Concepts of Learning in Artificial Neural Networks

Matthias Küntzel
email : kuentzel@math.uni-goettingen.de

13.01.2000

Abstract

This document gives a short introduction into the theory of artificial neural networks. After motivating the aims of computational neuroscience the McCulloch& Pitts neuron will be described. This model can be continually refined by taking more and more characteristics of the brain into account leading to Hebbian Learning and the PERCEPTRON algorithm, for which a proof of convergence is given. We derive the ADALINE rule from the gradient descent algorithm in the case of continuous–valued units. Finally a qualitative discussion of the storage capacity of simple neurons and an outlook on other models will be presented.

1 Introduction

Considering the number of neurons in the human brain (approx. 10^{11}) and the synapses connecting them (approx. 10^{14} give or take a few million) it seems at first glance illusionary to understand how they interact and what kind of concepts enable us to learn (and forget), memorize, associate etc. and do physics for that matter. On the other hand we realize the incredible advantages biological neural networks have compared with your ordinary von–Neumann computer. Although the processing speed of simple neurons appears to be laughably slow weighed against the blinding performance of their silicium counterparts, humans are nevertheless capable of recognizing patterns in a matter of milliseconds a task which computers had to solve by searching for a specific entry in a very large database. From the previous remarks it is obvious then that the architecture of the brain does not rely on sequential processes but is highly parallel in its nature. Moreover we deduce from experiments that information is not stored just in one specific place in the brain but is somewhat delocalized over a certain area. This fact combined with our ability to attribute the right patterns even when confronted with noisy input data shows the high error tolerance of biological neural networks, an ability that is not innate in computers (we have all had our share of blue screens). Other important inbuilt functions of the brain that make it worth studying can be found in the capacity to learn and generalize, which can be viewed as the basis for autonomous behaviour. All this is very well but it still leaves us with the question how to emulate the desired features. As physicists we are always ready to give up "unneccessary complexity" in favour of a simple model in the hope to explain collective phenomena and this is exactly what McCulloch and Pitts did in (1943). They tried to describe neurons as binary treshold units. We will see how this model in combination with certain architectures and algorithms can be quite successful although being in its simplicity far off biolgical reality . Before we go into further detail let's formulate a wishlist of the characteristics and abilities of the brain, that we would like to simulate :

- Learning, Forgetting, Unlearning
- Generalization
- Associative Memory with high error tolerance
- Parrallel architecture

2 From simple neurons to simple learning algorithms

Broadly speaking biological neurons consist of a lot of dentrites which serve as the input, a body (soma) in which some of the processing is believed to take place and an output area, the axon that is connected to other neurons by synapses. "Communication" between different units functions via so called neuro transmitters chemical substances that are sent out of the axon and trigger an electric response on the other side of the synaptic cavity. Happily throwing away any other information McCulloch and Pitts formulated their model of neurons in the following way:

$$O_i = \Theta(h_i) = \Theta\left(\sum_{k=1}^{N} w_{ik}\xi_k - \theta_i\right) \tag{1}$$

where we follow the notation :

$$\begin{aligned} O_i &= \text{output of the } i^{th} \text{ neuron} \\ \Theta(x) &= \begin{cases} 1 & ,x \geq 0 \\ 0 & ,\text{otherwise} \end{cases} \qquad \text{(step-function)} \\ w_{ik} &= \text{synaptic weight of the } k^{th} \text{ connection of the } i^{th} \text{ neuron} \\ \xi_k &= k^{th} \text{ input} \\ \theta_i &= \text{treshold of the } i^{th} \text{ neuron} \end{aligned}$$

This equation can be generalized by substituting $\Theta(x)$ with other functions $g(x)$. It can be seen that the neuron stays in the ground state ($O_i = 0$) until the weighted sum of the N binary inputs exceeds the treshold θ_i. If that happens the unit enters the excited state and "fires" ($O_i = 1$). Probably any biologist will club you to death with your own "Gerthsen" if you ask for a second opinion on this model. But we are firm believers in simplicity and take a look at what this concept holds. Consider just one neuron and let's assume that all inputs are $w_k = 1/N$. If we choose the treshold to be $\theta = 1/(N+1)$ this unit will act like a logical OR–gate. By increasing θ to $(N-1)/N$ an AND–gate results. McCulloch and Pitts showed, that any logical function can be realized in this way. Neurons differ from logical gates insofar as they could switch between functions just by adjusting their tresholds. But how would they do that ? If we can simulate this process that could be the key to learning.

