Available online at www.sciencedirect.com

ScienceDirect

Neural Networks 21 (2008) 786–795

Neural Networks

www.elsevier.com/locate/neunet

A learning rule for very simple universal approximators consisting of a single layer of perceptrons[☆]

Peter Auer[a], Harald Burgsteiner[b,*], Wolfgang Maass[c]

[a] *Chair for Information Technology, University of Leoben, Franz-Josef-Strasse 18, A-8700 Leoben, Austria*
[b] *Department of Health Care Engineering, Graz University of Applied Sciences, Eggenberger Allee 11, A-8020 Graz, Austria*
[c] *Institute for Theoretical Computer Science, Graz University of Technology, Inffeldgasse 16b/1, A-8010 Graz, Austria*

Received 21 March 2005; received in revised form 6 December 2007; accepted 6 December 2007

Abstract

One may argue that the simplest type of neural networks beyond a single perceptron is an array of several perceptrons in parallel. In spite of their simplicity, such circuits can compute any Boolean function if one views the majority of the binary perceptron outputs as the binary output 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. Note that in contrast to the familiar model of a "multi-layer perceptron" the parallel perceptron that we consider here has just binary values as outputs of gates on the hidden layer. For a long time one has thought that there exists no competitive learning algorithm for these extremely simple neural networks, which also came to be known as committee machines. It is commonly assumed that one has to replace the hard threshold gates on the hidden layer by sigmoidal gates (or RBF-gates) and that one has to tune the weights on at least two successive layers in order to achieve satisfactory learning results for any class of neural networks that yield universal approximators. We show that this assumption is not true, by exhibiting a simple learning algorithm for parallel perceptrons — the *parallel delta rule* (p-delta rule). In contrast to backprop for multi-layer perceptrons, the p-delta rule only has to tune a single layer of weights, and it does not require the computation and communication of analog values with high precision. Reduced communication also distinguishes our new learning rule from other learning rules for parallel perceptrons such as MADALINE. Obviously these features make the p-delta rule attractive as a biologically more realistic alternative to backprop in biological neural circuits, but also for implementations in special purpose hardware. We show that the p-delta rule also implements gradient descent – with regard to a suitable error measure – although it does not require to compute derivatives. Furthermore it is shown through experiments on common real-world benchmark datasets that its performance is competitive with that of other learning approaches from neural networks and machine learning. It has recently been shown [Anthony, M. (2007). On the generalization error of fixed combinations of classifiers. *Journal of Computer and System Sciences 73*(5), 725–734; Anthony, M. (2004). On learning a function of perceptrons. In *Proceedings of the 2004 IEEE international joint conference on neural networks* (pp. 967–972): *Vol. 2*] that one can also prove quite satisfactory bounds for the generalization error of this new learning rule.
© 2008 Elsevier Ltd. All rights reserved.

Keywords: Learning algorithm; Parallel delta rule; Parallel perceptrons; Committee machines; Reduced communication

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 Moerland and Fiesler (1996)).

[☆] Research for this article was partially supported by the Austrian Science Fund FWF, project #S9104-N13 and project #P17229, and project #FP6-015879 (FACETS) of the European Union, and by the IST Programme of the European Community, under the PASCAL Network of Excellence, IST-2002-506778.

[*] Corresponding author.
E-mail addresses: auer@unileoben.ac.at (P. Auer), harry@igi.tugraz.at (H. Burgsteiner), maass@igi.tugraz.at (W. Maass).

0893-6080/$ - see front matter © 2008 Elsevier Ltd. All rights reserved.
doi:10.1016/j.neunet.2007.12.036

For similar reasons it is quite dubious whether backprop is applied by biological neural systems to adapt their input/output behavior.

Backprop requires the computation and communication of analog numbers (derivatives) with high bit precision, which is difficult to achieve with noisy analog computing elements and noisy communication channels, such as those that are available in biological wetware. We will show in this article that there exists an alternative solution: a simple distributed learning algorithm for parallel perceptron 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. The learning algorithm consists of a simple extension of the familiar delta rule for a single perceptron, which we call the p-delta rule.

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 Widrow and Lehr (1990) and chapter 6 of Nilsson (1990) for an excellent survey). In fact, Rosenblatt's classic book (Rosenblatt, 1962) contains already an explicit algorithm for feedforward networks composed of binary elements. A major advantage of the p-delta rule over algorithms like MADALINE (and backprop for multi-layer perceptrons) is the fact that the p-delta rule requires only the transmission of less than 2 bits of communication (one of the three possible signals "up", "down", "neutral") from the central control to the local agents that control the weights of the individual perceptrons. In spite of its simplicity, the p-delta rule introduces a powerful new principle from machine learning into learning algorithms for committee machines: maximization of the margin of individual perceptrons. The superior generalization capability of support vector machines is largely based on this principle. We demonstrate that also in this new context (a distributed network) it provides good generalization behavior. We refer to Anthony (2007) for a rigorous analysis of this novel application of the margin maximization principle. There it is shown that the generalization error of the p-delta rule can be bounded in terms of the margins achieved by the constituent perceptrons.

