

4 Neural Computing

Neural Computing, e.g. Artificial Neural Networks, is one of the most interesting and rapidly growing areas of research, attracting researchers from a wide variety of scientific disciplines. Starting from the basics, Neural Computing covers all the major approaches, putting each in perspective in terms of their capabilities, advantages, and disadvantages.

An Artificial Neural Network (ANN) is an information processing paradigm that is inspired by the way of biological nervous systems, such as the brain, process information. The key element of this paradigm is the structure of the information processing system. It is composed of a large number of highly interconnected processing elements (neurons) working in unison to solve specific problems. ANNs, like people, learn by example. An ANN is configured for a specific application, such as pattern recognition or data classification, through a learning process. Learning in biological systems involves adjustments to the synaptic connections that exist between the neurones. This is true of ANNs as well.

4.1 The brain as an information processing system

The human brain contains about 10 billion nerve cells, or neurons. On average, each neuron is connected to other neurons through about 10 000 synapses. (The actual figures vary greatly, depending on the local neuroanatomy.) The brain's network of neurons forms a massively parallel information processing system. This contrasts with conventional computers in which a single processor executes a single series of instructions.



	processing elements	element size	energy use	processing speed	style of computation	fault tolerant	learns	intelligent, conscious
	10 ¹⁴ synapses	10 ⁻⁶ m	30 W	100 Hz	parallel, distributed	yes	yes	usually
	10 ⁸ transistors	10 ⁻⁶ m	30 W (CPU)	10 ⁹ Hz	serial, centralized	no	a little	not (yet)

Table 5: Description of selected DE Strategies (adapted from <http://www.idsia.ch>)

Against this, consider the time taken for each elementary operation: neurons typically operate at a maximum rate of about 100 Hz, while a conventional CPU carries out several hundred million machine level operations per second. Despite of being built with very slow hardware, the brain has quite remarkable capabilities:

- Its performance tends to degrade gracefully under partial damage. In contrast, most programs and engineered systems are brittle: if you remove some arbitrary parts, very likely the whole will cease to function.
- It can learn (reorganize itself) from experience.
- This means that partial recovery from damage is possible if healthy units can learn to take over the functions previously carried out by the damaged areas.
- It performs massively parallel computations extremely efficiently. For example, complex visual perception occurs within less than 100 ms, that is, 10 processing steps!

As a discipline of Artificial Intelligence, Neural Networks attempt to bring computers a little closer to the brain's capabilities by imitating certain aspects of information processing in the brain, in a highly simplified way. The comparison of computer and brain abilities is shown in Table 5.

The brain is not homogeneous. At the largest anatomical scale, we distinguish cortex, midbrain, brainstem, and cerebellum. Each of these can be hierarchically subdivided into many regions, and areas within each region, either according to the anatomical structure of the neural networks within it, or according to the function performed by them. The overall pattern of projections (bundles of neural connections) between areas is extremely complex, and only partially known. The best mapped (and largest) system in the human brain is the visual system, where the first 10 or 11 processing stages have been identified. We distinguish feedforward projections that go from earlier processing stages (near the sensory input) to later ones (near the motor output), from feedback connections that go in the opposite direction. In addition to these long-range connections, neurons also link up with many thousands of their neighbours. In this way they form very dense, complex local networks.

The basic computational unit in the nervous system is the nerve cell, or neuron. A biological neuron has, see Figure 61:

- Dendrites (inputs) a neuron
- Cell body
- Axon (output)

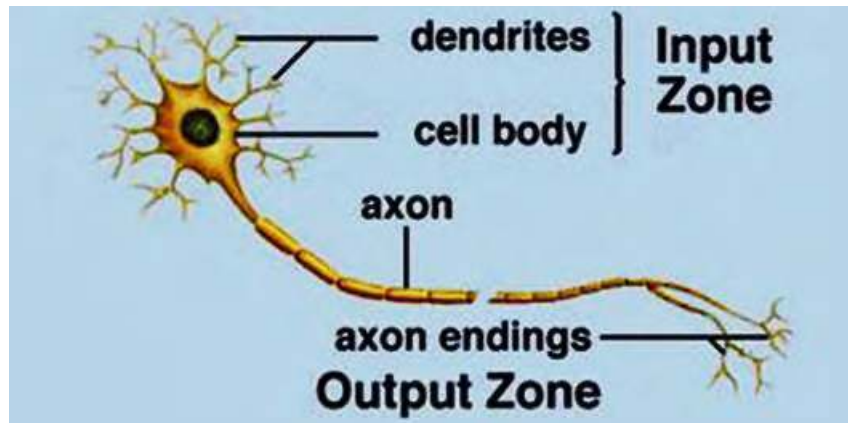


Figure 61: A biological neuron (adapted from <http://www.idsia.ch>)

A neuron receives input from other neurons (typically many thousands). Inputs sum (approximately). Once input exceeds a critical level, the neuron discharges a spike – an electrical pulse that travels from the body, down the axon, to the next neuron(s) (or other receptors). This spiking event is also called depolarization, and is followed by a refractory period, during which the neuron is unable to fire.

The axon endings (Output Zone) almost touch the dendrites or cell body of the next neuron. Transmission of an electrical signal from one neuron to the next is effected by neurotransmitters, chemicals which are released from the first neuron and which bind to receptors in the second. This link is called a synapse. The extent to which the signal from one neuron is passed on to the next depends on many factors, e.g. the amount of neurotransmitters available, the number and arrangement of receptors, amount of neurotransmitters reabsorbed, etc.

Brains learn. From what we know of neuronal structures, one way brains learn is by altering the strengths of connections between neurons, and by adding or deleting connections between neurons. Furthermore, they learn “on-line”, based on experience, and typically without the benefit of a benevolent teacher. The efficacy of a synapse can change as a result of experience, providing both memory and learning through long-term potentiation. One way this happens is through release of more neurotransmitters. Many other changes may also be involved, see Figure 62.

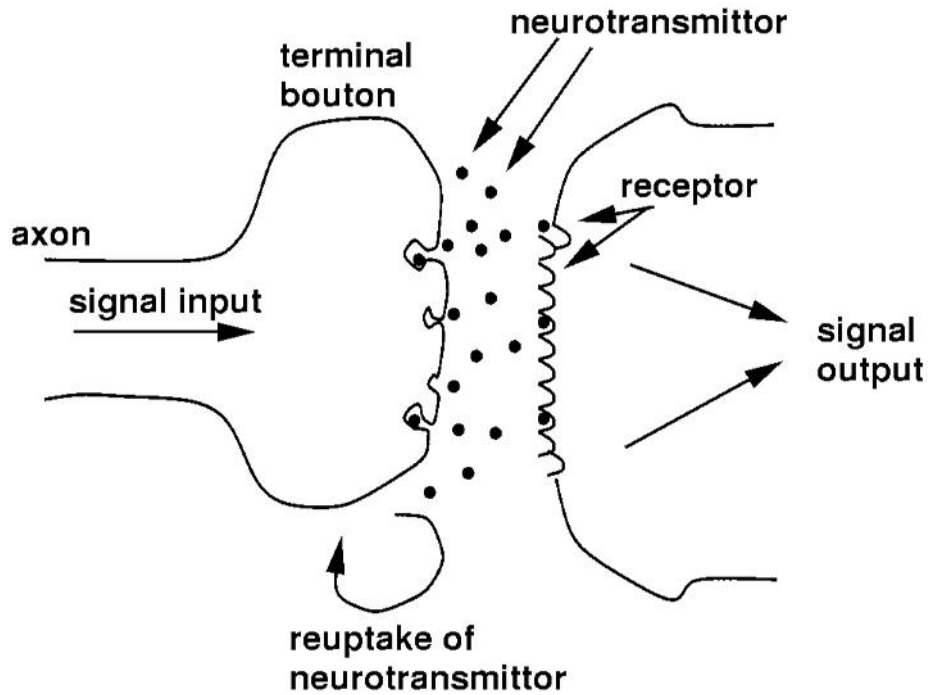


Figure 62: A biological neuron (adapted from <http://www.idisia.ch>)



Maastricht University *Leading in Learning!*

**Join the best at
the Maastricht University
School of Business and
Economics!**

Top master's programmes

- 33rd place Financial Times worldwide ranking: MSc International Business
- 1st place: MSc International Business
- 1st place: MSc Financial Economics
- 2nd place: MSc Management of Learning
- 2nd place: MSc Economics
- 2nd place: MSc Econometrics and Operations Research
- 2nd place: MSc Global Supply Chain Management and Change

Sources: Keuzegids Master ranking 2013; Elsevier 'Beste Studies' ranking 2012; Financial Times Global Masters in Management ranking 2012

Maastricht University is the best specialist university in the Netherlands
(Elsevier)

Visit us and find out why we are the best!
Master's Open Day: 22 February 2014

www.mastersopenday.nl



4.2 Introduction to neural networks

An artificial neural network is a connectionist massively parallel system, inspired by the human neural system. Its units, neurons (Figure 63), are interconnected by connections called synapse. Each neuron, as the main computational unit, performs only a very simple operation: it sums its weighted inputs and applies a certain activation function on the sum. Such a value then represents the output of the neuron. However great such a simplification is (according to the biological neuron), it has been found as plausible enough and is successfully used in many types of ANN, (Fausett 1994).

