

# The p-Delta Learning Rule for Parallel Perceptrons\*

Peter Auer, Harald M. Burgsteiner, Wolfgang Maass

Institute for Theoretical Computer Science

Technische Universität Graz

A-8010 Graz, Austria

March 11, 2002

## Abstract

A learning algorithm is presented for circuits consisting of a single layer of perceptrons (= threshold gates, or equivalently gates with a Heaviside activation function). We refer to such circuits as *parallel perceptrons*. In spite of their simplicity, these circuits can compute any boolean function if one views the majority of the binary perceptron outputs as the binary outputs of the parallel perceptron, and they are universal approximators for arbitrary continuous functions with values in  $[0, 1]$  if one views the fraction of perceptrons that output 1 as the analog output of the parallel perceptron. For a long time one has thought that there exists no competitive learning algorithms for these extremely simple circuits consisting of gates with binary outputs, which also became known as committee machines. It is commonly believed that one

---

\*Research for this article was partially supported by the ESPRIT Working Group NeuroCOLT, No. 8556, and the Fonds zur Förderung der wissenschaftlichen Forschung (FWF), Austria, project P12153.

has to replace the hard threshold gates by sigmoidal gates and that one has to tune the weights on at least two successive layers in order to get satisfactory learning results. We show that this is not true by exhibiting a simple learning algorithm for parallel perceptrons – the *parallel delta rule* ( $p$ -delta rule), whose performance is comparable to that of backprop for multilayer networks consisting of sigmoidal gates. In contrast to backprop, the  $p$ -delta rule does not require the computation and communication of analog values with high precision, although it does in fact implement gradient descent – with regard to a suitable error measure. Therefore it provides an interesting new hypothesis for the organization of learning in biological neural systems.

## 1 Introduction

In spite of its early successes, the perceptron along with the perceptron learning algorithm – the delta rule – have been abandoned because of the limited expressive power of a single perceptron. Instead, one has resorted to circuits consisting of at least two layers of modified perceptrons with a smooth activation function (sigmoidal neural nets, also called MLP's) and a gradient descent learning algorithm (backprop). “However, the realization of large backpropagation networks in analog hardware poses serious problems because of the need for separate or bidirectional circuitry for the backward pass of the algorithm. Other problems are the need of an accurate derivative of the activation function and the cascading of multipliers in the backward pass.” (quoted from [14]). For similar reasons it is quite dubious whether backprop is applied by biological neural systems to adapt their input/output behaviour.

We will show in this article that there exists an alternative solution: a simple distributed learning algorithm for a less involved class of neural networks with universal computational power that requires less than 2 bits of global communication. One gets already universal approximators for arbitrary

boolean and continuous functions if one takes a single layer of perceptrons in parallel, each with just binary output. One can view such single layer of perceptrons as a group of voters, where each vote (with value  $-1$  or  $1$ ) carries the same weight. Their majority vote can be viewed as a binary output of the circuit. Alternatively one can apply a simple squashing function to the percentage of votes with value  $1$  and thereby get universal approximators for arbitrary given continuous functions with values in  $[0, 1]$ . It is shown in this article that this very simple computational model – to which we will refer as *parallel perceptron* in the following – can be trained in an efficient manner to approximate about any given target function. The learning algorithm consists of a simple extension of the familiar delta rule for a single perception, which we call the  $p$ -delta rule. In contrast to the backprop algorithm, which requires transmission of analog values with high bit precision between the central control and local agents that update the weights of the sigmoidal gates, the  $p$ -delta rule requires only the broadcasting of one of three possible signals (up, down, neutral) from the central control to all local agents. Hence the  $p$ -delta rule provides new ideas for designing adaptive devices. It also yields a promising new hypothesis regarding the organization of learning in biological networks of neurons that overcomes deficiencies of previous approaches that were based on backprop. It can be readily applied to biologically realistic integrate-and-fire neuron models whose output is – on a short time scale – binary: they either fire an action potential or they don't. It also does not require a sophisticated “shadow-network” that computes and transmits fairly complex individual error signals to the neurons. The  $p$ -delta rule has already been applied very successfully to learning for a pool of spiking neurons [13].

We will show in section 4 of this article that in spite of the substantially less demanding communication requirements of the  $p$ -delta rule, and in spite of the much simpler computational structure of parallel perceptrons (which just have a *single* layer of weights), its performance for common benchmark data sets is comparable to that of backprop for MLP's and also to that of the

common decision tree algorithm C 4.5.

Parallel perceptrons and the  $p$ -delta rule are closely related to computational models and learning algorithms that had already been considered 40 years ago (under the name of committee machine, or as a particularly simple MADALINE; see [18] and chapter 6 of [15] for an excellent survey). Apparently the only ingredients that were missing at that time were mathematical tools for proving the universal approximation capability of such models, as well as theoretical and practical experience in machine learning.

In section 2 of this article we will discuss our computational model, the parallel perceptron. The  $p$ -delta learning rule for this model will be introduced and analyzed in section 3. Section 4 contains empirical data regarding the required size of parallel perceptrons and the performance of the  $p$ -delta rule on common benchmark datasets for classification problems. These data are compared with those for backprop and a state-of-the-art learning algorithm from machine learning.

## 2 The Parallel Perceptron

A perceptron (also referred to as threshold gate or McCulloch-Pitts neuron) with  $d$  inputs computes the following function  $f$  from  $\mathbb{R}^d$  into  $\{-1, 1\}$ :

$$f(\mathbf{z}) = \begin{cases} 1, & \text{if } \boldsymbol{\alpha} \cdot \mathbf{z} \geq 0 \\ -1, & \text{otherwise} \end{cases},$$

where  $\boldsymbol{\alpha} \in \mathbb{R}^d$  is the weight vector of the perceptron, and  $\boldsymbol{\alpha} \cdot \mathbf{z}$  denotes the usual vector product  $\boldsymbol{\alpha} \cdot \mathbf{z} := \sum_{j=1}^d \alpha(j)z(j)$  for  $\boldsymbol{\alpha} = \langle \alpha(1), \dots, \alpha(d) \rangle$  and  $\mathbf{z} = \langle z(1), \dots, z(d) \rangle$ <sup>1</sup>.

---

<sup>1</sup>For the sake of simplicity we assume that the threshold of each perceptron is normalized to 0. This is justified as long as one can assume that the actual input  $\langle z(1), \dots, z(d-1) \rangle \in \mathbb{R}^{d-1}$  is extended to a  $d$ -dimensional input  $\langle z(1), \dots, z(d) \rangle \in \mathbb{R}^d$  with  $z(d) := -1$ . In that case the  $d$ th component  $\alpha(d)$  of the weight vector  $\boldsymbol{\alpha}$  assumes the role of the threshold, since  $\sum_{j=1}^d \alpha(j)z(j) \geq 0 \Leftrightarrow \sum_{j=1}^{d-1} \alpha(j)z(j) \geq \alpha(d)$ .

A *parallel perceptron* is a single layer consisting of a finite number  $n$  of perceptrons. Let  $f_1, \dots, f_n$  be the functions from  $\mathbb{R}^d$  into  $\{-1, 1\}$  that are computed by these perceptrons. For input  $\mathbf{z}$  the output of the parallel perceptron is the value  $\sum_{i=1}^n f_i(\mathbf{z}) \in \{-n, \dots, n\}$ , more precisely the value  $s(\sum_{i=1}^n f_i(\mathbf{z}))$ , where  $s : \mathbb{Z} \rightarrow \mathbb{R}$  is a squashing function that scales the output into the desired range.

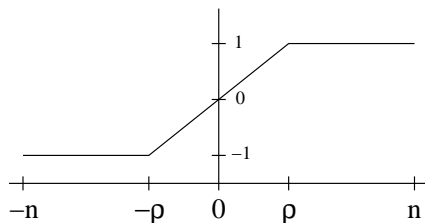


Figure 1: The squashing function  $s_\rho$  for a parallel perceptron with analog output.

