and colleagues trained monkeys to move a joystick towards a lit target.*[41]*[42] They found that a single neuron would fire for multiple target directions. However it would fire fastest for one direction and more slowly depending on how close the target was to the neuron's 'preferred' direction.

Kenneth Johnson originally derived that if each neuron represents movement in its preferred direction, and the vector sum of all neurons is calculated (each neuron has a firing rate and a preferred direction), the sum points in the direction of motion. In this manner, the population of neurons codes the signal for the motion. This particular population code is referred to as population vector coding. This particular study divided the field of motor physiologists between Evarts' "upper motor neuron" group, which followed the hypothesis that motor cortex neurons contributed to control of single muscles, and the Georgopoulos group studying the representation of movement directions in cortex.

Population coding has a number of advantages, including reduction of uncertainty due to neuronal variability and the ability to represent a number of different stimulus attributes simultaneously. Population coding is also much faster than rate coding and can reflect changes in the stimulus conditions nearly instantaneously.*[43] Individual neurons in such a population typically have different but overlapping selectivities, so that many neurons, but not necessarily all, respond to a given stimulus.

Typically an encoding function has a peak value such that activity of the neuron is greatest if the perceptual value is close to the peak value, and becomes reduced accordingly for values less close to the peak value.

It follows that the actual perceived value can be reconstructed from the overall pattern of activity in the set of neurons. The Johnson/Georgopoulos vector coding is an example of simple averaging. A more sophisticated mathematical technique for performing such a reconstruction is the method of maximum likelihood based on a multivariate distribution of the neuronal responses. These models can assume independence, second order correlations ,*[44] or even more detailed dependencies such as higher order maximum entropy models*[45] or copulas.*[46]

Correlation coding

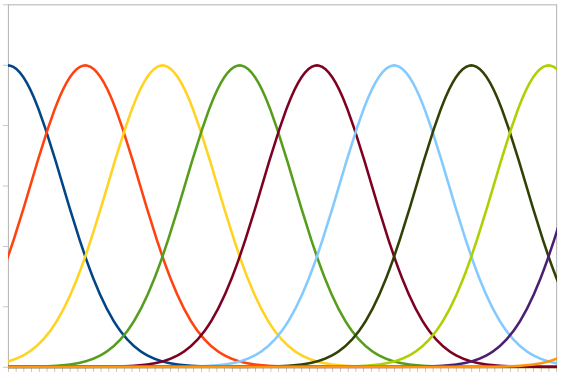
The correlation coding model of neuronal firing claims that correlations between action potentials, or "spikes" , within a spike train may carry additional information above and beyond the simple timing of the spikes. Early work suggested that correlation between spike trains can only reduce, and never increase, the total mutual information present in the two spike trains about a stimulus feature.*[47] However, this was later demonstrated to be incorrect. Correlation structure can increase information content if noise and signal correlations are of opposite sign.*[48] Correlations can also carry information not present in the average firing rate of two pairs of neurons. A good example of this exists in the pentobarbital-anesthetized marmoset auditory cortex, in which a pure tone causes an increase in the number of correlated spikes, but not an increase in the mean firing rate, of pairs of neurons.*[49]

Independent-spike coding

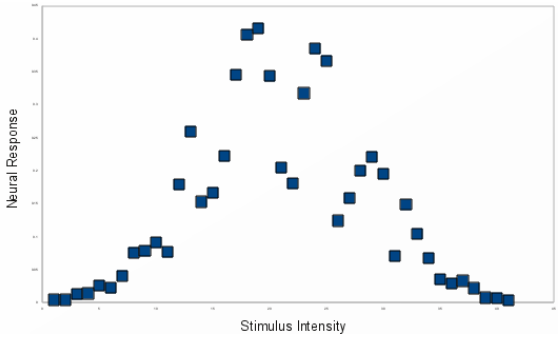
The independent-spike coding model of neuronal firing claims that each individual action potential, or "spike" , is independent of each other spike within the spike train.*[50]*[51]

Position coding

A typical population code involves neurons with a Gaussian tuning curve whose means vary linearly with the stimulus intensity, meaning that the neuron responds most strongly (in terms of spikes per second) to a stimulus near the mean. The actual intensity could be recovered as the stimulus level corresponding to the mean of the neuron with the greatest response. However, the noise inherent in neural responses means that a maximum likelihood estimation function is more accurate.



Plot of typical position coding



Neural responses are noisy and unreliable.

This type of code is used to encode continuous variables such as joint position, eye position, color, or sound frequency. Any individual neuron is too noisy to faithfully encode the variable using rate coding, but an entire population ensures greater fidelity and precision. For a population of unimodal tuning curves, i.e. with a single peak, the precision typically scales linearly with the number of neurons. Hence, for half the precision, half as many neurons are required. In contrast, when the tuning curves have multiple peaks, as in grid cells that represent space, the precision of the population can scale exponentially with the number of neurons. This greatly reduces the number of neurons required for the same precision.*[52]

6.3.4 Sparse coding

The sparse code is when each item is encoded by the strong activation of a relatively small set of neurons. For each item to be encoded, this is a different subset of all available neurons.

As a consequence, sparseness may be focused on temporal sparseness ("a relatively small number of time periods are active") or on the sparseness in an activated population of neurons. In this latter case, this may be defined in one time period as the number of activated neurons relative to the total number of neurons in the population. This seems to be a hallmark of neural computations since compared to traditional computers, information is massively distributed across neurons. A major result in neural coding from Olshausen et al.*[53] is that sparse coding of natural images produces wavelet-like oriented filters that resemble the receptive fields of simple cells in the visual cortex. The capacity of sparse codes may be increased by simultaneous use of temporal coding, as found in the locust olfactory system.*[54]

Given a potentially large set of input patterns, sparse coding algorithms (e.g. Sparse Autoencoder) attempt to automatically find a small number of representative patterns which, when combined in the right proportions, reproduce the original input patterns. The sparse coding for the input then consists of those representative patterns. For example, the very large set of English sentences can be encoded by a small number of symbols (i.e. letters, numbers, punctuation, and spaces) combined in a particular order for a particular sentence, and so a sparse coding for English would be those symbols.

Linear Generative Model

Most models of sparse coding are based on the linear generative model.*[55] In this model, the symbols are combined in a linear fashion to approximate the input.

More formally, given a k-dimensional set of real-numbered input vectors $\vec{\xi} \in \mathbb{R}^k$, the goal of sparse coding is to determine n k-dimensional basis vectors $\vec{b_1}, \ldots, \vec{b_n} \in \mathbb{R}^k$ along with a sparse n-dimensional vector of weights or coefficients $\vec{s} \in \mathbb{R}^n$ for each input vector, so that a linear combination of the basis vectors with proportions given by the coefficients results in a close approximation to the input vector: $\vec{\xi} \approx \sum_{j=1}^{n} s_j \vec{b_j}$.*[56]

The codings generated by algorithms implementing a linear generative model can be classified into codings with *soft sparseness* and those with *hard sparseness*.*[55] These refer to the distribution of basis vector coefficients for typical inputs. A coding with soft sparseness has a smooth Gaussian-like distribution, but peakier than Gaussian, with many zero values, some small absolute values, fewer larger absolute values, and very few very large absolute values. Thus, many of the basis vectors are active. Hard sparseness, on the other hand, indicates that there are many zero values, *no* or *hardly any* small absolute values, fewer larger absolute values, and very few very large absolute values, and thus few of the basis vectors are active. This is appealing from a metabolic perspective: less energy is used when fewer neurons are firing.*[55]

Another measure of coding is whether it is *critically complete* or *overcomplete*. If the number of basis vectors n is equal to the dimensionality k of the input set, the coding is said to be critically complete. In this case, smooth changes in the input vector result in abrupt changes in the coefficients, and the coding is not able to gracefully handle small scalings, small translations, or noise in the inputs. If, however, the number of basis vectors is larger

than the dimensionality of the input set, the coding is *overcomplete*. Overcomplete codings smoothly interpolate between input vectors and are robust under input noise.[*] [57] The human primary visual cortex is estimated to be overcomplete by a factor of 500, so that, for example, a 14 x 14 patch of input (a 196-dimensional space) is coded by roughly 100,000 neurons.[*] [55]

6.4 See also

- Models of neural computation
- Neural correlate
- Cognitive map
- Neural decoding
- Deep learning
- Autoencoder
- Vector quantization
- Binding problem
- Artificial neural network
- Grandmother cell
- Feature integration theory
- pooling
- Sparse distributed memory

6.5 References

[1] Brown EN, Kass RE, Mitra PP (May 2004). "Multiple neural spike train data analysis: state-of-the-art and future challenges". *Nat. Neurosci.* **7** (5): 456–61. doi:10.1038/nn1228. PMID 15114358.

[2] Thorpe, S.J. (1990). "Spike arrival times: A highly efficient coding scheme for neural networks" (PDF). In Eckmiller, R.; Hartmann, G.; Hauske, G. *Parallel processing in neural systems and computers* (PDF). North-Holland. pp. 91–94. ISBN 978-0-444-88390-2.

[3] Gerstner, Wulfram; Kistler, Werner M. (2002). *Spiking Neuron Models: Single Neurons, Populations, Plasticity*. Cambridge University Press. ISBN 978-0-521-89079-3.

[4] Stein RB, Gossen ER, Jones KE (May 2005). "Neuronal variability: noise or part of the signal?". *Nat. Rev. Neurosci.* **6** (5): 389–97. doi:10.1038/nrn1668. PMID 15861181.

[5] The Memory Code. http://www.scientificamerican.com/article/the-memory-code/

[6] Chen, G; Wang, LP; Tsien, JZ (2009). "Neural population-level memory traces in the mouse hippocampus". *PLoS One.* **4** (12): e8256. doi:10.1371/journal.pone.0008256. PMID 20016843.

[7] Zhang, H; Chen, G; Kuang, H; Tsien, JZ (Nov 2013). "Mapping and deciphering neural codes of NMDA receptor-dependent fear memory engrams in the hippocampus". *PLoS One.* **8** (11): e79454. doi:10.1371/journal.pone.0079454. PMID 24302990.

[8] Brain Decoding Project. http://braindecodingproject.org/

[9] The Simons Collaboration on the Global Brain. http://www.simonsfoundation.org/life-sciences/simons-collaboration-on-the-global-brain/

[10] Burcas G.T & Albright T.D. Gauging sensory representations in the brain. http://www.vcl.salk.edu/Publications/PDF/Buracas_Albright_1999_TINS.pdf

[11] Gerstner W, Kreiter AK, Markram H, Herz AV; Kreiter; Markram; Herz (November 1997). "Neural codes: firing rates and beyond". *Proc. Natl. Acad. Sci. U.S.A.* **94** (24): 12740–1. Bibcode:1997PNAS...9412740G. doi:10.1073/pnas.94.24.12740. PMC 34168. PMID 9398065.

[12] Aur D., Jog, MS., 2010 Neuroelectrodynamics: Understanding the brain language, IOS Press, 2010, doi:10.3233/978-1-60750-473-3-i

[13] Aur, D.; Connolly, C.I.; Jog, M.S. (2005). "Computing spike directivity with tetrodes". *J. Neurosci* **149** (1): 57–63. doi:10.1016/j.jneumeth.2005.05.006.

[14] Aur, D.; Jog, M.S. (2007). "Reading the Neural Code: What do Spikes Mean for Behavior?". *Nature Precedings.* doi:10.1038/npre.2007.61.1.

[15] Fraser, A.; Frey, A. H. (1968). "Electromagnetic emission at micron wavelengths from active nerves". *Biophysical Journal* **8** (6): 731–734. doi:10.1016/s0006-3495(68)86517-8.

[16] Aur, D (2012). "A comparative analysis of integrating visual information in local neuronal ensembles". *Journal of neuroscience methods* **207** (1): 23–30. doi:10.1016/j.jneumeth.2012.03.008. PMID 22480985.

[17] Kandel, E.; Schwartz, J.; Jessel, T.M. (1991). *Principles of Neural Science* (3rd ed.). Elsevier. ISBN 0444015620.

[18] Adrian ED & Zotterman Y. (1926). "The impulses produced by sensory nerve endings: Part II: The response of a single end organ." *J Physiol (Lond.)* **61**: 151–171.

[19] http://icwww.epfl.ch/~{}gerstner/SPNM/node7.html

[20] Dayan, Peter; Abbott, L. F. (2001). *Theoretical Neuroscience: Computational and Mathematical Modeling of Neural Systems*. Massachusetts Institute of Technology Press. ISBN 978-0-262-04199-7.

[21] Butts DA, Weng C, Jin J et al. (September 2007). "Temporal precision in the neural code and the timescales of natural vision". *Nature* **449** (7158): 92–5. Bibcode:2007Natur.449...92B. doi:10.1038/nature06105. PMID 17805296.

[22] J. Leo van Hemmen, TJ Sejnowski. 23 Problems in Systems Neuroscience. Oxford Univ. Press, 2006. p.143-158.

[23] Theunissen, F; Miller, JP (1995). "Temporal Encoding in Nervous Systems: A Rigorous Definition". *Journal of Computational Neuroscience* **2**: 149–162. doi:10.1007/bf00961885.

[24] Zador, Stevens, Charles, Anthony. "The enigma of the brain". *© Current Biology 1995, Vol 5 No 12*. Retrieved 4/08/12. Check date values in: |accessdate= (help)

[25] Kostal L, Lansky P, Rospars JP (November 2007). "Neuronal coding and spiking randomness". *Eur. J. Neurosci.* **26** (10): 2693–701. doi:10.1111/j.1460-9568.2007.05880.x. PMID 18001270.

[26] Geoffrois, E.; Edeline, J.M.; Vibert, J.F. (1994). "Learning by Delay Modifications". In Eeckman, Frank H. *Computation in Neurons and Neural Systems*. Springer. pp. 133–8. ISBN 978-0-7923-9465-5.

[27] Gollisch, T.; Meister, M. (22 February 2008). "Rapid Neural Coding in the Retina with Relative Spike Latencies". *Science* **319** (5866): 1108–1111. doi:10.1126/science.1149639.

[28] Wainrib, Gilles; Michèle, Thieullen; Khashayar, Pakdaman (7 April 2010). "Intrinsic variability of latency to first-spike". *Biological Cybernetics* **103** (1): 43–56. doi:10.1007/s00422-010-0384-8.

[29] Victor, Johnathan D (2005). "Spike train metrics". *Current Opinion in Neurobiology* **15** (5): 585–592. doi:10.1016/j.conb.2005.08.002.

[30] Hallock, Robert M.; Di Lorenzo, Patricia M. (2006). "Temporal coding in the gustatory system". *Neuroscience & Biobehavioral Reviews* **30** (8): 1145–1160. doi:10.1016/j.neubiorev.2006.07.005.

[31] Carleton, Alan; Accolla, Riccardo; Simon, Sidney A. (2010). "Coding in the mammalian gustatory system". *Trends in Neurosciences* **33** (7): 326–334. doi:10.1016/j.tins.2010.04.002.

[32] Wilson, Rachel I (2008). "Neural and behavioral mechanisms of olfactory perception". *Current Opinion in Neurobiology* **18** (4): 408–412. doi:10.1016/j.conb.2008.08.015.

[33] Karl Diesseroth, Lecture. "Personal Growth Series: Karl Diesseroth on Cracking the Neural Code." Google Tech Talks. November 21, 2008. http://www.youtube.com/watch?v=5SLdSbp6VjM

[34] Han X, Qian X, Stern P, Chuong AS, Boyden ES. "Informational lesions: optical perturbations of spike timing and neural synchrony via microbial opsin gene fusions." Cambridge, MA: MIT Media Lad, 2009.

[35] Montemurro MA, Rasch MJ, Murayama Y, Logothetis NK, Panzeri S (March 2008). "Phase-of-firing coding of natural visual stimuli in primary visual cortex". *Curr. Biol.* **18** (5): 375–80. doi:10.1016/j.cub.2008.02.023. PMID 18328702.

[36] Fries P, Nikolić D, Singer W (July 2007). "The gamma cycle". *Trends Neurosci.* **30** (7): 309–16. doi:10.1016/j.tins.2007.05.005. PMID 17555828.

[37] Havenith MN, Yu S, Biederlack J, Chen NH, Singer W, Nikolić D (June 2011). "Synchrony makes neurons fire in sequence, and stimulus properties determine who is ahead". *J. Neurosci.* **31** (23): 8570–84. doi:10.1523/JNEUROSCI.2817-10.2011. PMID 21653861.

[38] Spike arrival times: A highly efficient coding scheme for neural networks, SJ Thorpe - Parallel processing in neural systems, 1990

[39] Wu S, Amari S, Nakahara H (May 2002). "Population coding and decoding in a neural field: a computational study". *Neural Comput* **14** (5): 999–1026. doi:10.1162/089976602753633367. PMID 11972905.

[40] Maunsell JH, Van Essen DC (May 1983). "Functional properties of neurons in middle temporal visual area of the macaque monkey. I. Selectivity for stimulus direction, speed, and orientation". *J. Neurophysiol.* **49** (5): 1127–47. PMID 6864242.

[41] Intro to Sensory Motor Systems Ch. 38 page 766

[42] Science. 1986 Sep 26;233(4771):1416-9

[43] Hubel DH, Wiesel TN (October 1959). "Receptive fields of single neurones in the cat's striate cortex". *J. Physiol. (Lond.)* **148** (3): 574–91. PMC 1363130. PMID 14403679.

[44] Schneidman, E, Berry, MJ, Segev, R, Bialek, W (2006), *Weak Pairwise Correlations Imply Strongly Correlated Network States in a Neural Population* **440**, Nature 440, 1007-1012, doi:10.1038/nature04701

[45] Amari, SL (2001), *Information Geometry on Hierarchy of Probability Distributions*, IEEE Transactions on Information Theory 47, 1701-1711, CiteSeerX: 10.1.1.46.5226

[46] Onken, A, Grünewälder, S, Munk, MHJ, Obermayer, K (2009), *Analyzing Short-Term Noise Dependencies of Spike-Counts in Macaque Prefrontal Cortex Using Copulas and the Flashlight Transformation*, PLoS Comput Biol 5(11): e1000577, doi:10.1371/journal.pcbi.1000577

[47] Johnson, KO (Jun 1980). *J Neurophysiol* **43** (6): 1793–815. Missing or empty |title= (help)

[48] Panzeri, Schultz, Treves, Rolls, *Proc Biol Sci.* 1999 May 22;266(1423):1001-12.

[49] *Nature* **381** (6583): 610–3. Jun 1996. doi:10.1038/381610a0. Missing or empty |title= (help)

[50] Dayan P & Abbott LF. *Theoretical Neuroscience: Computational and Mathematical Modeling of Neural Systems*. Cambridge, Massachusetts: The MIT Press; 2001. ISBN 0-262-04199-5

[51] Rieke F, Warland D, de Ruyter van Steveninck R, Bialek W. *Spikes: Exploring the Neural Code*. Cambridge, Massachusetts: The MIT Press; 1999. ISBN 0-262-68108-0

[52] Mathis A, Herz AV, Stemmler MB; Herz; Stemmler (July 2012). "Resolution of nested neuronal representations can be exponential in the number of neurons". *Phys. Rev. Lett.* **109** (1): 018103. Bibcode:2012PhRvL.109a8103M. doi:10.1103/PhysRevLett.109.018103. PMID 23031134.

[53] Olshausen, Bruno A. "Emergence of simple-cell receptive field properties by learning a sparse code for natural images." Nature 381.6583 (1996): 607-609.

[54] Gupta, N; Stopfer, M (6 October 2014). "A temporal channel for information in sparse sensory coding.". *Current Biology* **24** (19): 2247–56. doi:10.1016/j.cub.2014.08.021. PMID 25264257.

[55] Rehn, Martin; Sommer, Friedrich T. (2007). "A network that uses few active neurones to code visual input predicts the diverse shapes of cortical receptive fields" (PDF). *Journal of Computational Neuroscience* **22**: 135–146. doi:10.1007/s10827-006-0003-9.

[56] Lee, Honglak; Battle, Alexis; Raina, Rajat; Ng, Andrew Y. (2006). "Efficient sparse coding algorithms" (PDF). *Advances in Neural Information Processing Systems*.

[57] Olshausen, Bruno A.; Field, David J. (1997). "Sparse Coding with an Overcomplete Basis Set: A Strategy Employed by V1?" (PDF). *Vision Research* **37** (23): 3311–3325. doi:10.1016/s0042-6989(97)00169-7.

6.6 Further reading

- Földiák P, Endres D, Sparse coding, Scholarpedia, 3(1):2984, 2008.
- Dayan P & Abbott LF. *Theoretical Neuroscience: Computational and Mathematical Modeling of Neural Systems*. Cambridge, Massachusetts: The MIT Press; 2001. ISBN 0-262-04199-5
- Rieke F, Warland D, de Ruyter van Steveninck R, Bialek W. *Spikes: Exploring the Neural Code*. Cambridge, Massachusetts: The MIT Press; 1999. ISBN 0-262-68108-0
- Olshausen, B. A.; Field, D. J. "Emergence of simple-cell receptive field properties by learning a sparse code for natural images". *Nature* **381** (6583): 607–9. doi:10.1038/381607a0.
- Tsien, JZ. et al. (2014). "On initial Brain Activity Mapping of episodic and semantic memory code in the hippocampus". *Neurobiology of Learning and Memory* **105**: 200–210. doi:10.1016/j.nlm.2013.06.019.