A neuron X_i obtains input signals x_j and relevant weights of connections w_j , optionally a value called bias b_i is added in order to shift the sum relative to the origin. The weighted sum of inputs is computed and the bias is added so that we obtain a value called stimulus or inner potential of the neuron s_i . After that it is transformed by an activation function f into output value o_i that is computed as it is shown in equations (see Figure 63):

$$s_i = \sum_{j=1}^n w_j x_j + b_i$$

$$o_i = \left(1 + e^{-s_i}\right)^{-1}$$

and may be propagated to other neurons as their input or be considered as an output of the network. Here, the activation function is a sigmoid, (Kondratenko and Kuperin 2003). The purpose of the activation function is to perform a threshold operation on the potential of the neuron.

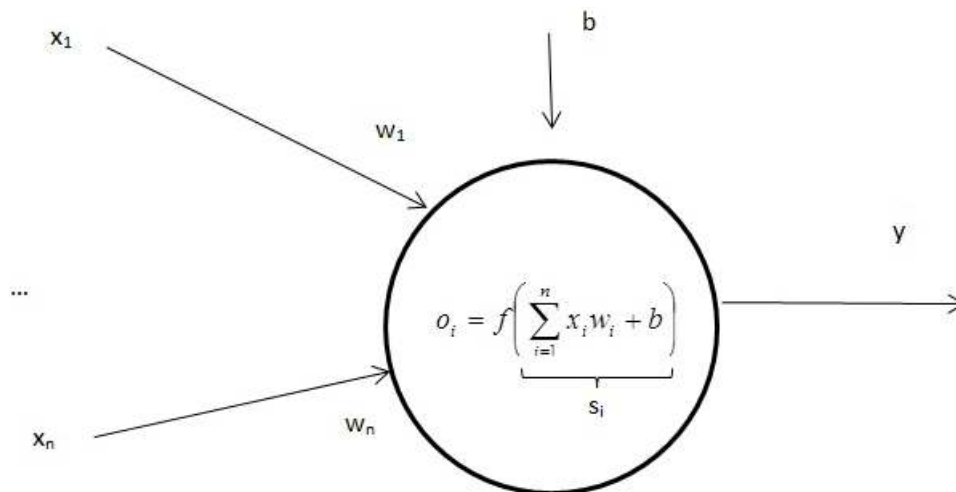


Figure 63: A simple artificial neuron

Activation functions

Most units in neural network transform their net inputs by using a scalar-to-scalar function called an *activation function*, yielding a value called the unit's activation. Except possibly for output units, the activation value is fed to one or more other units. Activation functions with a bounded range are often called squashing functions. Some of the most commonly used activation functions are the following (Fausett 1994).

$$f(x) = \begin{cases} 1 & x \geq 0 \\ 0 & x < 0 \end{cases} \quad \text{binary step function}$$

$$f(x) = \begin{cases} 1 & x > 1 \\ x & 0 \leq x \leq 1 \\ 0 & x < 0 \end{cases} \quad \text{saturated linear function}$$

$$f(x) = \frac{1}{1 + e^{-x}} \quad \text{standard sigmoid}$$

$$f(x) = \frac{1 - e^{-2x}}{1 + e^{-2x}} \quad \text{hyperbolic tangent}$$



> Apply now

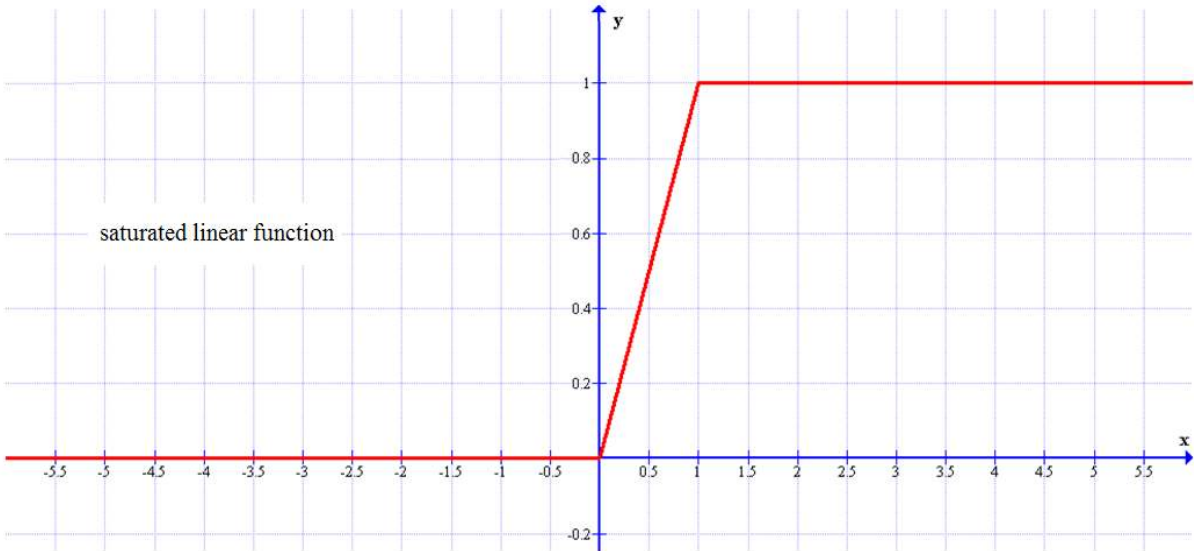
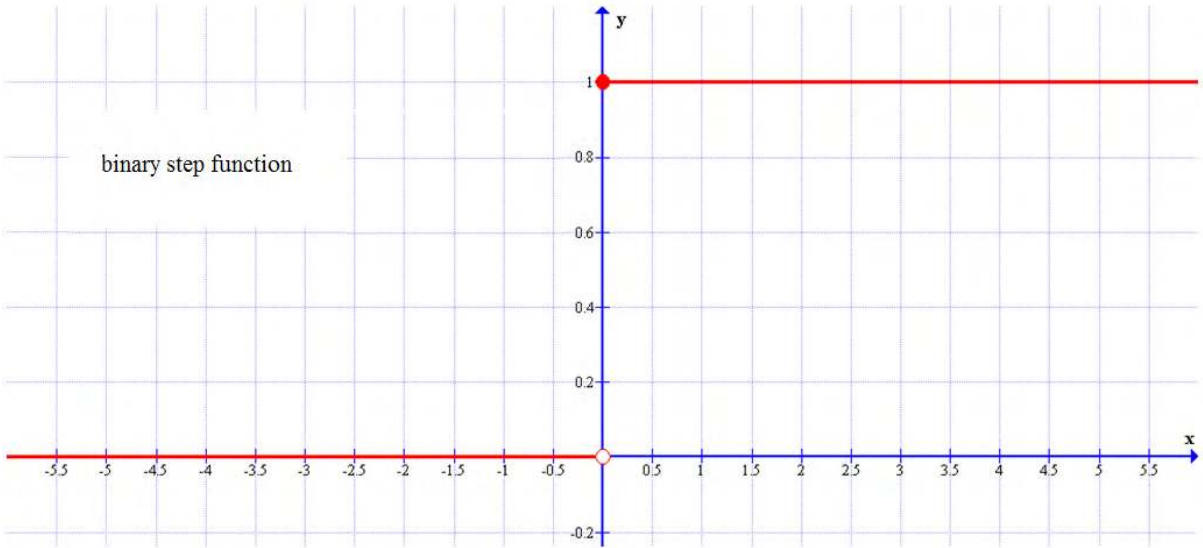
REDEFINE YOUR FUTURE
**AXA GLOBAL GRADUATE
 PROGRAM 2015**

redefining / standards 

agence.cdg. © Photonistop



Graphs of these activation functions are shown in Figure 64.