In this article we will restrict our attention to the piecewise linear squashing function  $s_\rho$  with range  $[-1, 1]$  that is plotted in Figure 1. The parameter  $\rho$  may be viewed as the resolution of the squashing function  $s_\rho$ , which is formally defined by

$$s_\rho(p) = \begin{cases} -1 & \text{if } p < -\rho \\ p/\rho & \text{if } -\rho \leq p \leq \rho \\ +1 & \text{if } p > \rho. \end{cases}$$

For  $\rho = 1$  this function maps into the binary range  $\{-1, 1\}$  and simply implements a majority vote of the  $n$  perceptrons (assuming that  $n$  is odd). In order to approximate a given continuous function with values in  $[-1, 1]$  up to some given  $\varepsilon > 0$  we will typically set  $\rho := 1/(2\varepsilon)$ .

In Figure 2 we give an example of a function from  $\mathbb{R}^2$  into  $\{-1, -\frac{1}{2}, 0, \frac{1}{2}, 1\}$  computed by a parallel perceptron consisting of  $n = 8$  perceptrons.

It is not difficult to prove that *every* Boolean function from  $\{-1, 1\}^d$  into  $\{-1, 1\}$  can be computed by a parallel perceptron (with  $\rho = 1$ ). Furthermore

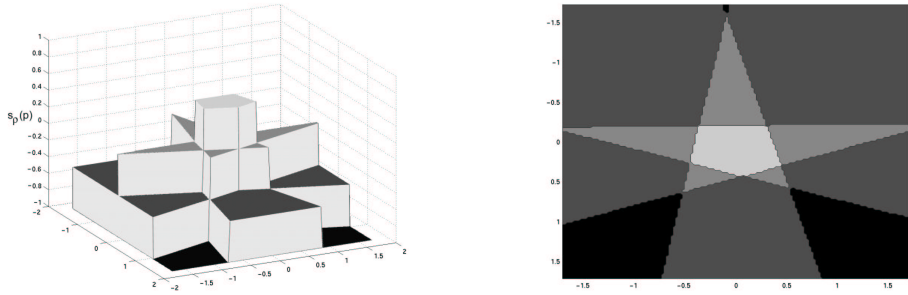


Figure 2: Perspective and top-view of an approximation of a 2D-Gaussian function by a parallel perceptron with  $n = 8$ ,  $\rho = 2$ ,  $\epsilon = 0.25$

we show that *every* continuous function  $g : \mathbb{R}^d \rightarrow [-1, 1]$  can be approximated by a parallel perceptron within any given error bound  $\epsilon$  on any closed and bounded subset of  $\mathbb{R}^d$ . Therefore parallel perceptrons are in fact universal approximators. Furthermore our empirical results show that many real-world classification problems can be solved by parallel perceptrons that consist of a very small number of perceptrons.

Parallel perceptrons are closely related to circuits involving a single hard or soft winner-take-all gate, whose computational properties were investigated in [10], [11]. We will show in the Appendix of this article that the claimed universal approximation property of parallel perceptrons follows immediately from that of winner-take-all circuits.

### 3 The p-delta learning rule

In this section we develop a learning rule for parallel perceptrons. We call this learning rule the p-delta rule. It consists of two ingredients. The first ingredient is the classical delta rule, which is applied to *some* of the individual perceptrons that make up the parallel perceptron. The second ingredient is a rule which decides whether the classical delta rule should be applied to a given individual perceptron. This rule takes inspiration from support vector

machines [7]. Support vector machines learn classifiers which maximize the *margin* of the classification: An input  $\mathbf{z} \in \mathbb{R}^d$  is classified by  $\text{sign}(\boldsymbol{\alpha} \cdot \mathbf{z})$ , and  $\boldsymbol{\alpha}$  is calculated such that for all training examples  $\mathbf{z}_k$  the dot product  $|\boldsymbol{\alpha} \cdot \mathbf{z}_k|$  is large. In a somewhat related fashion we apply the delta rule to those individual perceptrons that give the wrong output, and also to those that give the right output but with a too small margin. This ensures that the outputs of the individual perceptrons become stable against perturbations of the inputs, which improves both the stability of the learning process as well the performance — in respect to generalization — of the trained parallel perceptron.

In the next section we discuss the basic options regarding the application of the classical delta rule in the context of parallel perceptrons, and in Section 3.2 we show how to incorporate the *large margin* idea. In Section 3.3 we demonstrate that the resulting learning rule can be interpreted as gradient descent for a suitable error function. In Section 3.4 we discuss the relationship of our learning rule to support vector machines and large margin classifiers. Finally, we discuss the plausibility of the p-delta rule for biological systems in Section 3.5. Some remarks on the implementation of the p-delta rule on digital computers are given in Appendix A.

### 3.1 Getting the outputs right

The p-delta rule can be applied in batch or in incremental mode. In batch mode the weight vectors are updated once after each epoch, where in each epoch each training example is presented once. In incremental mode the weight vectors are updated after each presentation of a training example. For notational simplicity we assume that updates are performed in incremental mode. The modification for batch mode is straightforward.

Let  $(\mathbf{z}, o) \in \mathbb{R}^d \times [-1, +1]$  be the current training example and let  $\boldsymbol{\alpha}_1, \dots, \boldsymbol{\alpha}_n \in \mathbb{R}^d$  be the current weight vectors of the  $n$  individual perceptrons

in the parallel perceptron. Thus the current output of the parallel perceptron is calculated as

$$\hat{o} = s_\rho(p), \quad \text{where } p = \#\{i : \boldsymbol{\alpha}_i \cdot \mathbf{z} \geq 0\} - \#\{i : \boldsymbol{\alpha}_i \cdot \mathbf{z} < 0\}.$$

If  $|\hat{o} - o| \leq \varepsilon$  then the output of the parallel perceptron is within the desired accuracy of the target output  $o$ , and its weights need not be modified.

Consider now for example the case

$$\hat{o} > o + \varepsilon,$$

when the output of the parallel perceptron is too large. To lower the output of the parallel perceptron we need to reduce the number of weight vectors with  $\boldsymbol{\alpha}_i \cdot \mathbf{z} \geq 0$ . Applying the classical delta rule to such a weight vector yields the update

$$\boldsymbol{\alpha}_i \leftarrow \boldsymbol{\alpha}_i + \eta \Delta_i$$

where  $\eta > 0$  is the learning rate and

$$\Delta_i = -\mathbf{z}.$$

However it is not obvious which weight vectors with  $\boldsymbol{\alpha}_i \cdot \mathbf{z} \geq 0$  should be modified by this update rule. There are several plausible options (among which the first and second option require substantially more computational overhead and communication between local agents):

1. Update only one of the weight vectors with  $\boldsymbol{\alpha}_i \cdot \mathbf{z} \geq 0$ . For example choose the weight vector with minimal  $\boldsymbol{\alpha}_i \cdot \mathbf{z}$ .
2. Update  $N$  of the weight vectors with  $\boldsymbol{\alpha}_i \cdot \mathbf{z} \geq 0$ , where  $N$  is the minimal number of sign changes of individual perceptrons that are necessary to get the output  $\hat{o}$  of the parallel perceptron right. This approach was taken in [15, Section 6.3].
3. Update all weight vectors with  $\boldsymbol{\alpha}_i \cdot \mathbf{z} \geq 0$ .