The p-delta rule provides 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. Note that in contrast to backprop the p-delta rule does not require a sophisticated "shadow-network" for learning, that computes and transmits fairly complex individual error signals to the neurons. The p-delta rule can also be 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. In fact, the p-delta rule has already been applied successfully to the training of a pool of spiking neurons (Maass, Natschlaeger, & Markram, 2002). The empirical results from Maass et al. (2002) also demonstrate that the p-delta rule performs well not just for classification but also for regression problems.

Learning algorithms for committee machines have also been discussed in the physics literature, see e.g. Copelli and Caticha (1995) and the references in that article. Copelli and Caticha (1995) has analyzed online learning for a special case of the parallel perceptron with 3 hidden units and non-overlapping inputs of these 3 units. The emphasis is there on (in general quite complex) learning rules that minimize the generalization error for this particular architecture. In contrast to that, the p-delta rule is an extremely simple learning rule that requires very little communication within the network, but still generalizes quite well.

We first show in Section 2 of this article that the parallel perceptron is a surprisingly powerful computational model. The p-delta learning rule for parallel perceptrons is presented and motivated in Section 3. Further theoretical and empirical analysis of the p-delta rule is presented in Section 5. In particular, its performance on 8 common benchmark datasets is analyzed and compared with that of MADALINE and backprop, and with two state-of-the-art learning algorithms from machine learning (decision trees and support vector machines). It turns out that the performance of the p-delta rule is, in spite of its simplicity and extremely low organizational and communication requirement within the network, quite competitive with these other approaches. Possible applications to biological models and VLSI are discussed in Section 4.

Some results of this article were previously announced in Auer, Burgsteiner, and Maass (2002).

2. Parallel perceptrons are universal approximators

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. (We assume that one of the inputs is a constant bias input.)

We define a *parallel perceptron* as a single layer consisting of a finite number n of perceptrons (without lateral connections). Let $f_1, \ldots, f_n$ 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 $\sum_{i=1}^n f_i(\mathbf{z}) \in \{-n, \ldots, n\}$, more precisely the value $s(\sum_{i=1}^n f_i(\mathbf{z}))$, where

$s : \mathbb{Z} \to \mathbb{R}$ is a squashing function that scales the output into the desired range.[1]

In the case of binary classification the squashing function is a simple threshold function

$$s(p) = \begin{cases} -1 & \text{if } p < 0 \\ +1 & \text{if } p \geq 0. \end{cases}$$

It is not difficult to prove that *every* Boolean function from $\{-1, 1\}^d$ into $\{-1, 1\}$ can be computed by such a parallel perceptron. This type of parallel perceptrons is a special case of the MADALINE architecture (Widrow & Lehr, 1990).

For regression problems the squashing function could be piecewise linear,

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

where $1 \leq \rho \leq n$ denotes the resolution of the squashing function.

We now show that parallel perceptrons are in fact universal approximators: *every* continuous function $g : \mathbb{R}^d \to [-1, 1]$ can be approximated by a parallel perceptron within any given error bound ε on any closed and bounded subset of $\mathbb{R}^d$. This follows as Corollary 2.2 from the following theorem, which relates the (surprisingly large) computational power of soft-winner-take-all circuits to that of parallel perceptrons. A soft-winner-take-all gate g computes a function from $\mathbb{R}^n$ into $\mathbb{R}$ defined by

$$g(x_1, \ldots, x_n) = \pi\left(\frac{|\{j \in \{1, \ldots, n\} : x_n \geq x_j\}| - \frac{n}{2}}{k}\right)$$

where $\pi(x) = x$ for $x \in [0, 1]$, $\pi(x) = 1$ for $x > 1$, $\pi(x) = 0$ for $x < 0$, and $k \in \{1, \ldots, \lfloor \frac{n}{2} \rfloor\}$ (see Maass (2000)). Note that the output value of this gate grows with the rank of x_n in the linear order of the numbers $\{x_1, \ldots, x_n\}$.