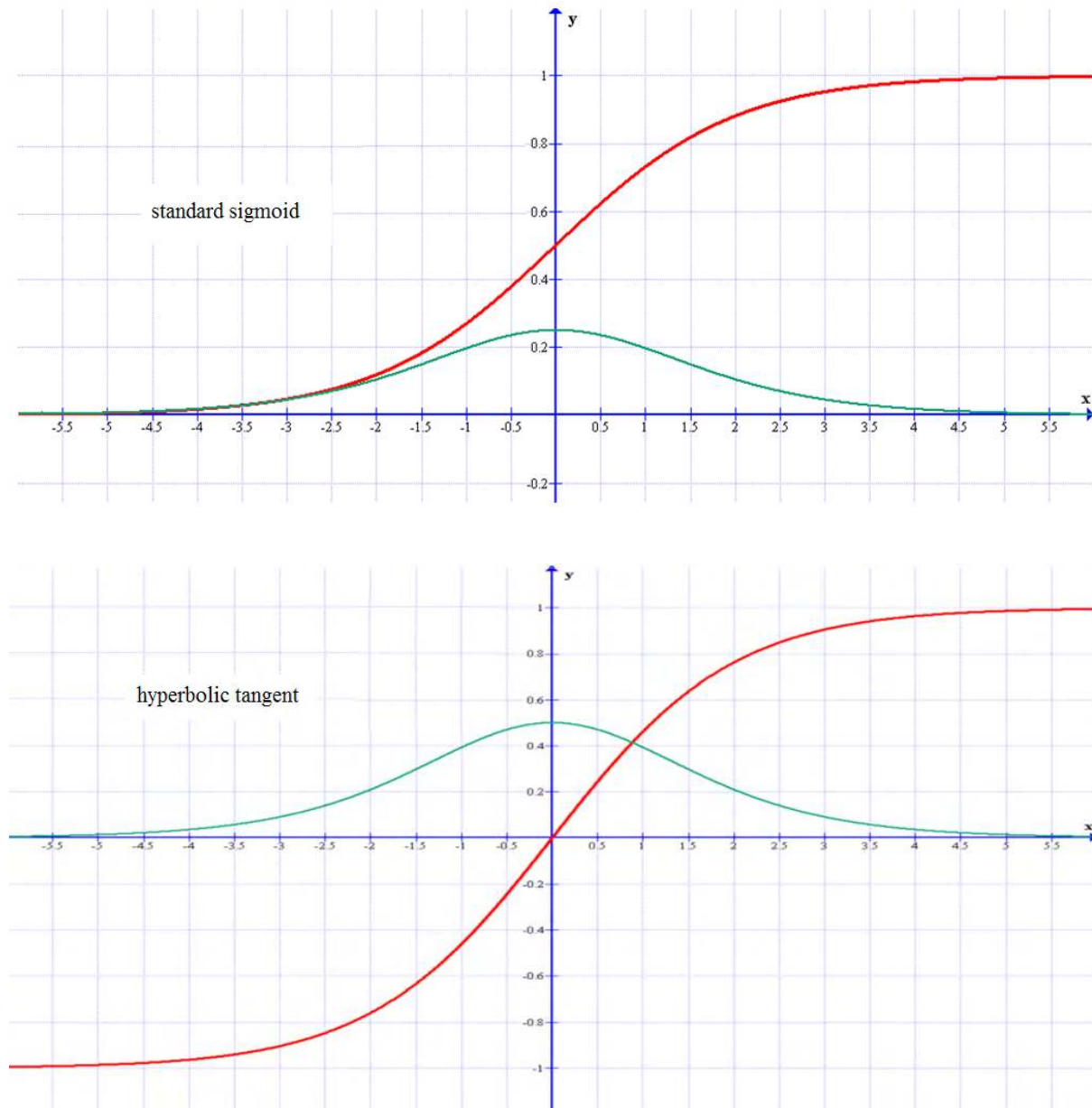


Figure 64: Graphs of activation functions.

Network topologies

Network topologies focus on the pattern of connections between the units and the propagation of data. The basic models are the following:

- *Feed-forward networks* (Figure 65), where the data flow from input to output units is strictly feed-forward. The data processing can extend over multiple (layers of) units, but no feedback connections are present, that is, connections extending from outputs of units to inputs of units in the same layer or previous layers.

- *Recurrent networks* (Figure 66) contain feedback connections. Contrary to feed-forward networks, the dynamical properties of the network are important. In some cases, the activation values of the units undergo a relaxation process such that the network will evolve to a stable state in which these activations do not change anymore. In other applications, the change of the activation values of the output neurons is significant such that the dynamical behavior constitutes the output of the network.

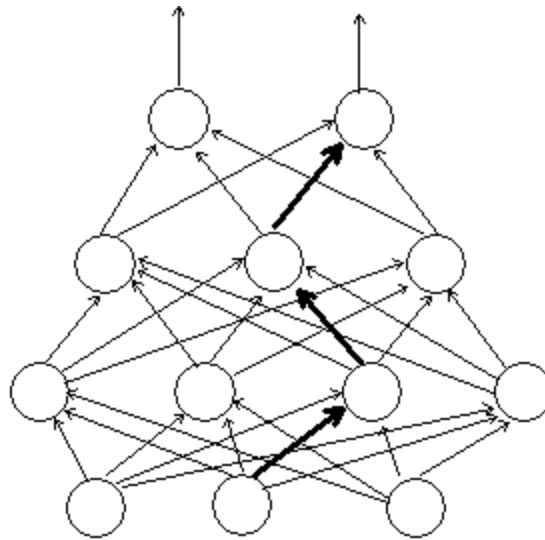


Figure 65: Feed-forward networks

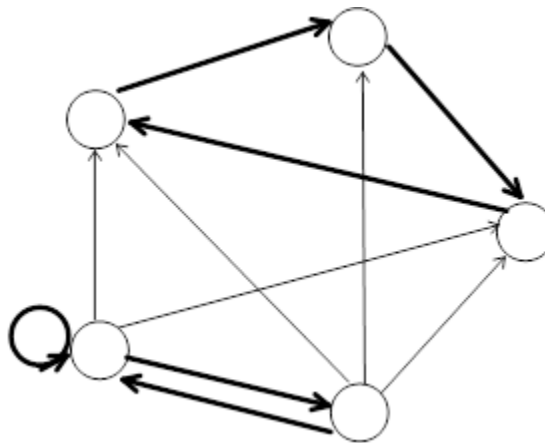


Figure 66: Recurrent networks

Classical examples of feed-forward networks are the Perceptron and Adaline. Examples of recurrent networks are Hopfield nets.

Training of artificial neural networks

A neural network has to be configured such that the application of a set of inputs produces (either 'direct' or via a relaxation process) the desired set of outputs. Various methods to set the strengths of the connections exist. One way is to set the weights explicitly, using a priori knowledge. Another way is to 'train' the neural network by feeding it teaching patterns and letting it change its weights according to some learning rule.

We can categorize the learning situations in two distinct sorts. These are:

- *Supervised learning* or *associative learning* in which the network is trained by providing it with input and matching output patterns. These input-output pairs can be provided by an external teacher, or by the system which contains the network (self-supervised).
- *Unsupervised learning* or *self-organization* in which an (output) unit is trained to respond to clusters of pattern within the input. In this paradigm the system is supposed to discover statistically salient features of the input population. Unlike the supervised learning paradigm, there is no a priori set of categories into which the patterns are to be classified; rather the system must develop its own representation of the input stimuli.

Hebb rule

Both learning paradigms discussed above result in an adjustment of the weights of the connections between units, according to some modification rule. Virtually all learning rules for models of this type can be considered as a variant of the Hebbian learning rule suggested by Hebb in the classic book *Organization of Behaviour* (Hebb 1949). The Hebb rule determines the change in the weight connection from u_i to u_j by $\Delta w_{ij} = \alpha * y_i * y_j$, where α is the learning rate and y_i, y_j represent the activations of u_i and u_j respectively. Thus, if both u_i and u_j are activated the weight of the connection from u_i to u_j should be increased.

Examples can be given of input/output associations which can be learned by a two-layer Hebb rule pattern associator. In fact, it can be proved that if the set of input patterns used in training are mutually orthogonal, the association can be learned by a two-layer pattern associator using Hebbian learning. However, if the set of input patterns are not mutually orthogonal, interference may occur and the network may not be able to learn associations. This limitation of Hebbian learning can be overcome by using the delta rule.

Delta rule

The delta rule (Russell 2005), also called the Least Mean Square (LMS) method, is one of the most commonly used learning rules. For a given input vector, the output vector is compared to the correct answer. If the difference is zero, no learning takes place; otherwise, the weights are adjusted to reduce this difference. The change in weight from u_i to u_j is given by: $\Delta w_{ij} = \alpha * y_i * e_j$, where α is the learning rate, y_i represents the activation of u_i and e_j is the difference between the expected output and the actual output of u_j . If the set of input patterns form a linearly independent set then arbitrary associations can be learned using the delta rule.

This learning rule not only moves the weight vector nearer to the ideal weight vector, it does so in the most efficient way. The delta rule implements a gradient descent by moving the weight vector from the point on the surface of the paraboloid down toward the lowest point, the vertex.

In the case of linear activation functions where the network has no hidden units, the delta rule will always find the best set of weight vectors. On the other hand, that is not the case for hidden units. The error surface is not a paraboloid and so does not have a unique minimum point. There is no such powerful rule as the delta rule for networks with hidden units. There have been a number of theories in response to this problem. These include the generalized delta rule and the unsupervised competitive learning model.



The image shows the BI Norwegian Business School logo, which is a central blue square with 'BI' in white, surrounded by a colorful, multi-colored starburst of lines. The lines are labeled with various business programs: Business, Strategic Marketing Management, International Business, Leadership & Organisational Psychology, Shipping Management, and Financial Economics. Below the logo is the text 'BI NORWEGIAN BUSINESS SCHOOL' and the EFMD EQUIS ACCREDITED logo.

Empowering People. Improving Business.

BI Norwegian Business School is one of Europe's largest business schools welcoming more than 20,000 students. Our programmes provide a stimulating and multi-cultural learning environment with an international outlook ultimately providing students with professional skills to meet the increasing needs of businesses.

BI offers four different two-year, full-time Master of Science (MSc) programmes that are taught entirely in English and have been designed to provide professional skills to meet the increasing need of businesses. The MSc programmes provide a stimulating and multi-cultural learning environment to give you the best platform to launch into your career.

- MSc in Business
- MSc in Financial Economics
- MSc in Strategic Marketing Management
- MSc in Leadership and Organisational Psychology

www.bi.edu/master



Generalizing the ideas of the delta rule, consider a hierarchical network with an input layer, an output layer and a number of hidden layers. We consider only the case where there is one hidden layer. The network is presented with input signals which produce output signals that act as input to the middle layer. Output signals from the middle layer in turn act as input to the output layer to produce the final output vector. This vector is compared to the desired output vector. Since both the output and the desired output vectors are known, we can calculate differences between both outputs and get an error of neural network. The error is backpropagated from the output layer through the middle layer to the unit which are responsible for generating that output. The delta rule can be used to adjust all the weights. More details are presented in (Fausett 1994).