For our p-delta rule we choose the third option. We proceed analogously in the case where  $\hat{o} < o - \varepsilon$ . Thus we arrive at the rule  $\boldsymbol{\alpha}_i \leftarrow \boldsymbol{\alpha}_i + \eta \Delta_i$  for all  $i$ , where

$$\Delta_i = \begin{cases} -\mathbf{z} & \text{if } \hat{o} > o + \varepsilon \text{ and } \boldsymbol{\alpha}_i \cdot \mathbf{z} \geq 0 \\ +\mathbf{z} & \text{if } \hat{o} < o - \varepsilon \text{ and } \boldsymbol{\alpha}_i \cdot \mathbf{z} < 0 \\ 0 & \text{otherwise.} \end{cases}$$

The problem with the other options is that they might try to update the wrong weight vectors. Updating all weight vectors ensures that at least some progress is made by the learning process. Although in this case too many weight vectors might be modified, this negative effect can be counteracted by the “clear margin” approach, which is discussed in the next section.

Note that the third option is the one which requires the least communications between a central control and agents that control the individual weight vectors  $\boldsymbol{\alpha}_i$ : each agent can determine on its own whether  $\boldsymbol{\alpha}_i \cdot \mathbf{z} \geq 0$ , and hence no further communication is needed to determine which agents have to update their weight vector once they are told to which class the global error of the output  $\hat{o}$  of the parallel perceptron belongs.

### 3.2 Stabilizing the outputs

For any of the 3 options discussed in the previous section, weight vectors are updated only if the output of the parallel perceptron is incorrect. Hence weight vectors remain unmodified as soon as the sign of  $\boldsymbol{\alpha}_i \cdot \mathbf{z}$  is “correct” in respect to the target output  $o$  and the output of the parallel perceptron  $\hat{o}$ . Thus at the end of training there are usually quite a few weight vectors for which  $\boldsymbol{\alpha}_i \cdot \mathbf{z}$  is very close to zero (for some training input  $\mathbf{z}$ ). Hence a small perturbation of the input  $\mathbf{z}$  might change the sign of  $\boldsymbol{\alpha}_i \cdot \mathbf{z}$ , and the output of the parallel perceptron is rather unstable. This reduces the generalization capabilities of the parallel perceptron. To stabilize the output of the parallel perceptron we modify the update rule of the previous section in order to keep  $\boldsymbol{\alpha}_i \cdot \mathbf{z}$  away from zero. In fact, we try to keep a *margin*  $\gamma$  around zero clear from any

dot products  $\boldsymbol{\alpha}_i \cdot \mathbf{z}$ . The margin  $\gamma > 0$  is a parameter of our algorithm. In Appendix A we discuss how this parameter can be set automatically by a learning algorithm.

Assume that  $\boldsymbol{\alpha}_i \cdot \mathbf{z} \geq 0$  has the correct sign but that  $\boldsymbol{\alpha}_i \cdot \mathbf{z} < \gamma$ . In this case we increase  $\boldsymbol{\alpha}_i \cdot \mathbf{z}$  by updating  $\boldsymbol{\alpha}_i$  as

$$\boldsymbol{\alpha}_i \leftarrow \boldsymbol{\alpha}_i + \eta \mu \mathbf{z} \tag{1}$$

for an appropriate parameter  $\mu > 0$ . The parameter  $\mu$  measures the importance of a clear margin: if  $\mu \approx 0$  then this update has little influence, if  $\mu$  is large then a clear margin is strongly enforced. A typical value of  $\mu$  is  $\mu = 1$ .

Observe that a margin  $\gamma$  is effective only if the weights  $\boldsymbol{\alpha}_i$  remain bounded: to satisfy condition  $|\boldsymbol{\alpha}_i \cdot \mathbf{z}| \geq \gamma$ , scaling up  $\boldsymbol{\alpha}_i$  by a factor  $C > 1$  or reducing  $\gamma$  by factor  $1/C$  is equivalent. Thus we keep the weights  $\boldsymbol{\alpha}_i$  bounded by the additional update

$$\boldsymbol{\alpha}_i \leftarrow \boldsymbol{\alpha}_i - \eta ( \|\boldsymbol{\alpha}_i\|^2 - 1 ) \boldsymbol{\alpha}_i$$

which moves  $\|\boldsymbol{\alpha}_i\|$  towards 1. Concluding, the p-delta rule is summarized in Figure 3. Note that  $\boldsymbol{\alpha}_i \cdot \mathbf{z}$  is always pushed away from 0 according to the 3rd or 4th line, unless it is pushed towards 0 according to one of the first two lines of the update rule in Fig. 3.

The rather informal arguments in this and the previous section can be made more precise by showing that the descent for an appropriate error function. This error function is given in the next section.

### 3.3 The error function of the p-delta rule

Let  $\boldsymbol{\alpha}_1, \dots, \boldsymbol{\alpha}_n$  be the current weight vectors of the parallel perceptron and let  $\mathbf{z}$  be an input vector for the parallel perceptron. Then the output of the parallel perceptron is calculated as  $\hat{o} = s_\rho (\sum_{i=1}^n f_i(\mathbf{z}))$  where  $f_i(\mathbf{z}) = +1$  if  $\boldsymbol{\alpha}_i \cdot \mathbf{z} \geq 0$  and  $f_i(\mathbf{z}) = -1$  if  $\boldsymbol{\alpha}_i \cdot \mathbf{z} < 0$ . An error function with respect to the

p-delta rule

For all  $i = 1, \dots, n$ :

$$\boldsymbol{\alpha}_i \leftarrow \boldsymbol{\alpha}_i - \eta (\|\boldsymbol{\alpha}_i\|^2 - 1) \boldsymbol{\alpha}_i + \eta \begin{cases} (-\mathbf{z}) & \text{if } \hat{o} > o + \varepsilon \text{ and } \boldsymbol{\alpha}_i \cdot \mathbf{z} \geq 0 \\ (+\mathbf{z}) & \text{if } \hat{o} < o - \varepsilon \text{ and } \boldsymbol{\alpha}_i \cdot \mathbf{z} < 0 \\ \mu(+\mathbf{z}) & \text{if } \hat{o} \leq o + \varepsilon \text{ and } 0 \leq \boldsymbol{\alpha}_i \cdot \mathbf{z} < \gamma \\ \mu(-\mathbf{z}) & \text{if } \hat{o} \geq o - \varepsilon \text{ and } -\gamma < \boldsymbol{\alpha}_i \cdot \mathbf{z} < 0 \\ 0 & \text{otherwise} \end{cases}$$

Figure 3: The p-delta rule

target output  $o$  can now be defined as

$$\begin{aligned} \text{Err}(\boldsymbol{\alpha}_1, \dots, \boldsymbol{\alpha}_n; \mathbf{z}, o) &= \frac{1}{2} \sum_{i=1}^n (\|\boldsymbol{\alpha}_i\|^2 - 1)^2 \\ &+ \begin{cases} \sum_{i:\boldsymbol{\alpha}_i \cdot \mathbf{z} \geq 0} (\mu\gamma + \boldsymbol{\alpha}_i \cdot \mathbf{z}) + \sum_{i:-\gamma < \boldsymbol{\alpha}_i \cdot \mathbf{z} < 0} (\mu\gamma + \mu\boldsymbol{\alpha}_i \cdot \mathbf{z}) & \text{if } \hat{o} > o + \varepsilon \\ \sum_{i:\boldsymbol{\alpha}_i \cdot \mathbf{z} < 0} (\mu\gamma - \boldsymbol{\alpha}_i \cdot \mathbf{z}) + \sum_{i:0 \leq \boldsymbol{\alpha}_i \cdot \mathbf{z} < +\gamma} (\mu\gamma - \mu\boldsymbol{\alpha}_i \cdot \mathbf{z}) & \text{if } \hat{o} < o - \varepsilon \\ \sum_{i:-\gamma < \boldsymbol{\alpha}_i \cdot \mathbf{z} < 0} (\mu\gamma + \mu\boldsymbol{\alpha}_i \cdot \mathbf{z}) + \sum_{i:0 \leq \boldsymbol{\alpha}_i \cdot \mathbf{z} < \gamma} (\mu\gamma - \mu\boldsymbol{\alpha}_i \cdot \mathbf{z}) & \text{if } |\hat{o} - o| \leq \varepsilon. \end{cases} \end{aligned}$$

