Optimal Gradient-Based Learning Using Importance Weights

Sepp Hochreiter and Klaus Obermayer
Bernstein Center for Computational Neuroscience
and
Technische Universität Berlin
10587 Berlin, Germany
{hochreit,oby}@cs.tu-berlin.de

***Abstract*— We introduce a novel "importance weight" method (IW) to speed up gradient based learning. The method is particularly useful for "difficult" data sets including features like unbalanced data, highly non-linear relationships between variables, or long-term dependencies in sequences. An importance weight is assigned to every data point of the training set. The weight controls the contribution of the data point to the total training error according to its informativeness for learning a good predictor. It can also be interpreted as an individual learning step size for the local gradient at this particular data point. The importance weights are obtained by solving a quadratic optimization problem which minimizes the absolute value of the change in the parameter vector during a learning step under the (soft) constraint, that the total error should be reduced by at least a given fixed value. For linear classifiers we show, that the new method is equivalent to standard support vector learning. We apply the IW method to feedforward multi-layer perceptrons and to recurrent neural networks (LSTM). Benchmarks with QuickProp and standard gradient descent methods are provided for toy data as well as for "real world" protein datasets. Results show that the new learning method is usually much faster in terms of epochs as well as in terms of absolute CPU time, and that it provides equal or better prediction results. In the "latching benchmark" for sequence prediction, the new approach was able to extract and exploit dependencies between sites which are 1,000,000 sequence elements apart – a new record.**

I. Introduction

Various methods have been proposed to improve or speed up gradient-based methods for supervised learning. Prominent methods include the natural gradient [1], step-size optimization [17], (pseudo) Newton methods [27], [3], [4], or momentum term methods [14], [20], and usually adapt the direction and/or the step-size of the gradient of the error function - given the local properties of the error surface. For certain data sets, however, gradient-based learning becomes unreliable and slow. This is due to the fact, that the gradient of the total training error becomes either very small or very large leading to a very small or a very large cumulative update.

Fig. 1 illustrates the problem of the vanishing gradient. Consider two points $\boldsymbol{x}^1$ and $\boldsymbol{x}^2$ close to one another in input space at a location, where the target function $y(\boldsymbol{x})$ changes rapidly in a nonlinear way. Gradient-based learning of a model function $f(\boldsymbol{x})$ using, for example, the squared error $E_{sqr}\left(\boldsymbol{x}^i\right) = \frac{1}{2}\left(y^i - f\left(\boldsymbol{x}^i\right)\right)^2$ may then lead to $\nabla_{\boldsymbol{w}} f\left(\boldsymbol{x}^1\right) \approx \nabla_{\boldsymbol{w}} f\left(x^2\right)$ for the current choice of the function's parameters. Since the targets y^1 and y^2 differ and $f\left(\boldsymbol{x}^1\right) \approx f\left(\boldsymbol{x}^2\right) \approx \frac{1}{2}\left(y^1 + y^2\right)$, we obtain $\nabla_{\boldsymbol{w}} E\left(\boldsymbol{x}^1\right) \approx -\nabla_{\boldsymbol{w}} E\left(\boldsymbol{x}^2\right)$. The cumulative effect of the individual gradients at $\boldsymbol{x}^1$ and $\boldsymbol{x}^2$ almost vanish and f hardly changes where it is required to change most. The situation depicted in Fig. 1 is often observed in sequence analysis, when sequences (for example proteins) which differ only at few positions may belong to different classes. If standard recurrent networks are used, the situation may become even worse, because recurrent networks face the vanishing gradient problem for long-term correlations [13], hence may not exploit this information.

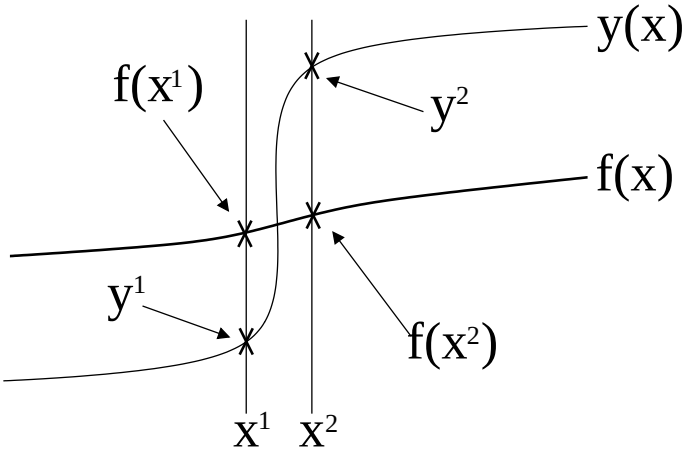

Fig. 1. Gradient contributions at x^1 and x^2 cancel each other.