Suppose you have a set of different patterns $\{\xi_k^\mu\}\quad \mu = 1,\ldots,p; k = 1,\ldots,N$ that can all be classified/encoded by several (N) binary characteristics. If we got a neuron to adapt its synaptic weights in such a way, that each pattern triggers a specific reproducible output we would say, that our network has learned those patterns. There are a lot of ways to enlightenment. We could imagine a teacher who presents the different specimen to the network and if the network guessed wrong he would either

1. reveal the right answer or
2. just keep its pocket money to show it the error of its ways.

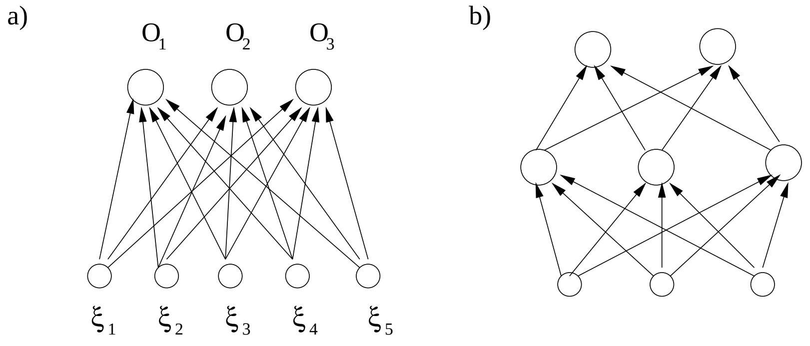

Figure 1: a)simple perceptron and b) perceptron with hidden units

This is done many times until all connections are adapted according to a certain algorithm and the set of patterns is stored with high fidelity. Both methods fall into the category of supervised learning. The second one is a special case of the first and is generally referred to as "reinforcement learning". A third method called unsupervised learning does without a teacher and leaves the network to itself in finding any interesting categories in the presented patterns (remember how you felt in your first semester at university). Here we will just focus on case 1 . In this chapter we will also confine ourselves to a special kind of network architecture, the so called feed–forward networks that consist of several layers of neurons whose synaptic connections all point in the same general direction as can be seen in figure (1) leading to an asymmetric matrix $\{w_{ik}\}$. Layered feed–forward–networks were called perceptrons when first studied in detail by Rosenblatt (et al.) in 1962. Ouput and input are not counted as layers. Any units belonging to layers different from the last one are called "hidden". We will just examine networks without any hidden units here. They are often known as simple perceptrons. We now choose for the nonlinear output function (activation function) $g(x) = \mathrm{sgn}(x)$. Also one can treat the treshold as the result of another constant input $\xi_0 = -1$ setting the connection strengths $w_{i0} = \theta_i$. Equation (1) then reads:

$$O_i = \mathrm{sgn}\left(\sum_{k=0}^{N} w_{ik}\xi_k\right) \tag{2}$$

ξ_1	ξ_2	ζ
0	0	-1
0	1	-1
1	0	-1
1	1	+1

Figure 2: modified thruth table for the Boolean AND–function

Suppose we want the network to produce the output $\{\zeta_i^\mu\}$ (target pattern) when presented with one of the p input patterns $\{\xi_k^\mu\}$

$$O_i^\mu \quad \stackrel{!}{=} \quad \zeta_i^\mu \quad \forall i, \mu \quad \text{(desired)} \tag{3}$$