The first line of the right hand side penalizes a deviation of  $\|\boldsymbol{\alpha}_i\|$  from 1 and therefore keeps the norms of the weights close to 1. Line 2 and 3 cover the case that the output of the parallel perceptron is too far from the target output, i.e.  $|o - \hat{o}| > \varepsilon$ . The sums  $\sum_{i:\boldsymbol{\alpha}_i \cdot \mathbf{z} \geq 0} (\mu\gamma + \boldsymbol{\alpha}_i \cdot \mathbf{z})$  and  $\sum_{i:\boldsymbol{\alpha}_i \cdot \mathbf{z} < 0} (\mu\gamma - \boldsymbol{\alpha}_i \cdot \mathbf{z})$  measure how far dot products  $\boldsymbol{\alpha}_i \cdot \mathbf{z}$  are on the wrong side (from 0), e.g. if  $\hat{o} > o + \varepsilon$  than there are too many  $\boldsymbol{\alpha}_i$  with  $\boldsymbol{\alpha}_i \cdot \mathbf{z} \geq 0$  and each such weight is penalized with  $\mu\gamma + \boldsymbol{\alpha}_i \cdot \mathbf{z}$ . The term  $\mu\gamma$  ensures compatibility with the other sums,  $\sum_{i:-\gamma < \boldsymbol{\alpha}_i \cdot \mathbf{z} < 0} (\mu\gamma + \mu\boldsymbol{\alpha}_i \cdot \mathbf{z})$  and  $\sum_{i:0 \leq \boldsymbol{\alpha}_i \cdot \mathbf{z} < +\gamma} (\mu\gamma - \mu\boldsymbol{\alpha}_i \cdot \mathbf{z})$ . These sums penalize weights for which  $\boldsymbol{\alpha}_i \cdot \mathbf{z}$  is within a margin  $\gamma$  around 0. Consider for example  $\sum_{i:0 \leq \boldsymbol{\alpha}_i \cdot \mathbf{z} < +\gamma} (\mu\gamma - \mu\boldsymbol{\alpha}_i \cdot \mathbf{z})$ . If  $0 \leq \boldsymbol{\alpha}_i \cdot \mathbf{z} < \gamma$  then the distance to the margin  $\gamma$  is  $\gamma - \boldsymbol{\alpha}_i \cdot \mathbf{z}$ . Weighting this distance by  $\mu$  gives the error term

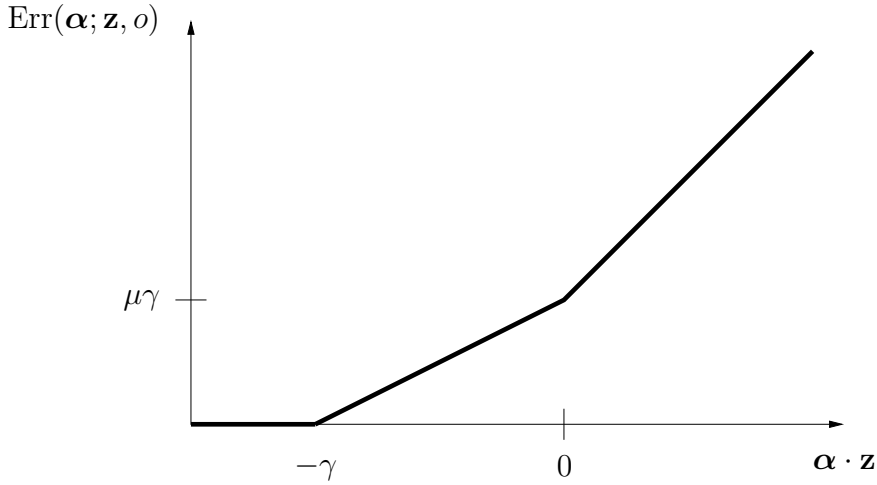


Figure 4: Plot of the error function  $\text{Err}(\boldsymbol{\alpha}; \mathbf{z}, o)$  for a single weight vector  $\boldsymbol{\alpha}$  versus the dot product  $\boldsymbol{\alpha} \cdot \mathbf{z}$ , assuming that  $\|\boldsymbol{\alpha}\| = 1$  and  $\hat{o} > o + \varepsilon$  for all displayed  $\boldsymbol{\alpha}$ . ( $\mu = 1/2$ )

$$\mu\gamma - \mu\boldsymbol{\alpha}_i \cdot \mathbf{z}.$$

The fact that the error terms — either for being on the wrong side or for being within the margin — are compatible is depicted in Figure 4. As long as the dot product is on the wrong side, the error grows with the distance from 0. If the dot product is on the right side the error grows with the distance from the margin  $\gamma$ , scaled by  $\mu$ . Thus the error function is continuous as long as  $|\hat{o} - o| \neq \varepsilon$ . If the weights change such that the output of the parallel perceptron moves from  $|\hat{o} - o| > \varepsilon$  to  $|\hat{o} - o| < \varepsilon$  then the error function jumps downwards since the sum  $\sum_{i:\boldsymbol{\alpha}_i \cdot \mathbf{z} \geq 0} (\mu\gamma + \boldsymbol{\alpha}_i \cdot \mathbf{z})$  (assuming  $\hat{o} > o + \varepsilon$ ) is replaced by the smaller sum  $\sum_{i:0 \leq \boldsymbol{\alpha}_i \cdot \mathbf{z} < \gamma} (\mu\gamma - \mu\boldsymbol{\alpha}_i \cdot \mathbf{z})$  (third line of (2)).

It is easy to see that the error function is non-negative and that it is equal to zero if and only if  $|\hat{o} - o| \leq \varepsilon$  and all weight vectors  $\boldsymbol{\alpha}_i$  satisfy  $|\boldsymbol{\alpha}_i \cdot \mathbf{z}| \geq \gamma$  and  $\|\boldsymbol{\alpha}_i\| = 1$ . Taking derivatives of the error function with respect to  $\boldsymbol{\alpha}_i$  we find that the p-delta rule performs gradient descent with respect to this error function. Thus by the p-delta rule the weight vectors will converge — for a sufficiently small learning rate — towards a local minimum of the error

function.

### **3.4 Generalization and relationship to support vector machines**

The idea of having a clear margin around the origin is not new and is heavily used with support vector machines [7]. In our setting we use the clear margin to stabilize the output of the parallel perceptron. As is known from the analysis of support vector machines such a stable predictor also gives good generalization performance on new examples. Since our parallel perceptron is an aggregation of many simple perceptrons with large margins (see also [4]), one expects that parallel perceptrons also exhibit good generalization. This is indeed confirmed by our empirical results reported in Section 4.

### **3.5 Relationship to Models for Biological Neural Systems**

It is shown in the Appendix (Corollary C.2) that parallel perceptrons have the “universal approximation property”, i.e., they can approximate any given continuous function uniformly on any compact domain. This approximation result has an interesting interpretation in the context of computational neuroscience, since one can view a parallel perceptron as a model for a population of biological neurons (without lateral connections inside the population). One can model the decision whether a biological neuron will fire within a given time interval (e. g., of length 5 ms) quite well with the help of a single perceptron (see [5], [9]). Hence a parallel perceptron may be viewed as a model for a population  $P$  of biological neurons, that predicts how many of these neurons will fire within such time interval.

Therefore the universal approximation property of parallel perceptrons implies that a single population  $P$  of biological neurons (without lateral connections) can in principle be used to approximate any given continuous function

$f$  (from static inputs  $\underline{x}$  to static outputs  $f(\underline{x})$ ). One just has to assume that the components of the input vector  $\underline{x}$  are given in the form of synaptic input currents to the neurons in the population  $P$ , and that the fraction of neurons in  $P$  that fire within a given short time interval represents the approximation to the target output value  $f(\underline{x})$ .