Chapter 7

Word embedding

Word embedding is the collective name for a set of language modeling and feature learning techniques in natural language processing where words from the vocabulary (and possibly phrases thereof) are mapped to vectors of real numbers in a low dimensional space, relative to the vocabulary size ("continuous space").

There are several methods for generating this mapping. They include neural networks,[*][1] dimensionality reduction on the word co-occurrence matrix,[*][2] and explicit representation in terms of the context in which words appear.[*][3]

Word and phrase embeddings, when used as the underlying input representation, have been shown to boost the performance in NLP tasks such as syntactic parsing[*][4] and sentiment analysis.[*][5]

7.1 See also

- Brown clustering

7.2 References

[1] Mikolov, Tomas; Sutskever, Ilya; Chen, Kai; Corrado, Greg; Dean, Jeffrey (2013). "Distributed Representations of Words and Phrases and their Compositionality". arXiv:1310.4546 [cs.CL].

[2] Lebret, Rémi; Collobert, Ronan (2013). "Word Emdeddings through Hellinger PCA". arXiv:1312.5542 [cs.CL].

[3] Levy, Omer; Goldberg, Yoav. "Linguistic Regularities in Sparse and Explicit Word Representations" (PDF). *Proceedings of the Eighteenth Conference on Computational Natural Language Learning, Baltimore, Maryland, USA, June. Association for Computational Linguistics. 2014*.

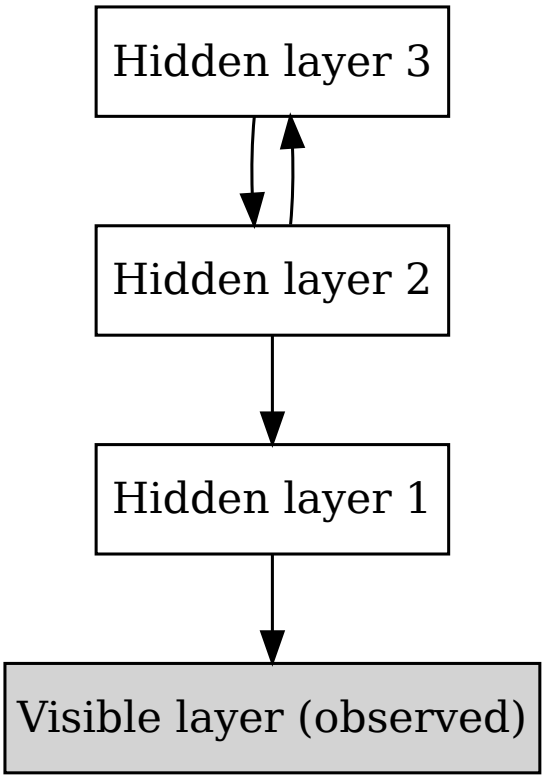
[4] Socher, Richard; Bauer, John; Manning, Christopher; Ng, Andrew. "Parsing with compositional vector grammars" (PDF). *Proceedings of the ACL conference. 2013*.

[5] Socher, Richard; Perelygin, Alex; Wu, Jean; Chuang, Jason; Manning, Chris; Ng, Andrew; Potts, Chris. "Recursive Deep Models for Semantic Compositionality Over a Sentiment Treebank" (PDF). *Conference on Empirical Methods in Natural Language Processing*.

Chapter 8

Deep belief network

In machine learning, a **deep belief network** (**DBN**) is a generative graphical model, or alternatively a type of deep neural network, composed of multiple layers of latent variables ("hidden units"), with connections between the layers but not between units within each layer.[*][1]



Schematic overview of a deep belief net. Arrows represent directed connections in the graphical model that the net represents.

When trained on a set of examples in an unsupervised way, a DBN can learn to probabilistically reconstruct its inputs. The layers then act as feature detectors on inputs.[*][1] After this learning step, a DBN can be further trained in a supervised way to perform classification.[*][2]

DBNs can be viewed as a composition of simple, unsupervised networks such as restricted Boltzmann machines (RBMs)[*][1] or autoencoders,[*][3] where each sub-network's hidden layer serves as the visible layer for the next. This also leads to a fast, layer-by-layer unsupervised training procedure, where contrastive divergence is applied to each sub-network in turn, starting from the "lowest" pair of layers (the lowest visible layer being a training set).

The observation, due to Geoffrey Hinton's student Yee-Whye Teh,[*][2] that DBNs can be trained greedily, one layer at a time, has been called a breakthrough in deep learning.[*][4][*]:6

8.1 Training algorithm

The training algorithm for DBNs proceeds as follows. Let X be a matrix of inputs, regarded as a set of feature vectors.[*][2]

1. Train a restricted Boltzmann machine on X to obtain its weight matrix, W. Use this as the weight matrix for between the lower two layers of the network.
2. Transform X by the RBM to produce new data X', either by sampling or by computing the mean activation of the hidden units.
3. Repeat this procedure with X ← X' for the next pair of layers, until the top two layers of the network are reached.

8.2 See also

- Bayesian network
- Deep learning

8.3 References

[1] Hinton, G. (2009). "Deep belief networks". *Scholarpedia* **4** (5): 5947. doi:10.4249/scholarpedia.5947.

[2] 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.

[3] Yoshua Bengio; Pascal Lamblin; Dan Popovici; Hugh Larochelle (2007). *Greedy Layer-Wise Training of Deep Networks* (PDF). NIPS.

[4] Bengio, Y. (2009). "Learning Deep Architectures for AI" (PDF). *Foundations and Trends in Machine Learning* **2**. doi:10.1561/2200000006.

8.4 External links

- "Deep Belief Networks". *Deep Learning Tutorials*.
- "Deep Belief Network Example". *Deeplearning4j Tutorials*.

Chapter 9

Convolutional neural network

For other uses, see CNN (disambiguation).

In machine learning, a **convolutional neural network** (or **CNN**) is a type of feed-forward artificial neural network where the individual neurons are tiled in such a way that they respond to overlapping regions in the visual field.*[1] Convolutional networks were inspired by biological processes*[2] and are variations of multilayer perceptrons which are designed to use minimal amounts of preprocessing.*[3] They are widely used models for image and video recognition.

9.1 Overview

When used for image recognition, convolutional neural networks (CNNs) consist of multiple layers of small neuron collections which look at small portions of the input image, called receptive fields. The results of these collections are then tiled so that they overlap to obtain a better representation of the original image; this is repeated for every such layer. Because of this, they are able to tolerate translation of the input image.*[4] Convolutional networks may include local or global pooling layers, which combine the outputs of neuron clusters.*[5]*[6] They also consist of various combinations of convolutional layers and fully connected layers, with pointwise nonlinearity applied at the end of or after each layer.*[7] It is inspired by biological process. To avoid the situation that there exist billions of parameters if all layers are fully connected, the idea of using a convolution operation on small regions, has been introduced. One major advantage of convolutional networks is the use of shared weight in convolutional layers, which means that the same filter (weights bank) is used for each pixel in the layer; this both reduces required memory size and improves performance.*[3]

Some Time delay neural networks also use a very similar architecture to convolutional neural networks, especially those for image recognition and/or classification tasks, since the "tiling" of the neuron outputs can easily be carried out in timed stages in a manner useful for analysis of images.*[8]

Compared to other image classification algorithms, convolutional neural networks use relatively little preprocessing. This means that the network is responsible for learning the filters that in traditional algorithms were hand-engineered. The lack of a dependence on prior-knowledge and the existence of difficult to design hand-engineered features is a major advantage for CNNs.

9.2 History

The design of convolutional neural networks follows the discovery of visual mechanisms in living organisms. In our brain, the visual cortex contains lots of cells. These cells are responsible for detecting light in small, overlapping sub-regions of the visual field, called receptive fields. These cells act as local filters over the input space. The more complex cells have larger receptive fields. A convolution operator is created to perform the same function by all of these cells.

Convolutional neural networks were introduced in a 1980 paper by Kunihiko Fukushima.*[7]*[9] In 1988 they were separately developed, with explicit parallel and trainable convolutions for temporal signals, by Toshiteru Homma, Les Atlas, and Robert J. Marks II.*[10] Their design was later improved in 1998 by Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner,*[11] generalized in 2003 by Sven Behnke,*[12] and simplified by Patrice Simard, David Steinkraus, and John C. Platt in the same year.*[13] The famous LeNet-5 network can classify digits successfully, which is applied to recognize checking numbers. However, given more complex problems the breadth and depth of the network will continue to increase which would become limited by computing resources. The approach used LeNet did not perform well with more complex problems.

With the rise of efficient GPU computing, it has become possible to train larger networks. In 2006 several publications described more efficient ways to train convolutional neural networks with more layers.*[14]*[15]*[16] In 2011, they were refined by Dan Ciresan et al. and were implemented on a GPU with impressive performance results.*[5] In 2012, Dan Ciresan et al. significantly improved upon the best performance in the literature for multiple image databases, including the MNIST

database, the NORB database, the HWDB1.0 dataset (Chinese characters), the CIFAR10 dataset (dataset of 60000 32x32 labeled RGB images),*[7] and the ImageNet dataset.*[17]

9.3 Details

9.3.1 Backpropagation

When doing propagation, the momentum and weight decay values are chosen to reduce oscillation during stochastic gradient descent. See Backpropagation for more.

9.3.2 Different types of layers

Convolutional layer

Unlike a hand-coded convolution kernel (Sobel, Prewitt, Roberts), in a convolutional neural net, the parameters of each convolution kernel are trained by the backpropagation algorithm. There are many convolution kernels in each layer, and each kernel is replicated over the entire image with the same parameters. The function of the convolution operators is to extract different features of the input. The capacity of a neural net varies, depending on the number of layers. The first convolution layers will obtain the low-level features, like edges, lines and corners. The more layers the network has, the higher-level features it will get.

ReLU layer

ReLU is the abbreviation of Rectified Linear Units. This is a layer of neurons that use the non-saturating activation function f(x)=max(0,x). It increases the nonlinear properties of the decision function and of the overall network without affecting the receptive fields of the convolution layer.

Other functions are used to increase nonlinearity. For example the saturating hyperbolic tangent f(x)=tanh(x), f(x)=|tanh(x)|, and the sigmoid function f(x)=(1+e^(-x))^(−1). Compared to tanh units, the advantage of ReLU is that the neural network trains several times faster.*[18]

Pooling layer

In order to reduce variance, pooling layers compute the max or average value of a particular feature over a region of the image. This will ensure that the same result will be obtained, even when image features have small translations. This is an important operation for object classification and detection.

Dropout "layer"

Since a fully connected layer occupies most of the parameters, it is prone to overfitting. The dropout method *[19] is introduced to prevent overfitting. That paper defines (the simplest form of) dropout as: "The only difference", from "Learning algorithms developed for Restricted_Boltzmann_machine such as Contrastive Divergence", "is that r", the number derived (usually by Sigmoid) from the incoming sum to the neural node from other nodes, "is first sampled and only the hidden units that are retained are used for training." and "dropout can be seen as multiplying by a Bernoulli_distribution random variable rb that takes the value 1/p with probability p and 0 otherwise." In other words, that simplest form of dropout is to take the chance and see if it happens or not, to observe if the neural node spikes/fires (instantly), instead of just remembering that chance happened.

Dropout also significantly improves the speed of training. This makes model combination practical, even for deep neural nets. Dropout is performed randomly. In the input layer, the probability of dropping a neuron is between 0.5 and 1, while in the hidden layers, a probability of 0.5 is used. The neurons that are dropped out, will not contribute to the forward pass and back propagation. This is equivalent to decreasing the number of neurons. This will create neural networks with different architectures, but all of those networks will share the same weights.

In Neural_coding#Sparse_coding bio neurons generally some react and some dont, at any moment of experience, as we are not perfect machines, not to say brains have the same chances and layer shapes used in simulated Dropout, but we learn anyways through it.

The biggest contribution of the dropout method is that, although it effectively generates 2^n neural nets, and as such, allows for model combination, at test time, only a single network needs to be tested. This is accomplished by performing the test with the un-thinned network, while multiplying the output weights of each neuron with the probability of that neuron being retained (i.e. not dropped out).

The "2^n neural nets" is accuracy but is pigeonholed by far less than 2^n bits having come in during the life of the neuralnet. We could the same way say Lambda_calculus takes exponential time if it werent for using base2 in memory instead of counting in Unary_numeral_system. So if each of 2^n abstract neural nets pushes at least 1 bit through those weights, they must have taken exponentially many turns sequentially since the bandwidth is not that wide. P_versus_NP_problem millenium prize remains unclaimed, even though there are great optimizations.

Loss layer

It can use different loss functions for different tasks. Softmax loss is used for predicting a single class of K mutually exclusive classes. Sigmoid cross-entropy loss is used for predicting K independent probability values in [0,1]. Euclidean loss is used for regressing to real-valued labels [-inf,inf]

9.4 Applications

9.4.1 Image recognition

Convolutional neural networks are often used in image recognition systems. They have achieved an error rate of 0.23 percent on the MNIST database, which as of February 2012 is the lowest achieved on the database.*[7] Another paper on using CNN for image classification reported that the learning process was "surprisingly fast"; in the same paper, the best published results at the time were achieved in the MNIST database and the NORB database.*[5]

When applied to facial recognition, they were able to contribute to a large decrease in error rate.*[20] In another paper, they were able to achieve a 97.6 percent recognition rate on "5,600 still images of more than 10 subjects" .*[2] CNNs have been used to assess video quality in an objective way after being manually trained; the resulting system had a very low root mean square error.*[8]

The ImageNet Large Scale Visual Recognition Challenge is a benchmark in object classification and detection, with millions of images and hundreds of object classes. In the ILSVRC 2014, which is large-scale visual recognition challenge, almost every highly ranked team used CNN as their basic framework. The winner GoogLeNet*[21] increased the mean average precision of object detection to 0.439329, and reduced classification error to 0.06656, the best result to date. Its network applied more than 30 layers. Performance of convolutional neural networks, on the ImageNet tests, is now close to that of humans.*[22] The best algorithms still struggle with objects that are small or thin, such as a small ant on a stem of a flower or a person holding a quill in their hand. They also have trouble with images that have been distorted with filters, an increasingly common phenomenon with modern digital cameras. By contrast, those kinds of images rarely trouble humans. Humans, however, tend to have trouble with other issues. For example, they are not good at classifying objects into fine-grained categories such as the particular breed of dog or species of bird, whereas convolutional neural networks handle this with ease.

In 2015 a many-layered CNN demonstrated the ability to spot faces from a wide range of angles, including upside down, even when partially occluded with competitive performance. The network trained on a database of 200,000 images that included faces at various angles and orientations and a further 20 million images without faces. They used batches of 128 images over 50,000 iterations.*[23]

9.4.2 Video analysis

Video is more complex than images since it has another temporal dimension. The common way is to fuse the features of different convolutional neural networks, which are responsible for spatial and temporal stream.*[24]*[25]

9.4.3 Natural Language Processing

Convolutional neural networks have also seen use in the field of natural language processing or NLP. Like the image classification problem, some NLP tasks can be formulated as assigning labels to words in a sentence. The neural network trained raw material fashion will extract the features of the sentences. Using some classifiers, it could predict new sentences.*[26]

9.4.4 Playing Go

Convolutional neural networks have been used in computer Go. In December 2014, Christopher Clark and Amos Storkey published a paper showing a convolutional network trained by supervised learning from a database of human professional games could outperform Gnu Go and win some games against Monte Carlo tree search Fuego 1.1 in a fraction of the time it took Fuego to play.*[27] Shortly after it was announced that a large 12-layer convolutional neural network had correctly predicted the professional move in 55% of positions, equalling the accuracy of a 6 dan human player. When the trained convolutional network was used directly to play games of Go, without any search, it beat the traditional search program GNU Go in 97% of games, and matched the performance of the Monte Carlo tree search program Fuego simulating ten thousand playouts (about a million positions) per move.*[28]

9.5 Fine-tuning

For many applications, only a small amount of training data is available. Convolutional neural networks usually require a large amount of training data in order to avoid over-fitting. A common technique is to train the network on a larger data set from a related domain. Once the network parameters have converged an additional training step is performed using the in-domain data to fine-tune the network weights. This allows convolutional networks to be successfully applied to problems with small training sets.

9.6 Common libraries

- Caffe: Caffe (replacement of Decaf) has been the most popular library for convolutional neural networks. It is created by the Berkeley Vision and Learning Center (BVLC). The advantages are that it has cleaner architecture and faster speed. It supports both CPU and GPU, easily switching between them. It is developed in C++, and has Python and MATLAB wrappers. In the developing of Caffe, protobuf is used to make researchers tune the parameters easily as well as adding or removing layers.
- Torch7 (www.torch.ch)
- OverFeat
- Cuda-convnet
- MatConvnet
- Theano: written in Python, using scientific python
- Deeplearning4j: Deep learning in Java and Scala on GPU-enabled Spark

9.7 See also

- Neocognitron
- Convolution
- Deep learning
- Time delay neural network

9.8 References

[1] "Convolutional Neural Networks (LeNet) - DeepLearning 0.1 documentation" . *DeepLearning 0.1*. LISA Lab. Retrieved 31 August 2013.

[2] Matusugu, Masakazu; Katsuhiko Mori; Yusuke Mitari; Yuji Kaneda (2003). "Subject independent facial expression recognition with robust face detection using a convolutional neural network" (PDF). *Neural Networks* **16** (5): 555–559. doi:10.1016/S0893-6080(03)00115-1. Retrieved 17 November 2013.

[3] LeCun, Yann. "LeNet-5, convolutional neural networks" . Retrieved 16 November 2013.

[4] Korekado, Keisuke; Morie, Takashi; Nomura, Osamu; Ando, Hiroshi; Nakano, Teppei; Matsugu, Masakazu; Iwata, Atsushi (2003). "A Convolutional Neural Network VLSI for Image Recognition Using Merged/Mixed Analog-Digital Architecture" . *Knowledge-Based Intelligent Information and Engineering Systems*: 169–176. CiteSeerX: 10.1.1.125.3812.

[5] Ciresan, Dan; Ueli Meier; Jonathan Masci; Luca M. Gambardella; Jurgen Schmidhuber (2011). "Flexible, High Performance Convolutional Neural Networks for Image Classification" (PDF). *Proceedings of the Twenty-Second international joint conference on Artificial Intelligence-Volume Volume Two* **2**: 1237–1242. Retrieved 17 November 2013.

[6] Krizhevsky, Alex. "ImageNet Classification with Deep Convolutional Neural Networks" (PDF). Retrieved 17 November 2013.

[7] Ciresan, Dan; Meier, Ueli; Schmidhuber, Jürgen (June 2012). "Multi-column deep neural networks for image classification" . *2012 IEEE Conference on Computer Vision and Pattern Recognition* (New York, NY: Institute of Electrical and Electronics Engineers (IEEE)): 3642–3649. arXiv:1202.2745v1. doi:10.1109/CVPR.2012.6248110. ISBN 9781467312264. OCLC 812295155. Retrieved 2013-12-09.

[8] Le Callet, Patrick; Christian Viard-Gaudin; Dominique Barba (2006). "A Convolutional Neural Network Approach for Objective Video Quality Assessment" (PDF). *IEEE Transactions on Neural Networks* **17** (5): 1316–1327. doi:10.1109/TNN.2006.879766. PMID 17001990. Retrieved 17 November 2013.

[9] Fukushima, Kunihiko (1980). "Neocognitron: A Self-organizing Neural Network Model for a Mechanism of Pattern Recognition Unaffected by Shift in Position" (PDF). *Biological Cybernetics* **36** (4): 193–202. doi:10.1007/BF00344251. PMID 7370364. Retrieved 16 November 2013.

[10] Homma, Toshiteru; Les Atlas; Robert Marks II (1988). "An Artificial Neural Network for Spatio-Temporal Bipolar Patters: Application to Phoneme Classification" (PDF). *Advances in Neural Information Processing Systems 1*: 31–40.

[11] LeCun, Yann; Léon Bottou; Yoshua Bengio; Patrick Haffner (1998). "Gradient-based learning applied to document recognition" (PDF). *Proceedings of the IEEE* **86** (11): 2278–2324. doi:10.1109/5.726791. Retrieved 16 November 2013.

[12] S. Behnke. Hierarchical Neural Networks for Image Interpretation, volume 2766 of Lecture Notes in Computer Science. Springer, 2003.

[13] Simard, Patrice, David Steinkraus, and John C. Platt. "Best Practices for Convolutional Neural Networks Applied to Visual Document Analysis." In ICDAR, vol. 3, pp. 958-962. 2003.

[14] Hinton, GE; Osindero, S; Teh, YW (Jul 2006). "A fast learning algorithm for deep belief nets.". *Neural computation* **18** (7): 1527–54. doi:10.1162/neco.2006.18.7.1527. PMID 16764513.

