Memorandum COSOR 94-10

Exact Solution and Learning of Binary Classification Problems with Simple Perceptrons

J.A. Matla
H.P. Stehouwer
J. Wessels

Eindhoven, March 1994
The Netherlands
Abstract: This paper discusses the effect of response functions on the performance of multi-layered perceptrons. It will be shown that the N-bit parity problem, and even any binary classification problem, is exactly solvable with a simple perceptron using the right (non-monotonic) response function. We discuss problems which arise, and present some results, when using non-monotonic response functions together with learning. As a new approach we introduce function-learning to adjust response functions during the learning process.

Keywords: Neural networks, multi-layer perceptron, N-bit parity, response functions, pattern classification, function-learning.

1. Introduction

During the last few years there has been an increasing interest in the use of multi-layered perceptrons (MLP’s) for various kinds of applications. As a result of this trend, there has been done a lot of research on this subject. Most efforts in today’s research are focused on improvements in the learning process of such an MLP.

The usual way to learn an MLP is by using the standard backpropagation learning algorithm (BP) [Rumelhart et al., 1986]. Since BP is relatively slow, in the last few years various improvements of BP have been proposed which, in many occasions, make learning faster. There are several ways to speed-up learning. We distinguish between two types of adjustments: on the learning algorithm and on the network-architecture.

An example of the first type of adjustment is by changing the learning-rule. Examples of this feature are Riedmillers’ RPROP [Schiffmann, 1993] and Fahlman’s Quickprop [1989]. RPROP is based on the Manhattan learning-rule. This learning-rule uses fixed update values, not influenced by the gradient. Quickprop comprise a collection of heuristic improvements and a second order approach, based on the false position principle, to get an approximation of the second derivative. Both heuristic methods can speed-up learning.

Besides changing the learning-rule, we can choose another error-measure than the usual quadratic measure in the least square error criterion. Van Ooyen and Nienhuis [1992] proposed an error-measure which is better fitting with a logistic response function. Another example is the hyperbolic arc tangent used by Fahlman in his Quickprop algorithm.

A possible adjustment of the network-architecture is the choice of the response function of a unit. Speed-up gained by such an adjustment is mostly caused by the shape of the response function and in particular the course of its slope. Examples of response functions are sigmoidal functions, like the logistic function (see figure 4.1) and the hyperbolic tangent [Kalman, 1992], or less obvious choices, like Gaussian functions [Umano et al., 1992] or goniometric functions [Lee, 1992].
The choice of the response function is important for the learning-performance. Nevertheless, there often seems to be no good justification for using a particular response function. Most of the time it seems like it is only a good guess, based on intuition and experience. As Xu et al. [1992] stated: 'other functions have been tried'. So, it looks like most of the inventors of such response functions seem to have just a vague idea why their particular response function is working well. Their idea is mostly based on the shape of the function; the slope is better differentiable, it is better fitting with the error-curve, or it is better in dealing with flat spots.

In this paper we focus on the effect of choosing a certain response function and we show why some response functions perform better than others. We show that it is possible to construct problem-suited response functions, with which we are able to exactly solve particular problems. An example of a constructed response function can be found in Stork [1992], where the N-bit parity problem is solved with just two hidden units.

Of course construction of exact MLP's is important, but there are many cases where we want the MLP to determine the dynamics of the problem at hand. A logical continuation of making problem-suited response functions will then be function-learning. By function-learning we intend to let a learning algorithm determine what shape of the response function is suitable for the problem at hand. Examples of function-learning can be found in Tawel [1989] and Umano et al. [1992], where the steepness of the response function is taken as a learning-parameter which can be adjusted by a learning-algorithm.

We discuss a more general approach of function-learning, based on non-monotonic response functions. As we will show, non-monotonic response functions can provide a better learning-performance, depending on the problem at hand. We will restrict ourselves to a first sketch of function-learning, besides we discuss some problems which arise when we want to implement function-learning.

The remaining part of this paper is organized as follows. First, in section 2, we will discuss a method to view binary classification problems (BCP's) as a way to order a number of vertices in hyperspaces by separating them with hyperplanes. In section 3 we conclude that any BCP can be exactly solved by a simple perceptron with a suitable response function. In section 4 we discuss problems which arise by using problem-suited response functions and we will sketch a algorithm for function-learning. In section 5 we show some results which we have reached with particular response functions. Finally, in section 6, we give some conclusions, remarks, and directions for further research.

