Neurocomputing

Neurocomputing
Vol. 71, Issue 13-15
August 2008

Variable Step Search Algorithm for Feedforward Networks

Mirosław Kordos and Włodzisław Duch

Abstract

A new class of search-based training algorithms for feedforward networks is introduced. These algorithms do not calculate analytical gradients and they do not use stochastic or genetic search techniques. The forward step is performed to calculate error in response to localized weight changes using systematic search techniques. One of the simplest variants of this type of algorithms, the Variable Step Search (VSS) algorithm, is studied in details. The VSS search procedure changes one network parameter at a time and thus does not impose any restrictions on the network structure or the type of transfer functions. Rough approximation to the gradient direction and the determination of the optimal step along this direction to find the minimum of cost function are performed simultaneously. Modifying the value of a single weight changes the signals only in a small fragment of the network, allowing for efficient calculation of contributions to errors. Several heuristics are discussed to increase the efficiency of VSS algorithm. Tests on benchmark data show that VSS performs not worse and sometimes even significantly better than such renown algorithms as the Levenberg-Marquardt or the scaled conjugate gradient.

Keywords: neural networks, Multi-Layer Perceptrons, neural training algorithms, search techniques, optimization

1. Introduction

Multilayer perceptrons (MLP) are usually trained using either analytical gradient-based algorithms with error backpropagation or (rarely) global optimization methods. Some of the most popular methods from the first group include standard backpropagation (BP) [1], various versions of RPROP [2]-[4], Quickprop [5], Levenberg-Marquardt (LM) [6][7] and the scaled conjugate gradient (SCG) [8] algorithms. The second group involves genetic algorithms [9]-[11], simulated annealing [12] and its variants such as Alopex [13], particle swarm optimization [14], tabu search [15] and several other algorithms [16],[17].

The training time of local gradient algorithms is usually significantly shorter than that of global methods. Sophisticated gradient techniques based on classical numerical analysis methods have been developed [18] and implemented in a large number of software packages. In theory global optimization methods should be able to find a better solution for complex problems, but in practice despite a lot of efforts (especially using the evolutionary computing algorithms) empirical results showing significant advantages of global optimization methods were difficult to obtain. Perhaps the benchmark problems analyzed were too simple. Applications to more difficult problems in bioinformatics show some advantages of genetically optimized neural networks [9]-[11]. In large parameter spaces the phenomenon of over-searching [19] may increase the chance that global optimization methods will find optimal solutions that for the test data will give worse results than solutions accessible by gradient methods. Thus extensive search may paradoxically make the problem of model selection quite difficult.

In this paper a new class of neural training algorithms based on local search techniques [20] is explored. The analysis and the algorithms described here can be used for feedforward networks of arbitrary structure, with arbitrary transfer functions (this is in fact one of the greatest advantages of this approach because changing transfer functions does not require development of new formulas or significant changes of the program). However, to be concise we shall focus only on the standard 3-layer MLP networks trained for data classification with logistic sigmoid transfer function $y(u)$ with the unit slope (β=1):

$$y(u) = \frac{1}{1+\exp(-\beta u)} \tag{1}$$

A “staircase” approximation to logistic functions will also be mentioned. Search-based optimization methods include stochastic methods [21], evolutionary algorithms and local systematic search techniques. So far algorithms based on systematic search have been largely ignored, with only a few papers mentioning their use in logical rule extraction from neural networks [22][23]. Analytical gradients are calculated assuming infinitesimal changes, but in computer implementations of the training algorithms changes are finite

and fast learning requires large steps, therefore numerical effects may degrade performance of analytical gradient algorithms. Localized perturbations, restricted to one or two weights are sufficient to provide numerical approximation to gradient direction. Inspection of the real learning processes (using also visualization techniques) led us to several interesting conclusions [21][25], briefly summarized in the following sections.

Remarks on gradients, search directions and search procedures are presented in section two. Lessons learned from experiments with search-based neural training algorithms were used to implement a new training method, called the Variable Step Search (VSS) algorithm. It uses a numerical rather than analytical approach in order to find optimal directions and step sizes in an iterative process. Several heuristics designed to improve performance of this algorithm are described in section three. Visualization of the VSS learning processes is presented in section four. Experimental results on several datasets, presented in section five in terms of convergence properties, accuracy and speed of calculations, are very promising. In many aspects VSS tends to perform very well, comparing favorably to the best neural training algorithms, such as the scaled conjugate gradient (SCG), the Levenberg-Marquardt and the Rprop algorithms. It is thus clear that VSS and other algorithms based on systematic search are worth investigating. The final section contains conclusions and remarks on the future work.

2. Gradients and search directions

In this section the background and the motivation for introduction of the VSS algorithm is presented. Analysis and comparison of analytical and numerical gradients is made and several remarks on the line search techniques for MLP training are made. Properties of the gradient directions calculated by backpropagation-based algorithms are discussed, and heuristics for finding optimal direction for the next search step are introduced.

