Reducing Communication for Distributed Learning in Neural Networks

Peter Auer Harald Burgsteiner Wolfgang Maass
{*pauer,harry,maass*}*@igi.tu-graz.ac.at*

Institute for Theoretical Computer Science
Technische Universität Graz
A-8010 Graz, Austria

Abstract. A learning algorithm is presented for circuits consisting of a single layer of perceptrons. We refer to such circuits as *parallel perceptrons*. In spite of their simplicity, these circuits are universal approximators for arbitrary boolean and continuous functions. In contrast to backprop for multi-layer perceptrons, our new learning algorithm – the *parallel delta rule* (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 such circuits such as those traditionally used for MADALINE. A theoretical analysis shows that the p-delta rule does in fact implement 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. Thus our algorithm also provides an interesting new hypothesis for the organization of learning in biological neural systems.

1 Introduction

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 show that there exists an alternative solution which has clear advantages for physical realization: a simple distributed learning algorithm for a class of neural networks with universal computational power that requires less than 2 bits of global communication. In order to get universal approximators for arbitrary continuous functions it suffices to take 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. The collective vote of these perceptrons can be rounded to -1 or 1 to yield a binary decision. Alternatively one can apply a squashing function to the collective vote and thereby get universal approximators for arbitrary continuous functions.

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 MADALINE; see chapter 6 of [7] for an excellent survey). At that time no mathematical tools were available to show the the universal approximation capability of these models. A major advantage of the p-delta rule over algorithms like MADALINE (and backprop for multiplayer 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. Hence it 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. The p-delta rule consists of a simple extension of the familiar delta rule for a single perception. It is shown in this article that parallel perceptrons can be trained in an efficient manner to approximate basically any practically relevant target function. The p-delta rule has already been applied very successfully to learning for a pool of spiking neurons [6]. The empirical results from [6] show also that the p-delta rule can be used efficiently not just for classification but also for regression problems.

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

A *parallel perceptron* is a single layer consisting of a finite number n of perceptrons. 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.

In this article we will restrict our attention to binary classification problems. We thus use as squashing function 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 a such parallel perceptron.

For regression problems one can use a piecewise linear squashing function. 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 $\mathbb{R}^d$. A prove is shown in the full paper [1].

3 The p-delta learning rule

3.1 Getting the outputs right

We discuss the p-delta rule for incremental updates where weights are updated after each presentation of a training example. The modifications for batch updates are straightforward. 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 perceptrons in the parallel perceptron. Thus the current output of the parallel perceptron is calculated as

$$\hat{o} = \begin{cases} -1 \text{ if } \#\{i : \boldsymbol{\alpha}_i \cdot \mathbf{z} \geq 0\} < \#\{i : \boldsymbol{\alpha}_i \cdot \mathbf{z} < 0\} \\ +1 \text{ if } \#\{i : \boldsymbol{\alpha}_i \cdot \mathbf{z} \geq 0\} \geq \#\{i : \boldsymbol{\alpha}_i \cdot \mathbf{z} < 0\}. \end{cases}$$

If $\hat{o} = o$ then the output of the parallel perceptron is correct and its weights need not be modified. If $\hat{o} = +1$ and $o = -1$ (we proceed analogously for $\hat{o} = -1$ and $o = +1$) the number of weight vectors with $\boldsymbol{\alpha}_i \cdot \mathbf{z} \geq 0$ needs to be reduced. Applying the classical delta rule to such a weight vector yields the update $\boldsymbol{\alpha}_i \leftarrow \boldsymbol{\alpha}_i - \eta\mathbf{z}$, where $\eta > 0$ is the learning rate. 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 is the MADALINE learning rule discussed in [7, 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. 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 whether $\hat{o} = o$, or $\hat{o} = +1$ and $o = -1$, or $\hat{o} = -1$ and $o = +1$.

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 output $\hat{o}$ of the parallel perceptron agrees with the target output 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 $\boldsymbol{\alpha}_i \cdot \mathbf{z}$. This reduces the generalization capabilities and the stability of the parallel perceptron. Therefore we modify the update rule of the previous section to keep $\boldsymbol{\alpha}_i \cdot \mathbf{z}$ away from zero. In fact, we try to keep a *margin*[1] γ around zero clear from any dot products $\boldsymbol{\alpha}_i \cdot \mathbf{z}$.

The idea of having a clear margin around the origin is not new and is heavily used by support vector machines [5, 3]. 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 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.