[15] Bengio, Yoshua; Lamblin, Pascal; Popovici, Dan; Larochelle, Hugo (2007). "Greedy Layer-Wise Training of Deep Networks" . *Advances in Neural Information Processing Systems*: 153–160.

[16] Ranzato, MarcAurelio; Poultney, Christopher; Chopra, Sumit; LeCun, Yann (2007). "Efficient Learning of Sparse Representations with an Energy-Based Model" (PDF). *Advances in Neural Information Processing Systems*.

[17] 10. Deng, Jia, et al. "Imagenet: A large-scale hierarchical image database."Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on. IEEE, 2009.

[18] Krizhevsky, A.; Sutskever, I.; Hinton, G. E. (2012). "Imagenet classification with deep convolutional neural networks" . *Advances in Neural Information Processing Systems 1*: 1097–1105.

[19] Srivastava, Nitish; C. Geoffrey Hinton; Alex Krizhevsky; Ilya Sutskever; Ruslan Salakhutdinov (2014). "Dropout: A Simple Way to Prevent Neural Networks from overfitting" (PDF). *Journal of Machine Learning Research* **15** (1): 1929–1958.

[20] Lawrence, Steve; C. Lee Giles; Ah Chung Tsoi; Andrew D. Back (1997). "Face Recognition: A Convolutional Neural Network Approach" . *Neural Networks, IEEE Transactions on* **8** (1): 98–113. doi:10.1109/72.554195. CiteSeerX: 10.1.1.92.5813.

[21] Szegedy, Christian, et al. "Going deeper with convolutions." arXiv preprint arXiv:1409.4842 (2014).

[22] O. Russakovsky et al., "ImageNet Large Scale Visual Recognition Challenge", 2014.

[23] "The Face Detection Algorithm Set To Revolutionize Image Search" . *Technology Review*. February 16, 2015. Retrieved February 2015.

[24] Karpathy, Andrej, et al. "Large-scale video classification with convolutional neural networks." IEEE Conference on Computer Vision and Pattern Recognition (CVPR). 2014.

[25] Simonyan, Karen, and Andrew Zisserman. "Two-stream convolutional networks for action recognition in videos." arXiv preprint arXiv:1406.2199 (2014).

[26] Collobert, Ronan, and Jason Weston. "A unified architecture for natural language processing: Deep neural networks with multitask learning."Proceedings of the 25th international conference on Machine learning. ACM, 2008.

[27] Clark, C., Storkey A.J. (2014), "Teaching Deep Convolutional Networks to play Go".

[28] Maddison C.J., Huang A., Sutskever I., Silver D. (2014), "Move evaluation in Go using deep convolutional neural networks".

9.9 External links

- A demonstration of a convolutional network created for character recognition
- Caffe
- Matlab toolbox
- MatConvnet
- Theano
- UFLDL Tutorial
- Deeplearning4j's Convolutional Nets

Chapter 10

Restricted Boltzmann machine

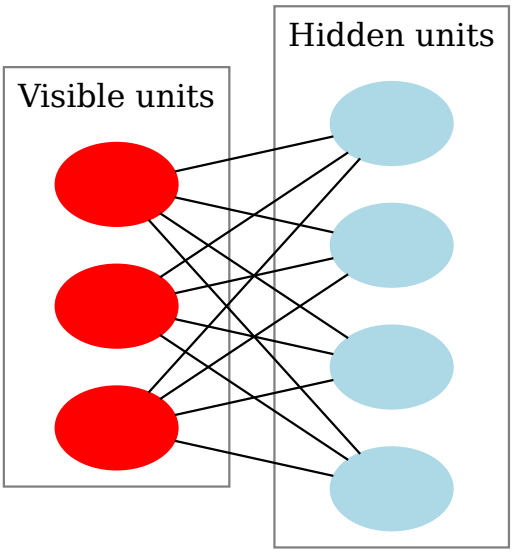


Diagram of a restricted Boltzmann machine with three visible units and four hidden units (no bias units).

A **restricted Boltzmann machine** (**RBM**) is a generative stochastic artificial neural network that can learn a probability distribution over its set of inputs. RBMs were initially invented under the name **Harmonium** by Paul Smolensky in 1986,[*][1] but only rose to prominence after Geoffrey Hinton and collaborators invented fast learning algorithms for them in the mid-2000s. RBMs have found applications in dimensionality reduction,[*][2] classification,[*][3] collaborative filtering,[*][4] feature learning[*][5] and topic modelling.[*][6] They can be trained in either supervised or unsupervised ways, depending on the task.

As their name implies, RBMs are a variant of Boltzmann machines, with the restriction that their neurons must form a bipartite graph: a pair of nodes from each of the two groups of units, commonly referred to as the "visible" and "hidden" units respectively, may have a symmetric connection between them, and there are no connections between nodes within a group. By contrast, "unrestricted" Boltzmann machines may have connections between hidden units. This restriction allows for more efficient training algorithms than are available for the general class of Boltzmann machines, in particular the gradient-based **contrastive divergence** algorithm.[*][7]

Restricted Boltzmann machines can also be used in deep learning networks. In particular, deep belief networks can be formed by "stacking" RBMs and optionally fine-tuning the resulting deep network with gradient descent and backpropagation.[*][8]

10.1 Structure

The standard type of RBM has binary-valued (Boolean/Bernoulli) hidden and visible units, and consists of a matrix of weights $W = (w_{i,j})$ (size $m \times n$) associated with the connection between hidden unit h_j and visible unit v_i , as well as bias weights (offsets) a_i for the visible units and b_j for the hidden units. Given these, the *energy* of a configuration (pair of boolean vectors) (v,h) is defined as

$$E(v,h) = -\sum_i a_i v_i - \sum_j b_j h_j - \sum_i \sum_j v_i w_{i,j} h_j$$

or, in matrix notation,

$$E(v,h) = -a^{\mathrm{T}} v - b^{\mathrm{T}} h - v^{\mathrm{T}} W h$$

This energy function is analogous to that of a Hopfield network. As in general Boltzmann machines, probability distributions over hidden and/or visible vectors are defined in terms of the energy function:[*][9]

$$P(v,h) = \frac{1}{Z} e^{-E(v,h)}$$

where Z is a partition function defined as the sum of $e^{-E(v,h)}$ over all possible configurations (in other words, just a normalizing constant to ensure the probability distribution sums to 1). Similarly, the (marginal) probability of a visible (input) vector of booleans is the sum over all possible hidden layer configurations:[*][9]

$$P(v) = \frac{1}{Z}\sum_{h} e^{-E(v,h)}$$

Since the RBM has the shape of a bipartite graph, with no intra-layer connections, the hidden unit activations are mutually independent given the visible unit activations and conversely, the visible unit activations are mutually independent given the hidden unit activations.*[7] That is, for m visible units and n hidden units, the conditional probability of a configuration of the visible units v, given a configuration of the hidden units h, is

$$P(v|h) = \prod_{i=1}^{m} P(v_i|h)$$

Conversely, the conditional probability of h given v is

$$P(h|v) = \prod_{j=1}^{n} P(h_j|v)$$

The individual activation probabilities are given by

$$P(h_j = 1|v) = \sigma\left(b_j + \textstyle\sum_{i=1}^{m} w_{i,j}v_i\right) \text{ and}$$
$$P(v_i = 1|h) = \sigma\left(a_i + \textstyle\sum_{j=1}^{n} w_{i,j}h_j\right)$$

where σ denotes the logistic sigmoid.

The visible units of RBM can be multinomial, although the hidden units are Bernoulli. In this case, the logistic function for visible units is replaced by the Softmax function

$$P(v_i^k = 1|h) = \frac{\exp(a_i^k + \Sigma_j W_{ij}^k h_j)}{\Sigma_{k=1}^{K}\exp(a_i^k + \Sigma_j W_{ij}^k h_j)}$$

where K is the number of discrete values that the visible values have. They are applied in Topic Modeling,*[6] and RecSys.*[4]

10.1.1 Relation to other models

Restricted Boltzmann machines are a special case of Boltzmann machines and Markov random fields.*[10]*[11] Their graphical model corresponds to that of factor analysis.*[12]

10.2 Training algorithm

Restricted Boltzmann machines are trained to maximize the product of probabilities assigned to some training set V (a matrix, each row of which is treated as a visible vector v),

$$\arg\max_{W} \prod_{v\in V} P(v)$$

or equivalently, to maximize the expected log probability of V :*[10]*[11]

$$\arg\max_{W} \mathbb{E}\left[\sum_{v\in V} \log P(v)\right]$$

The algorithm most often used to train RBMs, that is, to optimize the weight vector W , is the contrastive divergence (CD) algorithm due to Hinton, originally developed to train PoE (product of experts) models.*[13] *[14] The algorithm performs Gibbs sampling and is used inside a gradient descent procedure (similar to the way backpropagation is used inside such a procedure when training feedforward neural nets) to compute weight update.

The basic, single-step contrastive divergence (CD-1) procedure for a single sample can be summarized as follows:

1. Take a training sample v, compute the probabilities of the hidden units and sample a hidden activation vector h from this probability distribution.
2. Compute the outer product of v and h and call this the *positive gradient*.
3. From h, sample a reconstruction v' of the visible units, then resample the hidden activations h' from this. (Gibbs sampling step)
4. Compute the outer product of v' and h' and call this the *negative gradient*.
5. Let the weight update to $w_{i,j}$ be the positive gradient minus the negative gradient, times some learning rate: $\Delta w_{i,j} = \epsilon(vh^{\mathsf{T}} - v'h'^{\mathsf{T}})$.

The update rule for the biases a, b is defined analogously.

A Practical Guide to Training RBMs written by Hinton can be found in his homepage.*[9]

A restricted/layered Boltzmann machine (RBM) has either bit or scalar node values, an array for each layer, and between those are scalar values potentially for every pair of nodes one from each layer and an adjacent layer. It is run and trained using "weighted coin flips" of a chance calculated at each individual node. Those chances are the logistic sigmoid of the sum of scalar weights of whichever pairs of nodes are on at the time, divided by temperature which decreases in each round of Simulated annealing as potentially all the data is trained in again. If either node in a pair is off, that weight is not counted. To run it, you go up and down the layers, updating the chances

and weighted coin flips, until it converges to the coins in lowest layer (visible nodes) staying mostly a certain way. To train it, it is the same shape as running it except you observe the weights of the pairs that are on, the first time up you add the learning rate between those pairs, then go back down and up again and that time subtract the learning rate. As Geoffrey Hinton explained it, the first time up is to learn the data, and the second time up is to unlearn whatever its earlier reaction was to the data.

10.3 See also

- Autoencoder
- Deep learning
- Helmholtz machine
- Hopfield network

10.4 References

[1] Smolensky, Paul (1986). "Chapter 6: Information Processing in Dynamical Systems: Foundations of Harmony Theory" (PDF). In Rumelhart, David E.; McLelland, James L. *Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Volume 1: Foundations*. MIT Press. pp. 194–281. ISBN 0-262-68053-X.

[2] Hinton, G. E.; Salakhutdinov, R. R. (2006). "Reducing the Dimensionality of Data with Neural Networks" (PDF). *Science* **313** (5786): 504–507. doi:10.1126/science.1127647. PMID 16873662.

[3] Larochelle, H.; Bengio, Y. (2008). *Classification using discriminative restricted Boltzmann machines* (PDF). Proceedings of the 25th international conference on Machine learning - ICML '08. p. 536. doi:10.1145/1390156.1390224. ISBN 9781605582054.

[4] Salakhutdinov, R.; Mnih, A.; Hinton, G. (2007). *Restricted Boltzmann machines for collaborative filtering*. Proceedings of the 24th international conference on Machine learning - ICML '07. p. 791. doi:10.1145/1273496.1273596. ISBN 9781595937933.

[5] Coates, Adam; Lee, Honglak; Ng, Andrew Y. (2011). *An analysis of single-layer networks in unsupervised feature learning* (PDF). International Conference on Artificial Intelligence and Statistics (AISTATS).

[6] Ruslan Salakhutdinov and Geoffrey Hinton (2010). Replicated softmax: an undirected topic model. *Neural Information Processing Systems* **23**.

[7] Miguel Á. Carreira-Perpiñán and Geoffrey Hinton (2005). On contrastive divergence learning. *Artificial Intelligence and Statistics*.

[8] Hinton, G. (2009). "Deep belief networks". *Scholarpedia* **4** (5): 5947. doi:10.4249/scholarpedia.5947.

[9] Geoffrey Hinton (2010). *A Practical Guide to Training Restricted Boltzmann Machines*. UTML TR 2010–003, University of Toronto.

[10] Sutskever, Ilya; Tieleman, Tijmen (2010). "On the convergence properties of contrastive divergence" (PDF). *Proc. 13th Int'l Conf. on AI and Statistics (AISTATS)*.

[11] Asja Fischer and Christian Igel. Training Restricted Boltzmann Machines: An Introduction. Pattern Recognition 47, pp. 25-39, 2014

[12] María Angélica Cueto; Jason Morton; Bernd Sturmfels (2010). "Geometry of the restricted Boltzmann machine" (PDF). *Algebraic Methods in Statistics and Probability* (American Mathematical Society) **516**. arXiv:0908.4425.

[13] Geoffrey Hinton (1999). Products of Experts. *ICANN 1999*.

[14] Hinton, G. E. (2002). "Training Products of Experts by Minimizing Contrastive Divergence" (PDF). *Neural Computation* **14** (8): 1771–1800. doi:10.1162/089976602760128018. PMID 12180402.

10.5 External links

- Introduction to Restricted Boltzmann Machines. Edwin Chen's blog, July 18, 2011.
- Understanding RBMs. Deeplearning4j Documentation, December 29, 2014.

Chapter 11

Recurrent neural network

Not to be confused with Recursive neural network.

A **recurrent neural network** (**RNN**) is a class of artificial neural network where connections between units form a directed cycle. This creates an internal state of the network which allows it to exhibit dynamic temporal behavior. Unlike feedforward neural networks, RNNs can use their internal memory to process arbitrary sequences of inputs. This makes them applicable to tasks such as unsegmented connected handwriting recognition, where they have achieved the best known results.*[1]

11.1 Architectures

11.1.1 Fully recurrent network

This is the basic architecture developed in the 1980s: a network of neuron-like units, each with a directed connection to every other unit. Each unit has a time-varying real-valued activation. Each connection has a modifiable real-valued weight. Some of the nodes are called input nodes, some output nodes, the rest hidden nodes. Most architectures below are special cases.

For supervised learning in discrete time settings, training sequences of real-valued input vectors become sequences of activations of the input nodes, one input vector at a time. At any given time step, each non-input unit computes its current activation as a nonlinear function of the weighted sum of the activations of all units from which it receives connections. There may be teacher-given target activations for some of the 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. For each sequence, its error is 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. Algorithms for minimizing this error are mentioned in the section on training algorithms below.

In reinforcement learning settings, there is no teacher providing target signals for the RNN, instead a fitness function or reward function is occasionally used to evaluate the RNN's performance, which is influencing its input stream through output units connected to actuators affecting the environment. Again, compare the section on training algorithms below.

11.1.2 Hopfield network

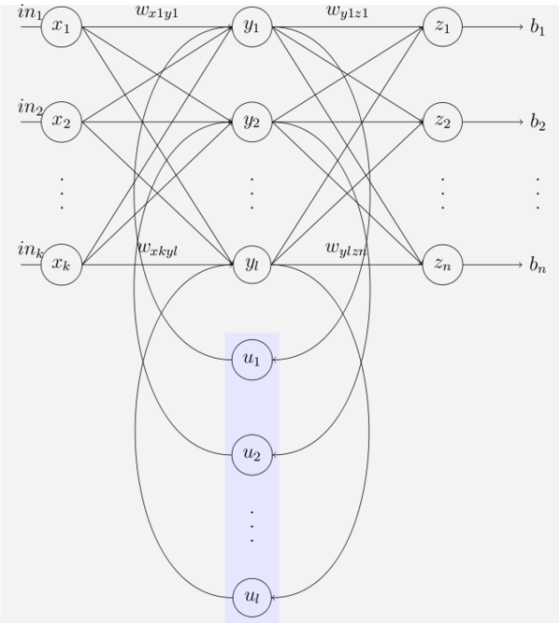
The Hopfield network is of historic interest although it is not a general RNN, as it is not designed to process sequences of patterns. Instead it requires stationary inputs. It is a RNN in which all connections are symmetric. Invented by John Hopfield in 1982, it guarantees that its dynamics will converge. If the connections are trained using Hebbian learning then the Hopfield network can perform as robust content-addressable memory, resistant to connection alteration.

A variation on the Hopfield network is the bidirectional associative memory (BAM). The BAM has two layers, either of which can be driven as an input, to recall an association and produce an output on the other layer.*[2]

11.1.3 Elman networks and Jordan networks

The following special case of the basic architecture above was employed by Jeff Elman. A three-layer network is used (arranged vertically as x, y, and z in the illustration), with the addition of a set of "context units" (u in the illustration). There are connections from the middle (hidden) layer to these context units fixed with a weight of one.*[3] At each time step, the input is propagated in a standard feed-forward fashion, and then a learning rule is applied. The fixed back connections result in the context units always maintaining a copy of the previous values of the hidden 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, due to Michael I. Jordan, are similar to Elman networks. The context units are however fed from



The Elman SRN

the output layer instead of the hidden layer. The context units in a Jordan network are also referred to as the state layer, and have a recurrent connection to themselves with no other nodes on this connection.*[3] Elman and Jordan networks are also known as "simple recurrent networks" (SRN).

11.1.4 Echo state network

The echo state network (ESN) is a recurrent neural network with a sparsely connected random hidden layer. The weights of output neurons are the only part of the network that can change and be trained. ESN are good at reproducing certain time series.*[4] A variant for spiking neurons is known as Liquid state machines.*[5]

11.1.5 Long short term memory network

The Long short term memory (LSTM) network, developed by Hochreiter & Schmidhuber in 1997,*[6] is an artificial neural net structure that unlike traditional RNNs doesn't have the vanishing gradient problem (compare the section on training algorithms below). It works even when there are long delays, and it can handle signals that have a mix of low and high frequency components. LSTM RNN outperformed other methods in numerous applications such as language learning*[7] and connected handwriting recognition.*[8]

11.1.6 Bi-directional RNN

Invented by Schuster & Paliwal in 1997,*[9] bi-directional RNN or BRNN use a finite sequence to predict or label each element of the sequence based on both the past and the future context of the element. This is done by adding the outputs of two RNN, 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 proved to be especially useful when combined with LSTM RNN.*[10]

11.1.7 Continuous-time RNN

A continuous time recurrent neural network (CTRNN) is a dynamical systems model of biological neural networks. A CTRNN uses a system of ordinary differential equations to model the effects on a neuron of the incoming spike train. CTRNNs are more computationally efficient than directly simulating every spike in a network as they do not model neural activations at this level of detail .

For a neuron i in the network with action potential y_i the rate of change of activation is given by:

$$\tau_i \dot{y}_i = -y_i + \sigma(\sum_{j=1}^{n} w_{ji} y_j - \Theta_j) + I_i(t)$$

Where:

- τ_i : Time constant of postsynaptic node
- y_i : Activation of postsynaptic node
- $\dot{y}_i$: Rate of change of activation of postsynaptic node
- w_{ji} : Weight of connection from pre to postsynaptic node
- $\sigma(x)$: Sigmoid of x e.g. $\sigma(x) = 1/(1 + e^{-x})$.
- y_j : Activation of presynaptic node
- Θ_j : Bias of presynaptic node
- $I_i(t)$: Input (if any) to node

CTRNNs have frequently been applied in the field of evolutionary robotics, where they have been used to address, for example, vision,*[11] co-operation*[12] and minimally cognitive behaviour.*[13]

11.1.8 Hierarchical RNN

There are many instances of hierarchical RNN whose elements are connected in various ways to decompose hierarchical behavior into useful subprograms.*[14]*[15]

11.1.9 Recurrent multilayer perceptron

Generally, a Recurrent Multi-Layer Perceptron (RMLP) consists of a series of cascaded subnetworks, each of which consists of multiple layers of nodes. Each of these subnetworks is entirely feed-forward except for the last layer, which can have feedback connections among itself. Each of these subnets is connected only by feed forward connections.*[16]

11.1.10 Second Order Recurrent Neural Network

Second order RNNs use higher order weights w_{ijk} instead of the standard w_{ij} weights, and inputs and states can be a product. This allows a direct mapping to a finite state machine both in training and in representation*[17]*[18] Long short term memory is an example of this.

11.1.11 Multiple Timescales Recurrent Neural Network (MTRNN) Model

MTRNN considered as a possible neural-based computational model that imitates to some extent, the brain activities.*[19] It has ability to simulate the functional hierarchy of the brain through self-organization that is not only depended on the spatial connection among neurons, but also on distinct types of neuron activities, each with distinct time properties. With such varied neuron activities, continuous sequences of any set of behavior are segmented into reusable primitives, which in turn are flexibly integrated into diverse sequential behaviors. The biological approval of such a type of hierarchy has been discussed on the memory-prediction theory of brain function by Jeff Hawkins in his book *On Intelligence*.