Very large cumulative updates - on the other hand - usually occur for unbalanced data sets, i.e. for data sets with unbalanced target values, e.g. if class cardinalities are strongly different or certain output values are dominant. In this case cumulative updates introduce an undesired bias towards the dominant target values. A similar effect appears for unbalanced input data, where the input data distribution has high density regions from which many training points are chosen. For unbalanced input data the cumulative update over-weights the error in high density regions, i.e. it does not take into account that a small weight update can improved the error of many data points. Prominent tasks, for which unbalanced data sets are common, are 3D-protein prediction [2] and text classification

[31], because the number of training examples of the positive class is usually small (and costly to obtain). The lack of balance usually leads to a poor performance on either precision or recall (false positives or negatives) [30] and to a small area under the receiver operating curve. A common strategy to tackle this problem is to under-sample the larger class (cf. [2]), but how should one select the examples? Another aspect is that unbalanced data sets often slow down learning because the output is attracted to the dominant value.

In order to overcome the abovementioned problems, we propose a novel strategy for improving gradient-based learning. Each data point is weighted by an "importance value", which determines its influence on the total gradient and which can be interpreted as an individual learning step. These factors depend on the location in input space and on the interaction between different data points. They are calculated at every iteration by solving a convex optimization problem as described in the next section. These weighting factors then lead - for example - to an amplification of differences between similar gradients (cf. Fig. 1 or a selection of only a few important locations in an unbalanced situation.

II. The Importance Weight Method

Consider N training examples $\boldsymbol{x}^i \in \mathbb{R}^M$, $1 \leq i \leq N$ together with their corresponding target values y^i. The relationship between the $\boldsymbol{x}^i$ and the y^i should be described by a model function $f(\boldsymbol{x};\boldsymbol{w})$ with parameter vector $\boldsymbol{w}$, for example a feedforward multilayer perceptron or a recurrent neural network. For every training data we define an individual error measure $E\left(\boldsymbol{x}^i\right)$ and use it to construct the total error $E = \sum_i E\left(\boldsymbol{x}^i\right)$ for the training set. Typically, $E\left(\boldsymbol{x}^i\right) = E_{sqr}\left(\boldsymbol{x}^i\right) = \frac{1}{2}\left(y^i - f\left(\boldsymbol{x}^i\right)\right)^2$ or $E\left(\boldsymbol{x}^i\right) = E_{abs}\left(\boldsymbol{x}^i\right) = \left|y^i - f\left(\boldsymbol{x}^i\right)\right|$.

We now construct an optimal learning step for the parameter vector $\boldsymbol{w}$ in order to achieve the following two goals: (a) The learning step should be such, that the individual error for every data point decreases by at least a value of p (if possible), and (b) the associated weight change $\Delta\boldsymbol{w}$ should be as small as possible. Consider the Taylor expansion

$$E(\boldsymbol{x}^i;\boldsymbol{w}+\Delta\boldsymbol{w}) = E(\boldsymbol{x}^i;\boldsymbol{w}) + \left\langle \nabla_{\boldsymbol{w}} E(\boldsymbol{x}^i;\boldsymbol{w}), \Delta\boldsymbol{w}\right\rangle + O\left(\|\Delta\boldsymbol{w}\|^2\right). \quad (1)$$

up to the first order. We then obtain the optimization problem

$$\begin{aligned} \min_{\Delta\boldsymbol{w}} \quad & \frac{1}{2}\|\Delta\boldsymbol{w}\|^2 \\ s.t. \quad & \left\langle -\nabla_{\boldsymbol{w}} E(\boldsymbol{x}^i;\boldsymbol{w}) \ , \ \Delta\boldsymbol{w}\right\rangle \geq p \ , \end{aligned}$$

where the constraints ensure goal (a) and the minimization ensures goal (b) in order to be consistent with the linear approximation, eq. 1. p is a free parameter which corresponds to a learning rate. Note, that this optimization does not lead to $\Delta\boldsymbol{w} \propto -\nabla_{\boldsymbol{w}} E$, but the constraints ensure a positive dot product: $\langle -\nabla_{\boldsymbol{w}} E \ , \ \Delta\boldsymbol{w}\rangle \geq N\ p$.

In general, one cannot guarantee that the error can be improved by a value of p at every training data position $\boldsymbol{x}^i$ at every iteration. Therefore, slack variables ξ_i are introduced which allow for a violation of the constraints but where large values are penalized by regularization parameter C. We then obtain the convex optimization problem

$$\begin{aligned} \min_{\Delta\boldsymbol{w}} \quad & \frac{1}{2}\|\Delta\boldsymbol{w}\|^2 + C\sum_i \xi_i \\ s.t. \quad & \left\langle -\nabla_{\boldsymbol{w}} E(\boldsymbol{x}^i) \ , \ \Delta\boldsymbol{w}\right\rangle \geq p - \xi_i, \quad 0 \leq \xi_i \ . \end{aligned} \quad (2)$$