4.3 The perceptron

The perceptron (Figure 67) is a simplest type of artificial neural networks that is linear and based on a threshold θ transfer function. The perceptron can only classify linearly separable cases with a binary target 1 or 0.

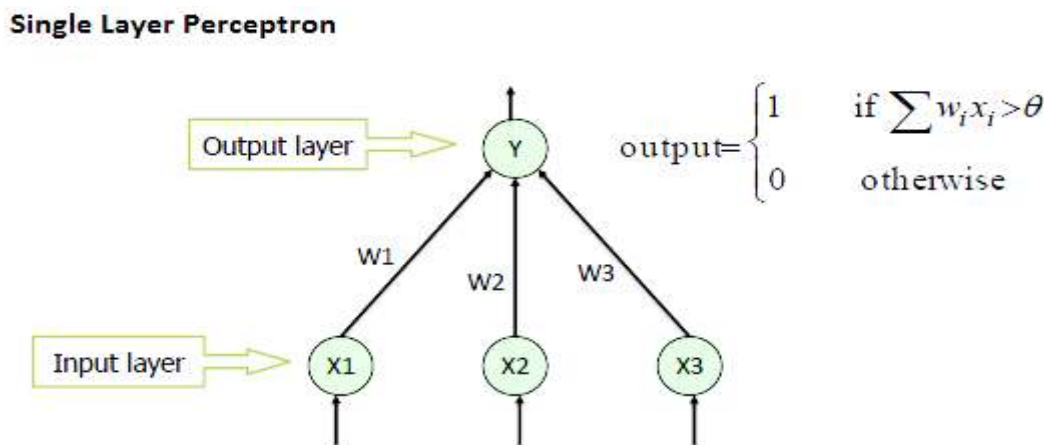


Figure 67: The perceptron (adapted from http://www.saedsayad.com/artificial_neural_network_bkp.htm)

The perceptron algorithm was invented in 1957 at the Cornell Aeronautical Laboratory by Frank Rosenblatt (Beale and Jackson 1992).

The single layer perceptron does not have a priori knowledge, so the initial weights are assigned randomly. The perceptron sums all the weighted inputs and if the sum is above the threshold (some predetermined value), it is said to be activated (output=1).

$$w_1 x_1 + w_2 x_2 + \dots + w_n x_n > \theta \quad \longrightarrow \quad \text{Output } 1$$

$$w_1 x_1 + w_2 x_2 + \dots + w_n x_n \leq \theta \quad \longrightarrow \quad 0$$

The input values are presented to the perceptron, and if the predicted output is the same as the desired output, then the performance is considered satisfactory and no changes to the weights are made. However, if the output does not match the desired output, then the weights need to be changed to reduce the error.

Perceptron Weight Adjustment is the following:

$$\Delta w = \alpha \times d \times x$$

α is a learning rate, usually less than 1,

d is a “predicted output – desired output”,

x represents input data.

As the perceptron is a linear classifier and if the cases are not linearly separable, the learning process will never reach a point where all the cases are classified properly. The most famous example of the inability of perceptron to solve problems with linearly non-separable cases is the XOR problem (Figure 68). However, a multi-layer perceptron using the backpropagation algorithm can successfully classify the XOR data.

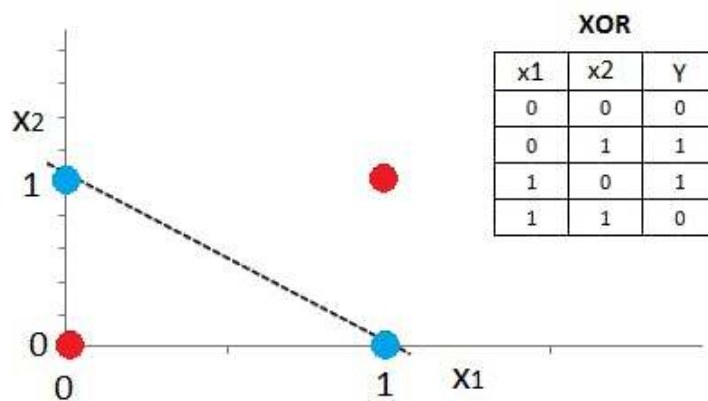


Figure 68: XOR problem (adapted from http://www.saedsayad.com/artificial_neural_network_bkp.htm)

Example

A perceptron for the AND function: binary inputs, binary targets. For simplicity, we take learning rate $\alpha = 1$ and threshold $\theta = 0,2$.

time	INPUT		OUTPUT			WEIGHT CHANGES			WEIGHTS		
	x_1	x_2	NET	OUT	TARGET	Δw_1	Δw_2	Δb	w_1	w_2	b
0									0	0	0
1	1	1	0	0	1	1	1	1	1	1	1
2	1	0	2	1	-1	-1	0	-1	0	1	0
3	0	1	1	1	-1	0	-1	-1	0	0	-1
4	0	0	-1	-1	-1	1	0	0	0	0	-1
...											
37	1	1	1	1	1	0	0	0	2	3	-4
38	1	0	-2	-1	-1	0	0	0	2	3	-4
39	0	1	-1	-1	-1	0	0	0	2	3	-4
40	0	0	-4	-1	-1	0	0	0	2	3	-4

Need help with your dissertation?

Get in-depth feedback & advice from experts in your topic area. Find out what you can do to improve the quality of your dissertation!

[Get Help Now](#)



Go to www.helpmyassignment.co.uk for more info



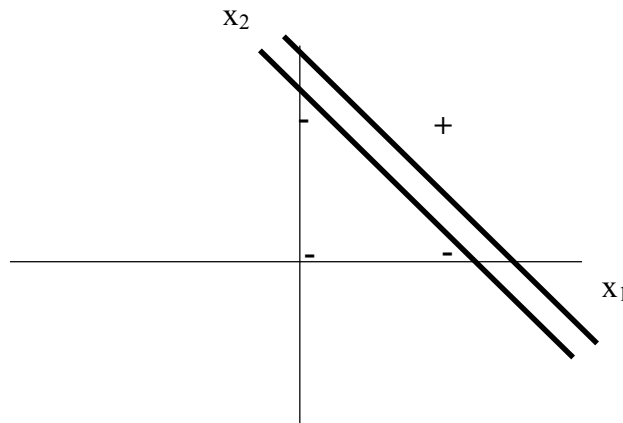


Figure 69: Final decision boundaries for AND function in perceptron learning

The positive response is given by all points such that: $2x_1 + 3x_2 - 4 > 0.2$

with boundary line: $x_2 = -\frac{2}{3}x_1 + \frac{7}{5}$.

The negative response is given by all points such that: $2x_1 + 3x_2 - 4 < -0.2$

with boundary line: $x_2 = -\frac{2}{3}x_1 + \frac{19}{15}$.

4.4 Multilayer networks

Many authors agree that multilayer feedforward neural networks (Figure 70) belong to the most common ones in practical use. Usually a fully connected variant is used, so that each neuron from the n -th layer is connected to all neurons in the $(n+1)$ -th layer, but it is not necessary and in general some connections may be missing. There are also no connections between neurons of the same layer. A subset of input units has no input connections from other units; their states are fixed by the problem. Another subset of units is designated as output units; their states are considered the result of the computation. Units that are neither input nor output are known as hidden units, (Hertz and Kogh 1991).

Each problem specifies a training set of associated pairs of vectors for the input units and output units. The full specification of a network to solve a given problem involves enumerating all units, the connections between them, and setting the weights on those connections. The first two tasks are commonly solved in an ad hoc or heuristic manner, while the final task is usually accomplished with the aid of a learning algorithm, such as backpropagation. This algorithm belongs to a group called “gradient descent methods”. An intuitive definition is that such an algorithm searches for the global minimum of the weight landscape by descending downhill in the most precipitous direction (Figure 71).