As the output units are not directly connected they are independent and can be examined one at a time. In this case the subscript i will be dropped. If one views the inputs as the components of an N–dimensional pattern vector ξ^μ and introduces the weight vector $\mathbf{w}$ =$(w_1, \ldots, w_N)$ our learning conditon becomes

$$\operatorname{sgn}(\mathbf{w} \cdot \xi^\mu) \quad \stackrel{!}{=} \quad \zeta^\mu \quad \forall \mu \quad \text{(desired)} \tag{4}$$

Geometrically speaking the hyperplane $\mathbf{w} \cdot \xi = 0$ divides our input space into two categories. It is often convenient to define

$$\mathbf{x}^\mu := \zeta^\mu \xi^\mu \tag{5}$$

turning condition (4) into

$$\mathbf{w} \cdot \mathbf{x}^\mu \geq 0 \quad \text{(desired)} \tag{6}$$

This means that the $\mathbf{x}$–vectors must all lie on the same side of the hyperplane through the origin perpendicular to $\mathbf{w}$

2.1 Linear Separability

We have found the condition that a given pattern can be classified by a perceptron: the solvability of a given problem depends on the existence of a hyperplane separating the $\zeta^\mu = +1$ patterns from the $\zeta^\mu = -1$ ones in ξ–space . A problem that meets this condition is called linear separable. As an example of this class we will consider the logical AND-function. A perceptron with two input neurons ξ_1, ξ_2 is needed. As can be seen in figures (4) and (5) this problem is easily solved by an adequate choice of connections. This looks promising enough. Sad to say there is an example for a basic problem our simple perceptron cannot cope with : the Boolean XOR–function Picture (6) reveals, that this problem is not linear separable. A new complication arises when three points lie on exactly the same line as shown in figure (7). Points are said to be in general position when this sort of special case *does not* occurs. Generalizing this term to higher dimensions it is sensible to define that in an N–dimensional space a set of points is in general position if no more than d of them lie on a d–dimensional hyperplane

ξ_1	ξ_2	ζ
0	0	-1
0	1	+1
1	0	+1
1	1	-1

Figure 3: modified thruth table for the Boolean XOR–function

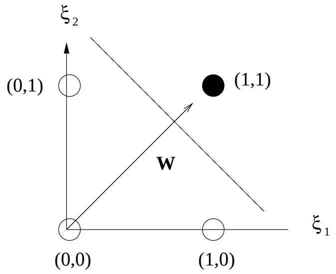

Figure 4: AND function realized by perceptron with explicit treshold

through the origin (in the case that the treshold is not given in explicit form). Failures of linear separability which are similar to the XOR–problem occur only when there are less inputs than patterns, while the second type described above can happen for any p.

2.2 The PERCEPTRON–algorithm at work

We will now discuss the question how our basic model actually contrives to learn the patterns presented to it. There are two basic timescales in a neural network. Upon sensory input patterns will be classified and adequate actions can be taken. This happens very quickly. On a much larger timescale the learning process takes place. Patterns have to be presented several times until they are "known". It has been indicated above that finding an algorithm that continually adapts the synaptic weights should be the issue here. This kind of notion originally dates back to Hebb, a physiologist, who in 1949 ventured the opinion that the learning process follows a local rule. He thought that the modification of synaptic strength is connected to the correlation between the presynaptic input signal and the triggered response in the postsynaptic neuron.

$$\Delta w_{ik} \quad = \quad \epsilon \, O_i \, \xi_k \tag{7}$$

For our purposes we consider a variant of this idea and choose the added weight in a learning step to be proportional to the product of the input and the desired output pattern. This can be formulated in the following way:

$$w_{ik}^{new} \quad = \quad w_{ik}^{old} + \Delta w_{ik} \tag{8}$$

$$\Delta w_{ik} \quad = \quad \eta \, (1 - \zeta_i^\mu O_i^\mu) \, \zeta_i^\mu \xi_k^\mu \tag{9}$$