and its dual formulation

$$\begin{aligned} \min_{\boldsymbol{\alpha}} \quad & \frac{1}{2}\sum_{i,j}\alpha_i\alpha_j \left\langle \nabla_{\boldsymbol{w}} E(\boldsymbol{x}^i), \nabla_{\boldsymbol{w}} E(\boldsymbol{x}^j)\right\rangle - p\sum_i \alpha_i \\ s.t. \quad & 0 \leq \alpha_i \leq C \ . \end{aligned} \quad (3)$$

for the change of the model parameters, where α_i are the Lagrange multipliers corresponding to the constraints. The dual problem can then be solved using the sequential minimal optimization (SMO) procedure [19]. A fast and efficient implementation can be obtained, if one takes into account that there are no equality constraints and if the SMO is always initialized with the values of the α_i from the previous learning step [12]. The new learning step can then be calculated using the Karush-Kuhn-Tucker conditions

$$\begin{aligned} \Delta\boldsymbol{w} &= -\sum_i \alpha_i \ \nabla_{\boldsymbol{w}} E(\boldsymbol{x}^i) \quad \text{and} \\ \|\Delta\boldsymbol{w}\|^2 &= p\sum_i \alpha_i + C\sum_{i:\alpha_i = C} \xi_i \ . \end{aligned} \quad (4)$$

Given eq. (4) we can now interpret the Lagrange multipliers α_i as the importance weights for training vectors and as the individual step sizes for the standard gradient update rule. The importance weights are determined by minimizing the contributions of the coupling strengths $\left\langle \nabla_{\boldsymbol{w}} E(\boldsymbol{x}^i), \nabla_{\boldsymbol{w}} E(\boldsymbol{x}^j)\right\rangle$, eq. (3), which relate the gradients of the error function for every pair of data points. A good solution is obtained, if at least one $\alpha_i = 0$ for as many pairs of data points with similar gradient information and if the values of the α_i are large for pairs whose gradients tend to be antiparallel. The number of non-zero importance weights is controlled by the values of the hyperparameters p and C, and we will show in the next section that their number can be very small. Data points which are never used during learning allow to construct a new bound on the generalization error using leave-one-out estimators in analogy to the construction of bounds for support vector machines [26].

For perceptrons, $f(\boldsymbol{x}^i;\boldsymbol{w}) = \left\langle \boldsymbol{x}^i, \boldsymbol{w}\right\rangle$, and binary classification tasks using the classification error $E\left(x^i\right) = \max\left\{0, -\ y_i\ f\left(\boldsymbol{x}^i\right)\right\}$, we obtain the error gradient is $\nabla_{\boldsymbol{w}} E(\boldsymbol{x}^i;\boldsymbol{w}) = -\ y_i\ \boldsymbol{x}^i$, if $y^i\ \left\langle \boldsymbol{x}^i \ , \ \boldsymbol{w}\right\rangle \leq 0$, and 0, otherwise. Since the constraints of the optimization problem (2) can be written in the form $y^i\ \left\langle \boldsymbol{x}^i \ , \ \Delta\boldsymbol{w}\right\rangle \geq p$ we obtain the the standard support vector machine learning rule [26]. In this case, gradient-based learning reduces to one iteration of the update rule eq. (4 for the initial values $\boldsymbol{w} = \boldsymbol{0}$.

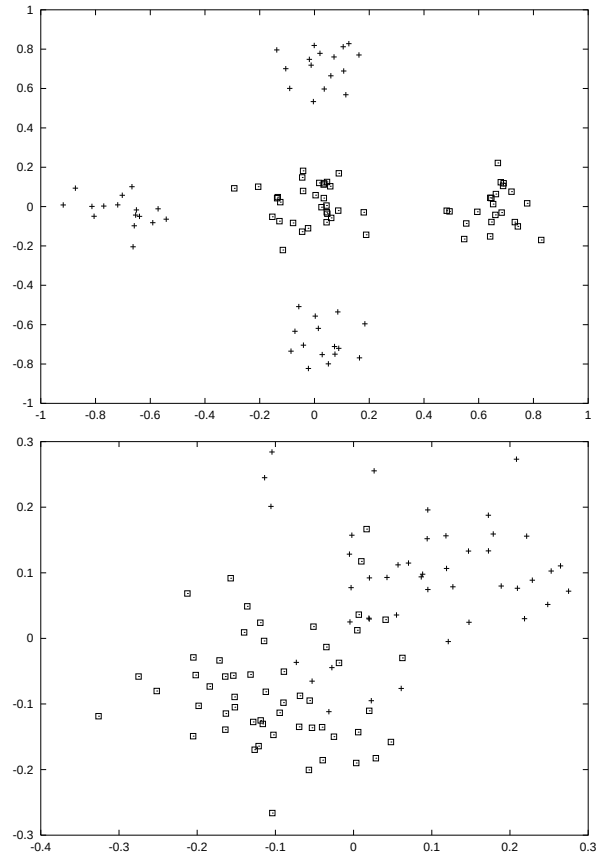

Fig. 2. Toy data sets for the binary classification task. Top: 4-Gaussian linear separable (L, data points from the central cluster are deleted and drawn again from the Gaussian which forms the right cluster) and "simple" 5-Gaussian nonlinear (NL1) dataset. Bottom: "difficult" nonlinear (NL2) dataset. Crosses (class +1) and squares (class -1) denote the location of the data points in a two-dimensional feature space.