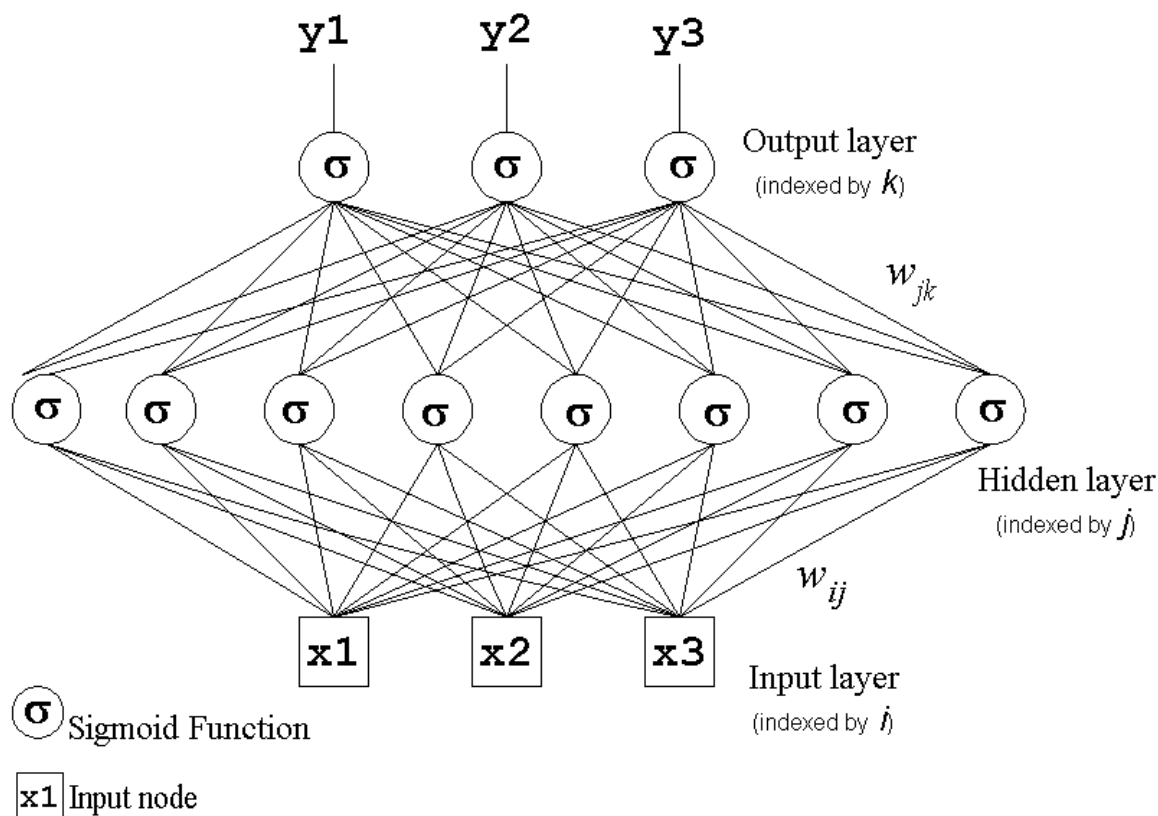


Figure 70: A multilayer feedforward neural network (adapted from <http://homepages.gold.ac.uk/nikolaev/311multi.htm>)

The initial position is set at random (note that there is no a priori knowledge about the shape of the landscape) selecting the weights of the network from some range (typically from -1 to 1 or from 0 to 1). It is obvious that the initial position on the weight landscape greatly influences both the length and the path made when seeking the global minimum. In some cases it is even impossible to get to the optimal position due to the occurrence of some deep local minima. Considering the different points, it is clear, that backpropagation using a fully connected neural network is not a deterministic algorithm. Now, a more formal definition of the backpropagation algorithm (for a three layer network) is presented, (Fausett 1994).

1. The input vector is presented to the network.
2. The feedforward is performed, so that each neuron computes its output following the formula over neurons in previous layer:

$$o_i = \frac{1}{1 + e^{\left(-\sum_{j=1}^n x_j w_j + b\right)}}$$

3. The error on the output layer is computed for each neuron using the desired output (y_j) on the same neuron:

$$err_j^0 = o_j(1 - o_j)(y_j - o_j)$$

4. The error is propagated back to the hidden layer over all the hidden neurons (h_i) and weights between each of them and over all neurons in the output layer:

$$err_i^h = h_i(1 - h_i) \sum_{j=1}^r err_j^0 w_j^0$$

5. Having values err_j^0 and err_i^h computed, the weights from the hidden to the output layer and from the input to the hidden layer can be adjusted using the following formulas

$$w_j^0(t+1) = w_j^0(t) + \alpha err_j^0 h_i$$

$$w_j^h(t+1) = w_j^h(t) + \alpha err_j^h x_i$$

where α is the learning coefficient and x_i is the i -th neuron in the input layer.



Brain power

By 2020, wind could provide one-tenth of our planet's electricity needs. Already today, SKF's innovative know-how is crucial to running a large proportion of the world's wind turbines.

Up to 25 % of the generating costs relate to maintenance. These can be reduced dramatically thanks to our systems for on-line condition monitoring and automatic lubrication. We help make it more economical to create cleaner, cheaper energy out of thin air.

By sharing our experience, expertise, and creativity, industries can boost performance beyond expectations. Therefore we need the best employees who can meet this challenge!

The Power of Knowledge Engineering

Plug into The Power of Knowledge Engineering.
Visit us at www.skf.com/knowledge

SKF



6. All the preceding steps are repeated until the total error of the network over all training pairs does not fall under certain level, where m is number of output neurons.

$$E = \frac{1}{2} \sum_{i=1}^m (y_i - o_i)^2$$

The formulas in step three and four are products of derivation of the error function on each node. A detailed explanation of this derivation as well as of the complete algorithm can be found in (Hertz and Kogh 1991).

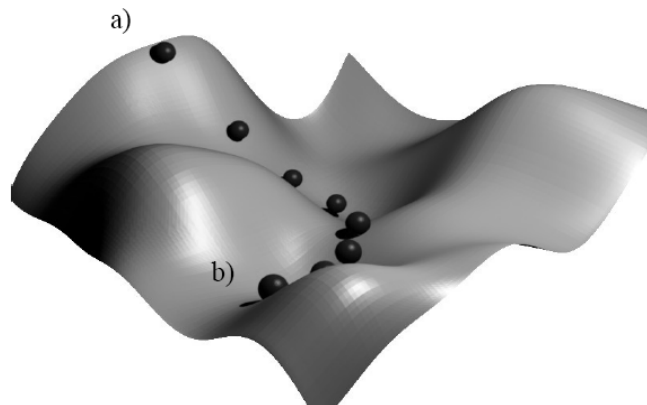


Figure 71: An intuitive approach to the gradient descent method, looking for the global minimum: a) is the starting point, b) is the final one.

4.5 Kohonen self-organizing maps

Kohonen Self-Organizing Maps (or just Self-Organizing Maps, or SOMs for short), are a type of neural network. They were developed in 1982 by Tuevo Kohonen, a professor emeritus of the Academy of Finland. Self-Organizing Maps are aptly named “Self-Organizing” because no supervision is required. SOMs learn on their own through unsupervised competitive learning. “Maps” is because they attempt to map their weights to conform to the given input data. The nodes in a SOM network attempt to become like the inputs presented to them. In this sense, this is how they learn.

SOM can also be called “Feature Maps”, as in Self-Organizing Feature Maps. Retaining principle ‘features’ of the input data is a fundamental principle of SOMs, and one of the things that makes them so valuable. Specifically, the topological relationships between input data are preserved when mapped to a SOM network. This has a pragmatic value of representing complex data. Figure 72 represents a map of the world quality-of-life. Yellows and oranges represent wealthy nations, while purples and blues are the poorer nations. From this view, it can be difficult to visualize the relationships between countries. However, represented by a SOM as shown in Figure 73, it is much easier to see what is going on. Here we can see the United States, Canada, and Western European countries, on the left side of the network, being the wealthiest countries. The poorest countries, then, can be found on the opposite side of the map (at the point farthest away from the richest countries), represented by the purples and blues. Figure 73 is a hexagonal grid. Each hexagon represents a node in the neural network. This is typically called a unified distance matrix, and is probably the most popular method of displaying SOMs. Another intrinsic property of SOMs is known as vector quantization. This is a data compression technique. SOMs provide a way of representing multidimensional data in a much lower dimensional space – typically one or two dimensions. This aids in their visualization benefit, as humans are more proficient at comprehending data in lower dimensions than higher dimensions, as can be seen in the comparison of Figure 72 to Figure 73. The above examples show how SOMs are a valuable tool in dealing with complex or vast amounts of data. In particular, they are extremely useful for the visualization and representation of these complex or large quantities of data in manner that is most easily understood by the human brain.



“I studied English for 16 years but...
...I finally learned to speak it in just six lessons”
Jane, Chinese architect

ENGLISH OUT THERE

Click to hear me talking before and after my unique course download



Structure of a SOM

The structure of a SOM is fairly simple, and is best understood with the use of an illustration such as Figure 74.

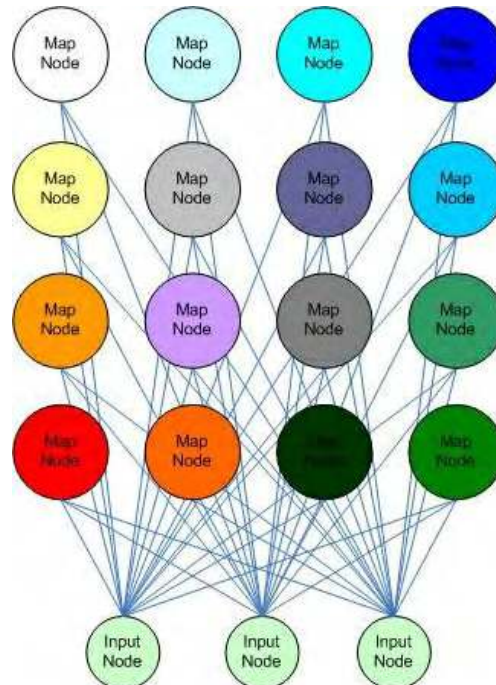


Figure 74: Structure of a SOM (adapted from <http://www.shy.am>).

Figure 74 is a 4×4 SOM network (4 nodes down, 4 nodes across). It is easy to overlook this structure as being trivial, but there are a few key things to notice. First, each map node is connected to each input node. For this small 4×4 node network, that is $4 \times 4 \times 3 = 48$ connections. Secondly, notice that map nodes are not connected to each other. The nodes are organized in this manner, as a 2-D grid makes it easy to visualize the results. This representation is also useful when the SOM algorithm is used. In this configuration, each map node has a unique (i, j) coordinate. This makes it easy to reference a node in the network, and to calculate the distances between nodes. Because of the connections only to the input nodes, the map nodes are oblivious as to what values their neighbours have. A map node will only update its' weights (explained next) based on what the input vector tells it.