2. Monotonic step response functions

We start with a few assumptions. First, the neural network models we consider in this paper will be multi-layer feed-forward networks (MLP's). When we talk about MLP's, we do not count the inputs as a layer, but only the layers with real processing units. With a simple perceptron (SP) we mean a perceptron with no hidden layers. For a more detail description, see also [Hertz et al., 1991]. We consider just one output unit, this will be no limitation because we will examine only SP's. In SP's the output units with their weights are mutually independent.

Furthermore we assume that any problem we discuss, if not explicitly stated otherwise, will be a binary classification problem (BCP). We consider such a BCP as a classification...
problem where a set of objects must be distributed among a set of subsets. We assume that any object, that must be classified, can be considered as a binary input, and any subset can be considered as a binary output. A BCP is then a problem to classify a set of inputs among a set of particular output categories. Which BCP's an SP can or cannot solve will strongly depends on the kind of response function used.

Consider an SP with N binary inputs, and one binary output (see figure 2.1).

![Figure 2.1: A simple perceptron.](image)

We define an input-output pair \( \mathbf{J}, \mathbf{I} \) as \( \{ \mathbf{J}, \mathbf{I} \} \), where \( \mathbf{J}, \mathbf{I} = (\mathbf{J}_1, \ldots, \mathbf{J}_N) \), \( \mathbf{I} \in \{0, 1\} \) denotes the input pattern, and \( \mathbf{I} \in \{0, 1\} \) denotes the output (target). Let \( \mathbf{w} = (w_1, \ldots, w_N) \) be the weight-vector. The projection of the input patterns, or net input of the output unit, is defined as the inner product of the weight-vector and the input-vector, \( \mathbf{w} \cdot \mathbf{J}, \mathbf{I} \). The net input for the output unit by presentation of input-output pair \( \mathbf{J}, \mathbf{I} \) is denoted as \( h_{\mathbf{J}, \mathbf{I}} = \mathbf{w} \cdot \mathbf{J}, \mathbf{I} - \theta \), where \( \theta \) denotes the bias. The output of the output unit is \( O = F(h_{\mathbf{J}, \mathbf{I}}) \), where \( F \) is called the response function.

Consider an SP with one unit, which can take only two output values, 0 and 1, depending on the value of the fan-in being below or above some fixed bias \( \theta \). In this section we consider as response function the unit step function, or Heaviside function \( \Theta \) (see figure 2.2):

\[
\Theta(x) = \begin{cases} 
1, & x \geq 0 \\
0, & \text{otherwise}
\end{cases}
\]

![Figure 2.2: The Heaviside function \( \Theta \).](image)

The output of the unit is a monotonic response function, similar to the \( \text{sgn} \) function, only restricted to Boolean variables instead of negative and positive ones. \( F \) defines a separating hyperplane \( \mathbf{w} \cdot \mathbf{J} - \theta = 0 \), which divides the \( \mathbf{J} \)-space into two parts: one in which the output value is 0, and one in which this value is 1. An SP that solves a given BCP only then exists if the BCP is linearly separable. Linear separability means that there exists a weight-vector
which provide a separating hyperplane, perpendicular to the weightvector, which divides the $\xi$-space into two parts, separating the inputs into the two output categories with $\xi=0$ and $\xi=1$ respectively. If there exists no such hyperplane, then the problem cannot be solved by an SP using units with monotonic response functions. A BCP can then be viewed as a problem to find a separating hyperplane for the output unit which separates the inputs, with the two different targets, into two different output classes.

An SP, using a monotonic response function as described above, is able to solve problems which are linearly separable. Problems which are not linearly separable, like the XOR problem, are not solvable with an SP using units with such monotonic response functions. To illustrate the concept of linear separability, we show two examples, the AND function which is linearly separable and the XOR problem which is not.

First let us take a look at the AND function with the truth-table shown in Table 2.1. The desired output is 1 when both inputs are 1, and 0 otherwise.