2.1. Numerical and Analytical Gradient Directions

The analytical gradient-based algorithms use an error backpropagation mechanism to assess the gradient component in each hidden weight direction. Assuming a single output Y, feedforward mapping $M(\mathbf{X};\mathbf{W})$ of the input vector $\mathbf{X}$ to Y, parameterized by the weight vector $\mathbf{W}$ and a standard quadratic error function [26] the formula for the gradient of the output weights w_k is:

$$\frac{\partial E(\mathbf{W})}{\partial w_k} = \left(M(\mathbf{X};\mathbf{W}) - Y\right)\frac{\partial M(\mathbf{X};\mathbf{W})}{\partial w_k} \tag{2}$$

The derivative of the mapping $M(\mathbf{X};\mathbf{W})$ is expressed using derivatives of the transfer functions and the errors made by the network. These errors are propagated to the input layer to calculate gradients for the remaining weights.

In the numerical gradient network training [24][25] a single weight w_k is subject to a small perturbation dw (positive or negative) and changes of the network error in response to this perturbation are used as a gradient component in w_k direction:

$$\frac{\partial E(\mathbf{W})}{\partial w_k} = \frac{E([w_1,\ldots,w_k+dw,\ldots,w_n]) - E(\mathbf{W})}{dw} \tag{3}$$

Thus the numerical gradient method roughly coincides with finite difference method of gradient approximation [27]. Accuracy of such calculation depends on the curvature of the error surface. The numerical gradient direction depends on dw, but this dependence is usually not too strong for dw in the range 0.002–0.2 with unit sigmoid slope (β=1).

The main difference between gradients computed using analytical and numerical formulas is seen for small gradient values (frequently associated with the hidden weight components at the beginning of the training). Small gradients tend to be smaller in the analytical gradient calculations, while the large values tend to be larger (see Fig. 1). This tendency is stronger for larger networks with more complex data. Numerical gradient calculates the descent directions taking into account the error values in two points thus examining a broader range of the error surface than analytical gradient, so it can "predict" more precisely the error value in a spot located at some distance from the current point.

In the dependence shown in Fig. 1, the analytical gradient was calculated using Eq. 2, and the numerical gradient using Eq. 3. The relation between analytical and numerical gradient components is more important than the absolute value of the components because the absolute value is always multiplied by a certain step size during network training. Therefore the product of the step size and the component value is the most important quantity. The gradient components shown in Fig. 1 are rescaled so that the lengths of the analytical and numerical gradient vector are the same.

Interesting empirical observations on numerical and analytical gradients have been made during training on several datasets. A finite step along numerical gradient direction leads in most cases to faster decrease of the error than the same step along analytical gradient direction. The difference is even stronger if minimization along the direction determined by backpropagation and numerical gradient is done.

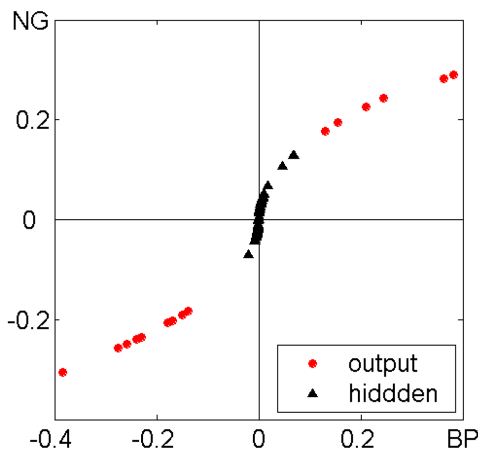

Fig. 1. A comparison of analytically (BP) and numerically (NG, with *dw*=0.02) determined gradient components in all the weight directions for the thyroid dataset at the first training epoch using logistics sigmoid transfer functions with unit slope (β=1). (21 inputs, 4 hidden, 3 outputs, 21-4-3 network).

Although the numerical gradient is still not the optimal direction of the learning trajectory, it tends to be much closer to it than the analytical gradient (see the next subsection for detailed discussion).

Backpropagation frequently gets stuck in apparent local minima or plateaus without reaching low values of the error [29]. The importance of local minima has been controversial for a long time [28]. Contrary to the common belief, local minima may not be a real problem in neural training; ill-conditioning and saddle points have much more direct effect that has important influence on the performance of training algorithms [30]. Backpropagation training is frequently stuck only in the apparent local minima, and in many cases switching to another training algorithm (for example a numerical gradient) leads to the further decrease of the error and to the final convergence. Analytical gradient algorithms get stuck because gradients on flat surfaces, flat saddle points calculated in analytical way may become very small [31][32], while a finite step numerical gradient is larger and may lead to the lower areas on the error surface. Sometimes the trajectory may be trapped in a highly situated ravine on the error surface and then also the numerical gradient method is unable to converge. Visualization of the error surface (see sec. IV) shows frequently such situations, but the local minima in form of "craters" are never observed. In summary, there are good reasons to use numerical, instead of analytical, gradients.