Theorem 2.1. *Let $\tilde{f} : \mathbb{R}^d \to [0, 1]$ be some arbitrary function computed by a soft-winner-take-all gate (in the terminology of Maass (2000)) applied to weighted sums of input components. Then the closely related function $f : \mathbb{R}^d \to [-1, 1]$ 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

$$\tilde{S}_j = \sum_{i=1}^{d} \alpha_i^j z_i \quad \text{for } j = 1, \ldots, \tilde{n}.$$

Thus $\tilde{f}(\mathbf{z}) = \pi\left(\frac{|\{j \in \{1,\ldots,\tilde{n}\} : \tilde{S}_{\tilde{n}}(\mathbf{z}) \geq \tilde{S}_j(\mathbf{z})\}| - \frac{\tilde{n}}{2}}{k}\right)$ for some $k \in \{1, \ldots \lfloor \frac{\tilde{n}}{2} \rfloor\}$. In order to compute the closely related function $f(\mathbf{z}) = 2\tilde{f}(\mathbf{z}) - 1$ (which simply rescales the output range $[0, 1]$ of $\tilde{f}$ into the output range $[-1, 1]$ of a parallel perceptron; this difference in range of the outputs is of course irrelevant) by a parallel perceptron, we only need to define an array of threshold gates which compare in parallel on the hidden layer the relative sizes of the (linear) weighted sums $\tilde{S}_j$ and $\tilde{S}_{\tilde{n}}$. The number of these threshold gates that output "1" can easily be transformed into the natural number $c(\mathbf{z})$ (see next paragraph), that corresponds to the value of the nominator in the quotient that appears in the preceding definition of $\tilde{f}(\mathbf{z})$. The activation function π in the definition of $\tilde{f}(\mathbf{z})$ can be simulated by the piecewise linear activation function $S = S_k$ at the output of a parallel perceptron.

Formally we proceed as follows. Define $\rho := k$, $n := \tilde{n} + k$, $S_j := \tilde{S}_{\tilde{n}} - \tilde{S}_j$ for $j = 1, \ldots, \tilde{n}$, and $S_j \equiv -1$ for $j = \tilde{n} + 1, \ldots, n$. Then we have

$$S_j \geq 0 \Leftrightarrow \tilde{S}_{\tilde{n}} \geq \tilde{S}_j \qquad \text{for } j = 1, \ldots, \tilde{n},$$

and $S_j < 0$ for $j = \tilde{n} + 1, \ldots, n$. Define $c(\mathbf{z}) := |\{j \in \{1, \ldots, n\} : S_j(\mathbf{z}) \geq 0\}|$. Then $c(\mathbf{z}) = |\{j \in \{1, \ldots, \tilde{n}\} : \tilde{S}_{\tilde{n}}(\mathbf{z}) \geq \tilde{S}_j(\mathbf{z})\}|$ 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 at the beginning of this section) 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 ρ and n imply that

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

and

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

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

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

Case 2: $\frac{c(\mathbf{z}) - \frac{\tilde{n}}{2}}{k} \geq 1$
Then $\frac{2c(\mathbf{z}) - n}{\rho} \geq 1$ and $f(\mathbf{z}) = 1 = 2 \cdot 1 - 1 = 2\tilde{f}(\mathbf{z}) - 1$.
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 2.2 (*Universal Approximation Theorem for Parallel Perceptrons*). *Assume that $h : D \to [-1, 1]$ 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 $f := \mathbb{R}^d \to [-1, 1]$ such that*

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

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*).*

[1] This could be interpreted as a space-rate coding in a biological interpretation.

Proof. Define $\tilde{h} : D \to [0, 1]$ 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 Maass (2000) one can compute a function $\tilde{f} : \mathbb{R}^d \to [0, 1]$ 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 2.1 one can compute the function $f : \mathbb{R}^d \to [-1, 1]$ 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 2.3. (a) The result of Corollary 2.2 is less obvious than it may appear at first sight. For 2-layer feedforward neural networks with *sigmoidal* gates on the hidden layer (which are usually referred to "multi-layer perceptrons", although this notion is misleading) the corresponding Universal Approximation Theorem is well known (Haykin, 1999). But the parallel perceptrons that we consider in this article (in particular in Corollary 2.2) have instead threshold gates on the hidden layer, which provide only binary outputs.

(b) The number of perceptrons of the parallel perceptron that is constructed in Corollary 2.2 is equal to the number $\tilde{n}$ of weighted sums that provide input to the soft-winner-take-all gate which computes an approximation $\tilde{f}$ to the given continuous function $\tilde{h}$ with precision $\frac{\varepsilon}{2}$. According to Remark 4.2 in Maass (2000) this number $\tilde{n}$ can be bounded in terms of the number of hidden units in a 2-layer circuit C of sigmoidal gates that approximates $\tilde{h}$ within $\frac{\varepsilon}{2}$, the maximal size of weights (expressed as multiple of the smallest nonzero weight) of the gate in the second layer of C, and on $\frac{1}{\varepsilon}$. Hence the required number of perceptrons (with binary output) in the approximating parallel perceptron of Corollary 2.2 is likely to be larger than the number of gates in C (which give analog outputs), but not astronomically larger.