11.1.12 Pollack's sequential cascaded networks

11.1.13 Neural Turing Machines

NTMs are method of extending the capabilities of 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.*[20]

11.1.14 Bidirectional Associative Memory (BAM)

First introduced by Kosko,*[21] BAM neural networks store 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.*[22]

11.2 Training

11.2.1 Gradient descent

To minimize total error, gradient descent can be used to change 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 Paul Werbos, Ronald J. Williams, Tony Robinson, Jürgen Schmidhuber, Sepp Hochreiter, Barak Pearlmutter, and others.

The standard method is called "backpropagation through time" or BPTT, and is a generalization of back-propagation for feed-forward networks,*[23]*[24] and like that method, is an instance of Automatic differentiation in the reverse accumulation mode or Pontryagin's minimum principle. A more computationally expensive online variant is called "Real-Time Recurrent Learning" or RTRL,*[25]*[26] 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*.*[27]*[28]

There also is an online hybrid between BPTT and RTRL with intermediate complexity,*[29]*[30] and there are variants for continuous time.*[31] 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.*[32] *[33] The Long short term memory architecture together with a BPTT/RTRL hybrid learning method was introduced in an attempt to overcome these problems.*[6]

11.2.2 Hessian Free Optimisation

Successful training on complex tasks has been achieved by employing Hessian Free Optimisation. The speedup compared with previous training methods now makes RNN applications feasible.*[34]

11.2.3 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.* [35]* [36]* [37]

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, henceforth; the whole network is represented as a single chromosome. The fitness function is evaluated as follows: 1) each weight encoded in the chromosome is assigned to the respective weight link of the network ; 2) the training set of examples is then presented to the network which propagates the input signals forward ; 3) the mean-squared-error is returned to the fitness function ; 4) this function will then drive the genetic selection process.

There are many chromosomes that make up the population; therefore, many different neural networks are evolved until a stopping criterion is satisfied. A common stopping scheme is: 1) when the neural network has learnt a certain percentage of the training data or 2) when the minimum value of the mean-squared-error is satisfied or 3) 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 neural network during training. Therefore, the goal of the genetic algorithm is to maximize the fitness function, hence, reduce 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.

11.3 Related fields and models

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

Recurrent neural networks 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, recurrent neural networks can appear as nonlinear versions of finite impulse response and infinite impulse response filters and also as a nonlinear autoregressive exogenous model (NARX)* [38]

11.4 Issues with recurrent neural networks

Most RNNs have had scaling issues. In particular, RNNs cannot be easily trained for large numbers of neuron units nor for large numbers of inputs units . Successful training has been mostly in time series problems with few inputs and in chemical process control.

11.5 References

[1] A. Graves, M. Liwicki, S. Fernandez, R. Bertolami, H. Bunke, J. Schmidhuber. A Novel Connectionist System for Improved Unconstrained Handwriting Recognition. IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 31, no. 5, 2009.

[2] Rául Rojas (1996). *Neural networks: a systematic introduction*. Springer. p. 336. ISBN 978-3-540-60505-8.

[3] Cruse, Holk; *Neural Networks as Cybernetic Systems*, 2nd and revised edition

[4] H. Jaeger. Harnessing nonlinearity: Predicting chaotic systems and saving energy in wireless communication. Science, 304:78–80, 2004.

[5] W. Maass, T. Natschläger, and H. Markram. A fresh look at real-time computation in generic recurrent neural circuits. Technical report, Institute for Theoretical Computer Science, TU Graz, 2002.

[6] Hochreiter, Sepp; and Schmidhuber, Jürgen; *Long Short-Term Memory*, Neural Computation, 9(8):1735–1780, 1997

[7] Gers, Felix A.; and Schmidhuber, Jürgen; *LSTM Recurrent Networks Learn Simple Context Free and Context Sensitive Languages*, IEEE Transactions on Neural Networks, 12(6):1333–1340, 2001

[8] 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

[9] Bidirectional recurrent neural networks. IEEE Transactions on Signal Processing, 45:2673–81, November 1997.

[10] A. Graves and J. Schmidhuber. Framewise phoneme classification with bidirectional LSTM and other neural network architectures. Neural Networks, 18:602–610, 2005.

[11] Harvey, Inman; Husbands, P. and Cliff, D. (1994). "Seeing the light: Artificial evolution, real vision" . *Proceedings of the third international conference on Simulation of adaptive behavior: from animals to animats 3*: 392–401.

[12] Quinn, Matthew (2001). "Evolving communication without dedicated communication channels" . *Advances in Artificial Life*. Lecture Notes in Computer Science **2159**: 357–366. doi:10.1007/3-540-44811-X_38. ISBN 978-3-540-42567-0.

[13] Beer, R.D. (1997). "The dynamics of adaptive behavior: A research program" . *Robotics and Autonomous Systems* **20** (2–4): 257–289. doi:10.1016/S0921-8890(96)00063-2.

[14] J. Schmidhuber. Learning complex, extended sequences using the principle of history compression. Neural Computation, 4(2):234-242, 1992

[15] R.W. Paine, J. Tani, "How hierarchical control self-organizes in artificial adaptive systems," Adaptive Behavior, 13(3), 211-225, 2005.

[16] "CiteSeerX —Recurrent Multilayer Perceptrons for Identification and Control: The Road to Applications" . Citeseerx.ist.psu.edu. Retrieved 2014-01-03.

[17] C.L. Giles, C.B. Miller, D. Chen, H.H. Chen, G.Z. Sun, Y.C. Lee, "Learning and Extracting Finite State Automata with Second-Order Recurrent Neural Networks," Neural Computation, 4(3), p. 393, 1992.

[18] C.W. Omlin, C.L. Giles, "Constructing Deterministic Finite-State Automata in Recurrent Neural Networks," Journal of the ACM, 45(6), 937-972, 1996.

[19] Y. Yamashita, J. Tani, "Emergence of functional hierarchy in a multiple timescale neural network model: a humanoid robot experiment," PLoS Computational Biology, 4(11), e1000220, 211-225, 2008. http://journals.plos.org/ploscompbiol/article?id=10.1371/journal.pcbi.1000220

[20] http://arxiv.org/pdf/1410.5401v2.pdf

[21] Kosko, B. (1988). "Bidirectional associative memories" . *IEEE Transactions on Systems, Man, and Cybernetics* **18** (1): 49–60. doi:10.1109/21.87054.

[22] Rakkiyappan, R.; Chandrasekar, A.; Lakshmanan, S.; Park, Ju H. (2 January 2015). "Exponential stability for markovian jumping stochastic BAM neural networks with mode-dependent probabilistic time-varying delays and impulse control" . *Complexity* **20** (3): 39–65. doi:10.1002/cplx.21503.

[23] P. J. Werbos. Generalization of backpropagation with application to a recurrent gas market model. Neural Networks, 1, 1988.

[24] David E. Rumelhart; Geoffrey E. Hinton; Ronald J. Williams. Learning Internal Representations by Error Propagation.

[25] A. J. Robinson and F. Fallside. The utility driven dynamic error propagation network. Technical Report CUED/F-INFENG/TR.1, Cambridge University Engineering Department, 1987.

[26] R. J. Williams and D. Zipser. Gradient-based learning algorithms for recurrent networks and their computational complexity. In Back-propagation: Theory, Architectures and Applications. Hillsdale, NJ: Erlbaum, 1994.

[27] J. Schmidhuber. A local learning algorithm for dynamic feedforward and recurrent networks. Connection Science, 1(4):403–412, 1989.

[28] Neural and Adaptive Systems: Fundamentals through Simulation. J.C. Principe, N.R. Euliano, W.C. Lefebvre

[29] J. Schmidhuber. A fixed size storage O(n3) time complexity learning algorithm for fully recurrent continually running networks. Neural Computation, 4(2):243–248, 1992.

[30] R. J. Williams. Complexity of exact gradient computation algorithms for recurrent neural networks. Technical Report Technical Report NU-CCS-89-27, Boston: Northeastern University, College of Computer Science, 1989.

[31] B. A. Pearlmutter. Learning state space trajectories in recurrent neural networks. Neural Computation, 1(2):263–269, 1989.

[32] S. Hochreiter. Untersuchungen zu dynamischen neuronalen Netzen. Diploma thesis, Institut f. Informatik, Technische Univ. Munich, 1991.

[33] S. Hochreiter, Y. Bengio, P. Frasconi, and J. Schmidhuber. Gradient flow in recurrent nets: the difficulty of learning long-term dependencies. In S. C. Kremer and J. F. Kolen, editors, A Field Guide to Dynamical Recurrent Neural Networks. IEEE Press, 2001.

[34] Martens, James, and Ilya Sutskever. "Training deep and recurrent networks with hessian-free optimization." In Neural Networks: Tricks of the Trade, pp. 479-535. Springer Berlin Heidelberg, 2012.

[35] F. J. Gomez and R. Miikkulainen. Solving non-Markovian control tasks with neuroevolution. Proc. IJCAI 99, Denver, CO, 1999. Morgan Kaufmann.

[36] Applying Genetic Algorithms to Recurrent Neural Networks for Learning Network Parameters and Architecture. O. Syed, Y. Takefuji

[37] F. Gomez, J. Schmidhuber, R. Miikkulainen. Accelerated Neural Evolution through Cooperatively Coevolved Synapses. Journal of Machine Learning Research (JMLR), 9:937-965, 2008.

[38] Hava T. Siegelmann, Bill G. Horne, C. Lee Giles, "Computational capabilities of recurrent NARX neural networks," IEEE Transactions on Systems, Man, and Cybernetics, Part B 27(2): 208-215 (1997).

- Mandic, D. & Chambers, J. (2001). *Recurrent Neural Networks for Prediction: Learning Algorithms, Architectures and Stability*. Wiley. ISBN 0-471-49517-4.

- Elman, J.L. (1990). "Finding Structure in Time". *Cognitive Science* **14** (2): 179–211. doi:10.1016/0364-0213(90)90002-E.

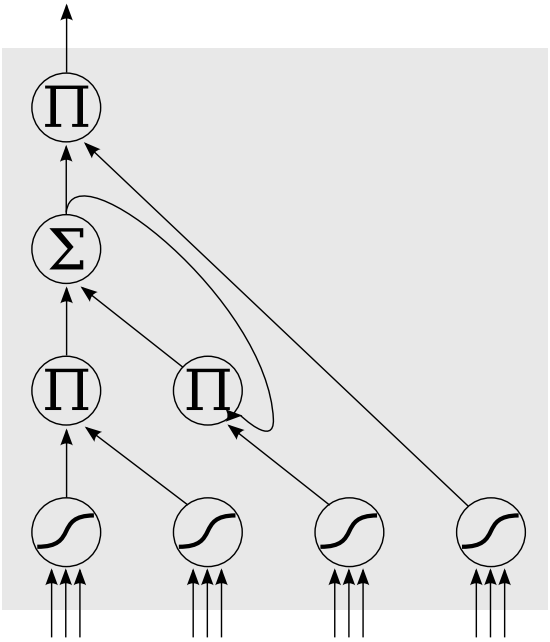
11.6 External links

- RNNSharp CRFs based on recurrent neural networks (C#, .NET)
- Recurrent Neural Networks with over 60 RNN papers by Jürgen Schmidhuber's group at IDSIA
- Elman Neural Network implementation for WEKA

Chapter 12

Long short term memory

Long short-term memory (**LSTM**) is a recurrent neural network (RNN) architecture (an artificial neural network) published[*][1] in 1997 by Sepp Hochreiter and Jürgen Schmidhuber. Like most RNNs, an LSTM network is universal in the sense that given enough network units it can compute anything a conventional computer can compute, provided it has the proper weight matrix, which may be viewed as its program. Unlike traditional RNNs, an LSTM network is well-suited to learn from experience to classify, process and predict time series when there are very long time lags of unknown size between important events. This is one of the main reasons why LSTM outperforms alternative RNNs and Hidden Markov Models and other sequence learning methods in numerous applications. For example, LSTM achieved the best known results in unsegmented connected handwriting recognition,[*][2] and in 2009 won the ICDAR handwriting competition. LSTM networks have also been used for automatic speech recognition, and were a major component of a network that recently achieved a record 17.7% phoneme error rate on the classic TIMIT natural speech dataset.[*][3]



A typical implementation of an LSTM block.

12.1 Architecture

An LSTM network is an artificial neural network that contains LSTM blocks instead of, or in addition to, regular network units. An LSTM block may be described as a "smart" network unit that can remember a value for an arbitrary length of time. An LSTM block contains gates that determine when the input is significant enough to remember, when it should continue to remember or forget the value, and when it should output the value.

A typical implementation of an LSTM block is shown to the right. The four units shown at the bottom of the figure are sigmoid units ($y = s(\sum w_i x_i)$, where s is some squashing function, such as the logistic function). The left-most of these units computes a value which is conditionally fed as an input value to the block's memory. The other three units serve as gates to determine when values are allowed to flow into or out of the block's memory. The second unit from the left (on the bottom row) is the "input gate". When it outputs a value close to zero, it zeros out the value from the left-most unit, effectively blocking that value from entering into the next layer. The second unit from the right is the "forget gate". When it outputs a value close to zero, the block will effectively forget whatever value it was remembering. The right-most unit (on the bottom row) is the "output gate". It determines when the unit should output the value in its memory. The units containing the Π symbol compute the product of their inputs ($y = \Pi x_i$). These units have no weights. The unit with the Σ symbol computes a linear function of its inputs ($y = \sum w_i x_i$.) The output of this unit is not squashed so that it can remember the same value for many time-steps without the value decaying. This value is fed back in so that the block can "remember" it (as long as the forget gate allows). Typically, this value is also fed into the 3 gating units to help them make gating decisions.

12.2 Training

To minimize LSTM's total error on a set of training sequences, iterative gradient descent such as backpropagation through time can be used to change each weight in proportion to its derivative with respect to the error. A major problem with gradient descent for standard RNNs is that error gradients vanish exponentially quickly with the size of the time lag between important events, as first realized in 1991.[*][4][*][5] With LSTM blocks, however, when error values are back-propagated from the output, the error becomes trapped in the memory portion of the block. This is referred to as an "error carousel", which continuously feeds error back to each of the gates until they become trained to cut off the value. Thus, regular backpropagation is effective at training an LSTM block to remember values for very long durations.

LSTM can also be trained by a combination of artificial evolution for weights to the hidden units, and pseudo-inverse or support vector machines for weights to the output units.[*][6] In reinforcement learning applications LSTM can be trained by policy gradient methods or evolution strategies or genetic algorithms.

12.3 Applications

Applications of LSTM include:

- Robot control[*][7]
- Time series prediction[*][8]
- Speech recognition[*][9][*][10][*][11]
- Rhythm learning[*][12]
- Music composition[*][13]
- Grammar learning[*][14][*][15][*][16]
- Handwriting recognition[*][17][*][18]
- Human action recognition[*][19]
- Protein Homology Detection[*][20]

12.4 See also

- Artificial neural network
- Prefrontal Cortex Basal Ganglia Working Memory (PBWM)
- Recurrent neural network
- Time series
- Long-term potentiation

12.5 References

[1] S. Hochreiter and J. Schmidhuber. Long short-term memory. Neural Computation, 9(8):1735–1780, 1997.

[2] A. Graves, M. Liwicki, S. Fernandez, R. Bertolami, H. Bunke, J. Schmidhuber. A Novel Connectionist System for Improved Unconstrained Handwriting Recognition. IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 31, no. 5, 2009.

[3] Graves, Alex; Mohamed, Abdel-rahman; Hinton, Geoffrey (2013). "Speech Recognition with Deep Recurrent Neural Networks". *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*: 6645–6649.

[4] S. Hochreiter. Untersuchungen zu dynamischen neuronalen Netzen. Diploma thesis, Institut f. Informatik, Technische Univ. Munich, 1991.

[5] S. Hochreiter, Y. Bengio, P. Frasconi, and J. Schmidhuber. Gradient flow in recurrent nets: the difficulty of learning long-term dependencies. In S. C. Kremer and J. F. Kolen, editors, A Field Guide to Dynamical Recurrent Neural Networks. IEEE Press, 2001.

[6] J. Schmidhuber, D. Wierstra, M. Gagliolo, F. Gomez. Training Recurrent Networks by Evolino. Neural Computation, 19(3): 757–779, 2007.

[7] H. Mayer, F. Gomez, D. Wierstra, I. Nagy, A. Knoll, and J. Schmidhuber. A System for Robotic Heart Surgery that Learns to Tie Knots Using Recurrent Neural Networks. Advanced Robotics, 22/13–14, pp. 1521–1537, 2008.

[8] J. Schmidhuber and D. Wierstra and F. J. Gomez. Evolino: Hybrid Neuroevolution / Optimal Linear Search for Sequence Learning. Proceedings of the 19th International Joint Conference on Artificial Intelligence (IJCAI), Edinburgh, pp. 853–858, 2005.

[9] A. Graves and J. Schmidhuber. Framewise phoneme classification with bidirectional LSTM and other neural network architectures. Neural Networks 18:5–6, pp. 602–610, 2005.

[10] S. Fernandez, A. Graves, J. Schmidhuber. An application of recurrent neural networks to discriminative keyword spotting. Intl. Conf. on Artificial Neural Networks ICANN'07, 2007.

[11] Graves, Alex; Mohamed, Abdel-rahman; Hinton, Geoffrey (2013). "Speech Recognition with Deep Recurrent Neural Networks". *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*: 6645–6649.

[12] F. Gers, N. Schraudolph, J. Schmidhuber. Learning precise timing with LSTM recurrent networks. Journal of Machine Learning Research 3:115–143, 2002.

[13] D. Eck and J. Schmidhuber. Learning The Long-Term Structure of the Blues. In J. Dorronsoro, ed., Proceedings of Int. Conf. on Artificial Neural Networks ICANN'02, Madrid, pages 284–289, Springer, Berlin, 2002.

[14] J. Schmidhuber, F. Gers, D. Eck. J. Schmidhuber, F. Gers, D. Eck. Learning nonregular languages: A comparison of simple recurrent networks and LSTM. Neural Computation 14(9):2039–2041, 2002.

[15] F. A. Gers and J. Schmidhuber. LSTM Recurrent Networks Learn Simple Context Free and Context Sensitive Languages. IEEE Transactions on Neural Networks 12(6):1333–1340, 2001.

[16] J. A. Perez-Ortiz, F. A. Gers, D. Eck, J. Schmidhuber. Kalman filters improve LSTM network performance in problems unsolvable by traditional recurrent nets. Neural Networks 16(2):241–250, 2003.

[17] A. Graves, J. Schmidhuber. Offline Handwriting Recognition with Multidimensional Recurrent Neural Networks. Advances in Neural Information Processing Systems 22, NIPS'22, pp 545–552, Vancouver, MIT Press, 2009.

[18] A. Graves, S. Fernandez,M. Liwicki, H. Bunke, J. Schmidhuber. Unconstrained online handwriting recognition with recurrent neural networks. Advances in Neural Information Processing Systems 21, NIPS'21, pp 577–584, 2008, MIT Press, Cambridge, MA, 2008.

[19] M. Baccouche, F. Mamalet, C Wolf, C. Garcia, A. Baskurt. Sequential Deep Learning for Human Action Recognition. 2nd International Workshop on Human Behavior Understanding (HBU), A.A. Salah, B. Lepri ed. Amsterdam, Netherlands. pp. 29–39. Lecture Notes in Computer Science 7065. Springer. 2011

[20] Hochreiter, S.; Heusel, M.; Obermayer, K. (2007). "Fast model-based protein homology detection without alignment" . *Bioinformatics* **23** (14): 1728–1736. doi:10.1093/bioinformatics/btm247. PMID 17488755.

12.6 External links

- Recurrent Neural Networks with over 30 LSTM papers by Jürgen Schmidhuber's group at IDSIA
- Gers PhD thesis on LSTM networks.
- Fraud detection paper with two chapters devoted to explaining recurrent neural networks, especially LSTM.
- Paper on a high-performing extension of LSTM that has been simplified to a single node type and can train arbitrary architectures.
- Tutorial: How to implement LSTM in python with theano

Chapter 13

Google Brain

Google Brain is a deep learning research project at Google.

13.1 History

Stanford University professor Andrew Ng who, since around 2006, became interested in using deep learning techniques to crack the problem of artificial intelligence, started a deep learning project in 2011 as one of the Google X projects. He then joined with Google Fellow Jeff Dean to develop "the Google brain" as it was first referred to.[*][1]

In June 2012, the *New York Times* reported that a cluster of 16,000 computers dedicated to mimicking some aspects of human brain activity had successfully trained itself to recognize a cat based on 10 million digital images taken from YouTube videos.[*][1] The story was also covered by National Public Radio[*][2] and SmartPlanet.[*][3]

In March 2013, Google hired Geoffrey Hinton, a leading researcher in the deep learning field, and acquired the company DNNResearch Inc. headed by Hinton. Hinton said that he would be dividing his future time between his university research and his work at Google.[*][4]

On 26 January 2014, multiple news outlets stated that Google had purchased DeepMind Technologies for an undisclosed amount. Analysts later announced that the company was purchased for £400 Million ($650M USD / €486M), although later reports estimated the acquisition was valued at over £500 Million.[*][5][*][6][*][7][*][8][*][9][*][10][*][11] The acquisition reportedly took place after Facebook ended negotiations with DeepMind Technologies in 2013, which resulted in no agreement or purchase of the company.[*][12] Google has yet to comment or make an official announcement on this acquisition.