2.2. Gradient Direction and the Optimal Next Step Direction

Gradient-based training methods make initially rapid progress, slowing significantly near the end of the training. Fig. 5 shows an MLP error surface projected on the first two PCA directions in the weight space (two directions capture typically over 95% of all variance). The error surface becomes almost flat in the areas that are located further from the starting point (initialization with small weights is assumed), and therefore reached by the learning trajectory at the final stage of the training. The analysis of such surfaces and one-dimensional crossections along single weight directions at the beginning and near the end of the training shows (Fig. 2) some interesting properties of MLP error surfaces. The hidden weights have in general rather low gradients at the beginning and at the end of the training, although their values change a lot; the output weights grow faster and have large gradients around minima, with large flat highly situated plateaus far from optimal values. This suggests deep ravines in the error function landscape.

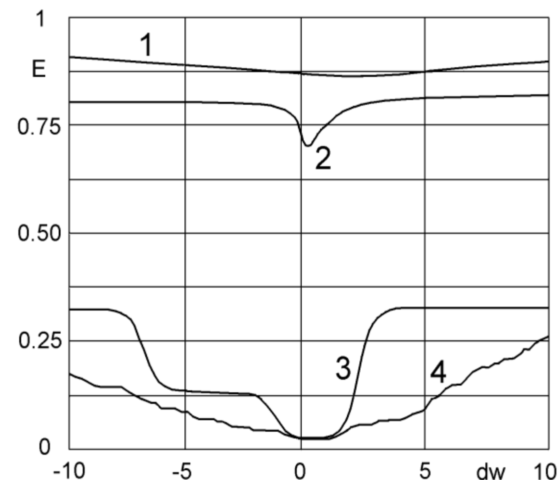

Fig. 2. Typical error surface crossections in the direction of: (1) hidden weight at the beginning of the training; (2) output weight at the beginning of the training, (3) output weight at the end of the training, (4) hidden weight at the end of the training.

In any case gradient direction is not the optimal step direction. The RPROP algorithm that takes into account only the sign of a derivative instead of the gradient performs usually not worse than BP, and frequently even better [2],[26]. Moreover, there exists a certain similarity between Rprop and VSS: both use individual update steps for each weight.

2.3. Gradients and Optimal Directions

It is instructive to assess the statistical relation between the size of the gradient component $dE(w)$ in the direction of weight w and the distance mw from the point $\mathbf{W}$ to the minimum of the error function in the direction of weight w (see Fig. 3). The error surface sections in a particular weight direction may differ significantly, although the curves shown in Fig. 2 are rather typical for most weights. We use numerical gradient in this section. The disproportion between

the analytical gradient component and the optimal step size is even bigger than for numerical gradient, because in this case it is the product of relations shown in Fig. 1 and in Fig. 3.

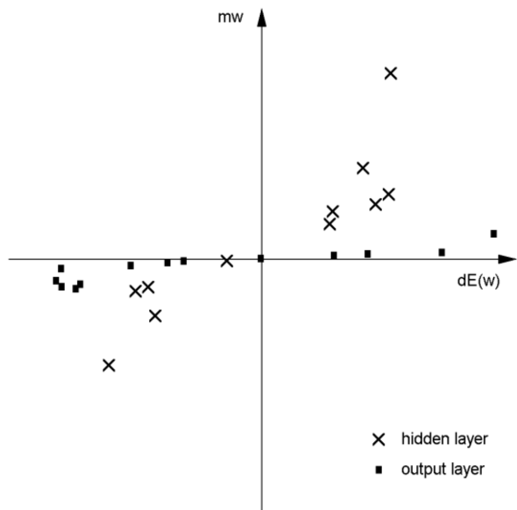

Fig. 3. Dependence between the numerical gradient component $dE(w)$ and the distance *mw* from the actual point to the minimum error in a given weight direction in the first training epoch (Iris 4-4-3 network).

Initially the minima for the output weights are located near the current point **W**, but for the hidden weights they are on average much further (Fig. 3). The error surface landscape changes as the training progresses. However, the changes are visible mostly in the hidden weight directions. Thus, at the end of the training the relations shown in Fig. 3 will look very similar for both the hidden and output weights (as in Fig. 2, lower curves).