The synaptic weights are only updated if the pattern has not been learned correctly. The parameter η ist called the learning rate. For the sake of stability

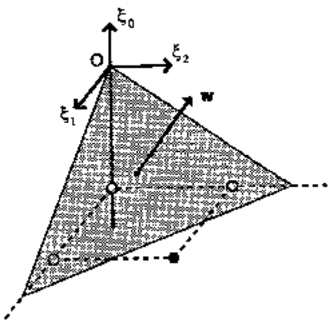

Figure 5: AND function : treshold θ treated as additional weight connected to a virtual constant input

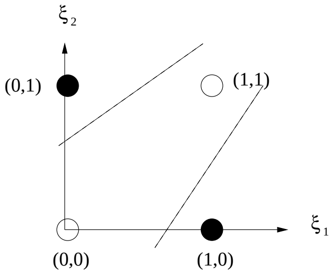

Figure 6: The XOR–function cannot be learned by the simple perceptron

it is often useful to require the size of h_i to be greater than a certain margin:

$$\zeta_i^\mu h_i^\mu \equiv \zeta_i \sum_k w_{ik} \xi_k^\mu > N\kappa \quad \kappa > 0 \tag{10}$$

The margin is chosen to scale with N in order to keep the margin size κ fixed for any number of input units. The new condition makes it necessary to transform (9) into

$$\Delta w_{ik} \quad = \quad \eta \Theta \left(N\kappa - \zeta_i^\mu h_i^\mu \right) \zeta_i^\mu \xi_k^\mu \tag{11}$$

This equation is the famous perceptron learning rule. For a single output unit (6) changes into

$$\mathbf{w} \cdot \mathbf{x}^\mu > N\kappa \quad \text{(desired)} \tag{12}$$

Consequently all points in $\mathbf{x}$–space must be further than $N\kappa/|\mathbf{w}|$ from the plane perpendicular to $\mathbf{w}$ to satisfy this condition. One can imagine that classification problems vary in difficulty. How can we measure the hardness of a given task ? Consider

$$D(\mathbf{w}) \quad = \quad \frac{1}{|\mathbf{w}|} \min_\mu (\mathbf{w} \cdot \mathbf{x}^\mu). \tag{13}$$

This quantity is exactly the distance of the worst pattern to the separating plane. Maximizing $D(\mathbf{w})$ with respect to the direction of $\mathbf{w}$ should therefore

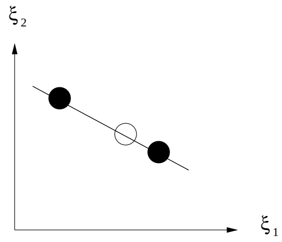

Figure 7: Patterns that are not in general position

give us the hyperplane best suited for our problem. It is the solution with minimum $|\mathbf{w}|$ and is called the optimal perceptron. Consequently the value

$$D_{max} \quad = \quad \max_{\mathbf{w}} D(\mathbf{w}) \tag{14}$$

can tell us how easy the problem is (the larger the easier, just in case ...). Let's take a look at the algorithm at work. For a single neuron/output–unit the pseudocode reads:

PERCEPTRON-Algorithm
BEGIN
Initialize $\mathbf{w}$
  DO
    randomly select pattern ξ^μ out of given set
      IF ($\zeta^\mu \sum w_k \xi_k^\mu < N\kappa$)
        THEN $\mathbf{w}^{\mathbf{new}} = \mathbf{w}^{\mathbf{old}} + \eta\Theta\left(N\kappa - \mathbf{w}\cdot\mathbf{x}^\mu\right)\mathbf{x}^\mu$
        ELSE $\mathbf{w}^{\mathbf{new}} = \mathbf{w}^{\mathbf{old}}$
  WHILE (there are unlearned patterns)
END