In December 2012, futurist and inventor Ray Kurzweil, author of *The Singularity is Near*, joined Google in a full-time engineering director role, but focusing on the deep learning project.[*][13] It was reported that Kurzweil would have "unlimited resources" to pursue his vision at Google.[*][14][*][15][*][16][*][17] However, he is leading his own team, which is independent of Google Brain.

13.2 In Google products

The project's technology is currently used in the Android Operating System's speech recognition system[*][18] and photosearch for Google+.[*][19]

13.3 Team

Google Brain was initially established by Google Fellow Jeff Dean and visiting Stanford professor Andrew Ng[*][20] (Ng since moved to lead the artificial intelligence group at Baidu[*][21]). As of 2014, team members included Jeff Dean, Geoffrey Hinton, Greg Corrado, Quoc Le,[*][22] Ilya Sutskever, Alex Krizhevsky, Samy Bengio, and Vincent Vanhoucke.

13.4 Reception

Google Brain has received in-depth coverage in *Wired Magazine*,[*][6][*][16][*][23] the *New York Times*,[*][23] Technology Review,[*][5][*][17] National Public Radio,[*][2] and Big Think.[*][24]

13.5 See also

- Google X
- Google Research
- Quantum Artificial Intelligence Lab run by Google in collaboration with NASA and Universities Space Research Association.

13.6 References

[1] Markoff, John (June 25, 2012). "How Many Computers to Identify a Cat? 16,000" . *New York Times*. Retrieved February 11, 2014.

[2] "A Massive Google Network Learns To Identify —Cats" . National Public Radio. June 26, 2012. Retrieved February 11, 2014.

[3] Shin, Laura (June 26, 2012). "Google brain simulator teaches itself to recognize cats" . SmartPlanet. Retrieved February 11, 2014.

[4] "U of T neural networks start-up acquired by Google" (Press release). Toronto, ON. 12 March 2013. Retrieved 13 March 2013.

[5] Regalado, Antonio (January 29, 2014). "Is Google Cornering the Market on Deep Learning? A cutting-edge corner of science is being wooed by Silicon Valley, to the dismay of some academics." . Technology Review. Retrieved February 11, 2014.

[6] Wohlsen, Marcus (January 27, 2014). "Google's Grand Plan to Make Your Brain Irrelevant" . *Wired Magazine*. Retrieved February 11, 2014.

[7] "Google Acquires UK AI startup Deepmind" . The Guardian. Retrieved 27 January 2014.

[8] "Report of Acquisition, TechCrunch". TechCrunch. Retrieved 27 January 2014.

[9] Oreskovic, Alexei. "Reuters Report". Reuters. Retrieved 27 January 2014.

[10] "Google Acquires Artificial Intelligence Start-Up DeepMind" . The Verge. Retrieved 27 January 2014.

[11] "Google acquires AI pioneer DeepMind Technologies" . Ars Technica. Retrieved 27 January 2014.

[12] "Google beats Facebook for Acquisition of DeepMind Technologies" . Retrieved 27 January 2014.

[13] Taylor, Colleen (December 14, 2012). "Ray Kurzweil Joins Google In Full-Time Engineering Director Role; Will Focus On Machine Learning, Language Processing" . *TechCrunch*. Retrieved February 11, 2014.

[14] Empson, Rip (January 3, 2013). "Imagining The Future: Ray Kurzweil Has "Unlimited Resources" For AI, Language Research At Google" . *TechCrunch*. Retrieved February 11, 2014.

[15] Ferenstein, Gregory (January 6, 2013). "Google's New Director Of Engineering, Ray Kurzweil, Is Building Your 'Cybernetic Friend' ". *TechCrunch*. Retrieved February 11, 2014.

[16] Levy, Steven (April 25, 2013). "How Ray Kurzweil Will Help Google Make the Ultimate AI Brain" . *Wired Magazine*. Retrieved February 11, 2014.

[17] Hof, Robert (April 23, 2013). "Deep Learning: With massive amounts of computational power, machines can now recognize objects and translate speech in real time. Artificial intelligence is finally getting smart." . Technology Review. Retrieved February 11, 2014.

[18] "Speech Recognition and Deep Learning" . *Google Research Blog*. Google. August 6, 2012. Retrieved February 11, 2014.

[19] "Improving Photo Search: A Step Across the Semantic Gap" . *Google Research Blog*. Google. June 12, 2013.

[20] Jeff Dean and Andrew Ng (26 June 2012). "Using large-scale brain simulations for machine learning and A.I." . *Official Google Blog*. Retrieved 26 January 2015.

[21] "Ex-Google Brain head Andrew Ng to lead Baidu's artificial intelligence drive" . South China Morning Post.

[22] "Quoc Le - Behind the Scenes" . Retrieved 20 April 2015.

[23] Hernandez, Daniela (May 7, 2013). "The Man Behind the Google Brain: Andrew Ng and the Quest for the New AI" . *Wired Magazine*. Retrieved February 11, 2014.

[24] "Ray Kurzweil and the Brains Behind the Google Brain" . Big Think. December 8, 2013. Retrieved February 11, 2014.

Chapter 14

Google DeepMind

Google DeepMind is a British artificial intelligence company. Founded in 2011 as **DeepMind Technologies**, it was acquired by Google in 2014.

14.1 History

14.1.1 2011 to 2014

In 2011 the start-up was founded by Demis Hassabis, Shane Legg and Mustafa Suleyman.[*][3][*][4] Hassabis and Legg first met at UCL's Gatsby Computational Neuroscience Unit.[*][5]

Since then major venture capitalist firms Horizons Ventures and Founders Fund have invested in the company,[*][6] as well as entrepreneur Scott Banister.[*][7] Jaan Tallinn was an early investor and an advisor to the company.[*][8]

In 2014, DeepMind received the "Company of the Year" award by Cambridge Computer Laboratory.[*][9]

The company has created a neural network that learns how to play video games in a similar fashion to humans[*][10] and a neural network that may be able to access an external memory like a conventional Turing machine, resulting in a computer that appears to possibly mimic the short-term memory of the human brain.[*][11]

14.1.2 Acquisition by Google

On 26 January 2014, Google announced[*][12] that it had agreed to acquire DeepMind Technologies. The acquisition reportedly took place after Facebook ended negotiations with DeepMind Technologies in 2013.[*][13] Following the acquisition, the company was renamed **Google DeepMind**.[*][1]

Estimates of the cost of acquisition vary, from $400 million[*][14] to over £500 million.[*][15][*][16][*][17][*][18][*][19]

One of DeepMind's conditions for Google was that they establish an AI Ethics committee.[*][20]

14.2 Research

DeepMind Technologies's goal is to "*solve intelligence*",[*][21] which they are trying to achieve by combining "*the best techniques from machine learning and systems neuroscience to build powerful general-purpose learning algorithms*". [*][21] They are trying to formalize intelligence[*][22] in order to not only implement it into machines, but also understand the human brain, as Demis Hassabis explains:

> [...] Attempting to distil intelligence into an algorithmic construct may prove to be the best path to understanding some of the enduring mysteries of our minds.
> —Demis Hassabis, Nature (journal), 23 February 2012[*][23]

Currently the company's focus is on publishing research on computer systems that are able to play games, and developing these systems, ranging from strategy games such as Go[*][24] to arcade games. According to Shane Legg human-level machine intelligence can be achieved "*when a machine can learn to play a really wide range of games from perceptual stream input and output, and transfer understanding across games[...].*"[*][25] Research describing an AI playing seven different Atari video games (Pong, Breakout, Space Invaders, Seaquest, Beamrider, Enduro, and Q*bert) reportedly led to their acquisition by Google.[*][10]

14.2.1 Deep reinforcement learning

As opposed to other AI's, such as IBM's Deep Blue or Watson, which were developed for a pre-defined purpose and only function within its merit, DeepMind claims that their system is not pre-programmed: it learns from experience, using only raw pixels as data input.[*][1][*][26] They test the system on video games, notably early arcade games, such as Space Invaders or Breakout.[*][26][*][27] Without altering the code, the AI begins to understand how to play the game, and after some time plays, for a few games (most notably Breakout), a more efficient game

than any human ever could.*[27] For most games though (Space Invaders, Ms Pacman, Q*Bert for example), DeepMind plays well below the current World Record. The application of DeepMind's AI to video games is currently for games made in the 1970s and 1980s, with work being done on more complex 3D games such as *Doom*, which first appeared in the early 1990s.*[27]

14.3 References

[1] Mnih1, Volodymyr; Kavukcuoglu1, Koray; Silver, David (26 February 2015). "Human-level control through deep reinforcement learning". *Nature*. doi:10.1038/nature14236. Retrieved 25 February 2015.

[2] "Forbes Report - Acquisition". Forbes. Retrieved 27 January 2014.

[3] "Google Buys U.K. Artificial Intelligence Company DeepMind". *Bloomberg*. 27 January 2014. Retrieved 13 November 2014.

[4] "Google makes £400m move in quest for artificial intelligence". *Financial Times*. 27 January 2014. Retrieved 13 November 2014.

[5] "Demis Hassabis: 15 facts about the DeepMind Technologies founder". The Guardian. Retrieved 12 October 2014.

[6] "DeepMind buy heralds rise of the machines". Financial Times. Retrieved 14 October 2014.

[7] "DeepMind Technologies Investors". Retrieved 12 October 2014.

[8] "Recode.net - DeepMind Technologies Acquisition". Retrieved 27 January 2014.

[9] "Hall of Fame Awards: To celebrate the success of companies founded by Computer Laboratory graduates.". Cambridge University. Retrieved 12 October 2014.

[10] "The Last AI Breakthrough DeepMind Made Before Google Bought It". The Physics arXiv Blog. Retrieved 12 October 2014.

[11] Best of 2014: Google's Secretive DeepMind Startup Unveils a "Neural Turing Machine", *MIT Technology Review*

[12] "Google to buy artificial intelligence company DeepMind". Reuters. 26 January 2014. Retrieved 12 October 2014.

[13] "Google beats Facebook for Acquisition of DeepMind Technologies". Retrieved 27 January 2014.

[14] "Computers, gaming". The Economist. 28 February 2015.

[15] "Google Acquires UK AI startup Deepmind". The Guardian. Retrieved 27 January 2014.

[16] "Report of Acquisition, TechCrunch". TechCrunch. Retrieved 27 January 2014.

[17] Oreskovic, Alexei. "Reuters Report". Reuters. Retrieved 27 January 2014.

[18] "Google Acquires Artificial Intelligence Start-Up DeepMind". The Verge. Retrieved 27 January 2014.

[19] "Google acquires AI pioneer DeepMind Technologies". Ars Technica. Retrieved 27 January 2014.

[20] "Inside Google's Mysterious Ethics Board". Forbes. 3 February 2014. Retrieved 12 October 2014.

[21] "DeepMind Technologies Website". DeepMind Technologies. Retrieved 11 October 2014.

[22] Shane Legg; Joel Veness (29 September 2011). "An Approximation of the Universal Intelligence Measure" (PDF). Retrieved 12 October 2014.

[23] Demis Hassabis (23 February 2012). "Model the brain's algorithms" (PDF). Nature. Retrieved 12 October 2014.

[24] Shih-Chieh Huang; Martin Müller (12 July 2014). "Investigating the Limits of Monte-Carlo Tree Search Methods in Computer Go". Springer.

[25] "Q&A with Shane Legg on risks from AI". 17 June 2011. Retrieved 12 October 2014.

[26] Volodymyr Mnih; Koray Kavukcuoglu; David Silver; Alex Graves; Ioannis Antonoglou; Daan Wierstra; Martin Riedmiller (12 December 2013). "Playing Atari with Deep Reinforcement Learning" (PDF). Retrieved 12 October 2014.

[27] *Deepmind artificial intelligence @ FDOT14*. 19 April 2014.

14.4 External links

- Google DeepMind

Chapter 15

Torch (machine learning)

Torch is an open source machine learning library, a scientific computing framework, and a script language based on the Lua programming language.[*][3] It provides a wide range of algorithms for deep machine learning, and uses an extremely fast scripting language LuaJIT, and an underlying C implementation.

15.1 torch

The core package of Torch is torch. It provides a flexible N-dimensional array or Tensor, which supports basic routines for indexing, slicing, transposing, type-casting, resizing, sharing storage and cloning. This object is used by most other packages and thus forms the core object of the library. The Tensor also supports mathematical operations like max, min, sum, statistical distributions like uniform, normal and multinomial, and BLAS operations like dot product, matrix-vector multiplication, matrix-matrix multiplication, matrix-vector product and matrix product.

The following exemplifies using torch via its REPL interpreter:

> a = torch.randn(3,4) > =a −0.2381 −0.3401 −1.7844 −0.2615 0.1411 1.6249 0.1708 0.8299 −1.0434 2.2291 1.0525 0.8465 [torch.DoubleTensor of dimension 3x4] > a[1][2] −0.34010116549482 > a:narrow(1,1,2) −0.2381 −0.3401 −1.7844 −0.2615 0.1411 1.6249 0.1708 0.8299 [torch.DoubleTensor of dimension 2x4] > a:index(1, torch.LongTensor{1,2}) −0.2381 −0.3401 −1.7844 −0.2615 0.1411 1.6249 0.1708 0.8299 [torch.DoubleTensor of dimension 2x4] > a:min() −1.7844365427828

The torch package also simplifies object oriented programming and serialization by providing various convenience functions which are used throughout its packages. The torch.class(classname, parentclass) function can be used to create object factories (classes). When the constructor is called, torch initializes and sets a Lua table with the user-defined metatable, which makes the table an object.

Objects created with the torch factory can also be serialized, as long as they do not contain references to objects that cannot be serialized, such as Lua coroutines, and Lua *userdata*. However, *userdata* can be serialized if it is wrapped by a table (or metatable) that provides read() and write() methods.

15.2 nn

The nn package is used for building neural networks. It is divided into modular objects that share a common Module interface. Modules have a forward() and backward() method that allow them to feedforward and backpropagate, respectively. Modules can be joined together using module composites, like Sequential, Parallel and Concat to create complex task-tailored graphs. Simpler modules like Linear, Tanh and Max make up the basic component modules. This modular interface provides first-order automatic gradient differentiation. What follows is an example use-case for building a multilayer perceptron using Modules:

> mlp = nn.Sequential() > mlp:add(nn.Linear(10, 25)) -- 10 input, 25 hidden units > mlp:add(nn.Tanh()) -- some hyperbolic tangent transfer function > mlp:add(nn.Linear(25, 1)) -- 1 output > =mlp:forward(torch.randn(10)) −0.1815 [torch.Tensor of dimension 1]

Loss functions are implemented as sub-classes of Criterion, which has a similar interface to Module. It also has forward() and backward methods for computing the loss and backpropagating gradients, respectively. Criteria are helpful to train neural network on classical tasks. Common criteria are the Mean Squared Error criterion implemented in MSECriterion and the cross-entropy criterion implemented in ClassNLLCriterion. What follows is an example of a Lua function that can be iteratively called to train an mlp Module on input Tensor x, target Tensor y with a scalar learningRate:

function gradUpdate(mlp,x,y,learningRate) local criterion = nn.ClassNLLCriterion() pred = mlp:forward(x) local err = criterion:forward(pred, y); mlp:zeroGradParameters(); local t =

criterion:backward(pred, y); mlp:backward(x, t); mlp:updateParameters(learningRate); end

It also has StochasticGradient class for training a neural network using Stochastic gradient descent, although the Optim package provides much more options in this respect, like momentum and weight decay regularization.

15.3 Other packages

Many packages other than the above official packages are used with Torch. These are listed in the torch cheatsheet. These extra packages provide a wide range of utilities such as parallelism, asynchronous input/output, image processing, and so on.

15.4 Applications

Torch is used by Google DeepMind,*[4] the Facebook AI Research Group,*[5] IBM,*[6] Yandex*[7] and the Idiap Research Institute.*[8] Torch has been extended for use on Android*[9] and iOS.*[10] It has been used to build hardware implementations for data flows like those found in neural networks.*[11]

Facebook has released a set of extension modules as open source software.*[12]

15.5 References

[1] "Torch: a modular machine learning software library". 30 October 2002. Retrieved 24 April 2014.

[2] Ronan Collobert. "Torch7". *GitHub*.

[3] Ronan Collobert; Koray Kavukcuoglu; Clement Farabet (2011). "Torch7: A Matlab-like Environment for Machine Learning" (PDF). *Neural Information Processing Systems*.

[4] What is going on with DeepMind and Google?

[5] KDnuggets Interview with Yann LeCun, Deep Learning Expert, Director of Facebook AI Lab

[6] Hacker News

[7] Yann Lecun's Facebook Page

[8] IDIAP Research Institute : Torch

[9] Torch-android GitHub repository

[10] Torch-ios GitHub repository

[11] NeuFlow: A Runtime Reconfigurable Dataflow Processor for Vision

[12] "Facebook Open-Sources a Trove of AI Tools". *Wired*. 16 January 2015.

15.6 External links

- Official website
- https://github.com/torch/torch7

Chapter 16

Theano (software)

Theano is a numerical computation library for Python.[*][1] In Theano, computations are expressed using a NumPy-like syntax and compiled to run efficiently on either CPU or GPU architectures.

Theano is an open source project[*][2] primarily developed by a machine learning group at the Université de Montréal.[*][3]

16.1 See also

- SciPy
- Torch

16.2 References

[1] Bergstra, J.; O. Breuleux, F. Bastien, P. Lamblin, R. Pascanu, G. Desjardins, J. Turian, D. Warde-Farley and Y. Bengio (30 June 2010). "Theano: A CPU and GPU Math Expression Compiler" (PDF). *Proceedings of the Python for Scientific Computing Conference (SciPy) 2010*.

[2] "Github Repository".

[3] "deeplearning.net".

Chapter 17

Deeplearning4j

Deeplearning4j is an open source deep learning library written for Java and the Java Virtual Machine*[1]*[2] and a computing framework with wide support for deep learning algorithms. Deeplearning4j includes implementations of the restricted Boltzmann machine, deep belief net, deep autoencoder, stacked denoising autoencoder and recursive neural tensor network, as well as word2vec, doc2vec and GloVe. These algorithms all include distributed parallel versions that integrate with Hadoop and Spark.*[3]

17.1 Introduction

Deeplearning4j relies on the widely used programming language, Java - though it is compatible with Clojure and includes a Scala API. It is powered by its own open-source numerical computing library, ND4J, and works with both CPUs and GPUs.*[4] *[5]

Deeplearning4j is an open source project*[6] primarily developed by a machine learning group in San Francisco led by Adam Gibson.*[7]*[8] Deeplearning4j is the only open-source project listed on Google's Word2vec page for its Java implementation.*[9]

Deeplearning4j has been used in a number of commercial and academic applications. The code is hosted on GitHub*[10] and a support forum is maintained on Google Groups.*[11]

The framework is composable, meaning shallow neural nets such as restricted Boltzmann machines, convolutional nets, autoencoders and recurrent nets can be added to one another to create deep nets of varying types.

17.2 Distributed

Training with Deeplearning4j takes place in the cluster, which means it can process massive amounts of data. Neural nets are trained in parallel via iterative reduce, which works on Hadoop/YARN and on Spark.*[7]*[12] Deeplearning4j also integrates with Cuda kernels to conduct pure GPU operations, and works with distributed GPUs.

17.3 Scientific Computing for the JVM

Deeplearning4j includes an n-dimensional array class using ND4J that allows for scientific computing in Java and Scala, similar to the functionality that Numpy provides to Python. It's effectively based on a library for linear algebra and matrix manipulation in a production environment. It relies on Matplotlib as a plotting package.

17.4 Canova Vectorization Lib for Machine-Learning

Canova vectorizes various file formats and data types using an input/output format system similar to Hadoop's use of MapReduce. A work in progress, Canova is designed to vectorize CSVs, images, sound, text and video. Since vectorization is a necessary step in preparing data to be ingested by neural nets, Canova solves one of the most important problems in machine learning. Canova can be used from the command line.

17.5 Text & NLP

Deeplearning4j includes a vector space modeling and topic modeling toolkit, implemented in Java and integrating with parallel GPUs for performance. It is specifically intended for handling large text collections.

Deeplearning4j includes implementations of tf–idf, deep learning, and Mikolov's word2vec algorithm, doc2vec and GloVe -- reimplemented and optimized in Java. It relies on TSNE for word-cloud visualizations.