The following relationships describe what a node essentially is:

1. $network \subset mapNode \subset float\ weights\ [numWeights]$
2. $inputVectors \subset inputVector \subset float\ weights\ [numWeights]$

The *first* relationship says that the network (the 4×4 grid above) contains map nodes. A single map node contains an array of floats, or its weights. *numWeights* will become more apparent during application discussion. The only other common item that a map node should contain is its (i, j) position in the network.

The *second* relationship says that the collection of input vectors (or input nodes) contains individual input vectors. Each input vector contains an array of floats, or its' weights. Note that *numWeights* is the same for both weight vectors. The weight vectors must be the same for map nodes and input vectors or the algorithm will not work.

The SOM algorithm

The Self-Organizing Map algorithm can be broken up into 6 steps (Beale and Jackson 1992).

1. Each node's weights are initialized.
2. A vector is chosen at random from the set of training data and presented to the network.
3. Every node in the network is examined to calculate which ones' weights are most like the input vector. The winning node is commonly known as the *Best Matching Unit* (BMU).

$$DistFromInput^2 = \sum_{i=0}^n (I_i - W_i)^2$$

I is current input vector
 W is node's weight vector
 n is number of weights

4. The radius of the neighborhood of the BMU is calculated. This value starts large. Typically it is set to be the radius of the network, diminishing each time-step.

Radius of the neighborhood:

$$\sigma(t) = \sigma_0 e^{(-t/\lambda)}$$

t is current iteration
 λ is time constant
 σ_0 is radius of the map

Time constant:

$$\lambda = numIterations / mapRadius$$

5. Any nodes found within the radius of the BMU calculated in 4) are adjusted to make them more like the input vector (Equation 3a, 3b).

New weight of a node:

$$W(t+1) = W(t) + \Theta(t)L(t)(I(t) - W(t))$$

Learning rate:

$$L(t) = L_0 e^{-t/\lambda}$$

The closer a node is to the BMU, the more its weights are altered:

Distance from BMU:

$$\Theta(t) = e^{(-\text{distFromBMU}^2 / (2\sigma^2(t)))}$$

6. Repeat 2) for N iterations.

There are some things to note about these formulas. Equation from step 3 represents simply the Euclidean distance formula, squared. It is squared because we are not concerned with the actual numerical distance from the input. We just need some sort of uniform scale in order to compare each node to the input vector. This equation provides that eliminating the need for a computationally expensive square root operation for every node in the network.



What do you want to do?

No matter what you want out of your future career, an employer with a broad range of operations in a load of countries will always be the ticket. Working within the Volvo Group means more than 100,000 friends and colleagues in more than 185 countries all over the world. We offer graduates great career opportunities – check out the Career section at our web site www.volvogroup.com. We look forward to getting to know you!

VOLVO
 AB Volvo (publ)
www.volvogroup.com

VOLVO TRUCKS | REHAULT TRUCKS | MACK TRUCKS | VOLVO BUSES | VOLVO CONSTRUCTION EQUIPMENT | VOLVO PENTA | VOLVO AERO | VOLVO IT
 VOLVO FINANCIAL SERVICES | VOLVO 3P | VOLVO POWERTRAIN | VOLVO PARTS | VOLVO TECHNOLOGY | VOLVO LOGISTICS | BUSINESS AREA ASIA



Equations from step 4 utilize exponential decay. At $t=0$ they are at their max. As t (the current iteration number) increases, they approach zero. This is exactly what we want. The radius should start out as the radius of the lattice, and approach zero, at which time the radius is simply the BMU node (see Figure 75). The time constant value is almost arbitrary and can be chosen. This provides a good value, though, as it depends directly on the map size and the number of iterations to perform.

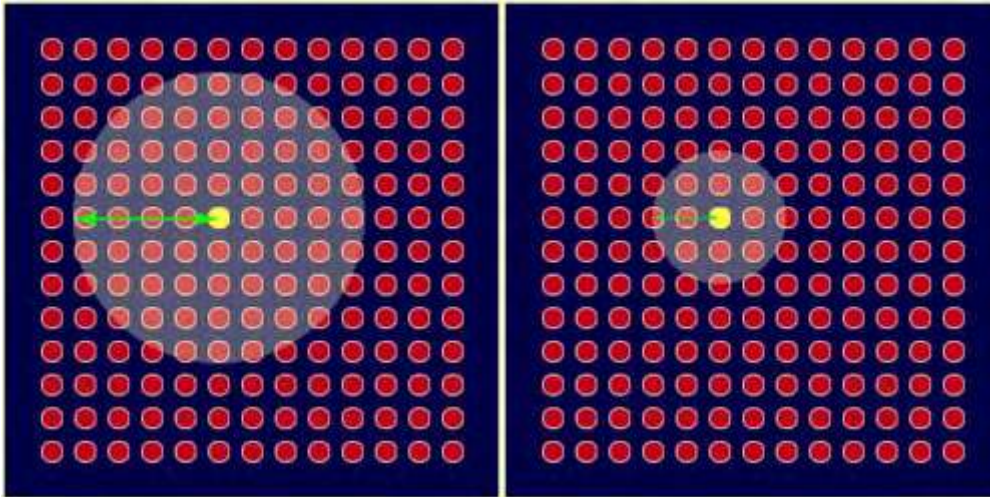


Figure 75: Radius of the neighbourhood (adapted from <http://www.shy.a>)

Equation from step 5 is the main learning function. $W(t+1)$ is the new, 'educated', weight value of the given node. Over time, this equation essentially makes a given node weight more like the currently selected input vector, I . A node that is very different from the current input vector will learn more than a node very similar to the current input vector. The difference between the node weight and the input vector are then scaled by the current learning rate of the SOM, and by $\Theta(t)$.

$\Theta(t)$ is used to make nodes closer to the BMU learn more than nodes on the outskirts of the current neighborhood radius. Nodes outside of the neighborhood radius are skipped completely. $distFromBMU$ is the actual number of nodes between the current node and the BMU, easily calculated as: $distFromBMU^2 = (bmuI - nodeI)^2 + (bmuJ - nodeJ)^2$

This can be done since the node network is just a 2-D grid of nodes. With this in mind, nodes on the very fringe of the neighbourhood radius will learn some fraction less 1.0. As $distFromBMU$ decreases, $\Theta(t)$ approaches 1.0. The BMU itself will have a $distFromBMU$ equal to 0, which gives $\Theta(t)$ its maximum value of 1.0. Again, this Euclidean distance remains squared to avoid the square root operation.

There exists a lot of variations regarding the equations used with the SOM algorithm. There is also a lot of research being done on the optimal parameters. Some things of particular heavy debate are the number of iterations, the learning rate, and the neighborhood radius. It has been suggested by Kohonen himself, however, that the training should be split into two phases. Phase 1 will reduce the learning coefficient from 0.9 to 0.1, and the neighbourhood radius from half the diameter of the lattice to the immediately surrounding nodes. Phase 2 will reduce the learning rate from 0.1 to 0.0, but over double or more the number of iterations in Phase 1. In Phase 2, the neighbourhood radius value should remain fixed at 1 (the BMU only). Analysing these parameters, Phase 1 allows the network to quickly ‘fill out the space’, while Phase 2 performs the ‘fine-tuning’ of the network to a more accurate representation.

Example – Colour Classification

Colour classification SOMs only use three weights per map and input nodes. These weights represent the (R,G,B) triplet for the colour. For example, colours may be presented to the network – (1,0,0) for red, (0,1,0) for green, etc. The goal for the network here is to learn how to represent all of these input colours on its 2-D grid while maintaining the intrinsic properties of a SOM such as retaining the topological relationships between input vectors. With this in mind, if dark blue and light blue are presented to the SOM, they should end up next to each other on the network grid.

To illustrate the process, we will step through the algorithm for the colour classification application. Step 1 is the initialisation of the network. Figure 76 shows a newly initialised network. Each square is a node in the network.



Figure 76: An initialised network (adapted from <http://www.shy.am>)

Step 1. The initialisation method used here is to assign a random value between 0.0 and 1.0 for each component (r, g, and b) of each node.

Step 2 is to choose a vector at random from the input vectors. Eight input vectors are used in this example, ranging from red to yellow to dark green.

Step 3 goes through every node and finds the BMU, as described earlier. Figure 77 shows the BMU being selected in the 4x4 network.