Table 2.1: The AND function

<table>
<thead>
<tr>
<th>$\xi_1$</th>
<th>$\xi_2$</th>
<th>$\xi$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Figure 2.3 (a) shows the problem in $\xi$-space: the solid dot means that the target of the input pattern is a 1, an open dot means that the target is a 0. It is easy to draw a line (a one-dimensional hyperplane) to separate the "one"-corner from the rest. So the problem is linearly separable and an SP can solve it. A matching SP is shown in figure 2.3 (b).

Figure 2.3 (a): The AND function is linearly separable.

A problem which is not linearly separable is the XOR problem with the truth-table shown in Table 2.2. The desired output is 1 if one or the other of the inputs is 1 and 0 if they are both...
1 or both 0. This is also the simplest case of the N-bit parity function which we will discuss later. When we look at figure 2.4, there is no possible way to draw a line that separates the two distinct classes of targets. With two lines it is possible to achieve such a partition, but with the monotonic Heaviside response function we are using here, we only can provide one such line. This is clear when we look at figure 2.4 and is also algebraically shown in Hertz et al. [p. 97, 1991].

Table 2.2: The XOR problem

<table>
<thead>
<tr>
<th>$\xi_1$</th>
<th>$\xi_2$</th>
<th>$\xi$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

Figure 2.4: The XOR problem is not linearly separable.

3. Non-monotonic step response functions

In this section we will show that it is rather easy to exactly solve the XOR problem with an SP using a non-monotonic response function. Afterwards we will show that the same idea holds for the N-bit parity problem and even for any BCP. Before we do so, let us again take a short look at the kind of response function we have used above. This function is a hard limit of a sigmoidal function, which is a monotonic response function. The reason that we are not able to solve the XOR problem with such a response function, is that it is not able to provide more than one separating hyperplane. As a result of this, it can only separate the $\xi$-space into two parts where we need at least three such parts.

The usual way to deal with this problem is to incorporate an extra layer with hidden units which can provide the extra hyperplane we need. Another approach, rarely used in literature, is the use of a non-monotonic response function which can provide more than one hyperplane. This is the approach we will use here. Again we will illustrate this idea by using step response functions with which it is easy to see how the idea works in $\xi$-space. Having such a step response function, it is not difficult to find a continuous analogon with the same characteristics.

First we will construct an SP that exactly solves the XOR problem using an unusual response function. Consider the SP in figure 3.1 (a) and associated step response function in figure 3.1 (b). An SP using this kind of response function actually solves the XOR problem exactly!
The reason why this works, is because of the choice of the response function which provides two separating lines in $\xi$-space. These two separating lines directly follow from the SP and the response function we have used, and are shown in figure 3.1 (c):

\[\xi_1 + \xi_2 = 0.5\]
\[\xi_1 + \xi_2 = 1.5\]

The same idea can be used for the construction of an SP that exactly solves the N-bit parity problem. The N-bit parity problem is based on the number of ones in the inputs. The problem is to decide whether or not this number is even, which means a 1, or odd, which means a 0. With a right response function this again is no problem. A general construction of an SP that solves the N-bit parity problem is then as follows:

- take N inputs $\xi_i$ (i=1,...,N), and one output unit,
- let the weights $w_i$ (i=1,...,N) have value 1, and let the bias $\theta$ have value 0,
- choose a step response function, which has value 0 if the (integer) argument is even, and value 1 if the argument is odd.

To illustrate this idea, let us take a look at figure 3.2 (a) which shows an SP that exactly solves the 4-bit parity problem if the response function of figure 3.2 (b) is applied.
The same idea holds when we try to solve the N-bit parity problem for any arbitrary large N. Again the problem is exactly solvable because of the capability of the response function to provide N (N-1)-dimensional hyperplanes, able to divide the N-dimensional space in N+1 slices.

Obviously, choosing a response function can have great influence on the performance of a perceptron. It can make the difference between a perceptron that is able to learn a given problem right and a perceptron which is not.

With the concepts discussed above in mind, we can think of a way to construct an SP which can exactly solve any BCP. The first thing we need is a right response function which can provide enough hyperplanes to divide the space into the number of parts we need. As we will show, given any BCP, it is no problem to construct an SP, with suitable response function, exactly solving it. The approach we will use here is by providing enough hyperplanes to separate all possible input patterns, which are vertices in ξ-space, from each other.