III. Numerical Experiments

A. Application to Feedforward Networks (MLP)

1) Toy Data – Binary Classification Problem: We applied the new learning method to the three artificial data sets shown in Fig. 2: a linear separable data set L (Fig. 2, left, without the central cluster), a "simple" nonlinear data set NL1 (Fig. 2, left), and a "difficult" nonlinear data set NL2 (Fig. 2, right). A multi-layer perceptron (4 layers, 10 units per hidden layer, one output unit, sigmoid transfer functions $\frac{1}{1+\exp(-x)}$ for the hidden units and $\frac{2}{1+\exp(-x)}-1$ for the output unit) was trained to predict the binary class labels $y^i \in \{-1,+1\}$ by minimizing the standard quadratic error function E_{sqr} on the training set. The IW method was compared with the "QuickProp" [10]. We chose "QuickProp" because in [23] and [24] the authors compared

- "Backprop" [6],
- "Backprop (batch mode)"[6],
- "Backprop (batch mode) + Eaton and Oliver" [9],
- "Backprop + Darken and Moody" [8],
- "J. Schmidhuber" [25],
- "R. Salomon" [22],
- "Chan and Fallside " [7],
- "Polak-Ribiere + line search" [15],
- "Conjugate gradient + line search" [18],
- "Silva and Almeida" [28],
- "SuperSAB" [29],
- "Delta-Bar-Delta" [14],
- "QuickProp" [10],
- "RPROP" [21], and
- "Cascade correlation" [11]

and concluded in [24]:

> "In terms of learning speed RPROP and Quickprop seems to be superior to all other training algorithms using fixed topologies."

TABLE I

Benchmark results for the toy data sets L, NL1, and NL2 shown in Fig. 2 and for a MLP (2-10-10-1) with sigmoid units. All weights were initialized with values drawn randomly and uniformly from $[-0.1, 0.1]$. A data point was correctly classified if the difference between the output value of the MLP and the target values ± 1 was less than 0.2. The columns show (left to right): learning method – QuickProp (QP) vs. importance weight (IW) method and data set in brackets, number of training examples N, values of C and p, "EP." number of training epochs needed until all data points are correctly classified (except for NL2, for which QuickProp was terminated after 1,000,000 epochs), number k of data points used for training ($\alpha_i \neq 0$ for at least one epoch, QuickProp requires all examples), number of misclassifications on the training set ("M"). The 300 training examples for NL1 are obtained by drawing additional 200 data points from the corresponding input distribution (5 Gaussians).

Method	N	parameters	EP.	k	M
IW (L)	100	$C=10^{-1}, p=10^{-2}$	4	3	0
IW (L)	100	$C=10^{-1}, p=10^{0}$	1	7	0
IW (NL1)	100	$C=10^{1}, p=10^{-1}$	5	63	0
IW (NL1)	100	$C=10^{1}, p=10^{-2}$	37	47	0
IW (NL1)	100	$C=10^{2}, p=10^{-2}$	46	29	0
IW (NL1)	100	$C=10^{3}, p=10^{-2}$	75	18	0
IW (NL1)	300	$C=10^{3}, p=10^{-2}$	86	34	0
QP (NL1)	100		95	100	0
IW (NL2)	100	$C=10^{3}, p=10^{-3}$	91	100	0
IW (NL2)	100	$C=10^{4}, p=10^{-3}$	63	100	0
QP (NL2)	100		10^6	100	11

The results are summarized in Table I. For data set L only 3 to 7 examples were selected by IW during training, all other data points were not considered ($\alpha_i = 0$) for learning the network's weights. For data set NL1 there is a trade-off between speed of convergence and the number of selected data points, depending on the values of the hyperparameters of IW. If only few examples are considered (18 out of 100) learning is slow (75 epochs), while learning becomes faster (5 epochs) if more data points are selected (63 out of 100). QuickProp needs more epochs than IW for convergence on NL1. For data set NL2 the importance weight method converged within the first 100 epochs whereas QuickProp did not find a solution

with zero misclassification error within 10^6 epochs.

2) Prediction of Protein Interactions: In this section we report benchmark results for five data sets describing protein-protein interactions for yeast proteins taken from the PepSPOT experiments of [16]. Each data set consists of the binding affinities between one out of five SH3 domains (Boi1, Boi2, Rvs167, Yfr024, Yhr016c). 672 peptides are selected by scanning with PatMatch the yeast proteome for the relaxed, 7 amino acids (AAs) long consensus pattern (R/K)xxPxxP (standard notation, "x" denotes "arbitrary amino acid"). The selected peptides of length 14 AAs were synthesized at high density on cellulose membranes by the SPOT technology. Then the membranes were probed by the corresponding SH3 domain fused to glutathion S-transferase (GST). Bound domains were detected by an anti-GST antibody and a secondary anti-immunoglobulin G (IgG) coupled to horseradish peroxidase (POD). Finally, spot intensity are given as real valued, positive Boehringer light units (BLU). For details of experiments and data sets see [16].