(c) 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 $\boldsymbol{\alpha}_i \cdot \mathbf{z}$ represents the input current to some spiking neuron that fires (at time 0) if and only if $\boldsymbol{\alpha}_i \cdot \mathbf{z} \geq 0$. Hence $p := \#\{1 \leq i \leq n : \boldsymbol{\alpha}_i \cdot \mathbf{z} \geq 0\}$ 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 computational analysis to computations on time series, which are represented by continuous functions $u_1, \ldots, u_m$ 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 $i = 1, \ldots, m$ 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 Maass and Sontag (2000) for the definition of the notions that are used.

Theorem 2.4. *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 $\underline{u} = \langle u_1, \ldots, u_n \rangle \in U^n$ into functions from $\mathbb{R}$ into $\mathbb{R}$.*

Then the following are equivalent[2]:

(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 $t \in \mathbb{R}$)
(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 Maass and Sontag (2000), which in turn relies on earlier arguments from Boyd and Chua (1985). Apart from Corollary 2.2 one uses the fact that any two different functions $v_1, v_2 : (-\infty, 0] \to \mathbb{R}$ can be separated via a suitable delay line, i.e., there exists some $\tau > 0$ such that $(D_\tau v_1)(0) \neq (D_\tau v_2)(0)$. ■

3. The p-delta learning rule

In this section we develop a learning rule for parallel perceptrons. We call this learning rule the p-deltarule. 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 (Guyon, Boser, & Vapnik, 1993). Support vector machines learn classifiers which maximize the *margin* of the classification: An input $\mathbf{z} \in \mathbb{R}^d$ is classified by 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 – with respect to generalization – of the trained parallel perceptron. The idea of training a single perceptron with a sufficiently large margin dates back to Mays' "increment adaptation rule" (Mays, 1963; Block, Boyd, & Chua, 1962).

In the following we discuss the basic options regarding the application of the classical delta rule in the context of parallel perceptrons, and we show how to incorporate the *large margin* idea. We first discuss the more general rule for regression and then state the simplified version for binary classification.

[2] The implication "$(b) \Rightarrow (c)$" was already shown in Boyd and Chua (1985).

3.1. Getting the outputs right

Let $(\mathbf{z}, o) \in \mathbb{R}^d \times [-1, +1]$ be the current training example and let $\boldsymbol{\alpha}_1, \ldots, \boldsymbol{\alpha}_n \in \mathbb{R}^d$ 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 $|\hat{o} - o| \leq \varepsilon$ where ε is the desired accuracy, then the output of the parallel perceptron is correct up to this accuracy and the weights need not be modified.

Consider now the case

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

where 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:

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 Nilsson (1990, 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

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

Although by our choice of the update option 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.

p-delta **rule**

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

(a)

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

(b)

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

Fig. 1. The p-delta rule.

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" with 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* γ around zero clear from any dot products $\boldsymbol{\alpha}_i \cdot \mathbf{z}$. The margin $\gamma > 0$ is a parameter of our 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

$$\boldsymbol{\alpha}_i \leftarrow \boldsymbol{\alpha}_i + \eta \mu \mathbf{z} \tag{3}$$

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

Observe that a margin γ 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 γ by a factor $1/C$ is equivalent. Thus we keep the weights $\boldsymbol{\alpha}_i$ normalized, $\|\boldsymbol{\alpha}_i\| = 1$.[3] In conclusion, the p-delta rule is summarized in Fig. 1. 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. 1.

4. Possible applications to biological models and analog VLSI

We have shown in Corollary 2.2 that parallel perceptrons have the "universal approximation property", i.e., they can approximate any given continuous function uniformly on any

[3] There exists substantial evidence that biological neurons are also genetically programmed to keep their incoming weights normalized, see Turrigiano (2004) for a recent review.

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. 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 Gerstner (1998) and Maass (1997)). Hence a parallel perceptron can be used to predict how many of these neurons will fire within such short time interval. Thus the universal approximation property of parallel perceptrons implies that a single population P of biological neurons (without lateral connections) can in principle approximate any given continuous function f. 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 discuss in this article, has already been tested in this biological context (Maass et al., 2002). The results of these simulations show that the p-delta learning rule is very successful in training populations of readout neurons to adopt a given population response. We are not aware of any other learning algorithm that could be used for that purpose. In particular we are exploiting here that, in contrast to backprop, the p-delta learning rule can be applied to neuron models that do not have a smooth activation function. The computer simulations discussed in Maass et al. (2002) also show that the p-delta rule can be used to train models for circuits of biological neurons to adopt a given temporal response, i. e., to approximate a given nonlinear filter.