Consider a BCP with N binary inputs. Then there are $2^N$ possible input patterns $\xi^\mu$, which we can number as follows. Take an arbitrary input pattern $\xi^\mu$, such an input pattern can be considered as a binary number. We number the input patterns $\xi^\mu$ by their corresponding decimal value. The way we can exactly solve the BCP with an SP is with the following construction (without loss of generalization we still consider just one output unit):

- take an SP with N inputs,
- give weight $w_i$ (i=1,...,N) value $2^{i-1}$,
- give the bias $\theta$ value 0,
- choose the response function so that the output $O$ of pattern $\xi^\mu$ ($\mu=0,...,2^N-1$) matches $\zeta^\mu$.

When we look at a BCP with an arbitrarily large number of N inputs, we see that this construction is able to provide $2^N-1$ (N-1)-dimensional hyperplanes which can separate the $2^N$ input patterns $\xi^\mu$. Again these hyperplanes simply follow from the SP and response function we used:

$$w_\xi k = k + \frac{1}{2} \text{ with } k=0,...,2^N-2.$$
Notice that this method holds for any possible input-output combination. To illustrate this idea, consider the solution of the XOR problem in figure 3.3 (a), which shows the SP, and figure 3.3 (b), which shows the associated response function, according to the construction given above.

![Figure 3.3 (a): A constructed SP that exactly solves the XOR problem with the response function of fig. 3.3 (b).](image)

![Figure 3.3 (b): The constructed response function for the XOR problem.](image)

Again we can look at the 2-dimensional plot of the $\xi$-space to show the positions of the separating hyperplanes, see figure 3.3 (c). We see that this method is able to provide three hyperplanes, although we only need two of them:

- plane 1: $\xi_1 + 2\xi_2 = 0.5$
- plane 2: $\xi_1 + 2\xi_2 = 1.5$
- plane 3: $\xi_1 + 2\xi_2 = 2.5$

![Figure 3.3 (c): The XOR problem in $\xi$-space.](image)

Notice the difference between the two SP’s that we have found to solve the XOR problem, and in particular the difference between the two response functions and the provided separating hyperplanes. In the first solution more information is used about the dynamics of the problem, while the second approach is more general. Despite a small difference, the general shape of the functions is the same in this sense that both use two separating hyperplanes to separate the two different kinds of outputs.

Considering larger parity problems, we can observe an obvious difference about the use of prior information. Take for example the 4-bit parity problem. In the example discussed in figure 3.2 (a)-(b), it is clearly seen that we use four hyperplanes to separate the $\xi$-space. When we look at the response function of the constructed SP for the 4-bit parity problem in
figure 3.4, we see that we here use fifteen \((2^5-1)\) hyperplanes, of which ten are effective.

![Figure 3.4: The constructed response function for the 4-bit parity problem.](image)

One drawback to this kind of method of constructing a suitable response function for BCP's is the fact that the perceptrons are problem-dependent, which means that its response function strongly depends on the structure of the problem we are trying to solve. Using such a fixed constructed response function means that there is hardly any freedom to use this function in more general problems. Without any hidden units, the problem is to choose a response function which provides the right number of hyperplanes, instead of choosing the right number of hidden units.

When we want to use the method of finding the right response function in more general cases, in which it is not always obvious what the desired outputs are, we must think of a kind of function-learning together with the usual weight-learning. Of course this creates extra difficulties. This is a result of the curse of dimensionality. This means that the problems of learning, first divided over a number of hidden units, now come back in a single layer. On the other hand, one major advantage is that we can put prior knowledge into the response function, which means that the learning-problem can become easier.

### 4. Continuous response functions and function-learning

When we want to train an MLP, we usually use the backpropagation learning-algorithm (BP) [Rummelhart et al., 1986]. Using BP requires differentiable (continuous) response functions [Hertz et al., 1991]. Therefore, step response functions are useless in that case. However, it usually is no problem to make a continuous variation on a step response function.