We will apply it to the Boolean AND–function already discussed above. It can be shown that for this problem $D_{max} = \sqrt{1/17}$. In order to get stable patterns we choose $N\kappa = 0.2$ using a learning rate $\eta = 0.5$. All components of $\mathbf{w}$ are initialized with zero. We set:

$$\xi^1 = \begin{pmatrix} -1 \\ 0 \\ 0 \\ printf(" \end{pmatrix} \qquad \zeta^1 = -1 \qquad \xi^2 = \begin{pmatrix} -1 \\ 1 \\ 0 \end{pmatrix} \qquad \zeta^2 = -1$$

$$\xi^3 = \begin{pmatrix} -1 \\ 0 \\ 1 \end{pmatrix} \qquad \zeta^3 = -1 \qquad \xi^4 = \begin{pmatrix} -1 \\ 1 \\ 1 \end{pmatrix} \qquad \zeta^4 = 1$$

The different patterns are randomly picked and fed into the network. Computer simulations indicate that at least 13 iterations will be necessary (using the chosen learning rate). A possible output is shown below.

iteration	pattern μ	weights	p1	p2	p3	p4
1	4	(-0.5 , 0.5 , 0.5)	-	-	-	-
2	1	(0.0 , 0.5 , 0.5)	-	-	-	+
3	1	(0.5 , 0.5 , 0.5)	-	-	-	+
4	2	(1.0 , 0.5 , 0.0)	+	-	-	+
5	4	(0.5 , 1.0 , 0.5)	+	+	+	-
6	3	(1.0 , 0.5 , 0.5)	+	-	-	+
7	4	(0.5 , 1.0 , 1.0)	+	+	+	-
8	2	(1.0 , 1.0 , 0.5)	+	-	-	+
9	3	(1.5 , 0.5 , 0.5)	+	+	-	+
10	4	(1.0 , 1.0 , 1.0)	+	+	+	-
11	3	(1.5 , 0.5 , 1.0)	+	-	-	+
12	4	(1.0 , 1.0 , 1.5)	+	+	+	-
13	2	(1.5 , 1.0 , 1.0)	+	-	-	+
13	2	(1.5 , 1.0 , 1.0)	+	+	+	+

Ha, you may say, he's made this up. Of course it works in this case ! Rest assured though: if the given problem is of the linear separable type i.e. there is a solution $\mathbf{w}^*$ the perceptron algorithm will converge. You still don't believe me ? Very well you have been warned. Here's the proof.

2.3 Proof of Convergence for the PERCEPTRON–Algorithm

Assuming the existence of a solution $\mathbf{w}^*$ of the problem ($\mathbf{w}^* \cdot \mathbf{x}^\mu > c$), we will show that the perceptron rule reaches it in a finite number of steps. At each time step t in the learning process a pattern is randomly chosen and the weights are only updated if $\mathbf{w} \cdot \mathbf{x}^\mu < 0$. At the beginning all weights are zero

$$\mathbf{w}(t=0) \quad = \quad 0 \tag{15}$$

During each learning step we have

$$\mathbf{w}(t+1) \quad = \quad \mathbf{w}(t) + \eta \mathbf{x}^\mu \tag{16}$$

Consequently

$$\begin{aligned} (\mathbf{w}(t+1))^2 &= (\mathbf{w}(t))^2 + 2\eta \mathbf{w}(t)\mathbf{x}^\mu + \eta^2 (\mathbf{x}^\mu)^2 && (17) \\ &\leq (\mathbf{w}(t))^2 + \eta^2 N && (18) \end{aligned}$$

Iterating the last equation from 0 to t results in

$$(\mathbf{w}(t))^2 \quad \leq \quad \eta^2 N t \tag{19}$$

Consider the scalar product $\mathbf{w}^* \cdot \mathbf{w}(t+1)$. We find:

$$\begin{aligned} \mathbf{w}^* \cdot \mathbf{w}(t+1) &= \mathbf{w}^* \cdot \mathbf{w}(t) + \eta \mathbf{w}^* \cdot \mathbf{x}^\mu \\ &\geq \mathbf{w}^* \cdot \mathbf{w}(t) + \eta c \qquad c > 0 && (20) \end{aligned}$$