The BLU values were mapped to the interval $[-1, 1]$ by $\frac{2\ (a/a_0)^2}{1\ +\ (a/a_0)^2}\ -\ 1$ (see table II, 3rd column). However, the data sets are highly unbalanced with most of the transformed BLU values (tBLU) being close to zero and only approx. 7% being of order one (see Table II, 2nd column).

The machine learning task is to predict - for each SH3 domain - the tBLU values from the amino acid sequence of the 14 amino acid long peptides. The sequence was coded using a 1-out-of-20 binary code for every of the 11 variable amino acids of the peptide and a indicator variable for the presence of R vs. K at position 4. We used a multi-layer perceptron with 221 units in the input layer, one unit in the hidden, and one unit in the output layer - more complex networks led to inferior performance. Hidden and output units had sigmoid transfer functions as in previous experiment. The hidden unit effect is equivalent to a slope parameter of the output sigmoid in an architecture without a hidden layer. The MLP was trained with IW and with batch backpropagation (BP) using a regularized (weight decay for the input weights with Laplacian prior) squared error cost-function:

$$E_{sqr-reg}\left(\boldsymbol{x}^i\right)\ =\ \frac{1}{2}\left(y^i\ -\ f\left(\boldsymbol{x}^i;\boldsymbol{w}\right)\right)^2\ +\ \kappa\sum_l |w_l|\ . \quad (5)$$

Note, that IW only uses gradients, therefore it can be applied to regularized cost functions. Additionally, early stopping was used. The first 400 peptides were used for training, the following 100 peptides served as a validation set to select both the regularization parameter κ and the stopping time, and the remaining 172 peptides were used as a final test set. All weights were initialized with values drawn randomly and uniformly from $[-0.1, 0.1]$. The BP learning rate was 0.001, higher values led to instabilities. The IW parameters were $p = 0.01$ and $C = 0.1$.

The benchmark results are summarized in Table II. The IW method does not only require much less epochs for learning, but it is also faster in terms of CPU time. The increased computational costs for every learning step due to the solution of the optimization problem eq. (3) is over compensated by the much lower number of epochs. The error on the test set achieved by the IW method is always less than the error achieved by the BP procedure.

TABLE II

BENCHMARK RESULTS FOR THE FIVE PROTEIN INTERACTION DATA SETS. THE COLUMNS SHOW (LEFT TO RIGHT): NAME OF THE SH3 DOMAIN, NUMBER OF PEPTIDES WITH A TARGET VALUE $\geq$ -0.8, VALUE OF a_0, NUMBER k OF DATA POINTS USED FOR TRAINING (VALIDATION EXAMPLES AND THOSE TRAINING EXAMPLES $\boldsymbol{x}^i$ WITH $\alpha_i \neq 0$ FOR AT LEAST ONE EPOCH, BP REQUIRES ALL TRAINING EXAMPLES), VALUE OF THE REGULARIZATION PARAMETER κ, MSE ON THE TRAINING SET, MSE ON THE TEST SET, NUMBER OF EPOCHS (EP.), AND CPU TIME. THE VALUES FOR BP AND IW ARE GIVEN IN THE UPPER AND LOWER PART OF THE TABLE.

SH3	$\geq$	a_0 10^3	u	κ 10^{-5}	train 10^{-2}	test 10^{-2}	EP. 10^3	CPU time
batch backpropagation method								
Boi1	73	5	500	0.1	5.2	16.3	46	422
Rvs167	39	5	500	0.1	3.2	12.1	64	595
Boi2	21	1	500	0.1	2.6	5.3	67	622
Yfr024	32	20	500	0.1	2.6	6.5	100	920
Yhr016c	34	5	500	0.1	2.6	8.6	100	944
importance weight method								
Boi1	73	5	363	50	4.2	14.3	0.14	56
Rvs167	39	5	394	50	2.2	11.6	0.15	62
Boi2	21	1	401	100	1.4	5.1	0.14	57
Yfr024	32	20	500	1	0.8	5.7	0.28	128
Yhr016c	34	5	500	1	1.1	7.9	0.43	202

Table III shows an analysis of the binding affinities predicted by the multilayer perceptron which was trained using the IW method. Peptides with a high predicted binding affinity (higher than a given threshold) were selected from the test set, and their total number, the number of true and false positives, and the number of true negatives are given for different threshold values and for the five different data sets. The results are very good

Fig. 3 shows the sequence logos for the five different data sets. Every column corresponds to one position within the peptide of 14 amino acid length. The size of each letter (the abbreviation for the amino acid) corresponds to the strength of positive input weights of the trained MLP to the corresponding component of the 20 dimensional input vector at every location (except for positions 8 and 11, which were not part of the input, see description of MLP architecture). These results are in agreement with the findings in [16], e.g. with the strict consensus binding motif RxFPxPP of the Rvs167 SH3 domain and with the class 2 binding motif PxxPxR for Yfr024c and Ysc84 SH3 domains.