Examples of a continuous approximation of the Heaviside response function, as described in section 2, are sigmoidal functions like the commonly used logistic function (see figure 4.1):
$S(x) = \frac{1}{1 + e^{-\alpha x T}}$, with temperature $T$.

Figure 4.1: The logistic function.

Justification of the choice of a sigmoidal function in literature is based on either neurobiological grounds, mathematical grounds, pragmatic grounds, or, most commonly, on intuitive grounds. Most of the intuition comes from vague arguments related to its squashing capabilities and to the fact that the gain (slope) through the axis can be easily manipulated. It is generally believed that "the exact details of the sigmoidal function are not critical in a backpropagation network" [p.19, Caudill 1990]. While this is true for the performance of an already-trained network, there are important differences that can affect training dramatically. Therefore, attention should focus on choosing a monotonic response function that exhibit the best properties for training [Kalman, 1992].

In contrast with monotonic response functions, the exact details of the shape of a non-monotonic response function are of great influence. We will show that in particular the mutual distance between the provided separating hyperplanes will be important. Beside that, the same attention should be focused on choosing non-monotonic response functions that exhibits the best properties for training.

For choosing a non-monotonic response function there are several candidates. For example, a simple continuous approximation of the response function of figure 3.1 (b) is a Gaussian function of the form:

$$G(x) = \exp\left(-\frac{1}{2} \left(\frac{x}{T}\right)^2\right),$$

where $T$ is the temperature, a measure of the steepness of the function.

Figure 4.2: The Gaussian function G.

For the response function in figure 3.2 (b), a goniometric function like the sinus can be useful, but for less regular step response functions like the one in figure 3.4 it will be more difficult to find a suitable continuous function. In cases where there is not a simple relation between the different input-output patterns, it sometimes will be hard to find a continuous approximation. Nevertheless, the shape of all possible suitable response functions for the
different BCP’s, will have a number of common characteristics:

- they all must be non-monotonic, able to provide more than one hyperplane,
- the distance between two different separating hyperplanes is not necessarily the same,
- most response functions are suitable for just a few BCP’s, so they are problem-dependent in this sense that they can deal with certain restrictions of the problem being observed.

As mentioned, sometimes it can be difficult to find a suitable response function for a particular problem. And for the general case of finding a suitable response function that can deal with all BCP’s it will be almost impossible. Nevertheless, there are many cases in which a general approach of finding a suitable response function would be useful. A solution then could be, what we will call, function-learning. With function-learning we mean a process which evolves a function during the learning-process. Of course this will be a difficult process, because all the difficulties which were first divided over the hidden layers, will come back in a single layer and response function.

Before we discuss function-learning any further, we will take a short look at the effect of the learning-parameters we already know, and their effects in input space. To do so, we go back to the case where we have a monotonic response function, which can provide just one hyperplane. The position of this hyperplane depends on the weights \( w \) and the bias \( \theta \) (see also figure 4.3):

- the direction of the weight-vector \( w \) defines the rotation of the hyperplane. A change in the direction of \( w \) will rotate the hyperplane.
- the bias \( \theta \) allows the hyperplane not to go through the origin. No bias \((\theta=0)\) implies that the hyperplane goes through the origin, a non-zero bias places the hyperplane on a distance \( \theta \) from the origin. The bias will shift the complete hyperplane. A positive change of the bias will move the hyperplane in the direction of the weight-vector and a negative change moves the hyperplane in the opposite direction.

![Figure 4.3: Effect of learning-parameters.](image-url)
For a non-monotonic response function, providing more than one hyperplane, these two learning-parameters have the same effect on all hyperplanes:

- the weights provide the same rotation for all hyperplanes,
- the bias provides the same shift (in the same direction) for all hyperplanes.

Because of the limitations of these learning-parameters, every pair of hyperplanes will always relatively keep the same distance to each other. A change in the position of one hyperplane, will always effect the position of the other hyperplanes. What we want is to be able to adjust the position of a single hyperplane. It will be clear that the weights and the bias, the usual learning-parameters of BP, have not an independent effect on the different hyperplanes. So, these parameters are useful for the refinement of the position and direction of all hyperplanes, but have not the effect we want to achieve on the different hyperplanes. Therefore, we must search for alternatives to make the positions of hyperplanes variable.