Heuristic approximation of the relation between $dE(w)$ and the $m(w)$ distance in the successive training epochs may be used to speed up search of the minimum. An example of such approximation for logistics sigmoid transfer function is given by the equation below.

$$m(w) = \mathrm{sign}\left(dE(w)\right)\left(1 + a \cdot \exp\left(-bt\right)\right) dE_1^c$$
$$dE_1 = \min\left(\left|dE(w)\right|, \min\left(\left|dE_{\max}\right|, 5\left|dE_m\right|\right)\right) \quad (4)$$

(1)

The first factor takes care of the sign of the gradient. The second factor (where t denotes the epoch number) expresses the fact that during training the error surface sections around the actual learning trajectory in the direction of hidden weights are asymptotically getting more and more similar to the sections in the direction of output weights (Fig. 2). For that reason a=0 for output weights, while for hidden weights this factor is expressed by an exponential function, which asymptotically approaches 1. The third factor (dE_1^C) approximates the dependence of $dE(w)$ on the distance $m(w)$ in a given epoch for a given layer. Parameters dE_1, a and b are determined fitting the function in Eq. 4 using the least mean square method on the data points (network weight values after each epoch) obtained while training the networks using the datasets described in section V with the numerical gradient method. dE_{max} is the greatest and dE_m is the mean of the error changes while changing the weight values by dw to determine the descent direction. The power $c \in (0,1)$ is a constant usually fixed at c =0.5, a=0 for the output layer, $a \in [10,20]$ range for the hidden layer, and b=[0.10,0.20] range.

Our aim at this point is to illustrate the situation rather than to find the best approximation, thus the purpose of Eq. 4 is to demonstrate that using the statistics from several network trainings such approximation may be defined, although it may not be an optimal approximation. Nevertheless, in computational experiments described below the use of Eq. 4 to calculate distance to the expected minimum contributed to an average reduction of the number of the training epochs required for convergence by 30-60% with both numerical gradient and with analytical gradients in standard backpropagation procedure.

The VSS algorithm, introduced in the next section, solves the problem of finding the trajectory direction in a different way, although still based on conclusions of the reasoning presented here.

2.4. In-Place and Progressive Search

The numerical gradient training [25] was based on the "in-place search": all weight changes were examined relatively to the current set of weights (point on the error surface), and then a single step was made searching for a minimum along the gradient direction. Thus numerical gradient simply replaced the analytical gradient in the backpropagation procedure.

The simplest search for the minimum of a function is based on progressive line search. Minimum is found separately for one parameter (using any line search method), and the process is repeated starting from the new point for the remaining parameters, as used in algorithms that search along the coordinate directions [33]. Parameters may be randomly reordered before each iteration. Although more sophisticated ways to choose directions may be introduced, for example using the conjugate gradient directions [33], it is worth trying the simplest approach first (i.e. moving along the directions of the coordinate axes). In fact moving along the individual parameter directions is also done in the first iteration of the Powell's quadratically convergent method [33], an iteration that usually leads to the largest reduction of the error. The search method used in the VSS algorithm is based on repetitive application of this first iteration.

VSS is a generalization of the simple search method that adds or subtracts from each parameter a fixed dw value, accepting only those changes that lead to the decrease of

errors. To avoid local minima stochastic algorithms, such as simulated annealing [12], Alopex [13], and several other global optimisation algorithms, accept (using specific probability distribution) changes that lead to an increase of the error. The VSS algorithm does not use this approach, relaying on the method of exploring the error surface that allows for effective MLP training, as long as the next point is within the same ravine of the error surface (Fig.5). Therefore in the "variable step search" (VSS) training algorithm the step size in each direction is determined by the line search.

Progressive search method updates the weight vector **W** immediately after the minimum along w_k direction is found, thus making as many steps (micro-iterations) in orthogonal directions during one iteration (training epoch) as the number of weights N_w. After each micro-iteration the weight vector **W** is changed, and thus also the error landscape and the value of the function $E(\mathbf{W})$ used for the next step. After the whole epoch the error function will undergo N_w modifications, which is in contrast to the standard backpropagation and other analytical approaches, where changes to all **W** components are made under the assumption of using the same $E(\mathbf{W})$.

The basic VSS algorithm is very simple and is outlined in the following pseudo code:

for i=1 to NumberOfEpochs **do**
  for j=1 to NumberOfWeights **do**
    find dw_j that minimizes $E(i,w_j+dw_j)$;
    $w_j \leftarrow w_j+dw_j$;
  end
  if $E < E_{min}$
    break;
  end
end

E_{min} is the error value at which the training stops; selecting the stopping criterion is not specific to VSS and can be done in the same way as for other MLP training algorithms. The method of calculating the error value E is shown in Fig. 4.

Any line search minimization method can be used to find the optimal dw, and the mean-square error (MSE) or any other error measure [26][31] may be used as optimization criterion. However, to increase the computational efficiency of VSS algorithm special methods to compute dw and $E(e,w+dw)$ are proposed below.