Step 4 of the algorithm calculates the neighbourhood radius. This is also shown in Figure 77. All the nodes tinted red are within the radius. Step 5 then applies the learning functions to all of these nodes. It is based on their distance from the BMU. The BMU (dark red) learns the most, while nodes on the outskirts of the radius (light pink) learn the least. Nodes outside of the radius (white) don't learn at all. We then go back to Step 2 and repeat. Figure 78 shows a trained SOM, representing all eight input colours. Notice how light green is next to dark green, and red is next to orange. An ideal map would probably have light blue next to dark blue. This is where the *Error Map* comes into play, which is described next.

gaiteye
Challenge the way we run

**EXPERIENCE THE POWER OF
FULL ENGAGEMENT...**

**RUN FASTER.
RUN LONGER..
RUN EASIER...**

**READ MORE & PRE-ORDER TODAY
WWW.GAITEYE.COM**



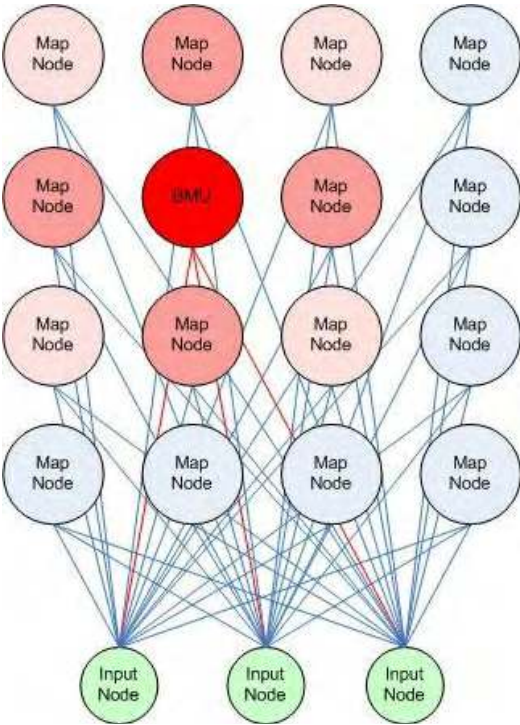


Figure 77: Best Matching Unit (BMU) (adapted from <http://www.shy.am>)

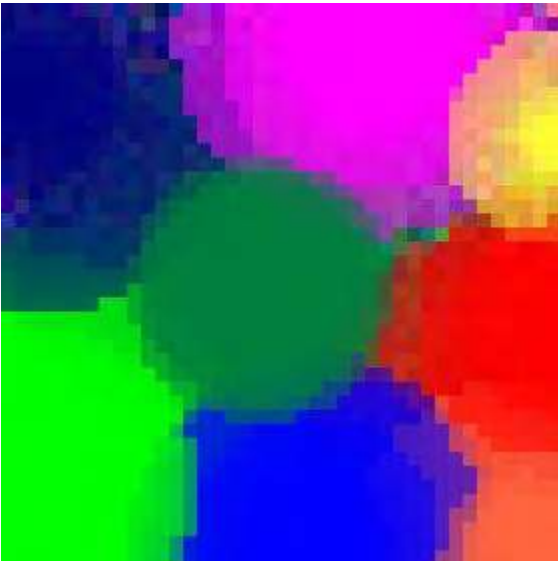


Figure 78: The BMU window (adapted from <http://www.shy.am>)

There are two other windows in the colour classification application. These are the BMU Window and the Error Map. These windows are not active until after the network is trained. First we describe the BMU window. Upon successful SOM training, this window will show small white dots. These white dots represent the N most frequently used BMU nodes, where N is the number of input vectors (unless $N < \#$ of iterations. Then, $N = \#$ of iterations). These nodes have been deemed to be a BMU the most times out of all the nodes in the network, presenting the least distance possible between a map node and the selected input vector for the given iteration. Figure 79 shows this window. Notice how Figure 79 could be placed on top of Figure 78, and the dots would correspond to the centers of the circles.

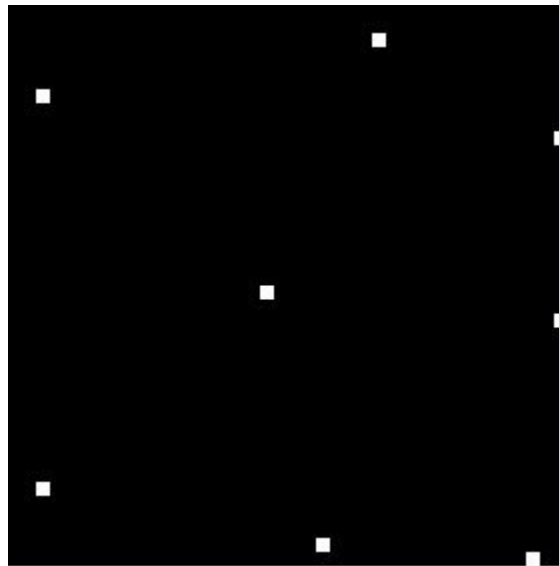


Figure 79: Dots correspond to the centres of the circles in Figure 76 (adapted from <http://www.shy.am>).

Next, the Error Map is calculated (Figure 79). Each time a SOM is trained, it can produce a completely different result given the same input data. This is because the network is initialised with random colours, presenting a unique setup prior to each training session. Also, this occurs because input vectors to be presented to the network are chosen at random. With this in mind, some SOMs may turn out 'better' than others, where 'better' is a measure of how well the topological data is preserved.

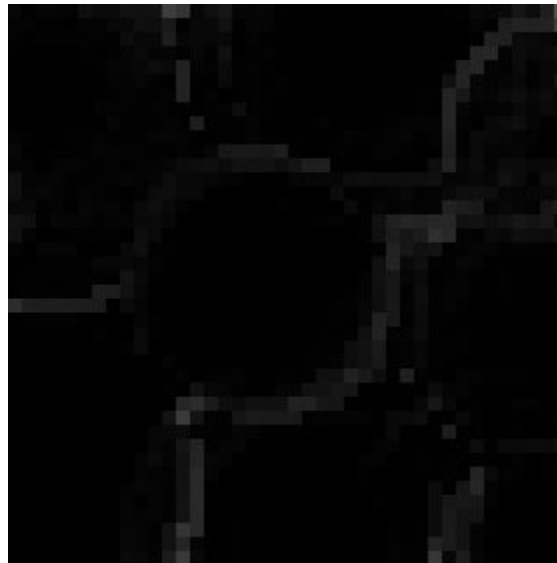


Figure 80: Error map corresponding with Figure 78 (adapted from <http://www.shy.am>).

To calculate an error map, loop through every map node of the network. Add up the distance (not the physical distance, but the weight distance. This is exactly the same as how the BMU is calculated) from the node we are currently evaluating, to each of its neighbours. Average this distance. Multiply this by 3 (the number of weights used), assuming no square root is used to calculate the distance between adjacent nodes. If the square root operation is used, multiply by $\sqrt{3}$ instead. Assign this value to the node. This gives each map node a nice value between 0.0 and 1.0. These values can then be used as the R = G = B values for each square of the Error Map window. Pure white represents the maximum possible distance between adjacent nodes, while black shows that adjacent nodes are all the same colour. Shades of gray in between give an even finer explanation, with darker grays being a better map than a map with light grays. Figure 80 shows an example. Notice the lines and how they line up with Figure 78.

4.6 Hopfield networks

A Hopfield network is a form of recurrent artificial neural network invented by John Hopfield. Hopfield nets serve as content-addressable memory systems with binary (bipolar) threshold nodes. They are guaranteed to converge to a local minimum, but convergence to a false pattern (wrong local minimum) rather than the stored pattern (expected local minimum) can occur. Hopfield networks also provide a model for understanding human memory.

Figure 81 shows a diagram representing a facial recognition system using a Hopfield network. The pixels are initially converted into black and white and the background is removed. Afterwards, the image is given to the network, which will eventually converge to the image on the right that the network used to train with.

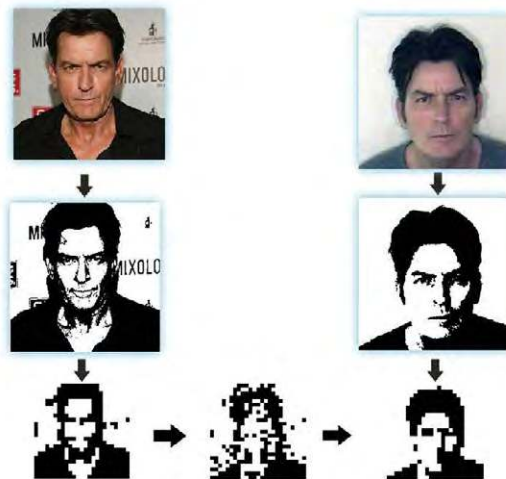


Figure 81: A diagram representing a facial recognition system using a Hopfield network
 (adapted from http://en.wikipedia.org/wiki/File:Face_recognition_with_hopfield_network.jpg)

The units in Hopfield nets only take on two different values for their states and the value is determined by whether or not the units' input exceeds their threshold. Hopfield nets normally have units that take on values of 1 and -1 resp. 0 and 1.



Every two units i and j of a Hopfield network have a connection that is described by the connectivity weight w_{ij} . In this sense, the Hopfield network can be formally described as a complete undirected graph. The connections in a Hopfield net typically have the following restrictions:

- $w_{ii} = 0, \forall i$ (no unit has a connection with itself)
- $w_{ij} = w_{ji}, \forall i, j$ (connections are symmetric)

The requirement that weights be symmetric is typically used, as it will guarantee that the energy function decreases monotonically while following the activation rules, and the network may exhibit some periodic or chaotic behaviour if non-symmetric weights are used.