A parameter which is sometimes used as a learning-parameter is the measure of the steepness of the response function, the so-called temperature T. Therefore, it can be viewed as a function-parameter, which can be adjusted to get a better tuning of the response function. A way to get such an adjustment is with the use of so-called temperature-learning. The derivation of temperature-learning is similar to standard BP and is proposed by Tawel [1989] and used with a Gauss function by Umano et al. [1992].

The effect of temperature-learning on non-monotonic response functions is that pairs of separating hyperplanes are not having a fixed mutual distance anymore. By changing the temperature the mutual distances of the hyperplanes will vary, although relatively, which allows the neural net to find a better adjustment to the problem.

One major drawback is the fact that we are still unable to treat every hyperplane independently. All three learning-variables (w, θ, and T) have the property that they change the position of every hyperplane relatively, with regard to every other hyperplane. What we want is a method to treat all hyperplanes individually. To accomplish this we can think of:

- individual weights for every hyperplane,
- an individual bias for every hyperplane,
- a parameterized response function.

The first two options are almost impossible to realize with the standard MLP and the kind of units we are using. The last alternative is, theoretically, easier to realize. The way in which this can be done is by an iterative process, which we call function-learning.

In the now following we construct a response function and sketch an algorithm of function-learning. With this response function we are able to treat all hyperplanes individually. The shape of the response function is again non-monotonic, able to provide a number of hyperplanes, achieved by the use of a couple of alternating tops and valleys. The response function we use is a constructive one, build up out of Gaussians. The reason we are using Gaussians is that they are easy to compute and naturally provide two hyperplanes. The way we construct the initial response function is as follows:

- for a top we take a common Gaussian G,
- for a valley we take a Gaussian-like function Ĝ=1-G,
• glue both functions together at the point where both reach value ½, and cut off the loose ends,
• repeat the last step until the initial response function provides enough hyperplanes.

We then get an interval response function, build up out of a number of Gaussian functions which we can treat individually in the following way (see figure 4.4). $g_i$ denotes the position of Gaussian $i$ and $b_i$ is a measure of the width of Gaussian $i$ ($i=-M,...,N$).

The obtained interval function is continuous, but not differentiable in $g_i \pm b_i$ (unless $b_i=b_j$ for all $i$ and $j$). Because the direction of the slope is most important, this will be no important impediment.

![Figure 4.4: A constructed interval response function, with variable distance between separating hyperplanes.](image)

Let us take a closer look at the Gaussian functions we are using:

$$G(x) = \exp\left(-\frac{1}{2}x^2\right) \quad \text{and} \quad \hat{G}(x) = 1 - \exp\left(-\frac{1}{2}x^2\right).$$

The constructed response function has the following form (see also figure 4.4):

$$F(x) = \begin{cases} 
G\left(\frac{x-g_i}{T_i}\right), & g_i - b_i \leq x \leq g_i - b_i \quad \text{and} \quad i \text{ even} \\
\hat{G}\left(\frac{x-g_i}{T_i}\right), & g_i - b_i \leq x \leq g_i + b_i \quad \text{and} \quad i \text{ odd} \\
G\left(\frac{x-g_M}{T_M}\right), & x < g_M - b_M \\
G\left(\frac{x-g_N}{T_N}\right), & x > g_N + b_N 
\end{cases}$$
Here $T_i$ is the temperature, a measure of the steepness of the Gaussian $i$. Notice that we have taken $M$ and $N$ even to get Gaussian tops at both ends of the response function. The parameters $g_i$ and $T_i$ can be written as an expression in $b_i$:

$$g_i = \begin{cases} 
  b_0 + 2 \sum_{j=1}^{i-1} b_j + b_i, & \text{if } i > 0 \\
  0, & \text{if } i = 0 \\
  -(b_0 + 2 \sum_{j=i+1}^{N} b_j + b_i), & \text{if } i < 0 
\end{cases}$$

$$T_i = \frac{b_i}{\sqrt{\ln 4}}, \text{ obtained from } F^{-1}(g_i \pm b_i) = \frac{\sqrt{2}}{2}.$$

Of course, there are many other ways to construct a response function out of Gaussians. One of them is to add a couple of Gaussians into a new function. A drawback of this method is that we get a function which will not immediately have the right shape, because of extreme values unequal to one or zero. Anyway our construction has also the advantage that it is easy to compute.