The p-delta learning rule for parallel perceptrons that we introduce in this article, has already been tested in this biological context through extensive computer simulations of biophysical models for populations of neurons [13]. The results of these simulations show that the p-delta learning rule is very successful in training such populations of spiking neurons to adopt a given population response. We are not aware of any other learning algorithm that could be used for that purpose. Among other things we are exploiting here the fact that, in contrast to backprop, the p-delta learning rule can be applied to neuron models that do not have a smooth activation function, such as a spiking neuron. The computer simulation discussed in [13] shows that the p-delta rule can actually be used to train models for circuits of biological neurons to adopt a given temporal response, i. e., to approximate a given nonlinear filter.

Whether the p-delta rule is biologically realistic is another matter. Relatively little is known about the actual learning mechanisms that modify synapses in vivo. In fact, frequently cited older experimental data on longterm synaptic plasticity have to be reconsidered in the light of new data (see e. g., [1]). Even less reliable data are available on biological mechanisms that might provide global control for synaptic plasticity in neural circuits. However in spite of this lack of knowledge one may argue that the p-delta rule has a larger chance of being biologically realistic than other learning algorithms, such as backprop, that have been developed for artificial neural networks. These other learning algorithms require the transmission of analog error signals (typically involving the values of derivatives) with high bit precision between a global control and local mechanisms that carry out the actual synaptic modifications. These communication requirements would be very costly to implement

in a noisy analog system. In contrast to that, the p-delta rule just requires to broadcast to all local agents whether the current population response  $\hat{o}$  was close to the desired target output  $o$ , or way too large, or way too small. Thus in contrast to backprop, the same information can be sent to all local agents, the message requires just 2 bits, and no complex computations (such as taking derivatives and multiplication of analog values) are needed to compute these 2 bits.

For the same reason the p-delta rule is better suitable than backprop for implementation in analog VLSI, for example in order to build adaptive “universal approximator chips” in analog VLSI with on-chip learning facilities.

## 4 Empirical Results

For an empirical evaluation of the p-delta rule we have chosen five datasets from the UCI machine learning repository [2]: Breast-cancer (BC), Pima Indian Diabetes (DI), Heart disease (HD), Ionosphere (IO), and Sonar (SN). For a more detailed description of the datasets see Appendix B. Our criteria were binary classification tasks, few missing values in the data, and published learning results for multilayer-perceptrons trained by backprop (MLP+BP). We also compared our results with the implementation of MLP+BP and C4.5 in WEKA<sup>2</sup>.

### 4.1 Comparison with published results

For this comparison we used the same experimental setup (split into training and test set, number of cross-validations, dealing with missing attributes) that was reported in the literature (for details see Appendix B). We added a constant bias to the data and initialized the weights of the parallel perceptron randomly. The results are shown in Table 1. Results for the p-delta rule are

---

<sup>2</sup>A complete set of Java Programs for Machine Learning, including datasets from UCI, available at <http://www.cs.waikato.ac.nz/~ml/weka/>.

Dataset	Gorman, Sejnowski [6]	Sigillito et al. [16]	Ster, Dobnikar [17]	p-delta $n = 1$	p-delta $n = 3$	p-delta $n = 7$
BC	-	-	96.7	65.52	96.72	96.71
DI	-	-	76.4	65.10	75.42	72.54
HD	-	-	81.3	56.14	82.97	78.94
IO	-	96.0	-	58.72	90.46	84.27
SN	$82.0 \pm 7.3$	-	-	53.70	76.63	74.57

Table 1: Comparison of the performance (= accuracy on test data) of the p-delta rule with backprop on common benchmark classification problems.

the averages of the performance on the testset over 10 independent runs of 10 fold cross validation. The p-delta rule was applied until its error function (Section 3.3) did not improve by at least 1% during the second half of the trials. Typically training stopped after a few hundred epochs, sometimes it took a few thousand epochs. For all datasets we used  $\mu = 1$ . The learning rate  $\eta$  and the margin  $\gamma$  were tuned automatically. Details for this automatic tuning are given in Appendix A.

The results show that the performance of the p-delta rule is comparable with that of multilayer-perceptrons. We also found that for the tested datasets small parallel perceptrons ( $n = 3$ ) suffice for good classification accuracy.

## 4.2 Comparison with state-of-the-art machine learning software

Since the experimental setup in [6, 16, 17] varied considerably we also tested all datasets with a fixed experimental setup using state-of-the-art machine learning software. We used MLP+BP and C4.5 from WEKA (version 3.2). Table 2 shows the average over 5 trials of 10 fold cross-validation. For this experiment we removed examples with missing values<sup>3</sup>. We ran p-delta with  $n = 3$ ,  $\rho = 1$ , and an automatic tuning of  $\eta$  and  $\gamma$  (the results are very similar for fixed values of these parameters, see Table 3 in Appendix A). The

<sup>3</sup>This has very little influence on the performance.

Dataset	C4.5	MLP+BP	p-delta
BC	95.31	95.99	96.87
DI	74.48	76.79	75.42
HD	74.66	80.74	82.97
IO	88.60	89.74	90.46
SN	72.60	82.59	76.63

Table 2: Comparison of the performance (= accuracy on test data) of the p-delta rule for  $n = 3$  perceptrons with that of backprop and decision trees (from the WEKA software package).

stopping condition was the same as described in Section 4.1. For the MLP we used 3 neurons in the hidden layer since this number of neurons in the hidden layer yielded good results in [6, 16, 17]. The learning rate was 0.01 and the momentum 0.8 as in [17]. The MLP was trained for 500 epochs, which was sufficient for convergence. For C4.5 we used the default parameter settings.

Again we find that the performance of parallel perceptrons is comparable with the performance of other state-of-the-art classification algorithms.

## 5 Discussion

Moerland and Fiesler [14] state that “the design of a compact digital neural network can be simplified considerably when Heaviside functions are used as activation functions instead of a differentiable sigmoidal activation function. While training algorithms for perceptrons with Heaviside functions abound, training multilayer networks with nondifferentiable Heaviside functions requires the development of new algorithms.” We have presented in this article a new learning algorithm — the p-delta rule — for parallel perceptrons, i.e., for neural networks consisting of a single layer of perceptrons. In contrast to other learning algorithms for parallel perceptrons, such as MADALINE-3 [18] and weight perturbation [8] (see the survey in [14]), this algorithm can be executed in parallel, with one global 2-bit error signal as the only communication

between local agents. In comparison with backprop for multi-layer perceptrons this new learning algorithm – which only requires to tune weights on a single layer – provides an alternative solution to the credit assignment problem that does not require the computation and communication of derivatives. Hence one may argue that the learning algorithm presented in this article provides a more compelling model for learning in biological neural circuits than the familiar backprop algorithm for multi-layer perceptrons. In fact it has already been successfully used in computer simulations of biological neural circuits for training a pool of spiking neurons to approximate an analog target function via space-rate coding [13].

It is shown in Appendix C that the parallel perceptron model is closely related to previously studied Winner-Take-All circuits. Hence the p-delta rule also provides a new learning algorithm for Winner-Take-All circuits.

## Acknowledgment

We gratefully acknowledge the support of the ESPRIT Working Group NeuroCOLT, No. 8556, and the support of the Fonds zur Förderung der wissenschaftlichen Forschung (FWF), Austria, project P12153.

## References

- [1] Abbott, L. F. & Nelson, S. B. (2000). Synaptic plasticity: taming the beast. *Nature Neuroscience*, 3:1178-1183.
- [2] Blake, C.L. & Merz, C.J. (1998). *UCI Repository of machine learning databases*, <http://www.ics.uci.edu/~mllearn/MLRepository.html>. Irvine, CA: University of California, Department of Information and Computer Science.
- [3] Boyd, S., & Chua, L. O. (1985). Fading memory and the problem of approximating nonlinear operators with Volterra series. *IEEE Trans. on Circuits and Systems*, 32:1150-11.