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. Most of 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 difficult to satisfy 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 more 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.

5. Further analysis of the *p*-delta rule

5.1. *The p-delta rule performs gradient descent for some error function*

In this section we construct an error function $\mathrm{Err}(\boldsymbol{\alpha}_1, \dots, \boldsymbol{\alpha}_n; \mathbf{z}, o)$ for input $\mathbf{z}$ and target output o with respect to the weights $\boldsymbol{\alpha}_i$ of the parallel perceptron, such that the p-delta rule performs gradient descent with respect to this error function.

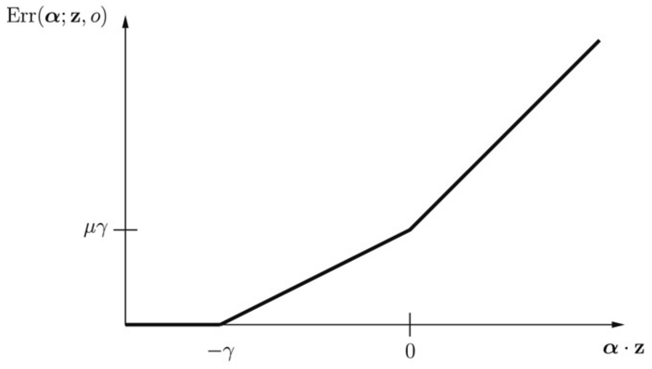

Fig. 2. Plot of the error function $\mathrm{Err}(\boldsymbol{\alpha}; \mathbf{z}, o)$ for a single weight vector $\boldsymbol{\alpha}$ versus the dot product $\boldsymbol{\alpha} \cdot \mathbf{z}$, assuming that $\|\boldsymbol{\alpha}\| = 1$ and $\hat{o} > o + \varepsilon$ for all displayed $\boldsymbol{\alpha}$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

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 $\hat{o} = s_\rho\left(\sum_{i=1}^n f_i(\mathbf{z})\right)$, 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 target output o can now be defined as

$$\mathrm{Err}(\boldsymbol{\alpha}_1, \dots, \boldsymbol{\alpha}_n; \mathbf{z}, o) = \begin{cases} \sum_{i:\boldsymbol{\alpha}_i \cdot \mathbf{z} \geq 0} (\mu\gamma + \boldsymbol{\alpha}_i \cdot \mathbf{z}) + \sum_{i:-\gamma<\boldsymbol{\alpha}_i \cdot \mathbf{z}<0} (\mu\gamma \\ \quad + \mu\boldsymbol{\alpha}_i \cdot \mathbf{z}) \quad \text{if } \hat{o} > o + \varepsilon \\ \sum_{i:\boldsymbol{\alpha}_i \cdot \mathbf{z} < 0} (\mu\gamma - \boldsymbol{\alpha}_i \cdot \mathbf{z}) + \sum_{i:0\leq\boldsymbol{\alpha}_i \cdot \mathbf{z}<+\gamma} (\mu\gamma \\ \quad - \mu\boldsymbol{\alpha}_i \cdot \mathbf{z}) \quad \text{if } \hat{o} < o - \varepsilon \\ \sum_{i:-\gamma<\boldsymbol{\alpha}_i \cdot \mathbf{z}<0} (\mu\gamma + \mu\boldsymbol{\alpha}_i \cdot \mathbf{z}) + \sum_{i:0\leq\boldsymbol{\alpha}_i \cdot \mathbf{z}<\gamma} (\mu\gamma \\ \quad - \mu\boldsymbol{\alpha}_i \cdot \mathbf{z}) \quad \text{if } |\hat{o} - o| \leq \varepsilon. \end{cases} \tag{4}$$

Lines 1 and 2 cover the case where the output of the parallel perceptron is too far from the target output, i.e. $|o - \hat{o}| > \epsilon$. 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 the dot products $\boldsymbol{\alpha}_i \cdot \mathbf{z}$ are on the wrong side (from 0), e.g. if $\hat{o} > o + \varepsilon$ then 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 γ 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 γ is $\gamma - \boldsymbol{\alpha}_i \cdot \mathbf{z}$. Weighting this distance by μ gives the error term $\mu\gamma - \mu\boldsymbol{\alpha}_i \cdot \mathbf{z}$.