Iterating the last equation from 0 to t leads to

$$\mathbf{w}^* \cdot \mathbf{w}(t) \quad \geq \quad \eta c t \tag{21}$$

Combining (19) and (21) we conclude using the Cauchy–Schwarz inequality $(\mathbf{w}^*)^2(\mathbf{w}(t))^2 \geq (\mathbf{w}^* \cdot \mathbf{w}(t))^2$:

$$\begin{aligned} \eta^2 N t (\mathbf{w}^*)^2 &\geq \eta^2 c^2 t^2 \\ \Longrightarrow \quad t &\leq N \frac{(\mathbf{w}^*)^2}{c^2} := t^* \end{aligned} \tag{22}$$

The number of iterated learning steps is therefore bounded by t^* and the algorithms converges in finite time. (q.e.d.) If the problem is a hard one the constant c will be quite small, resulting in a greater t^*.

2.4 The ADALINE learning rule

So far we have only discussed binary treshold–units. In this chapter we will examine the effects of using continuous–valued neurons endowed with a differentiable function $g(x)$. This generalization allows us to define an energy–function $E[\mathbf{w}]$ (differentiable with respect to the weights) which assesses the state of the system. Then known algorithms like gradient–descent can be used to minimize the error during the learning process. The adaline learning rule is derived from linear units with $g(h) = h$. Defining a cost–function by

$$\begin{aligned} E[\mathbf{w}] &= \frac{1}{2} \sum_{i,\mu} (\zeta_i^\mu - O_i^\mu)^2 \\ &= \frac{1}{2} \sum_{i,\mu} \left(\zeta_i^\mu - \sum_k w_{ik} \xi_k^\mu \right)^2 \end{aligned} \tag{23}$$

it is obvious that E approaches zero as we approach the solution satisfying

$$\zeta_i^\mu = \sum_k w_{ik} \xi_k^\mu \tag{24}$$

Consequently we have to move down the Energy surface defined by (23) in $\mathbf{w}$–space. The gradient descent algorithm suggests changing each w_{ik} by an amount proportional to the gradient of E at the present location:

$$\begin{aligned} \Delta w_{ik} &= -\eta \frac{\partial E}{\partial w_{ik}} \\ &= \eta \sum_\mu (\zeta_i^\mu - O_i^\mu)\, \xi_k^\mu \end{aligned} \tag{25}$$

This can also be achieved by successive presentation of the input patterns to the network using the updating rule

$$\Delta w_{ik} = \eta\, (\zeta_i^\mu - O_i^\mu)\, \xi_k^\mu \quad \text{ADA(ptive)LINE(ar) rule} \tag{26}$$

The last equation looks very similar to our rule for the treshold unit network. But now we are working with continuous–valued neurons. Moreover the condition for the existence of a solution to learn a set of patterns is a different one : the patterns have to be linearly indepentent. We also remark that gradient–descent will only approach the solution asymptotically. So why even consider adaline ? The reason lies in the fact that it can be easily generalized to more layers in contrast to the Hebb rule. Multi–layered networks can overcome a lot of the simple perceptron's limitations.

2.5 Storage capacity of the simple perceptron

It has been hinted at above that the PERCEPTRON–algorithm is more likely to despair on a problem when the number of patterns exceeds the number of neurons. This leads us to a general question : how many patterns can securely be stored in a simple perceptron ? For continuous–valued units (even with nonlinear output) this question can be easily answered since they require linear independent input patterns to learn properly. Consequently their storage capacity is $p_{max} = N$. Treshold units are somewhat harder to deal with. One finds that for continuous valued inputs

$$\lim_{N \to \infty} p_{max} = 2N \tag{27}$$

This result can be understood qualitatively by looking at the number of possible outputs $C(p, N)$ for any p points in general condition embedded in an N–dimensional space that can be separated by a hyperplane as a function of the load parameter $\alpha = p/N$. Some thought reveals that this number is given by

$$C(p, N) \quad = \quad 2 \sum_{i=0}^{N-1} \binom{p-1}{i} \tag{28}$$

From the symmetry of the binomial coefficients it follows that

$$C(2N, N) = 2^{N-1} \tag{29}$$

Consequently the graph in figure (8) goes through $1/2$ at $p = 2N$. For large N a kind of phase–transition seems to take place at this point that getting more and more distinct for increasing N. For $p \leq N$ we get : $C(p, N) = 2^p$, i.e. all possible outputs can be learned and consequently in this scope linear independence and general position are equivalent.