- [4] Freund, Y., & Schapire, R. E. (1999). Large margin classification using the Perceptron algorithm. *Machine Learning* 37(3):277–296.
- [5] Gerstner, W. (1998). Spiking Neurons. In: Maass, W., & Bishop, C. M. (eds.), *Pulsed Neural Networks*, MIT Press, 3-54. Available at [http://diwww.epfl.ch/lami/team/gerstner/wg\\_pub.html](http://diwww.epfl.ch/lami/team/gerstner/wg_pub.html).
- [6] Gorman, R.P. & Sejnowski, T.J. (1988). Analysis of Hidden Layer Units in a Layered Network Trained to Classify Sonar Targets. *Neural Networks*, 1:75-89.
- [7] Guyon I., Boser B., & Vapnik V. (1993). Automatic capacity tuning of very large VC-dimension classifiers. *Advances in Neural Information Processing Systems*, volume 5, Morgan Kaufmann (San Mateo) 147-155.
- [8] Jabri, M. & Flower, B. (1992). Weight Perturbation: An Optimal Architecture and Learning Technique for Analog VLSI Feedforward and Recurrent Multilayer Networks. *IEEE Transactions on Neural Networks*, vol. 3(1), 154-157.
- [9] Maass, W. (1997). Networks of spiking neurons: the third generation of neural network models. *Neural Networks*, 10:1659-1671.
- [10] Maass, W. (2000). Neural Computation with Winner-Take-All as the only Nonlinear Operation. Appears in: *Advances in Neural Information Processing Systems*, vol. 12, MIT Press, Cambridge, 293-299. Available at <http://www.igi.TUGraz.at/maass/116nips.ps.gz>.
- [11] Maass, W. (2000). On the Computational Power of Winner-Take-All. *Neural Computation*, 12(11):2519-2536. Available at <http://www.igi.TUGraz.at/maass/113j.ps.gz>.
- [12] Maass, W. & Sontag, E. D. (2000). Neural systems as nonlinear filters. *Neural Computation*, 12(8):1743-1772. Available at <http://www.tu-graz.ac.at/igi/maass/107rev.ps.gz>.
- [13] Maass, W., Natschlaeger, T., & Markram, H. (2001). *Real-time computing without stable status: a new framework for neural computation based on perturbations*, [http://www.igi.TUGraz.at/maass/liquid\\_state\\_machines.pdf](http://www.igi.TUGraz.at/maass/liquid_state_machines.pdf).

- [14] Moerland, P. & Fiesler, E. (1996). Neural Network Adaptations to Hardware Implementations. In: Fiesler, E. & Beale, R., eds., *Handbook of Neural Computation*, Institute of Physics Publishing and Oxford University Publishing, New York, E1.2:1-13.
- [15] Nilsson, N. J. (1990). *The Mathematical Foundations of Learning Machines*, Morgan Kauffmann Publishers, San Mateo (USA).
- [16] Sigillito, V.G., Wing, S.P., Hutton, L.V., Baker, K.B. (1989). Classification of radar returns from the ionosphere using neural networks. *Johns Hopkins APL Technical Digest*, 10:262-266.
- [17] Ster, B. & Dobnikar, A. (1996). Neural Networks in Medical Diagnosis: Comparison with Other Methods. In: *Bulsari, A. et al., eds, Proceedings of the International Conference on Engineering Applications of Neural Networks (EANN'96)*, 427-430.
- [18] Widrow, B. & Lehr, M. A. (1990). 30 years of adaptive neural networks: Perceptron, madaline, and backpropagation. *Proceedings of the IEEE*, 78(9):1415-1442.

## **A Appendix: Practical considerations for the implementation of the p-delta rule on a digital computer**

To check the validity of the parallel perceptron and of the p-delta rule we ran several experiments with benchmark machine learning datasets. When the p-delta rule is run on a digital computer and is applied to a specific learning task, a particular instantiation of the p-delta rule (and its parameters) has to be chosen. Thoughts about reasonable choices are discussed in this section.

The first thing to mention is that we did batch updates of the weights instead of updating the weights for each individual training example. We

accumulated the updates for all training examples and updated the weights once for each epoch. Doing batch updates allows also another modification of the update rule: instead of normalizing the weights implicitly by the update rule

$$\boldsymbol{\alpha}_i \leftarrow \boldsymbol{\alpha}_i - \eta (\|\boldsymbol{\alpha}_i\|^2 - 1) \boldsymbol{\alpha}_i$$

we explicitly normalized the weights by

$$\boldsymbol{\alpha}_i \leftarrow \boldsymbol{\alpha}_i / \|\boldsymbol{\alpha}_i\|$$

after each update.

Now it remains to choose the parameters of the p-delta rule,  $\eta$ ,  $\gamma$ , and  $\mu$ . It turns out that the choice of  $\mu$  is rather insignificant and we set  $\mu = 1$ . With  $\eta$  and  $\gamma$  the situation is more complicated. In particular the choice of  $\gamma$  is important for a good performance of the p-delta rule. Therefore we included a mechanism which automatically tunes  $\gamma$ . For this we constructed an update rule for  $\gamma$ . This update rule increases  $\gamma$  if for a training example  $(\mathbf{z}, o)$  only few dot products  $\boldsymbol{\alpha}_i \cdot \mathbf{z}$  are in the margin  $(-\gamma, +\gamma)$ , and the update rule decreases  $\gamma$  if there are many dot products within this margin. The number of dot products within the margin is calculated as

$$M_+ = \#\{i : 0 \leq \boldsymbol{\alpha}_i \cdot \mathbf{z} < +\gamma \text{ and } \hat{o} \leq o + \varepsilon\}$$

$$M_- = \#\{i : -\gamma < \boldsymbol{\alpha}_i \cdot \mathbf{z} < 0 \text{ and } \hat{o} \geq o - \varepsilon\}$$

(we count only dot products which have the correct sign in respect to  $o$  and  $\hat{o}$ ). Now the update rule is given by

$$\gamma \leftarrow \gamma + \eta (M_{\min} - \min\{M_{\max}, M_+ + M_-\})$$

where  $M_{\min}, M_{\max} > 0$  are suitable parameters. This means that at most  $M_{\max}$  of the dot products within the margin are considered, and that  $\gamma$  is increased if the number of dot products is smaller than  $M_{\min}$  and that  $\gamma$  is increased if there are more than  $M_{\min}$  dot products within the margin. Good values for  $M_{\min}$  and  $M_{\max}$  are  $M_{\min} = \varepsilon\rho$  and  $M_{\max} = 4M_{\min}$ .

Dataset	p-delta ( $n = 3$ ) (automatical tuning)	p-delta ( $n = 3$ ) (no tuning)
BC	96.87	96.82
DI	75.42	76.30
HD	82.97	78.85
IO	90.46	86.01
SN	76.63	78.80

Table 3: Comparison of the results of the p-delta rule with and without the parameters  $\eta$  and  $\gamma$  being set automatically. The results in the second column were obtained setting  $\eta_0 = 0.25$  and  $\gamma = 0.05$ .

For the last remaining parameter, the learning rate  $\eta$ , we used an adaptive learning rate schedule. If an update of the weights reduces the value of the error function, then the learning rate is increased (say, by a factor 1.1), if an update increases the value of the error function, then the learning rate is decreased (say, by a factor 0.5). Such an adaptive learning rate schedule yields a relatively fast but still stable convergence of the p-delta rule.