The fact that the error terms – either for being on the wrong side or for being within the margin – are compatible is depicted in Fig. 2. 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 γ, scaled by μ. 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

Table 1
The results for the empirical comparison show the average accuracy on the test set for 10 times 10-fold CV (MADALINE: $n = 3$, MLP: 3 hidden units, SVM: 2nd degree polynomial kernel) and the corresponding standard error

Dataset	p-delta ($n = 3$)	MADA LINE	WEKA MLP + BP	WEKA C4.5	WEKA SVM
BC	96.94% ± 0.20	96.28% ± 0.44	96.50% ± 0.19	95.46% ± 0.53	96.87% ± 0.16
CH	97.25% ± 0.23	97.96% ± 0.18	99.27% ± 0.10	99.40% ± 0.07	99.43% ± 0.08
CR	71.73% ± 0.82	70.51% ± 0.99	73.12% ± 0.76	72.72% ± 0.89	75.45% ± 0.75
DI	73.66% ± 1.03	73.37% ± 1.38	76.77% ± 0.60	73.74% ± 0.79	77.32% ± 0.55
HD	80.02% ± 1.19	78.82% ± 1.25	82.09% ± 1.08	76.25% ± 2.22	80.78% ± 1.19
IO	84.78% ± 1.57	86.52% ± 1.23	89.37% ± 0.80	89.74% ± 0.74	91.20% ± 0.53
SI	95.72% ± 0.21	95.73% ± 0.33	96.23% ± 0.27	98.67% ± 0.21	93.92% ± 0.16
SN	74.04% ± 2.96	78.85% ± 3.16	81.63% ± 1.24	73.32% ± 1.90	84.52% ± 1.08

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 (4)).

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 $|\boldsymbol{\alpha}_i \cdot \mathbf{z}| \geq \gamma$. Taking derivatives of the error function with respect to $\boldsymbol{\alpha}_i$ we find that the first part of the p-delta rule performs gradient descent with respect to this error function.

5.2. Generalization capability and relationship to support vector machines

The idea having a clear margin around the origin is not new and is heavily used by support vector machines (Guyon et al., 1993). 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 Freund and Schapire (1999)), one expects that parallel perceptrons also exhibit good generalization. This is indeed confirmed by our empirical results reported in the next section. Further justification for our approach is provided by the theoretical analysis in Anthony (2007, 2004). There it is shown that the generalization error of a parallel perceptron can indeed be bounded in terms of the margins of the simple perceptrons, such that the bound on the generalization error decreases with increasing margins.

5.3. Empirical results on machine learning datasets

For an empirical evaluation of the p-delta rule we have chosen eight datasets with binary classification tasks from the UCI machine learning repository (Blake & Merz, 1998): Wisconsin breast-cancer (BC), King–Rook vs. King–Pawn Chess Endgames (CH), German Numerical Credit Data (CR), Pima Indian Diabetes (DI), Cleveland heart disease (HD), Ionosphere (IO), Thyroid disease records (Sick) (SI) and Sonar (SN). For a more detailed description of the datasets see Appendix B. Our criteria were binary classification tasks and few missing values in the data.

We compared our results with the implementations in WEKA[4] of multi-layer perceptrons with backpropagation (MLP + BP), the decision tree Algorithm C4.5, and support vector machines (with SMO). We also compared our results with MADALINE. 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 are the errors of the algorithms on the test sets averaged over 10 independent runs of 10-fold crossvalidation. The learning was stopped, when the error function of the p-delta rule did not improve by at least 1% during the second-half of the trials so far. Typically this occurred after a few hundred epochs, sometimes it took a few thousand epochs. For all datasets we used $\gamma = 0.05$, $\mu = 1$, and a decreasing learning rate $\eta = \frac{1}{4\sqrt{t}}$ where t is the number of epochs so far. The parameters η and γ could also be tuned automatically. This would result in partially faster convergence of learning but yields no significantly better results than with fixed parameters. 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 other classification algorithms. We also found that for the tested datasets small parallel perceptrons ($n = 3$) suffice for good classification accuracy.

The performance of the p-delta rule scales similarly to the performance of the perceptron rule. In particular, the performance of the p-delta rule does not depend on the input dimension but scales with the squares of the margins of the individual perceptrons (Anthony, 2007, 2004).