3. Reduction of Computational Cost

There are many general methods that reduce computational costs of neural training, such as weight pruning [26], use of support vectors for neural training [34], statistical sampling for large training sets, etc. Because these methods can be used with almost any neural training algorithm they will not be discussed here. Instead, three methods specific to the VSS algorithm are considered: signal table for organizing updates of error calculations, non-differentiable transfer functions that may be computed faster than continuous sigmoidal functions and some heuristics to speed up the line search for a minimum in a given weight direction.

3.1. Signal Table

Because only one weight is changed at a time the input signals do not need to be propagated through the entire network to calculate the error. Propagation through the fragment of the network in which the signals may change as a result of the weight update is sufficient. The remaining signals incoming to all neurons of hidden and output layers are remembered for each training vector in an array called the "signal table". After a single weight is changed only the appropriate entries in the signal table are updated. The MSE error of each output neuron is also remembered and do not need to be recalculated again if a weight of another output neuron is changed.

At the beginning of the training the signals are propagated through the entire network (this is done only one time), thus filling in the signal table entries. The use of the signal table significantly shortens training time enabling effective training of larger networks. Table 1 contains the formulas for the number of arithmetical operations with and without the signal table. The formulas are based on the analysis of the signal flow. For example in the first formula, $N_o(N_h+1)$ is the total number of weights in the output layer, $N_h(N_i+1)$ in the hidden layer and (N_o+N_h) is the total number of activation functions in the network. Thus, calculating the network error after every single weight change the activation function would have to be calculated that many times.

The dimension of the signal table is $N_V(N_o+N_h)$, where N_V is the number of vectors in the training set and N_h and N_o are the numbers of hidden and output neurons, respectively. For example, for a network with 30 neurons and 10,000 training vectors, storing variables in 8 bytes (double type) the signal table needs only 2.3 MB of memory, that is two or more orders of magnitude less than the memory requirements for the LM algorithm, and also less than the requirements of SCG algorithm (see details in section five).

Table 1. The number of operations with and without the signal table required to calculate numerical gradient direction (for one training vector). N_i, N_h, N_o – number of input, hidden and output neurons.

operation type	number of operations without signal table	number of operations with signal table
calculating sigmoid value (neuron outputs)	$[N_o(N_h+1) + N_h(N_i+1)] (N_o+N_h)$	$N_o(N_h+1) + N_h(N_i+1)(1+N_o)$
adding incoming signals multiplied by weight values (neuron activations)	$[N_h(N_i+1) + N_o(N_h+1)]^2$	$2[N_o(N_h+1)+ N_h(N_i+1)(1+N_o)]$

3.2. Staircase Transfer Functions

Calculation of the value of sigmoidal transfer functions is quite time consuming; in our experiments it took over 8 times longer than a single multiplication (in Borland Delphi implementation on the Athlon XP processor). Due to the finite precision of numerical calculations in computer implementation the sigmoidal transfer functions are in fact non-differentiable staircase functions with a very large number of small steps. Because the VSS algorithm does not rely on analytical gradient the transfer functions do not have to be differentiable and an array with approximated values implementing a staircase transfer function with lower precision can be used, reducing the training times by more than half without compromising accuracy. At least 20 equally spaced values of sigmoidal function have been used, approximating the sigmoid with accuracy of 2-3 significant digits.

The signal table can reduce the number of operations required to calculate the weighted activation *u* for a large network by several orders of magnitude, updating the activations *u* in the epoch *i* for a single weight change w_k as:

$$u_i = u_{i-1} + x(w_{k,i} - w_{k,i-1}) \tag{5}$$

The number of operations required to calculate single neuron output *y*(*u*) is reduced on average by the signal table by less than one order of magnitude. With signal table the staircase transfer functions additionally shortens the training time up to several times. On the other hand without the signal table the gain due to the staircase approximation of sigmoidal functions is quite small, because the calculation time is dominated by multiplications that enter activations *u*.

3.3. Line Search Heuristics

The search algorithm should take advantage of the MLP error surface properties. The steepness of the error surface in different directions varies by orders of magnitude, and the ravines in which the MLP learning trajectories lay are usually curved, slowly changing their directions [35],[36],[37],[38]. Therefore one can expect that an optimal change of weight value *dw* for the same weight in two successive training cycles will not differ much, while *dw* for different weights in the same training cycle may have values that differ on orders of magnitude.

In each training cycle *i* the first guess of *dw*(*w,i*) for a given weight *w* might be the value *dw*(*w,i*–1) of the weight change in the previous training cycle. However, detailed empirical analysis of our implementation of the line search leads to the conclusion that for most cases convergence is faster when smaller $dw(w,i) \approx 0.35dw(w,i-1)$ are used, in spite of the fact that statistically the ratio $dw(w,i)/dw(w,i-1)$ is close to 1.