It has to be remarked, that getting the magnitudes of the parameters  $\gamma$  and  $\eta$  right is crucial for the performance of the p-delta rule, although the p-delta rule is quite robust against small changes of the parameters. Setting these parameters by hand can be – depending on the complexity of the training data – quite difficult. The considerations in this section provide a good way of tuning these parameters automatically, though this method is harder to implement in hardware (e.g. in analog VLSI) through the increased need of communication to the perceptrons.

### **A.1 Comparison of results with and without the parameters $\gamma$ and $\eta$ being set automatically**

In all experiments reported so far the p-delta rule was used with an automatical tuning of margin  $\gamma$  and learning rate  $\eta$ . Since the computation of  $\gamma$  involves information of the output of several perceptrons, this increases the number of

bits needed for the communication of the error signal. In neural network learning algorithms, one often uses a steadily decreasing learning rate (e.g.  $\eta = \frac{\eta_0}{\sqrt{t}}$ , where  $\eta_0$  is the base learning rate and  $t$  is the current epoch during learning). When we used this decreasing learning rate and fixed  $\gamma$  to a reasonable value, we found that the performance of the p-delta rule did not change much. Only the learning time (i.e. the number of epochs needed) increased: from an average of a few several hundred epochs with automatic tuning to about 2000 epochs without automatic tuning. Table 3 shows the average of 10 runs of 10-fold CV with decaying learning rate  $\eta = \frac{\eta_0}{\sqrt{t}}$ ,  $\eta_0 = 0.25$ , and the parameter  $\gamma$  fixed to  $\gamma = 0.05$  for all datasets, compared to the result with automatic tuning. The number of perceptrons was set to  $n = 3$ .

## B Appendix: Datasets for empirical evaluation

For an empirical evaluation of the p-delta rule we have chosen five datasets from the UCI machine learning repository [2]. Our criteria were binary classification tasks, few missing values in the data, and published learning results for multilayer-perceptrons trained by backprop (MLP+BP). Furthermore we looked for datasets where it was well documented how missing values should be dealt with. The datasets we have chosen are:

- Breast-cancer (BC): We used the original Winconsin breast-cancer dataset, consisting of 699 examples of breast-cancer medical data out of two classes. Each of the nine attributes is an integer value between 1 and 10. There are 16 examples containing missing values. 65.5% of the examples form the majority class in both cases, with and without the examples containing missing values.
- Pima Indian Diabetes (DI): This dataset contains 768 instances with 8 attributes each plus a binary class label. There are no missing values in

this dataset. All eight attributes are real values. The baseline accuracy is 65.1%.

- Heart Disease (HD): We used the Cleveland heart disease database. From the 13 attributes describing the two classes (healthy or diseased heart) there are 6 real valued attributes and 7 nominal attributes. 7 examples out of 303 contain missing values. The majority class is 54.5% including the examples with missing values and 54.1% without those examples.

The datasets BC, HD, and DI were used in [17] to evaluate the training of a 2-layer neural network with backpropagation and momentum, with and without an internal mechanism to avoid overfitting. The learning rate was set to 0.01 and the momentum was set to 0.8. They did 10-fold cross validation on the BC, HD and DI dataset. We report their results for 10 neurons in the hidden layer. They do not state exactly how the examples with missing values were treated. We replaced missing values by the corresponding average values.

- Ionosphere (IO): This database contains 351 examples of radar return signals from the ionosphere. Each example consists of 34 real valued attributes plus binary class information. There are no missing values. [16] report results for the IO dataset using a 2-layer sigmoidal neural network with different numbers (0,3,5,8,10,15) of neurons in the hidden layer, plus a result for a linear perceptron. They split the dataset into 200 training examples (101 "good" returns, 99 "bad" returns) and 151 test examples.
- Sonar (SN): The sonar database is a high-dimensional dataset describing sonar signals in 60 real-valued attributes. The two classes are 'rock' (97 examples) and 'metal' (111 examples). The dataset contains 208 instances and no missing values. On the sonar dataset (SN) [6] did a detailed analysis of sigmoidal neural networks with different numbers (2,3,6,12,24) of neurons in the hidden layer. We used their result for a 2-

	BC	DI	HD	IO	SN
Examples	683 (699)	768	296 (303)	351	208
Classes	2	2	2	2	2
Majority Class	65.5% (65.5%)	65.1%	54.1% (54.5%)	64.1%	53.4%
Ex. with missing	16	0	7	0	0
#Attributes	9	8	13	34	60
# numeric	9	8	6	34	60
# binary	0	0	3	0	0
# nominal $\geq 3$	0	0	4	0	0

Table 4: Description of the datasets used for empirical evaluation. The numbers in brackets show the corresponding values per dataset including examples with missing values.

layer sigmoidal neural network with 3 units in the hidden layer averaged over 10 trials of 13 fold cross validation of angle-independent training.

Table 4 gives an overview about the datasets being used. The values of the majority class in brackets is including examples with missing values.

## C Appendix: On the Computational Power of Parallel Perceptrons

We show in this Appendix that circuits that employ a soft winner-take-all gate as their only nonlinearity can be simulated by parallel perceptrons (Theorem C.1). Hence the universal approximation result of [11] implies a corresponding result for parallel perceptrons (see Corollary C.2). In addition we show in Theorem C.4 that a very large class of nonlinear filters on time series can be approximated if one adds delay lines to parallel perceptrons.

**Theorem C.1** *Let  $\tilde{f} : \mathbb{R}^d \rightarrow [0, 1]$  be some arbitrary function computed by a soft-winner-take-all gate (in the terminology of [11]) applied to weighted sums of input components. Then the closely related function  $f : \mathbb{R}^d \rightarrow [-1, 1]$  defined by  $f(\mathbf{z}) = 2\tilde{f}(\mathbf{z}) - 1$  can be computed by a parallel perceptron.*

**Proof:** Assume that  $\tilde{f}$  is computed by a circuit consisting of a soft-winner-take-all gate applied to the  $\tilde{n}$  weighted sums

$$\tilde{S}_j = \sum_{i=1}^d \alpha_i^j z_i + \alpha_o^j, \quad j = 1, \dots, \tilde{n}.$$

Thus  $\tilde{f}(\mathbf{z}) = \pi\left(\frac{|\{j \in \{1, \dots, \tilde{n}\} : \tilde{S}_n(\mathbf{z}) \geq \tilde{S}_j(\mathbf{z})\}| - \frac{\tilde{n}}{2}}{k}\right)$  for some  $k \in \{1, \dots, \lfloor \frac{\tilde{n}}{2} \rfloor\}$ , where  $\pi(x) = x$  for  $x \in [0, 1]$ ,  $\pi(x) = 1$  for  $x > 1$ , and  $\pi(x) = 0$  for  $x < 0$ .<sup>4</sup>

Define  $\rho := k$ ,  $n := \tilde{n} + k$ ,  $S_j := \tilde{S}_n - \tilde{S}_j$  for  $j = 1, \dots, \tilde{n}$ , and  $S_j \equiv -1$  for  $j = \tilde{n} + 1, \dots, n$ . Then we have

$$S_j \geq 0 \Leftrightarrow \tilde{S}_n \geq \tilde{S}_j \quad \text{for } j = 1, \dots, \tilde{n},$$

and  $S_j < 0$  for  $j = \tilde{n} + 1, \dots, n$ . Define  $c(\mathbf{z}) := |\{j \in \{1, \dots, n\} : S_j(\mathbf{z}) \geq 0\}|$ . Then  $c(\mathbf{z}) = |\{j \in \{1, \dots, \tilde{n}\} : \tilde{S}_n(\mathbf{z}) \geq \tilde{S}_j(\mathbf{z})\}|$  for all  $\mathbf{z} \in \mathbb{R}^d$ . Furthermore the function  $f(\mathbf{z}) := s(2c(\mathbf{z}) - n)$  from  $\mathbb{R}^d$  into  $[-1, 1]$  (where  $s$  is the activation function with  $\rho := k$  defined in Section 2) can be computed by a parallel perceptron. We will show that  $f(\mathbf{z}) = 2\tilde{f}(\mathbf{z}) - 1$  for all  $\mathbf{z} \in \mathbb{R}^d$ .