TABLE III

ANALYSIS OF BINDING AFFINITIES, PREDICTED BY THE IW METHOD FOR THE FIVE PROTEIN-PROTEIN INTERACTION DATA SETS. TABLE III SHOWS AN ANALYSIS OF THE MLP PREDICTIONS ON THE TEST SET W.R.T. THE CLASSIFICATION OF THE INPUTS INTO "BINDING" AND "NON-BINDING" PEPTIDES. THE COLUMNS SHOW (LEFT TO RIGHT): THE NAME OF THE DATA SET (SH3 DOMAIN), THE THRESHOLD USED FOR THE "BINDING" VS. "NON-BINDING" DECISION, THE NUMBER OF PREDICTED POSITIVES (PEPTIDES THAT BIND), THE NUMBER OF TRUE POSITIVES, THE NUMBER OF FALSE NEGATIVES, AND THE NUMBER FALSE POSITIVES. FIGURE 2 (RIGHT) SHOWS THE SEQUENCE LOGOS CONSTRUCTED FROM THE INPUT WEIGHTS OF THE TRAINED MLP. FOR DETAILS SEE TEXT.

SH3	threshold	P	TP	FN	FP
Boi1	-0.96	4	4	0	43
	-0.85	4	3	1	18
Rvs167	-0.97	7	7	0	36
	-0.85	7	4	3	5
Boi2	-0.98	2	2	0	16
	-0.85	2	1	1	3
Yfr024	-0.995	8	8	0	125
	-0.99	8	7	1	34
	-0.98	8	6	2	11
	-0.96	8	5	3	6
Yhr016c	-0.99	4	4	0	16
	-0.85	4	2	2	1

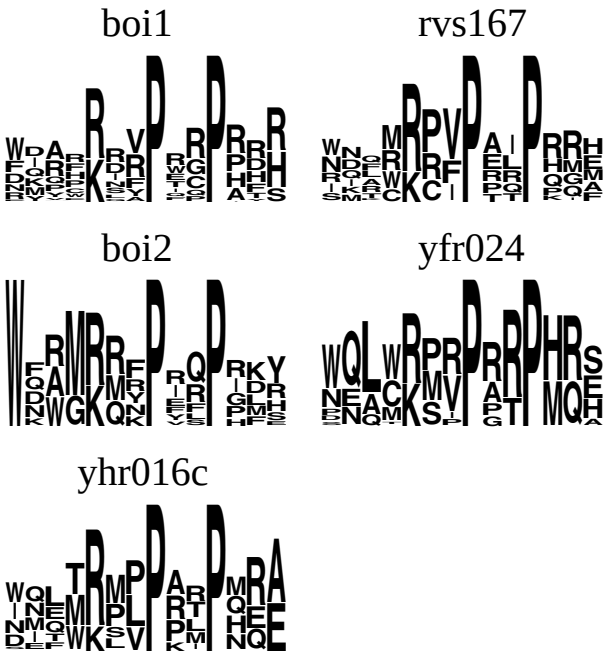

Fig. 3. Sequence logos of 14 amino acid peptide binding pattern for the five experiments.

B. Application to Recurrent Networks (LSTM)

1) Sequence Analysis – The Latching Benchmark: The latching experiment [5] is one of the best known benchmark for the ability of a new method to recognize long-term dependencies in sequences. The data set consists of sequences of real numbers which are drawn randomly and uniformly from the interval $[-0.1, 0.1]$, except for the first and the last element, which are either both 1 or -1. The task is to learn a predictor for the last element of the sequence, based on the other elements.

In [13] it was shown, that a "Long Short-Term Memory" (LSTM) recurrent network was able to successfully solve this problem for sequences up to a length of 1000. Here we generated 200 sequences – 100 for training and 100 for test – of length 1,000,002, i.e. the first element of the sequence must be stored over one million (!) steps in order to predict the value of the last element. An LSTM network with two memory cells of size 1, no hidden units, one input unit and one output unit (activation function $\frac{2}{1+\exp(-x)} - 1$). The LSTM network has standard memory cells as described in [13] and all non-input units have a bias weight which leads to 51 weights. The network was trained by the IW methods ($p = 0.01$, $C = 10000$) using the squared error cost function (E_{sqr}), where the first and second input gate had an initial bias of -10 and and -15. We stopped learning if the all training sequences were processed correctly, i.e. the absolute prediction error of the final element was below 0.2. In 10 trials IW stopped on average after 83 epochs whereafter all test examples were processed correctly. This is a new record for recurrent neural networks in the detection of long term dependencies.