6. Discussion

Moerland and Fiesler (1996) 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 multi-layer networks with nondifferentiable Heaviside functions requires the development of new algorithms. We have presented in this article a new

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

learning algorithm – the p-delta rule – for parallel perceptrons, i.e., for neural networks consisting of a single layer of perceptrons (which use the Heaviside function as the activation function). This learning algorithm employs (implicitly) the method of maximizing the margin between examples from different classes, which is used very successfully for support vector machines. To the best of our knowledge, the p-delta rule is the first application of this strategy to hidden units in a multi-layer network. In contrast to other learning algorithms for parallel perceptrons, such as MADALINE-3 (Widrow & Lehr, 1990) and weight perturbation (Jabri & Flower, 1992) (see the survey in Moerland and Fiesler (1996)), the p-delta rule 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 neither requires smooth activation functions nor the computation and communication of derivatives. The recent results of Anthony (2004, 2007) provide attractive bounds for the generalization error of the p-delta rule. The p-delta rule involves besides the learning rate μ (which appears in virtual all learning algorithms) an additional parameter γ that regulates the sensitivity of the parallel perceptrons to small variations in the input values. Our computer simulations in Section 5.3 have shown that fixed default settings of both of these parameters perform well for a large variety of benchmark classification tasks.

Owing to the small amount of communication between perceptrons which the p-delta rule requires, one may argue that it provides a more compelling paradigm for distributed learning in biological neural circuits than the familiar backprop algorithm. It has already been used in computer simulations of biological neural circuits for training pools of neurons to approximate an analog target function via space-rate coding (Maass et al., 2002). Alternative approaches for training models for biological neural circuits, based on reinforcement learning, have recently been proposed in Fiete and Seung (2006), Xie and Seung (2004) and Izhikevich (2007). These reinforcement learning algorithms also require little communication between neurons, but more complex local computations of "eligibility" (i.e., credit assignment). It will be interesting to see a comparison of the performance of these quite different approaches. So far no results of applications of algorithms based on reinforcement learning to complex real-world classification problems (such as the benchmark problems considered in Section 5.3 of this paper) have been documented in the literature.

We have shown in Theorem 2.1 that the parallel perceptron model is closely related to the previously studied Winner-Take-All circuits (in particular it shares their universal approximation capability for arbitrary continuous functions, as we have shown in Corollary 2.2). Hence the p-delta rule also provides a new learning algorithm for Winner-Take-All circuits. The relevance of Winner-Take-All circuits for modelling computation in cortical microcircuits has recently been re-emphasized in Douglas and Martin (2004).

Appendix A. 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 also allows another modification of the update rule: instead of normalizing the weights implicitly by the update rule

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

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, η, γ, and μ. It turns out that the choice of μ is rather insignificant and we set $\mu = 1$. With η and γ the situation is more complicated. In particular the choice of γ is important for a good performance of the p-delta rule. Therefore we included a mechanism which automatically tunes γ. For this we constructed an update rule for γ. This update rule increases γ 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 γ if there are many dot products within this margin. The number of dot products within the margin is calculated as

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

(we count only dot products which have the correct sign with 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 γ is increased if the number of dot products is smaller than $M_{\min}$ and that γ is decreased 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 $M_{\max} = 4M_{\min}$.

For the last remaining parameter, the learning rate η, 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 γ and η right is crucial for the performance of the

Table 2
Numerical description of the datasets used for empirical evaluation

	BC	CH	CR	DI	HD	IO	SI	SN
Examples	683 (699)	3196	1000	768	296 (303)	351	2643 (3772)	208
Classes	2	2	2	2	2	2	2	2
Majority Class	65.5%	52.2%	70%	65.1%	54.1%	64.1%	92.0%	53.4%
Ex. with missing	16	0	0	0	7	0	1129	0
#Attributes	9	36	24	8	13	34	27	60
# numeric	9	0	24	8	6	34	6	60
# binary	0	34	0	0	3	0	20	0
# nominal ≥ 3	0	2	0	0	4	0	1	0

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. We found that the choice of the meta-parameters for the adaptation of γ and η is far less sensitive than the choice of the original parameters, which makes the automatic tuning of γ and η an effective approach.

Appendix B. Datasets used for empirical evaluation

For an empirical evaluation of the p-delta rule we have chosen eight datasets from the UCI machine learning repository (Blake & Merz, 1998). Our criteria were binary classification tasks, few missing values in the data, and published learning results for multi-layer perceptrons trained by backprop (MLP + BP). All examples containing missing values have been removed from the data. Nominal attributes were transformed into corresponding binary vectors. Table 2 gives an overview about the datasets being used. The numbers of the examples in brackets show the original size of the dataset before examples containing missing values had been removed. The rest of the values in the table are for the dataset without the examples containing missing values. 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. 16 examples containing missing values have been removed. 65.5% of the examples form the majority class.
- King–Rook vs. King–Pawn chess endgames (CH): This database consists of examples from chess endgame positions with king and rook on one side against only king and pawn as the opponents. The attributes describe various positions on the chessboard. The class is which side won in the corresponding game. None of the 3196 examples contains missing values. The majority class is 52.2%.
- German credit data (CR): The German credit database consists of 1000 examples of credit approvements. Two versions are available: one with and one without nominal and categorical attributes. We used the numerical version of this database that had also been used in Statlog. This database does not contain examples with missing values and the majority class is 70%.
- 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 contained missing values. These examples were removed from the dataset for the experiments. The majority class is 54.1% without those examples.
- 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.
- Sick (SI): This is the thyroid disease database from the Garavan Institute and Ross Quinlan. The original database contained 3772 examples of which 1129 contain missing values. These examples together with the original attributes 27 and 28 have been removed from the database. The majority class in the resulting dataset is 92%.
- 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.