Fig. 4 shows a diagram for determining the change *dw* of a single weight *w* in one training cycle *i*. OE (Old Error) is the MSE error (or another error measure) before the weight change is applied, and NE (New Error) is the error after the weight change is applied. Parameters max_n (maximum number of iterations), max_*w* (maximum allowed absolute value of the weight) and max_*d* (maximum allowed change of a weight in one training cycle) are introduced to prevent excessive growth of the weights. These parameters are optional and can have very large values or even be set to infinity. Experimentally determined optimal values of other parameters used in Fig. 4 are in the following ranges: $c1\in[0.3,0.4]$, $c2\in[2,3]$, $c3\in[0.1,0.3]$. However, the algorithm is not very sensitive to the values of these parameters, therefore they were set to their middle values and never changed in the experiments reported below. It should also be stressed that the results of VSS algorithm do not depend on these parameters; they may only influence the speed of convergence.

Before the training starts the weights are initialized with random values from the (−1,+1) interval. In the first training cycle $d=d0\in[0.2,0.3]$. Since $dw(w,0)=0$, for each weight *w* in the first training cycle the first guess $dw(w,1)=d0$ is taken. Because close to the starting (initialization) point the ravine leading to a minimum on the error surface is rather narrow, *d*0 must be sufficiently small to avoid overshooting and to keep the trajectory within the ravine.

Another heuristics has been derived from the observation that calculation of the minima along each weight direction to a high precision (e.g. by repeated parabolic interpolations where the curvature is not convex) increases the number of the training cycles, quite opposite to the expectations. Therefore only rough estimation of the step size *dw* in each direction is made. On average determining a single weight value in one training cycle the error in the line search algorithm needs to be calculated only about 3 times. (If the error does not change at the first attempt the weight value is kept unchanged for this iteration.) It is possible to increase *c*1 and *c*2 parameters so that the error will be calculated on average only twice, but this increases the number of training epochs and therefore does not reduce the total computational cost of the training.

In Fig. 4 block 3 deals with weights that did not change in the previous training cycle. This usually means that more precise weight tuning is needed, therefore a smaller value $d=d1\cdot\text{sign}(w)$ is added to that weight, preserving the direction but changing its value in the next training cycle.

For that reason *d*1 is multiplied by sign(*w*) to minimize the number of operations. The error value *NE* is calculated in blocks 4, 6 and 9. The functionality of block 10 is analogical to the momentum term used in backpropagation. If $c3\cdot(VE-OE)>NE-OE$ then the point is accepted, although the error in the previous point could have been a bit lower, since it is likely to bring gain in the next training cycle. *VE* is

the error recorded one step before *OE*, that is *NE*=error(n), *OE*=error(n−1), and *VE*=error(n−2).

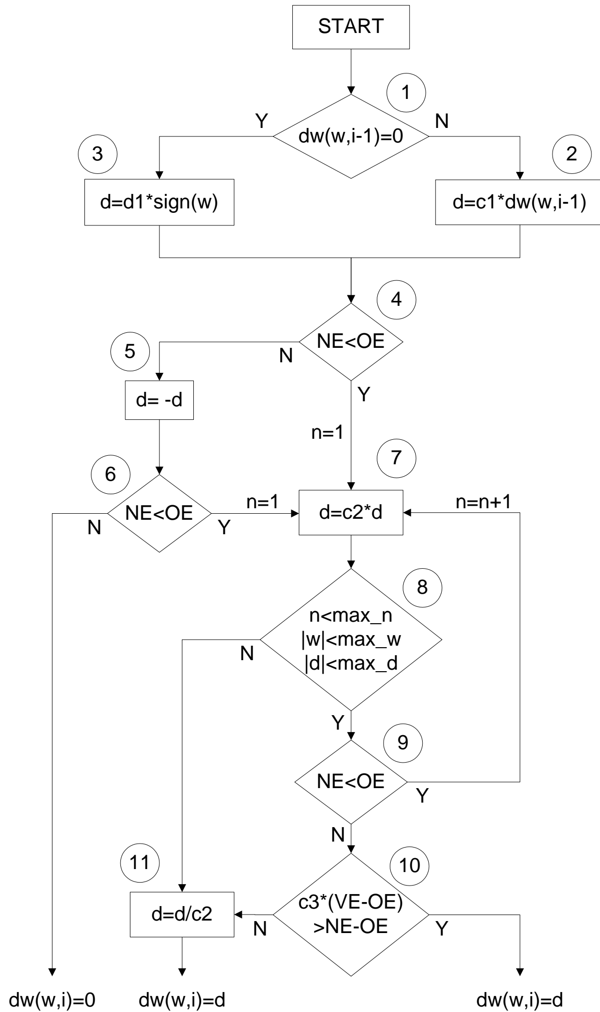

Fig. 4. Sketch of the Variable Step Search algorithm determining a single weight value in one training cycle.