2) Sequence Analysis – Protein Classification: In this section we report benchmark results for three data sets of labeled sequences of proteins. Protein sequences from the three classes IG_MHC (360 sequences), UPF (25 sequences), and MYELIN (18) were chosen from the Prosite database and complemented by a large number of randomly selected "negative" examples from unrelated families (IG_MHC-dataset: 1959 "negative" sequences, UPF-dataset: 1836 "negative" sequences, MYELIN-dataset: 1930 "negative" sequences). For each of the three data sets, the machine learning task was to correctly predict class membership based on the sequence of amino acids. By construction, the data sets were highly unbalanced and had an average sequence length of about 200. Long-term dependencies result from the fact that class relevant information may be located at the beginning of a sequence but prediction must be performed at sequence end. Further, long-term dependencies between groups of amino acids at distant location may be important features for a given class.

The classification task was solved using a recurrent LSTM network. The idea is to learn pattern recognizers for amino acid patterns indicative for a given class and use those at the input of a memory cell in order to store the occurrence of a pattern until the end of the sequence. The LSTM input gates, which serve as the pattern recognizers, receive their (only) input from a window of 20 adjacent amino acids (1-out-of-20 binary coding for every amino acid). The window is shifted over the sequence, and the network's prediction is evaluated after one full sweep. The initial bias of the input gates is set to -8.0. The LSTM architecture consisted of only one memory cell and no hidden units. The LSTM network was trained using gradient descent method (LSTM learning) from [13] as well as

with the new IW method ($p = 0.01$). The size of the training and validation sets were ("positives/all"): 338/2072 and 43/268 for IG_MHC, 22/1638 and 3/223 for UPF, and 15/1625 and 3/323 for MYELIN.

TABLE IV

BENCHMARK RESULTS FOR THE PROTEIN CLASSIFICATION DATA SETS. THE TABLE ON THE TOP SHOWS THE RESULTS FOR LSTM AND GRADIENT DESCENT (CF. [13]), THE TABLE ON THE BOTTOM THE IW RESULTS. COLUMNS SHOW (LEFT TO RIGHT): THE DATA SET, THE LEARNING PARAMETERS (LEARNING RATE FOR GRADIENT DESCENT AND C FOR IW), MSE ON THE TRAINING SET, BALANCED ERROR (MEAN OF FALSE POSITIVE AND FALSE NEGATIVE RATE ON THE TRAINING SET, MSE ON THE TEST SET, BALANCED ERROR ON THE TEST SET, NUMBER OF EPOCHS, AND CPU-TIME.

Prot. class	par.	train MSE	train BE	test MSE	test BE	ep.	CPU min
	gradient descent						
IG_MHC	0.1	0.050	33.8	0.047	4.3	3177	150
IG_MHC	0.2	0.050	33.8	0.048	4.3	3188	150
IG_MHC	0.5	0.051	33.8	0.048	4.3	3183	150
UPF	0.2	10e-6	0	10e-6	0	1360	32
MYELIN	0.2	10e-6	0	10e-6	0	440	10.1
	importance weight method						
IG_MHC	10	0.001	6.9	0.006	2.6	64	74
UPF	1	6x10e-4	0	2x10e-4	0	61	3.4
MYELIN	10	10e-6	0	10e-6	0	100	7.5

Table IV summarizes the results. The IW methods was always faster than gradient descent learning, in terms of epochs as well as in terms of CPU time. The validation mean squared error was better or equal for IG_MHC and MYELIN, and a bit worse for the UPF dataset. However, the average balanced classification error of the IW method (calculated as the average between the false positive and false negative rates) was always better or equal to error achieved by standard BP.

ACKNOWLEDGMENTS

We thank Martin Heusel and Rene Pfeifer for their help with the experiments. This work was funded in part by BMBF project no. 10025304. and by the DFG (SFB 618).

REFERENCES

[1] S.-I. Amari. Natural gradient works efficiently in learning. *Neural Computation*, 10(2):251–276, 1998.

[2] P. Baldi, G. Pollastri, C. A. Andersen, and S. Brunak. Matching protein beta-sheet partners by feedforward and recurrent neural networks. In *Proc. Int. Conf. Int. Systems for Molecular Biology 8*, pages 25–36, 2000.

[3] R. Battiti. First- and second-order methods for learning: between steepest descent and Newton's method. *Neural Computation*, 4:141–166, 1992.

[4] H. S. M. Beigi and C. J. Li. Learning algorithms for neural networks based on quasi-Newton with self-scaling. *Intelligent Control Systems*, 23:23–28, 1990.

[5] Y. Bengio, P. Simard, and P. Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE Trans. on Neur. Net.*, 5(2):157–166, 1994.

[6] C. M. Bishop. *Neural Networks for Pattern Recognition*. Clarendon Press, Oxford, 1995.