The preceding definitions of  $\rho$  and  $n$  imply that

$$\frac{c(\mathbf{z}) - \frac{\tilde{n}}{2}}{k} = \frac{c(\mathbf{z}) - \frac{n}{2} + \frac{k}{2}}{k} = \frac{2c(\mathbf{z}) - n + k}{2k} = \frac{2c(\mathbf{z}) - n}{2\rho} + \frac{1}{2} \quad (2)$$

and

$$\frac{2c(\mathbf{z}) - n}{\rho} = 2\left(\frac{c(\mathbf{z}) - \frac{\tilde{n}}{2}}{k}\right) - 1. \quad (3)$$

**Case 1:**  $\frac{c(\mathbf{z}) - \frac{\tilde{n}}{2}}{k} \in (0, 1)$ .

Then  $\frac{2c(\mathbf{z}) - n}{\rho} \in (-1, 1)$  and we have

$$f(\mathbf{z}) = \frac{2c(\mathbf{z}) - n}{\rho} = 2\left(\frac{c(\mathbf{z}) - \frac{\tilde{n}}{2}}{k}\right) - 1 = 2\tilde{f}(\mathbf{z}) - 1.$$

---

<sup>4</sup>We assume here without loss of generality that the  $\tilde{n}$ th output of the soft-winner-take-all gate is interpreted as the output of the circuit.

$$\text{Case 2: } \frac{c(\mathbf{z}) - \frac{n}{2}}{k} \geq 1$$

Then  $\frac{2c(\mathbf{z}) - n}{\rho} \geq 1$  and  $f(\mathbf{z}) = 1 = 2 \cdot 1 - 1 = 2\tilde{f}(\mathbf{z}) - 1$ .

$$\text{Case 3: } \frac{c(\mathbf{z}) - \frac{n}{2}}{\rho} \leq -1$$

Then  $\frac{2c(\mathbf{z}) - n}{\rho} \leq -1$  and  $f(\mathbf{z}) = -1 = 2 \cdot 0 - 1 = 2\tilde{f}(\mathbf{z}) - 1$ . ■

### Corollary C.2 (Universal Approximation Theorem for Parallel Perceptrons)

Assume that  $h : D \rightarrow [-1, 1]$  is some arbitrary continuous function with a compact domain  $D \subseteq \mathbb{R}^d$  (for example  $D = [-1, 1]^d$ ), and  $\varepsilon > 0$  is some given parameter. Then there exists a parallel perceptron that computes a function  $f : \mathbb{R}^d \rightarrow [-1, 1]$  such that

$$|f(\mathbf{z}) - h(\mathbf{z})| < \varepsilon \text{ for all } \mathbf{z} \in D .$$

Furthermore any Boolean function can be computed by rounding the output of a parallel perceptron (i.e., output 1 if the number  $p$  of positive weighted sums is above some threshold, else output 0).

**Proof:** Define  $\tilde{h} : D \rightarrow [0, 1]$  by  $\tilde{h}(\mathbf{z}) := \frac{1}{2}(h(\mathbf{z}) + 1)$ . Then  $\tilde{h}$  is a continuous function from  $D$  into  $[0, 1]$ . According to Theorem 4.1 in [11] one can compute a function  $\tilde{f} : \mathbb{R}^d \rightarrow [0, 1]$  by a soft-winner-take-all gate applied to weighted sums so that  $|\tilde{h}(\mathbf{z}) - \tilde{f}(\mathbf{z})| < \frac{\varepsilon}{2}$  for all  $\mathbf{z} \in D$ . According to Theorem C.1 one can compute the function  $f : \mathbb{R}^d \rightarrow [-1, 1]$  defined by  $f(\mathbf{z}) = 2\tilde{f}(\mathbf{z}) - 1$  by a parallel perceptron. We have for all  $\mathbf{z} \in D$  :  $|h(\mathbf{z}) - f(\mathbf{z})| = |(2\tilde{h}(\mathbf{z}) - 1) - (2\tilde{f}(\mathbf{z}) - 1)| = 2|\tilde{h}(\mathbf{z}) - \tilde{f}(\mathbf{z})| < \varepsilon$ .

Since any Boolean function from  $\{0, 1\}^d$  into  $\{0, 1\}$  can be interpolated by a continuous function, the preceding result implies the last sentence of the claim. ■

**Remark C.3** Circuits with a soft-winner-take-all gate have an obvious biological interpretation via lateral inhibition between pools of spiking neurons. Parallel perceptrons support a somewhat different biological interpretation on the level of individual spiking neurons, rather than pools of spiking neurons, without lateral inhibition: One may assume that each weighted sum  $\alpha_i \cdot \mathbf{z}$  represents the input current to some spiking neuron that fires (at time 0) if and only if  $\alpha_i \cdot \mathbf{z} \geq 0$ . Hence  $p := \#\{1 \leq i \leq n : \alpha_i \cdot \mathbf{z} \geq 0\}$  can be interpreted as the number of spikes that are sent out at time 0 by an array consisting of  $n$  spiking neurons.

In the subsequent theorem we extend our model to computations on time series, which are represented by continuous functions  $u_1, \dots, u_m$  from  $\mathbb{R}$  into  $\mathbb{R}$ . We analyze the computational power of circuits consisting of a parallel perceptron applied to some finite set of delayed versions  $D_\tau u_i$  of inputs for  $i = 1, \dots, m$  and  $\tau > 0$ , where  $D_\tau u_i$  is the function from  $\mathbb{R}$  into  $\mathbb{R}$  defined by  $(D_\tau u_i)(t) = u_i(t - \tau)$ . We will refer to these very simple circuits as *parallel perceptrons with delay-lines*. The following result shows that basically any interesting nonlinear filter can be uniformly approximated by such circuits. We refer to [12] for the definition of the notions that are used.

**Theorem C.4** *Assume that  $U$  is the class of functions from  $\mathbb{R}$  into  $[B_0, B_1]$  which satisfy  $|u(t) - u(s)| \leq B_2 \cdot |t - s|$  for all  $t, s \in \mathbb{R}$ , where  $B_0, B_1, B_2$  are arbitrary real-valued constants with  $B_0 < B_1$  and  $0 < B_2$ . Let  $\mathcal{F}$  be an arbitrary filter that maps vectors  $\underline{u} = \langle u_1, \dots, u_n \rangle \in U^n$  into functions from  $\mathbb{R}$  into  $\mathbb{R}$ .*

*Then the following are equivalent:*<sup>5</sup>

(a)  $\mathcal{F}$  can be approximated by parallel perceptrons with delay lines

(i.e., for any  $\varepsilon > 0$  there exists such circuit  $\mathcal{C}$  such that  $|\mathcal{F}\underline{u}(t) - \mathcal{C}\underline{u}(t)| < \varepsilon$  for all  $\underline{u} \in U^n$  and all  $t \in \mathbb{R}$ )

---

<sup>5</sup>The implication “(b)  $\Rightarrow$  (c)” was already shown in [3].

(b)  $\mathcal{F}$  is time invariant and has fading memory

(c)  $\mathcal{F}$  can be approximated by a sequence of (finite or infinite) Volterra series.

**Proof:** The proof is a simple variation of the proof of the corresponding Theorem 3.1 in [12], which in turn relies on earlier arguments from [3]. Apart from Corollary C.2 one uses the fact that any two different functions  $v_1, v_2 : (-\infty, 0] \rightarrow \mathbb{R}$  can be separated via a suitable delay line, i.e., there exists some  $\tau > 0$  such that  $(D_\tau v_1)(0) \neq (D_\tau v_2)(0)$ . ■