Many experiments aimed at estimation of the optimal weight change sequence were performed, but various sequences did not have significant influence on the training efficiency. Therefore the weights are changed either in a random order or one after another in a systematic way, first all weights from the hidden layer, and than all weights from the output layer, or vice versa. If the change of a given weight does not significantly reduce the error for two iterations the weight is frozen, and if the weight is quite small it may be pruned.

The diagram shown in Fig. 4 presents the VSS algorithm incorporating the best heuristics found so far. It should be stressed that implementation of these heuristics is not necessary for the algorithm to work, but they are useful to increase its efficiency. Backpropagation-based algorithms also use a number of heuristics for the same purpose [26],[31]. Although the diagram in Fig. 4 seems to be complicated because several conditions are checked to incorporate various heuristics in fact implementation is quite simple as there is no need to program complex formulas with matrices and derivatives, as is the case for backpropagation-based methods. The VSS algorithm applied to MLP training proved to be quite stable, on most datasets leading to convergence in a very few training cycles.

4. VSS Learning Progress

Principal Component Analysis (PCA) can be used to reduce the weight space dimensionality for purpose of learning trajectory and the error surface visualization [24],[35]-[37]. Weight vectors $\mathbf{W}(t)$ at the starting point t=0, and after each training epoch $t=1..t_{\max}$, are collected in the weight matrix $\mathbf{W}=[\mathbf{W}(0), \mathbf{W}(1),.. \mathbf{W}(t_{\max})]$ with n rows and $t_{\max}+1$ columns. To determine principal components Singular Value Decomposition (SVD) is performed on the weight covariance matrix [33]. Each entry in the weight covariance matrix is calculated as:

$$c_{ij} = \frac{1}{t_{\max}} \sum_{t=0}^{t_{\max}} \left(W_i(t) - \overline{W}_i \right) \left(W_j(t) - \overline{W}_j \right) \quad (6)$$

where $\overline{W}_i$ is the i-th weight mean over all $t_{\max}+1$ epochs. A subset of the training epochs may be used to focus on some part of the learning trajectory.

For each point (c_1, c_2) in the PCA weight space $\mathbf{W}(c_1, c_2) = c_1\mathbf{V}_1 + c_2\mathbf{V}_2 + \mathbf{W}_0$ is defined, where $\mathbf{W}_0$ may be selected as one of the points on the learning trajectory (for example the starting point) and $\mathbf{V}_1$ and $\mathbf{V}_2$ are the unit vectors in the first and second principal component directions. The error surface plot (Fig. 5) shows the relative error $E_r(\mathbf{W}) = E(\mathbf{W})/N_V N_C$ on the vertical axis, and distances (c_1, c_2) in $\mathbf{V}_1$ and $\mathbf{V}_2$ directions on the horizontal axes. N_V is the number of vectors and N_C is the number of classes in the training set. For all error functions based on the Minkovsky's metric $||\cdot||$ when the output layer transfer function is bounded by 0 and 1 the error values are bounded from above by $N_V N_C$. Thus, the relative error is bounded by 1. The mean square error (MSE) is the most frequently used error measure, but replacing it with some other error measure in the VSS algorithm is quite trivial.

Typically the first principal component captures about 90% of the variance and the first two components contain together more than 95% of the total variance, therefore the plots reflect learning trajectory properties quite well. Although restoration of the error surface from only two PCA components is not ideal, to a significant degree projection of the learning trajectories tend to adhere to this surface. The beginning of a trajectory lies often over the error surface projection and its end under (the error surface projections are often flatter than original error surface on which the trajectory lies). The trajectories in n-dimensional weight space are bent, and their mean direction corresponds to the

direction of the error surface ravine in the PCA projection. Nevertheless visualization of the error surfaces and trajectories helps to understand the learning dynamics of neural algorithms [37].

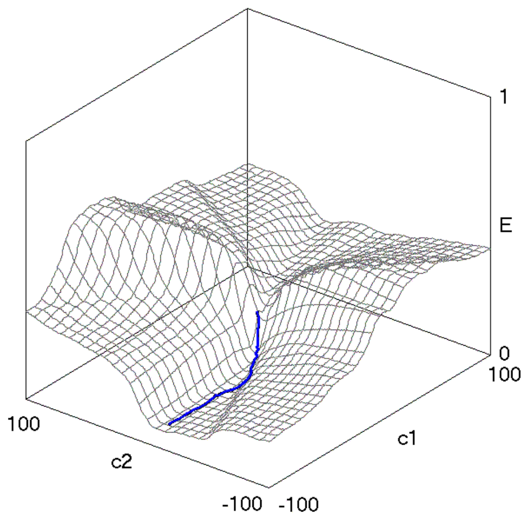

Fig. 5. Error surface and the learning trajectory of Iris (4-4-3) trained with VSS algorithm.