17.6 See also

- Torch
- Theano

17.7 References

[1] Metz, Cade (2014-06-02). "The Mission to Bring Google's AI to the Rest of the World". *Wired.com*. Retrieved 2014-06-28.

[2] Vance, Ashlee (2014-06-03). "Deep Learning for (Some of) the People". *Bloomberg Businessweek*. Retrieved 2014-06-28.

[3] TV, Functional (2015-02-12). "Adam Gibson, DeepLearning4j on Spark and Data Science on JVM with nd4j, SF Spark @Galvanize 20150212". *SF Spark Meetup*. Retrieved 2015-03-01.

[4] Harris, Derrick (2014-06-02). "A startup called Skymind launches, pushing open source deep learning". *GigaOM.com*. Retrieved 2014-06-29.

[5] Novat, Jordan (2014-06-02). "Skymind launches with open-source, plug-and-play deep learning features for your app". Retrieved 2014-06-29.

[6] "Github Repository".

[7] "deeplearning4j.org".

[8] "Crunchbase Profile".

[9] "Google Code".

[10] Deeplearning4j source code

[11] Deeplearning4j Google Group

[12] "Iterative reduce".

17.8 External links

- Official website
- "Github Repositories".
- "Deeplearning4j vs. Torch vs. Caffe vs. Pylearn".
- "Canova: A General Vectorization Lib for Machine Learning".
- "Apache Flink".

Chapter 18

Gensim

Gensim is an open-source vector space modeling and topic modeling toolkit, implemented in the Python programming language, using NumPy, SciPy and optionally Cython for performance. It is specifically intended for handling large text collections, using efficient online algorithms.

Gensim includes implementations of tf–idf, random projections, deep learning with Google's word2vec algorithm [*][1] (reimplemented and optimized in Cython), hierarchical Dirichlet processes (HDP), latent semantic analysis (LSA) and latent Dirichlet allocation (LDA), including distributed parallel versions.[*][2]

Gensim has been used in a number of commercial as well as academic applications.[*][3][*][4] The code is hosted on GitHub[*][5] and a support forum is maintained on Google Groups.[*][6]

Gensim accompanied the PhD dissertation *Scalability of Semantic Analysis in Natural Language Processing* of Radim Řehůřek (2011).[*][7]

18.1 Gensim's tagline

- Topic Modelling for Humans [*][8]

18.2 References

[1] Deep learning with word2vec and gensim

[2] Radim Řehůřek and Petr Sojka (2010). Software framework for topic modelling with large corpora. Proc. LREC Workshop on New Challenges for NLP Frameworks.

[3] Interview with Radim Řehůřek, creator of gensim

[4] gensim academic citations

[5] gensim source code

[6] gensim mailing list

[7] Rehurek, Radim (2011). "Scalability of Semantic Analysis in Natural Language Processing" (PDF). *http://radimrehurek.com/". Retrieved 27 January 2015. my open-source* gensim *software package that accompanies this thesis*

[8] Rehurek, Radim. "Gensim". *http://radimrehurek.com/". Retrieved 27 January 2015. Gensim's tagline: "**Topic Modelling for Humans**"*

18.3 External links

- Official website

Chapter 19

Geoffrey Hinton

Geoffrey (Geoff) Everest Hinton FRS (born 6 December 1947) is a British-born cognitive psychologist and computer scientist, most noted for his work on artificial neural networks. He now divides his time working for Google and University of Toronto.*[1] He is the co-inventor of the backpropagation and contrastive divergence training algorithms and is an important figure in the deep learning movement.*[2]

19.1 Career

Hinton graduated from Cambridge in 1970, with a Bachelor of Arts in experimental psychology, and from Edinburgh in 1978, with a PhD in artificial intelligence. He has worked at Sussex, University of California San Diego, Cambridge, Carnegie Mellon University and University College London. He was the founding director of the Gatsby Computational Neuroscience Unit at University College London, and is currently a professor in the computer science department at the University of Toronto. He holds a Canada Research Chair in Machine Learning. He is the director of the program on "Neural Computation and Adaptive Perception" which is funded by the Canadian Institute for Advanced Research. Hinton taught a free online course on Neural Networks on the education platform Coursera in 2012.*[3] Hinton joined Google in March 2013 when his company, DNNresearch Inc, was acquired. He is planning to "divide his time between his university research and his work at Google".*[4]

19.2 Research interests

An accessible introduction to Geoffrey Hinton's research can be found in his articles in *Scientific American* in September 1992 and October 1993. He investigates ways of using neural networks for learning, memory, perception and symbol processing and has authored over 200 publications in these areas. He was one of the researchers who introduced the back-propagation algorithm for training multi-layer neural networks that has been widely used for practical applications. He co-invented Boltzmann machines with Terry Sejnowski. His other contributions to neural network research include distributed representations, time delay neural network, mixtures of experts, Helmholtz machines and Product of Experts. His current main interest is in unsupervised learning procedures for neural networks with rich sensory input.

19.3 Honours and awards

Hinton was the first winner of the David E. Rumelhart Prize. He was elected a Fellow of the Royal Society in 1998.*[5]

In 2001, Hinton was awarded an Honorary Doctorate from the University of Edinburgh.

Hinton was the 2005 recipient of the IJCAI Award for Research Excellence lifetime-achievement award.

He has also been awarded the 2011 Herzberg Canada Gold Medal for Science and Engineering.*[6]

In 2013, Hinton was awarded an Honorary Doctorate from the Université de Sherbrooke.

19.4 Personal life

Hinton is the great-great-grandson both of logician George Boole whose work eventually became one of the foundations of modern computer science, and of surgeon and author James Hinton.*[7] His father is Howard Hinton.

19.5 References

[1] Daniela Hernandez (7 May 2013). "The Man Behind the Google Brain: Andrew Ng and the Quest for the New AI". *Wired*. Retrieved 10 May 2013.

[2] "How a Toronto professor's research revolutionized artificial intelligence". *Toronto Star*, Kate Allen, Apr 17 2015

[3] https://www.coursera.org/course/neuralnets

[4] "U of T neural networks start-up acquired by Google" (Press release). Toronto, ON. 12 March 2013. Retrieved 13 March 2013.

[5] "Fellows of the Royal Society". The Royal Society. Retrieved 14 March 2013.

[6] "Artificial intelligence scientist gets M prize". *CBC News*. 14 February 2011.

[7] The Isaac Newton of logic

19.6 External links

- Geoffrey E. Hinton's Academic Genealogy
- Geoffrey E. Hinton's Publications in Reverse Chronological Order
- Homepage (at UofT)
- "The Next Generation of Neural Networks" on YouTube
- Gatsby Computational Neuroscience Unit (founding director)
- Encyclopedia article on Boltzmann Machines written by Geoffrey Hinton for Scholarpedia

Chapter 20

Yann LeCun

Yann LeCun (born 1960) is a computer scientist with contributions in machine learning, computer vision, mobile robotics and computational neuroscience. He is well known for his work on optical character recognition and computer vision using convolutional neural networks (CNN), and is a founding father of convolutional nets.[*][1][*][2] He is also one of the main creators of the DjVu image compression technology (together with Léon Bottou and Patrick Haffner). He co-developed the Lush programming language with Léon Bottou.

20.1 Life

Yann LeCun was born near Paris, France, in 1960. He received a Diplôme d'Ingénieur from the Ecole Superieure d'Ingénieur en Electrotechnique et Electronique (ESIEE), Paris in 1983, and a PhD in Computer Science from Université Pierre et Marie Curie in 1987 during which he proposed an early form of the back-propagation learning algorithm for neural networks.[*][3] He was a postdoctoral research associate in Geoffrey Hinton's lab at the University of Toronto.

In 1988, he joined the Adaptive Systems Research Department at AT&T Bell Laboratories in Holmdel, New Jersey, USA, where he developed a number of new machine learning methods, such as a biologically inspired model of image recognition called Convolutional Neural Networks,[*][4] the "Optimal Brain Damage" regularization methods,[*][5] and the Graph Transformer Networks method (similar to conditional random field), which he applied to handwriting recognition and OCR.[*][6] The bank check recognition system that he helped develop was widely deployed by NCR and other companies, reading over 10% of all the checks in the US in the late 1990s and early 2000s.

In 1996, he joined AT&T Labs-Research as head of the Image Processing Research Department, which was part of Lawrence Rabiner's Speech and Image Processing Research Lab, and worked primarily on the DjVu image compression technology,[*][7] used by many websites, notably the Internet Archive, to distribute scanned documents. His collaborators at AT&T include Léon Bottou and Vladimir Vapnik.

After a brief tenure as a Fellow of the NEC Research Institute (now NEC-Labs America) in Princeton, NJ, he joined New York University (NYU) in 2003, where he is Silver Professor of Computer Science Neural Science at the Courant Institute of Mathematical Science and the Center for Neural Science. He is also a professor at Polytechnic Institute of New York University.[*][8][*][9] At NYU, he has worked primarily on Energy-Based Models for supervised and unsupervised learning,[*][10] feature learning for object recognition in Computer Vision,[*][11] and mobile robotics.[*][12]

In 2012, he became the founding director of the NYU Center for Data Science.[*][13] On December 9, 2013, LeCun became the first director of Facebook AI Research in New York City.,[*][14] and stepped down from the NYU-CDS directorship in early 2014.

LeCun is the recipient of the 2014 IEEE Neural Network Pioneer Award.

In 2013, he and Yoshua Bengio co-founded the International Conference on Learning Representations, which adopted a post-publication open review process he previously advocated on his website. He was the chair and organizer of the "Learning Workshop" held every year between 1986 and 2012 in Snowbird, Utah. He is a member of the Science Advisory Board of the Institute for Pure and Applied Mathematics[*][15] at UCLA, and has been on the advisory board of a number of companies, including MuseAmi, KXEN Inc., and Vidient Systems.[*][16] He is the Co-Director of the Neural Computation & Adaptive Perception research program of CIFAR[*][17]

20.2 References

[1] *Convolutional Nets and CIFAR-10: An Interview with Yann LeCun*. Kaggle 2014

[2] LeCun, Yann; Léon Bottou; Yoshua Bengio; Patrick Haffner (1998). "Gradient-based learning applied to document recognition" (PDF). *Proceedings of the IEEE* **86** (11): 2278–2324. doi:10.1109/5.726791. Retrieved 16 November 2013.

[3] Y. LeCun: Une procédure d'apprentissage pour réseau a seuil asymmetrique (a Learning Scheme for Asymmetric

Threshold Networks), Proceedings of Cognitiva 85, 599–604, Paris, France, 1985.

[4] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard and L. D. Jackel: Backpropagation Applied to Handwritten Zip Code Recognition, Neural Computation, 1(4):541-551, Winter 1989.

[5] Yann LeCun, J. S. Denker, S. Solla, R. E. Howard and L. D. Jackel: Optimal Brain Damage, in Touretzky, David (Eds), Advances in Neural Information Processing Systems 2 (NIPS*89), Morgan Kaufmann, Denver, CO, 1990.

[6] Yann LeCun, Léon Bottou, Yoshua Bengio and Patrick Haffner: Gradient Based Learning Applied to Document Recognition, Proceedings of IEEE, 86(11):2278–2324, 1998.

[7] Léon Bottou, Patrick Haffner, Paul G. Howard, Patrice Simard, Yoshua Bengio and Yann LeCun: High Quality Document Image Compression with DjVu, Journal of Electronic Imaging, 7(3):410–425, 1998.

[8] "People - Electrical and Computer Engineering". Polytechnic Institute of New York University. Retrieved 13 March 2013.

[9] http://yann.lecun.com/

[10] Yann LeCun, Sumit Chopra, Raia Hadsell, Ranzato Marc'Aurelio and Fu-Jie Huang: A Tutorial on Energy-Based Learning, in Bakir, G. and Hofman, T. and Schölkopf, B. and Smola, A. and Taskar, B. (Eds), Predicting Structured Data, MIT Press, 2006.

[11] Kevin Jarrett, Koray Kavukcuoglu, Marc'Aurelio Ranzato and Yann LeCun: What is the Best Multi-Stage Architecture for Object Recognition?, Proc. International Conference on Computer Vision (ICCV'09), IEEE, 2009

[12] Raia Hadsell, Pierre Sermanet, Marco Scoffier, Ayse Erkan, Koray Kavackuoglu, Urs Muller and Yann LeCun: Learning Long-Range Vision for Autonomous Off-Road Driving, Journal of Field Robotics, 26(2):120–144, February 2009.

[13] http://cds.nyu.edu

[14] https://www.facebook.com/yann.lecun/posts/10151728212367143

[15] http://www.ipam.ucla.edu/programs/gss2012/ Institute for Pure and Applied Mathematics

[16] Vidient Systems.

[17] "Neural Computation & Adaptive Perception Advisory Committee Yann LeCun". CIFAR. Retrieved 16 December 2013.

20.3 External links

- Yann LeCun's personal website
- Yann LeCun's lab website at NYU
- Yann LeCun's List of PhD Students
- Yann LeCun's publications
- Convolutional Neural Networks
- DjVuLibre website
- Lush website

Chapter 21

Jürgen Schmidhuber

Jürgen Schmidhuber (born 17 January 1963 in Munich) is a computer scientist and artist known for his work on machine learning, Artificial Intelligence (AI), artificial neural networks, digital physics, and low-complexity art. His contributions also include generalizations of Kolmogorov complexity and the Speed Prior. From 2004 to 2009 he was professor of Cognitive Robotics at the Technische Universität München. Since 1995 he has been co-director of the Swiss AI Lab IDSIA in Lugano, since 2009 also professor of Artificial Intelligence at the University of Lugano. Between 2009 and 2012, the recurrent neural networks and deep feedforward neural networks developed in his research group have won eight international competitions in pattern recognition and machine learning.*[1] In honor of his achievements he was elected to the European Academy of Sciences and Arts in 2008.

21.1 Contributions

21.1.1 Recurrent neural networks

The dynamic recurrent neural networks developed in his lab are simplified mathematical models of the biological neural networks found in human brains. A particularly successful model of this type is called Long short term memory.*[2] From training sequences it *learns* to solve numerous tasks unsolvable by previous such models. Applications range from automatic music composition to speech recognition, reinforcement learning and robotics in partially observable environments. As of 2010, his group has the best results on benchmarks in automatic handwriting recognition, obtained with deep neural networks*[3] and recurrent neural networks.*[4]

21.1.2 Artificial evolution / genetic programming

As an undergrad at TUM Schmidhuber evolved computer programs through genetic algorithms. The method was published in 1987 as one of the first papers in the emerging field that later became known as genetic programming. In the same year he published the first work on Meta-genetic programming. Since then he has co-authored numerous additional papers on artificial evolution. Applications include robot control, soccer learning, drag minimization, and time series prediction. He received several best paper awards at scientific conferences on evolutionary computation.

21.1.3 Neural economy

In 1989 he created the first learning algorithm for neural networks based on principles of the market economy (inspired by John Holland's bucket brigade algorithm for classifier systems): adaptive neurons compete for being active in response to certain input patterns; those that are active when there is external reward get stronger synapses, but active neurons have to pay those that activated them, by transferring parts of their synapse strengths, thus rewarding "hidden" neurons setting the stage for later success.*[5]

21.1.4 Artificial curiosity and creativity

In 1990 he published the first in a long series of papers on artificial curiosity and creativity for an autonomous agent. The agent is equipped with an adaptive predictor trying to predict future events from the history of previous events and actions. A reward-maximizing, reinforcement learning, adaptive controller is steering the agent and gets *curiosity reward* for executing action sequences that improve the predictor. This discourages it from executing actions leading to boring outcomes that are either predictable or totally unpredictable.*[6] Instead the controller is motivated to learn actions that help the predictor to learn new, previously unknown regularities in its environment, thus improving its model of the world, which in turn can greatly help to solve externally given tasks. This has become an important concept of developmental robotics. Schmidhuber argues that his corresponding formal theory of creativity explains essential aspects of art, science, music, and humor.*[7]

21.1.5 Unsupervised learning / factorial codes

During the early 1990s Schmidhuber also invented a neural method for nonlinear independent component analysis (ICA) called predictability minimization. It is based on co-evolution of adaptive predictors and initially random, adaptive feature detectors processing input patterns from the environment. For each detector there is a predictor trying to predict its current value from the values of neighboring detectors, while each detector is simultaneously trying to become as unpredictable as possible.*[8] It can be shown that the best the detectors can do is to create a factorial code of the environment, that is, a code that conveys all the information about the inputs such that the code components are statistically independent, which is desirable for many pattern recognition applications.

21.1.6 Kolmogorov complexity / computer-generated universe

In 1997 Schmidhuber published a paper based on Konrad Zuse's assumption (1967) that the history of the universe is computable. He pointed out that the simplest explanation of the universe would be a very simple Turing machine programmed to systematically execute all possible programs computing all possible histories for all types of computable physical laws.*[9]*[10] He also pointed out that there is an optimally efficient way of computing all computable universes based on Leonid Levin's universal search algorithm (1973). In 2000 he expanded this work by combining Ray Solomonoff's theory of inductive inference with the assumption that quickly computable universes are more likely than others.*[11] This work on digital physics also led to limit-computable generalizations of algorithmic information or Kolmogorov complexity and the concept of *Super Omegas*, which are limit-computable numbers that are even more random (in a certain sense) than Gregory Chaitin's *number of wisdom* Omega.*[12]

21.1.7 Universal AI

Important research topics of his group include universal learning algorithms and universal AI*[13]*[14] (see Gödel machine). Contributions include the first theoretically optimal decision makers living in environments obeying arbitrary unknown but computable probabilistic laws, and mathematically sound general problem solvers such as the remarkable asymptotically fastest algorithm for all well-defined problems, by his former postdoc Marcus Hutter. Based on the theoretical results obtained in the early 2000s, Schmidhuber is actively promoting the view that in the new millennium the field of general AI has matured and become a real formal science.

21.1.8 Low-complexity art / theory of beauty

Schmidhuber's low-complexity artworks (since 1997) can be described by very short computer programs containing very few bits of information, and reflect his formal theory of beauty*[15] based on the concepts of Kolmogorov complexity and minimum description length.

Schmidhuber writes that since age 15 or so his main scientific ambition has been to build an optimal scientist, then retire. First he wants to build a scientist better than himself (he quips that his colleagues claim that should be easy) who will then do the remaining work. He claims he "cannot see any more efficient way of using and multiplying the little creativity he's got" .

21.1.9 Robot learning

In recent years a robotics group with focus on intelligent and learning robots, especially in the fields of swarm and humanoid robotics was established at his lab.*[16] The lab is equipped with a variety of mobile and flying robots and is one of the around 20 labs in the world owning an iCub humanoid robot. The group has applied a variety of machine learning algorithms, such as reinforcement learning and genetic programming, to improve adaptiveness and autonomy of robotic systems.

Recently his work on evolutionary robotics, with a focus on using genetic programming to evolve robotic skills, especially in robot vision have allowed for quick and robust object detection in humanoid robots.*[17]*[18]*[19] IDSIA's work with the iCub humanoid won the 2013 AAAI Student Video competition.*[20]*[21]

21.2 References

[1] 2012 Kurzweil AI Interview with Jürgen Schmidhuber on the eight competitions won by his Deep Learning team 2009-2012

[2] S. Hochreiter and J. Schmidhuber. Long Short-Term Memory. Neural Computation, 9(8):1735–1780, 1997.

[3] D. C. Ciresan, U. Meier, L. M. Gambardella, J. Schmidhuber. Deep Big Simple Neural Nets For Handwritten Digit Recognition. Neural Computation 22(12): 3207-3220.

[4] A. Graves, M. Liwicki, S. Fernandez, R. Bertolami, H. Bunke, J. Schmidhuber. A Novel Connectionist System for Improved Unconstrained Handwriting Recognition. IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 31, no. 5, 2009.

[5] J. Schmidhuber. A local learning algorithm for dynamic feedforward and recurrent networks. Connection Science, 1(4):403–412, 1989

[6] J. Schmidhuber. Curious model-building control systems. In Proc. International Joint Conference on Neural Networks, Singapore, volume 2, pages 1458–1463. IEEE, 1991

[7] J. Schmidhuber. Formal Theory of Creativity, Fun, and Intrinsic Motivation (1990–2010). IEEE Transactions on Autonomous Mental Development, 2(3):230–247, 2010.