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 $\boldsymbol{\alpha}_i$ by $\boldsymbol{\alpha}_i \leftarrow \boldsymbol{\alpha}_i + \eta\mu\mathbf{z}$ 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. Observe that a larger margin γ is effective only if the weights $\boldsymbol{\alpha}_i$ remain bounded: one could trivially satisfy condition $|\boldsymbol{\alpha}_i \cdot \mathbf{z}| \geq \gamma$ by scaling up $\boldsymbol{\alpha}_i$ by a factor $C > 1$, but this would have no impact on the sign of $\boldsymbol{\alpha}_i \cdot \mathbf{z}$ for new test examples. Thus we keep the weights $\boldsymbol{\alpha}_i$ bounded by the additional update $\boldsymbol{\alpha}_i \leftarrow \boldsymbol{\alpha}_i - \eta \left(||\boldsymbol{\alpha}_i||^2 - 1 \right) \boldsymbol{\alpha}_i$, which moves $||\boldsymbol{\alpha}_i||$ towards 1. Concluding, we can summarize the p-delta rule for binary outcomes $o \in \{-1, +1\}$ as follows:

<u>p-delta **rule**</u>

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

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

The rather informal arguments in this and the previous section can be made more precise by showing that the p-delta rule performs gradient descent on an appropriate error function. This error function is zero iff $\hat{o} = o$ and all weight vectors $\boldsymbol{\alpha}_i$ satisfy $|\boldsymbol{\alpha}_i \cdot \mathbf{z}| \geq \gamma$ and $||\boldsymbol{\alpha}_i|| = 1$. The details of the error function are given in the full paper [1].

3.3 Application to networks of spiking neurons

One can model the decision whether a biological neuron will fire (and emit an action potential or "spike") within a given time interval (e.g., of length 5 ms)

[1] The margin needs to be set appropriately for each learning problem. In the full paper [1] we define a rule for setting this parameter automatically.

quite well with the help of a single perceptron. Hence a parallel perceptron may be viewed as a model for a population P of biological neurons (without lateral connections inside the population), where the current output value of the parallel perceptron corresponds to the current firing activity of this pool of neurons. The p-delta learning rule for parallel perceptrons has already been tested in this biological context through extensive computer simulations of biophysical models for populations of neurons [6]. 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 (i.e. regression problem), even for transformations on time series such as spike trains. We are not aware of any other learning algorithm that could be used for that purpose.

Table 1. Empirical comparison. Accuracy on test set (10 times 10-fold CV).

Dataset[a]	#exam.	#attr.	p-delta ($n = 3$)	MADA-LINE	WEKA MLP+BP	WEKA C4.5	WEKA SVM[b]
BC	683	9	96.94 %	96.28 %	96.50 %	95.46 %	96.87 %
CH	3196	36	97.25 %	97.96 %	99.27 %	99.40 %	99.43 %
CR	1000	24	71.73 %	70.51 %	73.12 %	72.72 %	75.45 %
DI	768	8	73.66 %	73.37 %	76.77 %	73.74 %	77.32 %
HD	296	13	80.02 %	78.82 %	82.10 %	76.25 %	80.78 %
IO	351	34	84.78 %	86.52 %	89.37 %	89.74 %	91.20 %
SI	2643	31	95.72 %	95.73 %	96.23 %	98.67 %	93.92 %
SN	208	60	74.04 %	78.85 %	81.63 %	73.32 %	84.52 %

[a] BC = Wisconsin breast-cancer, CH = King-Rook vs. King-Pawn Chess Endgames, DI = Pima Indian Diabetes, GE = German Numerical Credit Data, HD = Cleveland heart disease, IO = Ionosphere, SI = Thyroid disease records (Sick), SN = Sonar.

[b] MADALINE: n=3, MLP: 3 hidden units, SVM: 2nd degree polynomial kernel

4 Empirical evaluation

For another empirical evaluation of the p-delta rule we have chosen eight datasets with binary classification tasks from the UCI machine learning repository [2]. We compared our results with the implementations in WEKA[0] of multilayer 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[1]. The results are shown in Table 1. Results are averaged over 10 independent runs of 10-fold crossvalidation. The p-delta rule was applied until its error function did not improve by at least 1% during the

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

[1] For a more detailed description of our experiments see [1].

second half of the trials. Typically training stopped after a few hundred epochs, sometimes it took a few thousand epochs.

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.

5 Discussion

We have presented a learning algorithm — the p-delta rule — for parallel perceptrons, i.e., for neural networks consisting of a single layer of perceptrons. It presents an alternative solution to the credit assignment problem that neither requires smooth activation functions nor the computation and communication of derivatives. Because of the small amount of necessary communication it is argued that this learning algorithm 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 for computer simulations in that context.

We show in the full paper [1] that the parallel perceptron model is closely related to previously studied Winner-Take-All circuits. With nearly no modification the p-delta rule also provides a new learning algorithm for WTA circuits.

References

1. Auer, P., Burgsteiner, H. & Maass, W. *The p-delta Learning Rule*, submitted for publication, `http://www.igi.TUGraz.at/maass/p_delta_learning.pdf`.
2. Blake, C.L. & Merz, C.J. (1998). *UCI Repository of machine learning databases*, `http://www.ics.uci.edu/~mlearn/MLRepository.html`. Irvine, CA: University of California, Department of Information and Computer Science.
3. N. Cristianini, N., Shawe-Taylor J. (2000). *An Introduction to Support Vector Machines*, Cambridge University Press.
4. Freund, Y., and Schapire, R. E. (1999). *Large margin classification using the Perceptron algorithm*, Machine Learning 37(3):277–296.
5. Guyon I., Boser B., and 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.
6. Maass, W., Natschlaeger, T., and 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`.
7. Nilsson, N. J. (1990). *The Mathematical Foundations of Learning Machines*, Morgan Kauffmann Publishers, San Mateo (USA).