Function-learning is then a process of adjusting the parameters $b_i$, in such a way that the individual hyperplanes will get the right position. The variable width's $b_i$ of the partial functions provides the variable distance between the hyperplanes, and the $g_i$ determine the positions. To get the right adjustment, we can vary the width's $b_i$ according to some learning-rule. We get the following algorithm for function-learning:

- construct an initial response function, like the one proposed above, with a fixed number of provided hyperplanes,
- check the error deviation of all input-output patterns,
- adjust the response function, according to some learning-rule, by varying the width $b_i$ of every partial (Gaussian) function,
- repeat the last two steps until conditions are satisfied.

Another possibility of function-learning is to let the number of hyperplanes be variable, by making it possible to add or delete hyperplanes. This can be done by putting in an extra top and valley, or by just deleting them when they are not necessary, according to some learning-rule. Of course this can be done only with two hyperplanes every time. In practice deletion of hyperplanes will probably be the easiest to detect (delete hyperplanes when there is no vertex between them). So, we propose a method which starts with plenty of hyperplanes. If hyperplanes are not necessary, they can be deleted or simply shoved away into infinity.
Function-learning must be used in combination with the usual weight-learning. We then get an extended BP with function-learning. In which way both kind of learnings must be used through one another will be a matter of research.

The implementation of function-learning (and especially the latter extension with a variable number of hyperplanes) may be hard to realize in hardware, because we will need a new sort of unit, which can deal with dynamic response functions. Anyway, we think that using non-monotonic response functions, with function-learning, can add new possibilities to learning.

5. Results.

To examine the mentioned ideas of non-monotonic response functions, we have done some tests on some toy problems, in particular the XOR and the N-bit parity problem. With these tests we have not used any kind of function-learning, but have restricted ourselves to non-monotonic response functions with a fixed number of separating hyperplanes with constant mutual distance.

The response functions we used, were constructions of Gaussians of the kind described above. In this way we constructed symmetric response functions (M=N) with several tops, say t tops with t odd. All b_i and T_i were taken equal, b=b and T=T, for all i=−N,...,N. This means that the distance between succeeding hyperplanes was fixed. Despite this imposed restriction, we were able to obtain quite satisfying results.

Training was done with standard BP with a momentum term. Weights were adjusted per sample, which means that every time an input-output pair was presented to the network the weights were adjusted. Learning times are reported as the number of epochs which were necessary to train the network and are the averages of 25 or 100 training results. An epoch stands for one presentation of all input-output pairs.

When adding more tops, we also enlarged the steepness T_i of the response function. This was done to make sure that the separating hyperplanes were not situated too far away, in which case they have no influence and are superfluous. Therefore, we also have done some tests to get an idea of an optimal top-density. The results showed that it is advisable to let all tops be positioned in a fixed range, which means that the steepness has to be dependent on the number of tops. As a result of varying the steepness, we also observed that the learning-rate of the weights has to be dependent on the steepness of the response function used. The reason for this is that larger steepness provides larger deviation, which caused a too large adjustment of the weights. Therefore we used a top-depending steepness, with an adjusted learning-rate.

We first trained a 2-layer perceptron (2-MLP) with Gaussian response functions to learn the N-bit parity problem, results are shown in Table 5.1. As shown, the idea of using more hyperplanes (tops) is working very well, compared to a logistic response function with just one hyperplane. We cannot use too much hyperplanes, there seems to be an optimal number. There are two reasons for this. The first reason is that adding more tops will eventually have no effect because of the fact that there are already enough separating hyperplanes. The other reason is due to the fact that we have adjusted the top-density. It may be the case that because of this, the provided hyperplanes will be situated too close to each other. As a result of this, the input space will become full with separating hyperplanes. Positioning of hyperplanes will then become too precise.
Because of the stiffness of the response function used, with more hyperplanes the positioning of the hyperplanes becomes more precise. When using a 2-layer MLP, this problem can be overcome with the use of hidden units. When using an SP, we have no hidden units. In this situation it will become more important to adjust the response function to the problem. Tests on the N-bit parity problem (2≤N≤5) have shown that an SP is able to solve those problems, but that it is important to choose the right number of tops (planes) and the right steepness (mutual distance between hyperplanes). When these parameters are well chosen, it takes few epochs to learn. Otherwise, the SP will get stuck in a local minimum. This last observation is a result of the fact that the response function we have used is very stiff and requires more refinement than a sigmoidal function, because of the positioning of several hyperplanes (with the same mutual distance) instead of just one. Therefore we think that function-learning can be of great importance to avoid local minima.