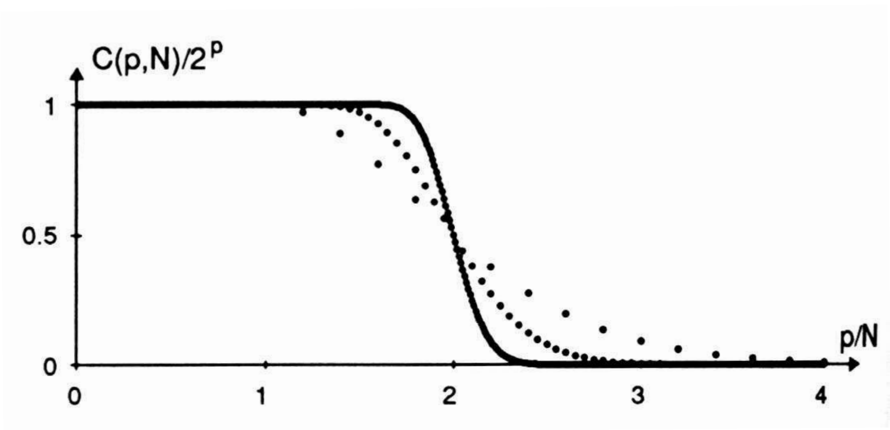

Figure 8: C(p,N) plotted against the load parameter for N =5, 20 and 100

3 Outlook

After having learned the desired patterns a perceptron is able to generalize, i.e. an input not encountered so far can also be processed. Whether the achieved output can be regarded as sensible is a different issue. Interesing questions arising form this problem are therefore :

- how well can a particular function from input– to output–space be approximated by training a network with a fixed number of examples and
- how long does it take to achieve a given performance level ?

Also different learning rules, activation functions and network architectures have to be examined. Multilayered perceptrons for example are able to solve the XOR–problem simply by recoding the input in the hidden units.

The extreme case that all neurons are connected with each other can be used to simulate associative memory; all neurons serve as both input–and output–units. If the connection strengths have the right values certain input patterns are stable, i.e they are reproduced. The important property of this network is that input patterns close to the stable patterns caused for example by a noisy environment converge on the same output as the neurons are updated using a Hebbian–like learning rule based on (7). Consequently one can think of memories as stable firing patterns in this architecture. When the network connections w_{ik} are assumed to be symmetric the Hopfield model results. In this case an energy function can be introduced making it possible to use well known statistical mechanics methods for its analysis because of the formal equivalence to spin glasses.

Mildly said the possible scope of neural networks is very large. Rumour has it that they are the second best way of doing anything. Neural networks can be used for pattern recognition, for solving optimization problems like "The travelling salesman" and some people claim that neural networks are even able to analyze and predict the stock market. On the other end of the income–ladder network models more closely related to biological reality help scientists to understand the underlying processes governing the brain.

I would like to conclude this paper with a quote:

> I used to think that the brain was the most wonderful organ in my body. Then I realized who was telling me this.
> (Erno Philip)

References

[1] E. Domany, J.L. van Hemmen, K. Schulten (Eds.) : Models of Neural Networks I, Springer (1995) :

W. Kinzel, M. Opper : Dynamics of Learning (p. 157–178)

[2] J. Hertz, A. Krogh, R.G. Palmer : Introduction to the theory of neural computation, Addison–Wesley (1991)