[8] J. Schmidhuber. Learning factorial codes by predictability minimization. Neural Computation, 4(6):863–879, 1992

[9] J. Schmidhuber. A computer scientist's view of life, the universe, and everything. Foundations of Computer Science: Potential – Theory – Cognition, Lecture Notes in Computer Science, pages 201–208, Springer, 1997

[10] Brian Greene, Chapter 10 of: The Hidden Reality: Parallel Universes and the Deep Laws of the Cosmos, Knopf, 2011

[11] J. Schmidhuber. The Speed Prior: A New Simplicity Measure Yielding Near-Optimal Computable Predictions. Proceedings of the 15th Annual Conference on Computational Learning Theory (COLT 2002), Sydney, Australia, LNAI, 216–228, Springer, 2002

[12] J. Schmidhuber. Hierarchies of generalized Kolmogorov complexities and nonenumerable universal measures computable in the limit. International Journal of Foundations of Computer Science 13(4):587–612, 2002

[13] J. Schmidhuber. Ultimate Cognition à la Gödel. Cognitive Computation 1(2):177–193, 2009

[14] J. Schmidhuber. Optimal Ordered Problem Solver. Machine Learning, 54, 211–254, 2004

[15] J. Schmidhuber. Low-Complexity Art. Leonardo, Journal of the International Society for the Arts, Sciences, and Technology, 30(2):97–103, MIT Press, 1997

[16] http://robotics.idsia.ch/ The IDSIA Robotics Lab

[17] J. Leitner, S. Harding, P. Chandrashekhariah, M. Frank, A. Förster, J. Triesch and J. Schmidhuber. Learning Visual Object Detection and Localisation Using icVision. Biologically Inspired Cognitive Architectures, Vol. 5, 2013.

[18] J. Leitner, S. Harding, M. Frank, A. Förster and J. Schmidhuber. Humanoid Learns to Detect Its Own Hands. IEEE Congress on Evolutionary Computing (CEC), 2013.

[19] S. Harding, J. Leitner and J. Schmidhuber. Cartesian Genetic Programming for Image Processing (CGP-IP). Genetic Programming Theory and Practice X (Springer Tract on Genetic and Evolutionary Computation). pp 31-44. ISBN 978-1-4614-6845-5. Springer, Ann Arbor, 2013.

[20] http://www.aaaivideos.org/2013/ AAAI Video Competition 2013.

[21] M. Stollenga, L. Pape, M. Frank, J. Leitner, A. Förster and J. Schmidhuber. Task-Relevant Roadmaps: A Framework for Humanoid Motion Planning. IROS, 2013.

21.3 Sources

- Google Scholar: Numerous scientific articles referencing Schmidhuber's work
- Scholarpedia article on Universal Search, discussing Schmidhuber's Speed Prior, Optimal Ordered Problem Solver, Gödel machine
- German article on Schmidhuber in CIO magazine: "Der ideale Wissenschaftler" (the ideal scientist)
- Build An Optimal Scientist, Then Retire: Interview with J. Schmidhuber in H+ magazine, 2010
- Video of Schmidhuber's talk on artificial curiosity and creativity at the Singularity Summit 2009, NYC
- TV clip: Schmidhuber on computable universes on *Through the Wormhole* with Morgan Freeman.
- On-going research on the iCub humanoid at the IDSIA Robotics Lab

21.4 External links

- Home page
- Publications
- Videos of Juergen Schmidhuber & the Swiss AI Lab IDSIA

Chapter 22

Jeff Dean (computer scientist)

Jeffrey Adgate "Jeff" Dean (born 1968) is an American computer scientist and software engineer. He is currently a Google Senior Fellow in the Systems and Infrastructure Group.

22.1 Personal life and education

Dean received a Ph.D. in Computer Science from the University of Washington, working with Craig Chambers on whole-program optimization techniques for object-oriented languages. He received a B.S., *summa cum laude* from the University of Minnesota in Computer Science & Economics in 1990. He was elected to the National Academy of Engineering in 2009, which recognized his work on "the science and engineering of large-scale distributed computer systems."

22.2 Career in computer science

Prior to joining Google, he was at DEC/Compaq's Western Research Laboratory, where he worked on profiling tools, microprocessor architecture, and information retrieval.

Prior to graduate school, he worked at the World Health Organization's Global Programme on AIDS, developing software for statistical modeling and forecasting of the HIV/AIDS pandemic.

22.3 Career at Google

Dean joined Google in mid-1999, and is currently a Google Senior Fellow in the Systems Infrastructure Group. While at Google, he has designed and implemented large portions of the company's advertising, crawling, indexing and query serving systems, along with various pieces of the distributed computing infrastructure that sits underneath most of Google's products. At various times, he has also worked on improving search quality, statistical machine translation, and various internal software development tools and has had significant involvement in the engineering hiring process.

Among others, the projects he's worked on include:

- Spanner - a scalable, multi-version, globally distributed, and synchronously replicated database
- Some of the production system design and statistical machine translation system for Google Translate.
- BigTable, a large-scale semi-structured storage system.
- MapReduce a system for large-scale data processing applications.
- Google Brain a system for large-scale artificial neural networks

22.4 Awards and honors

- Elected to the National Academy of Engineering (2009)
- Fellow of the Association for Computing Machinery (2009)
- ACM-Infosys Foundation Award (2012)

22.5 Major publications

- Jeffrey Dean and Sanjay Ghemawat. 2004. MapReduce: Simplified Data Processing on Large Clusters. OSDI'04: Sixth Symposium on Operating System Design and Implementation (December 2004)

22.6 See also

- Big Table
- MapReduce
- Spanner

22.7 External links

- Jeff Dean's Google home page.
- Meet Google's Baddest Engineer, Jeff Dean
- The Optimizer

Chapter 23

Andrew Ng

Andrew Yan-Tak Ng (Chinese: 吳恩達; born 1976) is Chief Scientist at Baidu Research in Silicon Valley. In addition, he is an associate professor in the Department of Computer Science and the Department of Electrical Engineering by courtesy at Stanford University. He is chairman of the board of Coursera, an online education platform that he co-founded with Daphne Koller.

He researches primarily in machine learning and deep learning. His early work includes the Stanford Autonomous Helicopter project, which developed one of the most capable autonomous helicopters in the world,[*][2][*][3] and the STAIR (STanford Artificial Intelligence Robot) project,[*][4] which resulted in ROS, a widely used open-source robotics software platform.

Ng is also the author or co-author of over 100 published papers in machine learning, robotics and related fields, and some of his work in computer vision has been featured in a series of press releases and reviews.[*][5] In 2008, he was named to the MIT *Technology Review* TR35 as one of the top 35 innovators in the world under the age of 35.[*][6][*][7] In 2007, Ng was awarded a Sloan Fellowship. For his work in Artificial Intelligence, he is also a recipient of the Computers and Thought Award.

On May 16, 2014, Ng announced from his Coursera blog that he would be stepping away from his day-to-day responsibilities at Coursera, and join Baidu as Chief Scientist, working on deep learning.[*][8]

23.1 Machine learning research

In 2011, Ng founded the Google Brain project at Google, which developed very large scale artificial neural networks using Google's distributed computer infrastructure.[*][9] Among its notable results was a neural network trained using deep learning algorithms on 16,000 CPU cores, that learned to recognize higher-level concepts, such as cats, after watching only YouTube videos, and without ever having been told what a "cat" is.[*][10][*][11] The project's technology is currently also used in the Android Operating System's speech recognition system.[*][12]

23.2 Online education

Ng started the Stanford Engineering Everywhere (SEE) program, which in 2008 placed a number of Stanford courses online, for free. Ng taught one of these courses, Machine Learning, which consisted of video lectures by him, along with the student materials used in the Stanford CS229 class.

The "applied" version of the Stanford class (CS229a) was hosted on ml-class.org and started in October 2011, with over 100,000 students registered for its first iteration; the course featured quizzes and graded programming assignments and became one of the first successful MOOCs made by Stanford professors.[*][14] His work subsequently led to the founding of Coursera in 2012.

23.3 Personal life

Ng was born in the UK in 1976. His parents were both Hongkongers. He spent time in Hong Kong and Singapore[*][1] and later graduated from Raffles Institution in Singapore as the class of 1992 and received his undergraduate degree in computer science from Carnegie Mellon University in Pittsburgh, Pennsylvania as the class of 1997. Then, he attained his master's degree from Massachusetts Institute of Technology in Cambridge, Massachusetts as the class of 1998 and received his PhD from University of California, Berkeley in 2002. He started working at Stanford University during that year; he currently lives in Palo Alto, California. He married Carol E. Reiley in 2014.

23.4 References

[1] Seligman, Katherine (3 December 2006). "If Andrew Ng could just get his robot to assemble an Ikea bookshelf, we'd all buy one". *SFGate*. Retrieved 12 February 2013.

[2] "From Self-Flying Helicopters to Classrooms of the Future". *Chronicle of Higher Education*. 2012.

[3] "Stanford Autonomous Helicopter Project".

[4] John Markoff (18 July 2006). "Brainy Robots Start Stepping Into Daily Life". *New York Times*.

[5] New algorithm improves robot vision

[6] "2008 Young Innovators Under 35". Technology Review. 2008. Retrieved August 15, 2011.

[7] Technology Review: TR35

[8] "A personal message from Co-founder Andrew Ng". Coursera blog. 2014. Retrieved May 16, 2014.

[9] Claire Miller and Nick Bilton (3 November 2011). "Google's Lab of Wildest Dreams". *New York Times*.

[10] John Markoff (25 June 2012). "How Many Computers to Identify a Cat? 16,000.". *New York Times*.

[11] Ng, Andrew; Dean, Jeff (2012). "Building High-level Features Using Large Scale Unsupervised Learning" (PDF).

[12] "Speech Recognition and Deep Learning". *Google Research Blog*. Google. 6 August 2012. Retrieved 29 January 2013.

[13] "Interview with Coursera Co-Founder Andrew Ng". Degree of Freedom. Retrieved May 19, 2013.

[14] Theresa Johnson. "Stanford for All". *Stanford Magazine*.

23.5 See also

- Robot Operating System

23.6 External links

- Homepage
- STAIR Homepage
- Publications
- Academic Genealogy
- Machine Learning (CS 229) Video Lecture
- Lecture videos
- From Self-Flying Helicopters to Classrooms of the Future
- Coursera-Leadership

23.7 Text and image sources, contributors, and licenses

23.7.1 Text

- **Artificial neural network** *Source:* http://en.wikipedia.org/wiki/Artificial%20neural%20network?oldid=662892335 *Contributors:* Magnus Manske, Ed Poor, Iwnbap, PierreAbbat, Youandme, Susano, Hfastedge, Mrwojo, Michael Hardy, Erik Zachte, Oliver Pereira, Bobby D. Bryant, Zeno Gantner, Parmentier~enwiki, Delirium, Pieter Suurmond, (, Alfio, 168..., Ellywa, Ronz, Snoyes, Den fjättrade ankan~enwiki, Cgs, Glenn, Cyan, Hike395, Hashar, Novum, Charles Matthews, Guaka, Timwi, Reddi, Andrewman327, Munford, Furrykef, Bevo, Fvw, Raul654, Nyxos, Unknown, Pakcw, Robbot, Chopchopwhitey, Bkell, Hadal, Wikibot, Diberri, Xanzzibar, Wile E. Heresiarch, Connelly, Giftlite, Rs2, Markus Krötzsch, Spazzm, Seabhcan, BenFrantzDale, Zigger, Everyking, Rpyle731, Wikiwikifast, Foobar, Edrex, Jabowery, Wildt~enwiki, Wmahan, Neilc, Quadell, Beland, Lylum, Gene s, Sbledsoe, Mozzerati, Karl-Henner, Jmeppley, Asbestos, Fintor, AAAAA, Splatty, Rich Farmbrough, Pak21, NeuronExMachina, Michal Jurosz, Pjacobi, Mecanismo, Zarutian, Dbachmann, Bender235, ZeroOne, Violetriga, Mavhc, One-dimensional Tangent, Gyll, Stephane.magnenat, Mysteronald, .:Ajvol:., Fotinakis, Nk, Tritium6, JesseHogan, Mdd, Passw0rd, Zachlipton, Alansohn, Jhertel, Anthony Appleyard, Denoir, Arthena, Fritz Saalfeld, Sp00n17, Rickyp, Hu, Tyrell turing, Cburnett, Notjim, Drbreznjev, Forderud, Oleg Alexandrov, Mogigoma, Madmardigan53, Justinlebar, Olethros, Ylem, Dr.U, Gengiskanhg, Male1979, Bar0n, Waldir, Eslip17, Yoghurt, Ashmoo, Graham87, Qwertyus, Imersion, Grammarbot, Rjwilmsi, Jeema, Venullian, SpNeo, Intgr, Predictor, Kri, BradBeattie, Plarroy, Windharp, Mehran.asadi, Commander Nemet, Wavelength, Borgx, IanManka, Rsrikanth05, Philopedia, Ritchy, David R. Ingham, Grafen, Nrets, Exir Kamalabadi, Deodar~enwiki, Mosquitopsu, Jpbowen, Dennis!, JulesH, Moe Epsilon, Supten, DeadEyeArrow, Eclipsed, SamuelRiv, Tribaal, Chase me ladies, I'm the Cavalry, CWenger, Donhalcon, Banus, Shepard, John Broughton, A13ean, SmackBot, PinstripeMonkey, McGeddon, CommodiCast, Jfmiller28, Stimpy, Commander Keane bot, Feshmania, ToddDeLuca, Diegotorquemada, Patrickdepinguin, KYN, Gilliam, Bluebot, Oli Filth, Gardoma, Complexica, Nossac, Hongooi, Pdtl, Izhikevich, Trifon Triantafillidis, SeanAhern, Neshatian, Vernedj, Dankonikolic, Rory096, Sina2, SS2005, Kuru, Plison, Lakinekaki, Bjankuloski06en~enwiki, IronGargoyle, WMod-NS, Dicklyon, Citicat, StanfordProgrammer, Ojan, Chi3x10, Aeternus, CapitalR, Atreys, George100, Gveret Tered, Devourer09, SkyWalker, CmdrObot, Leonoel, CBM, Mcstrother, MarsRover, CX, Arauzo, Peterdjones, Josephorourke, Kozuch, ClydeC, NotQuiteEXPComplete, Irigi, Mbell, Oldiowl, Tolstoy the Cat, Headbomb, Mitchell.E.Timin, Davidhorman, Sbandrews, KrakatoaKatie, QuiteUnusual, Prolog, AnAj, LinaMishima, Whenning, Hamaryns, Daytona2, JAnDbot, MER-C, Dcooper, Extropian314, Magioladitis, VoABot II, Amitant, Jimjamjak, SSZ, Robotman1974, David Eppstein, User A1, Martynas Patasius, Pmbhagat, JaGa, Tuhinsubhrakonar, SoyYo, Nikoladie~enwiki, R'n'B, Maproom, K.menin, Gill110951, Tarotcards, Plasticup, Margareta, Paskari, Jamesontai, Kiran uvpce, Jamiejoseph, Error9312, Jlaramee, Jeff G., A4bot, Singleheart, Ebbedc, Lordvolton, Ask123, CanOfWorms, Mundhenk, Wikiisawesome, M karamanov, Enkya, Blumenkraft, Twikir, Mikemoral, Oldag07, Smsarmad, Flyer22, Janopus, Bwieliczko, Dhatfield, F.j.gaze, Mark Lewis Epstein, S2000magician, PuercoPop, Martarius, ClueBot, Ignacio Javier Igjav, Ahyeek, The Thing That Should Not Be, Fadesga, Zybler, Midiangr, Epsilon60198, Thomas Tvileren, Wduch, Excirial, Three-quarter-ten, Skbkekas, Chaosdruid, Aprock, Qwfp, Jean-claude perez, Achler, XLinkBot, AgnosticPreachersKid, BodhisattvaBot, Stickee, Cmr08, Porphyro, Fippy Darkpaw, Addbot, DOI bot, AndrewHZ, Thomblake, Techjerry, Looie496, MrOllie, Transmobilator, Jarble, Yobot, Blm19732008, Nguyengiap84~enwiki, SparkOfCreation, AnomieBOT, DemocraticLuntz, Tryptofish, Trevithj, Jim1138, Durran65, MockDuck, JonathanWilliford, Materialscientist, Citation bot, Eumolpo, Twri, NFD9001, Isheden, J04n, Omnipaedista, Mark Schierbecker, RibotBOT, RoodyBeep, Gunjan verma81, FrescoBot, X7q, Ömer Cengiz Çelebi, Outback the koala, Citation bot 1, Tylor.Sampson, Calmer Waters, Skyerise, Trappist the monk, Krassotkin, Cjlim, Fox Wilson, The Strategist, LilyKitty, Eparo, בן גרשון, Jfmantis, Mehdiabbasi, VernoWhitney, Wiknn, BertSeghers, DASHBot, EmausBot, Nacopt, Dzkd, Racerx11, Japs 88, GoingBatty, RaoInWiki, Roposeidon, Epsiloner, Stheodor, Benlansdell, Radshashi, K6ka, D'oh!, Thisisentchris87, Aavindraa, Chire, Glosser.ca, IGeMiNix, Donner60, Yoshua.Bengio, Shinosin, Venkatarun95, ChuckNorrisPwnedYou, Petrb, ClueBot NG, Raghith, Robiminer, Snotbot, Tideflat, Frietjes, Gms3591, Ryansandersuk, Widr, MerlIwBot, Helpful Pixie Bot, Trepier, BG19bot, Thwien, Adams7, Rahil2000, Michaelmalak, Compfreak7, Kirananils, Altaïr, J.Davis314, Attleboro, Pratyya Ghosh, JoshuSasori, Ferrarisailor, Eugenecheung, Mtschida, ChrisGualtieri, Dave2k6inthemix, Whebzy, APerson, JurgenNL, Oritnk, Stevebillings, Djfrost711, Sa publishers, 剋, Mark viking, Markus.harz, Deeper Learning, Vinchaud20, Soueumxm, Toritris, Evolution and evolvability, Sboddhu, Sharva029, Paheld, Putting things straight, Rosario Berganza, Monkbot, Buggiehuggie, Santoshwriter, Likerhayter, Joma.huguet, Bclark401, Rahulpratapsingh06, Donkeychee, Michaelwine, Xsantostill, Jorge Guerra Pires, Wfwhitney, Loïc Bourgois, KasparBot and Anonymous: 488

- **Deep learning** *Source:* http://en.wikipedia.org/wiki/Deep%20learning?oldid=662755651 *Contributors:* The Anome, Ed Poor, Michael Hardy, Meekohi, Glenn, Bearcat, Nandhp, Giraffedata, Jonsafari, Oleg Alexandrov, Justin Ormont, BD2412, Qwertyus, Rjwilmsi, Kri, Bgwhite, Tomdooner, Bhny, Malcolma, Arthur Rubin, Mebden, SeanAhern, Dicklyon, JHP, ChrisCork, Lfstevens, A3nm, R'n'B, Like.liberation, Popoki, Jshrager, Strife911, Bfx0, Daniel Hershcovich, Pinkpedaller, Dthomsen8, Addbot, Mamikonyana, Yobot, AnomieBOT, Jonesey95, Zabbarob, Wyverald, RjwilmsiBot, Larry.europe, Helwr, GoingBatty, Sergey WereWolf, SlowByte, Yoshua.Bengio, JuyangWeng, Renklauf, Bittenus, Widr, BG19bot, Lukas.tencer, Kareltje63, Synchronist, Gameboy97q, IjonTichyIjonTichy, Mogism, AlwaysCoding, Mark viking, Cagarie, Deeper Learning, Prisx, Wikiyant, Underflow42, GreyShields, Opokopo, Prof. Oundest, Gigavanti, Sevensharpnine, Yes deeper, Monkbot, Chieftains337, Samueldg89, Nikunj157, Engheta, Aspurdy, Velvel2, Deng629, Zhuikov, Stergioc, Jerodlycett, DragonbornXXL and Anonymous: 94

- **Feature learning** *Source:* http://en.wikipedia.org/wiki/Feature%20learning?oldid=661746836 *Contributors:* Phil Boswell, Tobias Bergemann, Qwertyus, Rjwilmsi, Mcld, Kotabatubara, Dsimic, Yobot, AnomieBOT, BG19bot, Mavroudisv, TonyWang0316, Ixjlyons and Anonymous: 7

- **Unsupervised learning** *Source:* http://en.wikipedia.org/wiki/Unsupervised%20learning?oldid=660135356 *Contributors:* Michael Hardy, Kku, Alfio, Ahoerstemeier, Hike395, Ojigiri~enwiki, Gene s, Urhixidur, Alex Kosorukoff, Aaronbrick, Bobo192, 3mta3, Tablizer, Denoir, Nkour, Qwertyus, Rjwilmsi, Chobot, Roboto de Ajvol, YurikBot, Darker Dreams, Daniel Mietchen, SmackBot, CommodiCast, Trebor, DHN-bot~enwiki, Lambiam, CRGreathouse, Carstensen, Thijs!bot, Jaxelrod, AnAj, Peteymills, David Eppstein, Agentesegreto, Maheshbest, Timohonkela, Ng.j, EverGreg, Algorithms, Kotsiantis, Auntof6, PixelBot, Edg2103, Addbot, EjsBot, Yobot, Les boys, AnomieBOT, Salvamoreno, D'ohBot, Skyerise, Ranjan.acharyya, BertSeghers, EmausBot, Fly by Night, Rotcaeroib, Stheodor, Daryakav, Ida Shaw, Chire, Candace Gillhoolley, WikiMSL, Helpful Pixie Bot, Majidjanz and Anonymous: 40