As shown the usage of more hyperplanes, in case of parity, is acting very well and with hidden units it leads to superior results. This is a result of the fact that we can separate more vertices with more hyperplanes. In this way we add extra degrees of freedom, which can make learning easier, but, also worse.

6. Concluding remarks and suggestions for further research.

We have shown that choosing a certain response function is of great influence on capabilities and performance of an MLP. When restricted to monotonic response functions, not linearly separable BCP’s are not solvable with an SP. On the contrary, when using non-monotonic response functions, it is possible to exactly solve BCP’s with less, and even without, hidden units. This suggests that for general classification problems, certain constraints known a priori can be incorporated into the response functions themselves, before learning begins.

Because we are able to put certain constraints (i.e., the necessary number of separating hyperplanes) of the observed BCP into the response function, we are able to use fewer hidden units. Therefore, all difficulties of learning will be concentrated in fewer units and in particular in the choice of response functions. Therefore, choosing appropriate response functions is important.

In this paper we have concentrated on the effect of response functions. It was shown that this effect strongly depends on the number and positions of the separating hyperplanes. We saw that, given a BCP, it is easy to construct an SP exactly solving it. Nevertheless, when we wanted to use a learning algorithm with continuous non-monotonic response functions,
we encountered some problems. Those problems were due to the fact that the mutual distance between separating hyperplanes was constant because of the stiffness of the response function. Standard BP is only able to move all hyperplanes at once, not to change mutual distances between hyperplanes.

The solution we found was function-learning. With function-learning we can adjust the response function in a way that we are able to treat individual hyperplanes independently, concerning their mutual distance. We sketched an algorithm for function-learning, but have not tried to specify learning-rules. Our first aim was to show the effects and the importance of non-monotonic response functions.

There are still many open problems before we can use extended BP with function-learning, e.g., when and how we are going to use function-learning during the learning-process. The problem of how to use function-learning is mainly concentrated in the enforcement of learning-rules, e.g., how to adjust the variable width’s $b_i$. We also mentioned the possibility to add or delete hyperplanes (tops) within the response function during the learning-process. This process will also require sophisticated learning-rules.

We concentrated on varying the mutual distances between hyperplanes. Another possible research topic will be the problem to rotate every hyperplane individually (for instance with an individual bias per hyperplane). We were not able to find an acceptable solution for that problem with the current kind of unit. This problem is the main reason why we still need hidden units to learn some problems, because with function-learning we are able to vary the mutual distance between the hyperplanes, but not to rotate them.

Beside the theoretical algorithmic problems, there are also technical problems to overcome. It may be problematical to implement non-monotonic response functions in hardware, and it will cause even more problems if those response functions are variable. Nevertheless, construction of more powerful units, with the usage of non-monotonic response functions and function-learning, could be a great improvement in building networks with fewer units. Besides, incorporating prior constraints of problems in response functions can speed-up learning dramatically (see also the results on the N-bit parity problem). A result of this may be that we can enlarge the problem size which can be solved with MLP’s within reasonable network size and learning-time.

Acknowledgements

The authors would like to thank E.H.L. Aarts, R.E.J.M. Waucomont, and P.J. Zwietering (Eindhoven University of Technology) for their useful remarks. Also we like to thank Umano et al. (Osaka University) because their paper was the motive for our work.
References


CAUDILL, M. [1990], *Neural Networks Primer*, San Francisco, Miller Freeman Publications.


STORK, D.G. AND J.D. ALLEN [1992], ‘How to Solve the N-Bit Parity Problem With Two Hidden Units’, *Neural Networks* 5, 923-926.