[7] L. W. Chan and F. Fallside. An adaptive training algorithm for backpropagation networks. *Computer Speech and Language*, 2:205–218, 1987.

[8] C. Darken and J. Moody. Note on learning rate schedules for stochastic optimization. In J. E. Moody, S. J. Hanson, and R. P. Lippmann, editors, *Advances in Neural Information Processing Systems 4*, pages 832–838. Morgan Kauffmann, 1992.

[9] H. A. C. Eaton and T. L. Oliver. Learning coefficient dependence on training set size. *Neural Networks*, 5:283–288, 1992.

[10] S. E. Fahlman. Faster-learning variations on back-propagation: an empirical study. In *Proc. of the 1988 Conn. Models Summer School*, pages 38–51, 1989.

[11] S. E. Fahlman and C. Lebiere. The cascade-correlation learning algorithm. In D. S. Touretzky, editor, *Advances in Neural Information Processing Systems 2*, pages 525–532. San Mateo, CA: Morgan Kaufmann, 1990.

[12] S. Hochreiter and K. Obermayer. Classification, regression, and feature selection on matrix data. Technical Report 2004/2, Technische Universität Berlin, Fakultät für Elektrotechnik und Informatik, 2004.

[13] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997.

[14] R. A. Jacobs. Increased rates of convergence through learning rate adaptation. *Neural Networks*, 1:295–307, 1988.

[15] A. H. Kramer and A. Sangiovanni-Vincentelli. Efficient parallel learning algorithms for neural networks. In D. S. Touretzky, editor, *Advances in Neural Information Processing Systems 1*, pages 40–48. San Mateo, CA: Morgan Kaufmann, 1989.

[16] C. Landgraf, S. Panni, L. Montecchi-Palazzi, L. Castagnoli, J. Schneider-Mergener, R. Volkmer-Engert, and G. Cesareni. Protein interaction networks by proteome peptide scanning. *PLoS Biol.*, 2(1):94–103, 2004.

[17] Y. LeCun, P. Simard, and B. Pearlmutter. Automatic learning rate maximization by on-line estimation of the Hessian's eigenvectors". In *Advances in Neural Information Processing Systems 5*, pages 156–163, 1993.

[18] J. Leonard and M. A. Kramer. Improvement of the backpropagation algorithm for training neural networks. *Computers Chemical Engineering*, 14(3):337–341, 1990.

[19] J. Platt. Fast training of support vector machines using sequential minimal optimization. In B. Schölkopf, C. J. C. Burges, and A. J. Smola, editors, *Advances in Kernel Methods — Support Vector Learning*, pages 185–208, 1999.

[20] N. Qian. On the momentum term in gradient descent learning algorithms. *Neural Networks*, 12(1):145–151, 1999.

[21] M. Riedmiller and H. Braun. A direct adaptive method for faster backpropagation learning: The RPROP algorithm. In *Proc. of the IEEE International Conference on Neural Networks*, pages 586–591, San Francisco, CA, 1993.

[22] R. Salomon. Improved convergence rate of back-propagation with dynamic adaption of the learning rate. In *Parallel Problem Solving From Nature (Dortmund, 1990)*, volume 496 of *Lecture Notes in Computer Science*, pages 269–273, 1991.

[23] W. Schiffmann, M. Joost, and R. Werner. Comparison of optimized backpropagation algorithms. In *Proc. of the European Symposium on Artificial Neural Networks, ESANN 93*, pages 97–104, 1993.

[24] W. Schiffmann, M. Joost, and R. Werner. Optimization of the backpropagation algorithm for training multilayer perceptrons. Technical report, Institute of Physics, University of Koblenz, Koblenz, Germany, 1994.

[25] J. Schmidhuber. Accelerated learning in back-propagation nets. In R. Pfeifer, Z. Schreter, Z. Fogelman, and L. Steels, editors, *Connectionism in Perspective*, pages 429–438. Amsterdam: Elsevier, North-Holland, 1989.

[26] B. Schölkopf and A. J. Smola. *Learning with Kernels — Support Vector Machines, Regularization, Optimization, and Beyond*. MIT Press, 2002.

[27] R. Setiono and L. Hui. Use of a quasi-Newton method in a feedforward neural network construction algorithm. *IEEE Trans. Neural Net.*, 6(1):273–277, 1995.

[28] F. M. Silva and L. B. Almeida. Speeding up back-propagation. In R. Eckmiller, editor, *Advanced Neural Computers*, pages 151–158, Amsterdam, 1990. Elsevier.

[29] T. Tollenaere. SuperSAB: fast adaptive back propagation with good scaling properties. *Neural Networks*, 3:561–573, 1990.

[30] G. M. Weiss and F. Provost. Learning when training data are costly: The effect of class distribution on tree induction. *J. Artifi. Int. Res.*, 19:315–354, 2003.

[31] T. Zhang and V. S. Iyengar. Recommender systems using linear classifiers. *Journal of Machine Learning Research*, 2:313–334, 2002.