- **Generative model** *Source:* http://en.wikipedia.org/wiki/Generative%20model?oldid=660109222 *Contributors:* Awaterl, Michael Hardy, Hike395, Andrewman327, Benwing, Cagri, Rama, Serapio, Jonsafari, Male1979, Qwertyus, Zanetu, Dicklyon, Repied, Barticus88, RichardSocher~enwiki, Camrn86, YinZhang, Melcombe, Lfriedl, Shanttashjean, Omnipaedista, Geoffrey I Webb, Naxingyu, DanielWaterworth, Mekarpeles, ClueBot NG, Gilgoldm, C2oo56, Hestendelin, L T T H U and Anonymous: 14

- **Neural coding** *Source:* http://en.wikipedia.org/wiki/Neural%20coding?oldid=662182422 *Contributors:* Ed Poor, Apraetor, AndrewKeenanRichardson, LindsayH, CanisRufus, Kghose, Jheald, Woohookitty, BD2412, Rjwilmsi, RDBrown, Colonies Chris, Henrikhenrik, OrphanBot, Radagast83, NickPenguin, Vina-iwbot~enwiki, Clicketyclack, RomanSpa, Sohale, Rji, Andorin, Goodwillein, Anthonyhcole, Nick Number, Davidm617617, Lova Falk, Addbot, Looie496, Luckas-bot, Yobot, Amirobot, Tryptofish, Citation bot, Xhuo, SassoBot, FrescoBot, Xbcj0843hck3, Albertzeyer, TjeerdB, Jonkerz, Helwr, Bethnim, ZéroBot, Ego White Tray, Bibcode Bot, BG19bot, FlinZ, ChrisGualtieri, Nicolenewell, Iamozy, Kernsters, Shirindora, Phleg1, Monkbot, PghJesse, Giedroid, AngevineMiller and Anonymous: 33
- **Word embedding** *Source:* http://en.wikipedia.org/wiki/Word%20embedding?oldid=623976145 *Contributors:* Qwertyus, Daniel Hershcovich, Yobot and Citation bot
- **Deep belief network** *Source:* http://en.wikipedia.org/wiki/Deep%20belief%20network?oldid=650454380 *Contributors:* Glenn, Qwertyus, Kri, Like.liberation, Smuckola, Isthmuses, BG19bot, VivamusAmemus, Schurgast and Anonymous: 1
- **Convolutional neural network** *Source:* http://en.wikipedia.org/wiki/Convolutional%20neural%20network?oldid=662811606 *Contributors:* Glenn, Phil Boswell, Bearcat, GregorB, BD2412, Rjwilmsi, Mario23, Serg3d2, Mcld, Chris the speller, Frap, Dr.K., Lfstevens, TheSeven, Like.liberation, Att159, XLinkBot, Yobot, Anypodetos, AnomieBOT, Cugino di mio cugino, Citation bot, Dithridge, Jhbdel, Alvin Seville, Hobsonlane, RandomDSdevel, Arinelle, Peaceray, Osnetwork, BG19bot, BattyBot, Gameboy97q, APerson, MartinLjungqvist, Zieglerk, Monkbot, VivamusAmemus, Velvel2, Fight123456, Stri8ted, Gnagyusa, Xksteven and Anonymous: 30
- **Restricted Boltzmann machine** *Source:* http://en.wikipedia.org/wiki/Restricted%20Boltzmann%20machine?oldid=650005600 *Contributors:* Shd~enwiki, Glenn, Dratman, Qwertyus, Kri, Gareth Jones, Deepdraft, Mcld, Hongooi, Dicklyon, Abhineetnazi, Tomtheebomb, Dsimic, Arsi Warrior, Yobot, LilHelpa, UbaiSandouk, Gilo1969, Racerx11, Leopd, Enerjiparki, Sevensharpnine, Velvel2, Broido, Dharmablues and Anonymous: 22
- **Recurrent neural network** *Source:* http://en.wikipedia.org/wiki/Recurrent%20neural%20network?oldid=662220310 *Contributors:* Glenn, Barak~enwiki, Charles Matthews, Urhixidur, Aaronbrick, Tyrell turing, RJFJR, Alai, Male1979, DavidFarmbrough, Seliopou, Nehalem, Bhny, Rwalker, Banus, That Guy, From That Show!, SmackBot, Moxon, Yume149~enwiki, Charivari, Kissaki0, MichaelGasser, Terry Bollinger, Dicklyon, Fyedernoggersnodden, Thijs!bot, Daytona2, Curdeius, JaGa, Jamalex~enwiki, KylieTastic, Itb3d, Gdupont, Jwray, Headlessplatter, Daniel Hershcovich, DumZiBoT, SlaterDeterminant, Achler, Addbot, DOI bot, LatitudeBot, Roux, Fadyone, Yobot, AnomieBOT, Flewis, Citation bot, Omnipaedista, FrescoBot, Adrian Lange, Citation bot 1, Dmitry St, Epsiloner, Mister Mormon, DASHBotAV, ClueBot NG, RichardTowers, MerlIwBot, Helpful Pixie Bot, BG19bot, Frze, Justinyap88, Ivan Ukhov, طاها, Djfrost711, Randykitty, Jmander, Ukpg, Minky76, OhGodItsSoAmazing, Mknjbhvg, Slashdottir, Monkbot, DoubleDr, Jungkanji and Anonymous: 46
- **Long short term memory** *Source:* http://en.wikipedia.org/wiki/Long%20short%20term%20memory?oldid=659668087 *Contributors:* Michael Hardy, Glenn, Rich Farmbrough, Denoir, Woohookitty, SmackBot, Derek farn, Ninjakannon, Magioladitis, Barkeep, Pwoolf, Headlessplatter, M4gnum0n, Muhandes, Jncraton, Yobot, Dithridge, Omnipaedista, Albertzeyer, Silenceisgod, Epsiloner, Ego White Tray, Mister Mormon, Hmainsbot1, Mogism, Velvel2 and Anonymous: 13
- **Google Brain** *Source:* http://en.wikipedia.org/wiki/Google%20Brain?oldid=657438840 *Contributors:* Bearcat, Dfrankow, Vipul, Daranz, JorisvS, Dicklyon, Aeternus, Diaa abdelmoneim, AnomieBOT, Cnwilliams, Chire, BG19bot, Q6637p, Stephen Balaban, BattyBot, Blharp and Anonymous: 8
- **Google DeepMind** *Source:* http://en.wikipedia.org/wiki/Google%20DeepMind?oldid=654694254 *Contributors:* Ciphergoth, Bearcat, Dratman, Bgwhite, Robofish, Aeternus, Edwardx, Yellowdesk, Magioladitis, Touch Of Light, Steel1943, Ciphershort, Wikiisawesome, Sprachpfleger, Green Cardamom, Larry.europe, BG19bot, IjonTichyIjonTichy, Megab, William 2239, Bipper1024, Jon Jonathan, Harisonpop and Anonymous: 13
- **Torch (machine learning)** *Source:* http://en.wikipedia.org/wiki/Torch%20(machine%20learning)?oldid=661012919 *Contributors:* Qwertyus, Strife911, Niceguyedc, Larry.europe, BG19bot, Lor and Anonymous: 6
- **Theano (software)** *Source:* http://en.wikipedia.org/wiki/Theano%20(software)?oldid=641284109 *Contributors:* Bearcat, Qwertyus, Andre.holzner, Strife911, Mrocklin, MartinThoma, Turn685 and Monkbot
- **Deeplearning4j** *Source:* http://en.wikipedia.org/wiki/Deeplearning4j?oldid=662808219 *Contributors:* Rwalker, Cydebot, Like.liberation, NinjaRobotPirate, Daniel Hershcovich, Dawynn, Yobot, Wcherowi and Anonymous: 3
- **Gensim** *Source:* http://en.wikipedia.org/wiki/Gensim?oldid=651026231 *Contributors:* Thorwald, Qwertyus, Dawynn, Matěj Grabovský, Smk65536, Armbrust, Lightlowemon, Delusion23, Velvel2 and Anonymous: 10
- **Geoffrey Hinton** *Source:* http://en.wikipedia.org/wiki/Geoffrey%20Hinton?oldid=660779435 *Contributors:* Edward, Zeno Gantner, Rainer Wasserfuhr~enwiki, Flockmeal, MOiRe, Diberri, Risk one, Just Another Dan, Lawrennd, Rich Farmbrough, Davidswelt, Marudubshinki, Qwertyus, Rjwilmsi, Winterstein, Misterwindupbird, RussBot, Welsh, Gareth Jones, DaveWF, BorgQueen, InverseHypercube, CRKingston, Jsusskin, Onkelschark, OrphanBot, Jmlk17, Dl2000, Shoeofdeath, Aeternus, Adam Newbold, Cydebot, Michael Fourman, Roweis, Waacstats, Destynova, MetsBot, David Eppstein, STBot, FMAFan1990, AlexGreat, Sidcool1234, XLinkBot, Galyet, G7valera, Addbot, דוד ש, Omnipaedista, Plucas58, Morton Shumway, RjwilmsiBot, Larry.europe, Gumbys, Wpeaceout, Onionesque, BG19bot, ChrisGualtieri, Makecat-bot, Anne Delong, Silas Ropac, Putting things straight, Annaflagg, Justanother109, Jonarnold1985, Velvel2, Mathewk1300 and Anonymous: 24
- **Yann LeCun** *Source:* http://en.wikipedia.org/wiki/Yann%20LeCun?oldid=662361291 *Contributors:* Zeno Gantner, Klemen Kocjancic, Rich Farmbrough, Runner1928, Rschen7754, Bgwhite, RussBot, Gareth Jones, Jpbowen, Angoodkind, Cydebot, Studerby, Bongwarrior, Waacstats, 72Dino, Aboutmovies, Falcon8765, Profshadoko, Cajunbill, M4gnum0n, Berean Hunter, MystBot, Addbot, Yobot, Bunnyhop11, FrescoBot, Lbottou, Jvsarun1993, Uprightmanpower, JJRambo, Zzym, Hgsa5, AdjSilly, Polochicken, Ljjjp, Crickavery, Justanother109, Voltdye, Velvel2, Visionscholar, Mathewk1300, KasparBot, Algorith and Anonymous: 3
- **Jürgen Schmidhuber** *Source:* http://en.wikipedia.org/wiki/J%C3%BCrgen%20Schmidhuber?oldid=660776145 *Contributors:* Michael Hardy, Kosebamse, Hike395, Charles Matthews, Wik, Randomness~enwiki, Goedelman, Psb777, Newbie~enwiki, Juxi, Robin klein, Klemen Kocjancic, D6, On you again, Kelly Ramsey, Rpresser, Derumi, Woohookitty, GregorB, Male1979, FlaBot, Bgwhite, RussBot, SmackBot, Hongooi, Ben Moore, Phoxhat, JimStyle61093475, IDSIAupdate, Mpotse, Blaisorblade, Marek69, A1bb22, Waacstats, JaGa, Gwern, Kornfan71, R'n'B, DadaNeem, BOTijo, Noveltyghost, Tbsdy lives, Addbot, Lightbot, Yobot, Fleabox, Omnipaedista, Underlying lk, RjwilmsiBot, Angrytoast, Epsiloner, Helpful Pixie Bot, VIAFbot, It's here!, Midnightplunge, Laanaae, Hildensia, KasparBot and Anonymous: 16

- **Jeff Dean (computer scientist)** *Source:* http://en.wikipedia.org/wiki/Jeff%20Dean%20(computer%20scientist)?oldid=657563156 *Contributors:* Klemen Kocjancic, Lockley, AntonioDsouza, Wavelength, Gareth Jones, Elkman, CommonsDelinker, Aboutmovies, Mr.Z-bot, Omnipaedista, Patchy1, Iamjamesbond007, BG19bot, Stephen Balaban, Stephenbalaban, Timothy Gu, ChrisGualtieri, Mogism and Anonymous: 9
- **Andrew Ng** *Source:* http://en.wikipedia.org/wiki/Andrew%20Ng?oldid=659094106 *Contributors:* Michael Hardy, Msm, Simon Lacoste-Julien, Prenju, Klemen Kocjancic, Rich Farmbrough, Vipul, Rd232, Mr Tan, Mandarax, Alex Bakharev, Gareth Jones, Pyronite, Johndburger, InverseHypercube, Smallbones, Raysonho, CmdrDan, Cydebot, Batra, Utopiah, Magioladitis, Rootxploit, CommonsDelinker, Coolg49964, Bcnof, Hoising, Maghnus, XKL, Arbor to SJ, Martarius, Chaosdruid, XLinkBot, Addbot, Mortense, Yobot, Azylber, AnomieBOT, Chrisvanlang, Omnipaedista, Velblod, FrescoBot, Abductive, Amhey, RjwilmsiBot, Mmm333k, ZéroBot, Bemanna, Clue-Bot NG, Kashthealien, Geistcj, Rrrlf, ArmbrustBot, Happyhappy001, E8xE8, Gnomy7, Linuxjava, AcrO O, Calisacole, Velvel2, Vision-scholar, Demdim0, Csisawesome and Anonymous: 41

23.7.2 Images

- **File:Ambox_important.svg** *Source:* http://upload.wikimedia.org/wikipedia/commons/b/b4/Ambox_important.svg *License:* Public domain *Contributors:* Own work, based off of Image:Ambox scales.svg *Original artist:* Dsmurat (talk · contribs)
- **File:Ann_dependency_(graph).svg** *Source:* http://upload.wikimedia.org/wikipedia/commons/d/dd/Ann_dependency_%28graph%29.svg *License:* CC BY-SA 3.0 *Contributors:* Vector version of File:Ann dependency graph.png *Original artist:* Glosser.ca
- **File:Colored_neural_network.svg** *Source:* http://upload.wikimedia.org/wikipedia/commons/4/46/Colored_neural_network.svg *License:* CC BY-SA 3.0 *Contributors:* Own work, Derivative of File:Artificial neural network.svg *Original artist:* Glosser.ca
- **File:Commons-logo.svg** *Source:* http://upload.wikimedia.org/wikipedia/en/4/4a/Commons-logo.svg *License:* ? *Contributors:* ? *Original artist:* ?
- **File:Deep_belief_net.svg** *Source:* http://upload.wikimedia.org/wikipedia/commons/f/fa/Deep_belief_net.svg *License:* CC0 *Contributors:* Own work *Original artist:* Qwertyus
- **File:Edit-clear.svg** *Source:* http://upload.wikimedia.org/wikipedia/en/f/f2/Edit-clear.svg *License:* Public domain *Contributors:* The *Tango! Desktop Project*. *Original artist:*

  The people from the Tango! project. And according to the meta-data in the file, specifically: "Andreas Nilsson, and Jakub Steiner (although minimally)."
- **File:Elman_srnn.png** *Source:* http://upload.wikimedia.org/wikipedia/commons/8/8f/Elman_srnn.png *License:* CC BY 3.0 *Contributors:* Own work *Original artist:* Fyedernoggersnodden
- **File:Emoji_u1f4bb.svg** *Source:* http://upload.wikimedia.org/wikipedia/commons/d/d7/Emoji_u1f4bb.svg *License:* Apache License 2.0 *Contributors:* https://code.google.com/p/noto/ *Original artist:* Google
- **File:Fisher_iris_versicolor_sepalwidth.svg** *Source:* http://upload.wikimedia.org/wikipedia/commons/4/40/Fisher_iris_versicolor_sepalwidth.svg *License:* CC BY-SA 3.0 *Contributors:* en:Image:Fisher iris versicolor sepalwidth.png *Original artist:* en:User:Qwfp (original); Pbroks13 (talk) (redraw)
- **File:Folder_Hexagonal_Icon.svg** *Source:* http://upload.wikimedia.org/wikipedia/en/4/48/Folder_Hexagonal_Icon.svg *License:* Cc-by-sa-3.0 *Contributors:* ? *Original artist:* ?
- **File:Free_Software_Portal_Logo.svg** *Source:* http://upload.wikimedia.org/wikipedia/commons/3/31/Free_and_open-source_software_logo_%282009%29.svg *License:* Public domain *Contributors:* FOSS Logo.svg *Original artist:* Free Software Portal Logo.svg (FOSS Logo.svg): ViperSnake151
- **File:Gensim_logo.png** *Source:* http://upload.wikimedia.org/wikipedia/en/b/b1/Gensim_logo.png *License:* Fair use *Contributors:* http://radimrehurek.com/gensim/_static/images/logo-gensim.png *Original artist:* ?
- **File:Internet_map_1024.jpg** *Source:* http://upload.wikimedia.org/wikipedia/commons/d/d2/Internet_map_1024.jpg *License:* CC BY 2.5 *Contributors:* Originally from the English Wikipedia; description page is/was here. *Original artist:* The Opte Project
- **File:Lstm_block.svg** *Source:* http://upload.wikimedia.org/wikipedia/en/8/8d/Lstm_block.svg *License:* PD *Contributors:*

  Headlessplatter (talk) (Uploads) *Original artist:*

  Headlessplatter (talk) (Uploads)
- **File:NoisyNeuralResponse.png** *Source:* http://upload.wikimedia.org/wikipedia/en/6/66/NoisyNeuralResponse.png *License:* PD *Contributors:* ? *Original artist:* ?
- **File:Nuvola_apps_arts.svg** *Source:* http://upload.wikimedia.org/wikipedia/commons/e/e2/Nuvola_apps_arts.svg *License:* GFDL *Contributors:* Image:Nuvola apps arts.png *Original artist:* Manco Capac
- **File:P_vip.svg** *Source:* http://upload.wikimedia.org/wikipedia/en/6/69/P_vip.svg *License:* PD *Contributors:* ? *Original artist:* ?
- **File:People_icon.svg** *Source:* http://upload.wikimedia.org/wikipedia/commons/3/37/People_icon.svg *License:* CC0 *Contributors:* OpenClipart *Original artist:* OpenClipart
- **File:PopulationCode.svg** *Source:* http://upload.wikimedia.org/wikipedia/commons/a/a1/PopulationCode.svg *License:* Public domain *Contributors:* Image:PopulationCode.png at English Wikipedia *Original artist:*
- Original: AndrewKeenanRichardson
- **File:Portal-puzzle.svg** *Source:* http://upload.wikimedia.org/wikipedia/en/f/fd/Portal-puzzle.svg *License:* Public domain *Contributors:* ? *Original artist:* ?
- **File:Question_book-new.svg** *Source:* http://upload.wikimedia.org/wikipedia/en/9/99/Question_book-new.svg *License:* Cc-by-sa-3.0 *Contributors:*

  Created from scratch in Adobe Illustrator. Based on Image:Question book.png created by User:Equazcion *Original artist:*

  Tkgd2007

- **File:Recurrent_ann_dependency_graph.png** *Source:* http://upload.wikimedia.org/wikipedia/commons/7/79/Recurrent_ann_dependency_graph.png *License:* CC-BY-SA-3.0 *Contributors:* ? *Original artist:* ?
- **File:Restricted_Boltzmann_machine.svg** *Source:* http://upload.wikimedia.org/wikipedia/commons/e/e8/Restricted_Boltzmann_machine.svg *License:* CC BY-SA 3.0 *Contributors:* Own work *Original artist:* Qwertyus
- **File:Science-symbol-2.svg** *Source:* http://upload.wikimedia.org/wikipedia/commons/7/75/Science-symbol-2.svg *License:* CC BY 3.0 *Contributors:* en:Image:Science-symbol2.png *Original artist:* en:User:AllyUnion, User:Stannered
- **File:Synapse_deployment.jpg** *Source:* http://upload.wikimedia.org/wikipedia/en/2/22/Synapse_deployment.jpg *License:* CC-BY-SA-2.5 *Contributors:* ? *Original artist:* ?
- **File:Text_document_with_red_question_mark.svg** *Source:* http://upload.wikimedia.org/wikipedia/commons/a/a4/Text_document_with_red_question_mark.svg *License:* Public domain *Contributors:* Created by bdesham with Inkscape; based upon Text-x-generic.svg from the Tango project. *Original artist:* Benjamin D. Esham (bdesham)
- **File:Torch_2014_logo.png** *Source:* http://upload.wikimedia.org/wikipedia/en/f/f5/Torch_2014_logo.png *License:* Fair use *Contributors:* https://github.com/torch *Original artist:* ?
- **File:Wave.svg** *Source:* http://upload.wikimedia.org/wikipedia/commons/4/40/Wave.svg *License:* BSD *Contributors:* http://duke.kenai.com/wave/index.html (new), https://duke.dev.java.net/images/wave/index.html (old) *Original artist:* sbmehta converted to SVG from Sun Microsystems AI version.
- **File:Wikibooks-logo-en-noslogan.svg** *Source:* http://upload.wikimedia.org/wikipedia/commons/d/df/Wikibooks-logo-en-noslogan.svg *License:* CC BY-SA 3.0 *Contributors:* Own work *Original artist:* User:Bastique, User:Ramac et al.
- **File:Wikiversity-logo.svg** *Source:* http://upload.wikimedia.org/wikipedia/commons/9/91/Wikiversity-logo.svg *License:* CC BY-SA 3.0 *Contributors:* Snorky (optimized and cleaned up by verdy_p) *Original artist:* Snorky (optimized and cleaned up by verdy_p)

23.7.3 Content license

- Creative Commons Attribution-Share Alike 3.0