References

Anthony, M. (2004). On learning a function of perceptrons. In *Proceedings of the 2004 IEEE international joint conference on neural networks* (pp. 967–972):*Vol. 2*.

Anthony, M. (2007). On the generalization error of fixed combinations of classifiers. *Journal of Computer and System Sciences*, *73*(5), 725–734.

Auer, P., Burgsteiner, H., & Maass, W. (2002). Reducing communication for distributed learning in neural networks. In J. R. Dorronsoro (Ed.), *Lecture notes in computer science*: *Vol. 2415. Proc. of the international conference on artificial neural networks* (pp. 123–128). Springer.

Blake, C. L., & Merz, C. J. (1998). UCI Repository of machine learning databases. Irvine, CA: University of California, Department of Information and Computer Science. http://www.ics.uci.edu/mlearn/MLRepository.html.

Block, H. (1962). The perceptron: A model for brain functioning, I. *Reviews of Modern Physics*, *34*, 123–135.

Boyd, S., & Chua, L. O. (1985). Fading memory and the problem of approximating nonlinear operators with Volterra series. *IEEE Transactions on Circuits and Systems*, *32*, 1150–1171.

Copelli, M., & Caticha, N. (1995). *Journal of Physics A. Mathematical and General*, *28*, 1615–1625.

Douglas, R. J., & Martin, K. A. (2004). Neuronal circuits of the neocortex. *Annual Review of Neurosciences*, *27*, 419–451.

Fiete, I. R., & Seung, H. S. (2006). Gradient learning in spiking neural networks by dynamic perturbation of conductances. *Physical Review Letters*, *97*, 048104.

Freund, Y., & Schapire, R. E. (1999). Large margin classification using the Perceptron algorithm. *Machine Learning*, *37*(3), 277–296.

Gerstner, W. (1998). Spiking Neurons. In W. Maass, & C. M. Bishop (Eds.), *Pulsed neural networks* (pp. 3–54). MIT Press, Available at: http://diwww.epfl.ch/lami/team/gerstner/wg_pub.html.

Guyon, I., Boser, B., & Vapnik, V. (1993). Automatic capacity tuning of very large VC-dimension classifiers. In *Advances in neural information processing systems*: *vol. 5* (pp. 147–155). San Mateo: Morgan Kaufmann.

Haykin, S. (1999). *Neural networks: A comprehensive foundation*. New Jersey: Prentice Hall.

Izhikevich, E. M. (2007). Solving the distal reward problem through linkage of STDP and dopamine signaling. *Cerebral Cortex*, Jan 13 [Epub ahead of print].

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*, *3*(1), 154–157.

Maass, W. (1997). Networks of spiking neurons: The third generation of neural network models. *Neural Networks*, *10*, 1659–1671.

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.

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.

Maass, W., Natschlaeger, T., & Markram, H. (2002). Real-time computing without stable status: A new framework for neural computation based on perturbations. *Neural Computation*, *14*(11), 2531–2560.

Mays, C. H. (1963). Adaptive threshold logic, *Ph.D. thesis, Tech. Rep. 1557-1*. Stanford Electron.Labs., Stanford, CA.

Moerland, P., & Fiesler, E. (1996). In E. Fiesler, & R. Beale (Eds.), *Handbook of neural computation*: *E1.2. Neural network adaptations to hardware implementations* (pp. 1–13). New York: Institute of Physics Publishing and Oxford University Publishing.

Nilsson, N. J. (1990). *The mathematical foundations of learning machines*. San Mateo USA: Morgan Kauffmann Publishers.

Rosenblatt, J. F. (1962). *Principles of neurodynamics*. New York: Spartan Books.

Turrigiano, G. (2004). A competitive game of synaptic tag. *Neuron*, *44*(6), 903–904.

Widrow, B., & Lehr, M. A. (1990). 30 years of adaptive neural networks: Perceptron, madaline, and backpropagation. *Proceedings of the IEEE*, *78*(9), 1415–1442.

Xie, X., & Seung, H. S. (2004). Learning in neural networks by reinforcement of irregular spiking. *Physical Review E*, *69*, 041909.