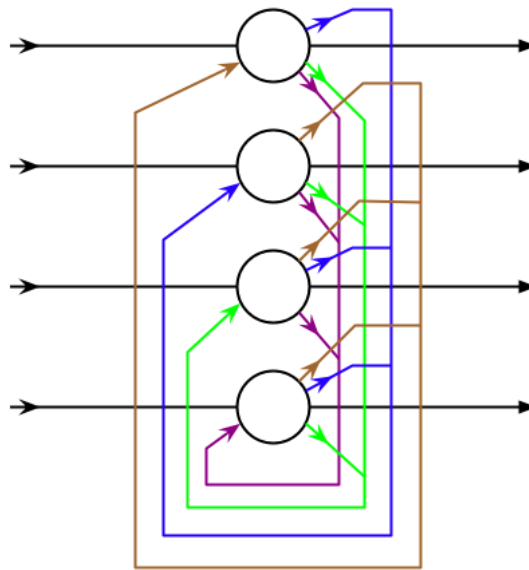


Figure 82: Hopfield network.

(adapted from <http://upload.wikimedia.org/wikipedia/commons/9/95/Hopfield-net.png>)

Updating one unit (node in the graph simulating the artificial neuron) in the Hopfield network is performed using the following rule:

$$s_i = \begin{cases} 1 & \text{if } \sum_j w_j s_j > \theta_i \\ -1 & \text{otherwise.} \end{cases}$$

where:

w_{ij} is the strength of the connection weight from unit j to unit i (the weight of the connection).

s_j is the state of unit j .

θ_i is the threshold of unit i .

Updates in the Hopfield network can be performed in two different ways:

- **Asynchronous:** Only one unit is updated at a time. This unit can be picked at random, or a pre-defined order can be imposed from the very beginning.
- **Synchronous:** All units are updated at the same time. This requires a central clock to the system in order to maintain synchronisation. This method is less realistic, since biological or physical systems lack a global clock that keeps track of time.

Neurons attract or repel each other. The weight between two units has a powerful impact upon the values of the neurons. Consider the connection weight w_{ij} between two neurons i and j . If $w_{ij} > 0$, the updating rule implies that:

- when $s_j = 1$, the contribution of j in the weighted sum is positive. Thus, s_i is pulled by j towards its value $s_j = 1$
- when $s_j = -1$, the contribution of j in the weighted sum is negative. Then again, s_i is pulled by j towards its value $s_j = -1$

Thus, the values of neurons i and j will converge if the weight between them is positive. Similarly, they will diverge if the weight is negative.

Energy

Hopfield nets have a scalar value associated with each state of the network referred to as the “energy” (Figure 83), E , of the network, where:

$$E = -\frac{1}{2} \sum_{i,j} w_{ij} s_i s_j + \sum_i \theta_i s_i$$

This value is called the “energy” because the definition ensures that when units are randomly chosen to update, the energy E will either lower in value or stay the same. Furthermore, under repeated updating the network will eventually converge to a state which is a local minimum in the energy function. Thus, if a state is a local minimum in the energy function, it is a stable state for the network. Note that this energy function belongs to a general class of models in physics.

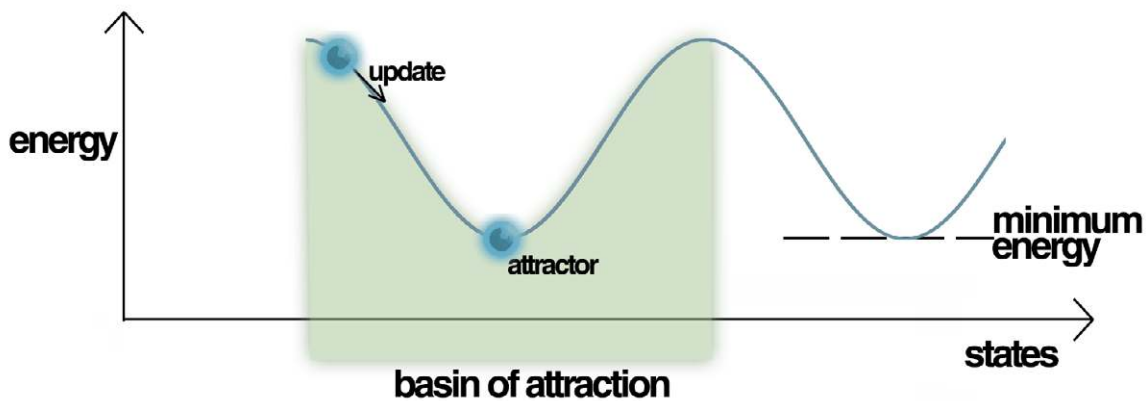


Figure 83: Energy Landscape of a Hopfield Network, highlighting the current state of the network (up the hill), an attractor state to which it will eventually converge, a minimum energy level and a basin of attraction shaded in green. Note how the update of the Hopfield Network is always going down in Energy.

(adapted from http://upload.wikimedia.org/wikipedia/commons/4/49/Energy_landscape.png)

Training a Hopfield net

Initialization of the Hopfield Networks is done by setting the values of the units to the desired start pattern. Repeated updates are then performed until the network converges to an attractor pattern. In the context of Hopfield Networks, an attractor pattern is a pattern that cannot change any value within it under updating.

The advertisement features a background image of a complex, glowing orange and red structural grid. At the top left, the URL 'www.sylvania.com' is displayed. The main text reads: 'We do not reinvent the wheel we reinvent light.' Below this, a paragraph describes the company's focus on innovative lighting technologies and the benefits of working with them. At the bottom right, the OSRAM SYLVANIA logo is shown, consisting of the word 'OSRAM' in orange, 'SYLVANIA' in white, and a lightbulb icon with 'SYLVANIA' written inside it.



Training a Hopfield net involves lowering the energy of states that the net should “remember”. This allows the net to serve as a content addressable memory system, that is to say, the network will converge to a “remembered” state if it is given only part of the state. The net can be used to recover from a distorted input to the trained state that is most similar to that input. This is called associative memory because it recovers memories on the basis of similarity.

There are various different learning rules that can be used to store information in the memory of the Hopfield Network. It is desirable for a learning rule to have both of the following two properties:

- *Local*: A learning rule is local if each weight is updated using information available to neurons on either side of the connection that is associated with that particular weight.
- *Incremental*: New patterns can be learned without using information from the old patterns that have been also used for training. That is, when a new pattern is used for training, the new values for the weights only depend on the old values and on the new pattern (Storkey and Valabregue 1999).

These properties are desirable, since a learning rule satisfying them is more biologically plausible. For example, since the human brain is always learning new concepts, one can reason that human learning is incremental. A learning system that would not be incremental would generally be trained only once, with a huge batch of training data.

The Hebbian Theory has been introduced by Donald Hebb (Hebb 1949), in order to explain “associative learning” in which simultaneous activation of neuron cells leads to pronounced increases in synaptic strength between those cells. It is often summarized as “Neurons that fire together, wire together. Neurons that fire out of sync, fail to link”.

The Hebbian rule is both local and incremental. For the Hopfield networks, it is implemented in the following manner, when learning n binary patterns:

$$w_{ij} = \frac{1}{n} \sum_{k=1}^n x_i^k x_j^k$$

where x_i^k represents bit i from pattern k .

If the bits corresponding to neurons i and j are equal in pattern k , then the product $x_i^k x_j^k$ will be positive. This would, in turn, have a positive effect on the weight w_{ij} and the values of i and j will tend to become equal. The opposite happens if the bits corresponding to neurons i and j are different.

The Network capacity of the Hopfield network model is determined by neuron amounts and connections within a given network. Therefore, the number of memories that are able to be stored are dependent on neurons and connections. Therefore, it is evident that many mistakes will occur if you try to store a large number of vectors. When the Hopfield model does not recall the right pattern, it is possible that an intrusion has taken place, since semantically related items tend to confuse the individual, and recollection of the wrong pattern occurs. Therefore, the Hopfield network model is shown to confuse one stored item with that of another upon retrieval.

Human memory

The Hopfield model accounts for associative memory through the incorporation of memory vectors. Memory vectors can be slightly used, and this would spark the retrieval of the most similar vector in the network. However, we will find out that due to this process, intrusions can occur. In associative memory for the Hopfield network, there are two types of operations: auto-association and hetero-association. The first being when a vector is associated with itself, and the latter being when two different vectors are associated in storage. Furthermore, both types of operations are possible to store within a single memory matrix, but only if that given representation matrix is not one or the other of the operations, but rather the combination (auto-associative and hetero-associative) of the two. It is important to note that Hopfield network model utilizes the same learning rule as Hebb learning rule, which basically tried to show that learning occurs as a result of the strengthening of the weights by when activity is occurring.

(Rizzuto and Kahana 2001) were able to show that the neural network model can account for repetition on recall accuracy by incorporating a probabilistic-learning algorithm. During the retrieval process, no learning occurs. As a result, the weights of the network remains fixed, showing that the model is able to switch from a learning stage to a recall stage. By adding contextual drift we are able to show the rapid forgetting that occurs in a Hopfield model during a cued-recall task. The entire network contributes to the change in the activation of any single node.

(McCullough and Pitts 1943), dynamical rule, which describes the behavior of neurons, does so in a way that shows how the activations of multiple neurons map onto the activation of a new neuron's firing rate, and how the weights of the neurons strengthen the synaptic connections between the new activated neuron (and those that activated it). Hopfield would use McCullough-Pitts's dynamical rule in order to show how retrieval is possible in the Hopfield network. However, it is important to note that Hopfield would do so in a repetitious fashion. Hopfield would use a nonlinear activation function, instead of using a linear function. This would therefore create the Hopfield dynamical rule and with this, Hopfield was able to show that with the nonlinear activation function, the dynamical rule will always modify the values of the state vector in the direction of one of the stored patterns.