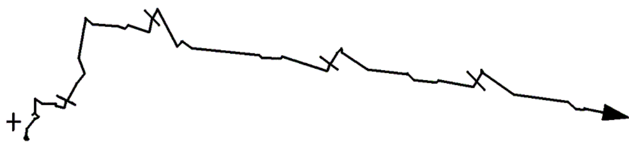

Fig. 6a. Projection of the Iris (4-4-3) learning trajectory trained with VSS in the first and second PCA direction. The cross shows the zero point in the weight space, and short bars separate the training epochs.

Fig. 6b. Projection of the Iris (4-4-3) learning trajectory trained with VSS in the third and fourth PCA direction. The cross shows the zero point in the weight space, and little bars separate the training epochs.

Fig. 6a and 6b present the directions of the weight changes in the first two PCA components during the VSS training. In each iteration correct direction of the error function's ravine which leads the trajectory towards minimum is quickly found and maintained. This figure is based on weights that have been updated in a systematic way, starting from the hidden weights, and ending with the output weights. The directions change sharply in the middle of iteration, when one of the hidden weight values is changed by a large amount, and then near the end of the epoch, when output weights are changed. Trajectories displayed in directions corresponding to higher PCA components seem to be quite chaotic (Fig. 6b) and do not carry much information.

4.1. Network Error

In contrast to the typical training algorithms each epoch in the VSS algorithm consists of N_w micro iterations. The number of epochs needed for convergence is quite small, for simple data it can be as low as 2 or 3. Fig. 7 shows the accuracy A, MSE error E and the total weight norm growth $\boldsymbol{W}$ (without using any methods of weight growth reduction) during the training of an MLP network with 4-4-3 structure on the Iris data.

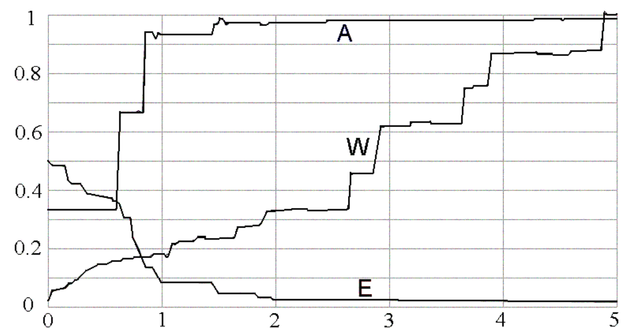

Fig. 7. MSE error (E), classification accuracy on the training set (A) and normalized weight ||W(i)||/||W(5)|| vector length (W) during the first 5 training cycles for the Iris (4-4-3) network

The error reaches minimum value already after two epochs, while the accuracy is already at the maximum. In the subsequent iterations most contribution to reduction of error comes from growing quickly weights, in effect making the sigmoidal functions steeper, although the direction of the weight vector is changing very little. The error minima are frequently in infinity (infinite growth of output layer weights). When the norm $||\boldsymbol{W}||=1$ is imposed on the network parameters, or a regularization term is added to the error function, the minima are moved from infinity to a point at the finite distance from the starting point.

The discussion and illustrations of error surface of network trained with more complex data sets with different error functions can be found in [24] and [37]. However, the general conclusions drawn from the network training in the Iris dataset can be extended to those cases.

4.2. Weight Values

Fig. 8 and 9 present changes of the hidden layer weights trained by the VSS and the LM algorithms. Although training

the network with VSS beyond the 4-th epoch does improve classification the training is continued here to show how the weights change in this process. In VSS these weights change very rapidly in the initial phase of the training and quickly reach their optimal values. In LM (and in other backpropagation-based algorithms) changes are slower and continue for larger number of epochs. In the second-order algorithms (such as LM) the hidden layer weights grow faster than in the first order ones, but because the step size in a given weight direction is approximately proportional to the ratio of the first to the second derivative, the hidden layer weights tend still to be underestimated. VSS on the other hand does not estimate weight changes but directly changes each particular weight to a value that approximately corresponds to the error minimum in this weight direction. The output layer weights change in a similar manner in both algorithms; faster than the hidden weights in LM, but slower than the hidden weights in VSS. Another difference is that usually both layer weights change in a more monotonic way in VSS than in LM.

VSS does not decrease the step when the gradient decreases because this algorithm does not rely on gradient information, but takes into account the learning history contained in the trajectory. This is advantageous because also the final part of the network training is relatively fast. On the other hand it may lead to very large final weights. This would stop the training process in gradient-based methods because the volume of the parameter space where gradients are non-zero shrinks to zero. For the VSS algorithm it is not a big problem because the gradients are not used, but the error surface becomes very flat, so the direction of the weight changes is simply maintained and learning continues. Large weights change in effect the sigmoid transfer functions into a step-like function, and the final prediction into a binary decision.

In some applications softer outputs may be preferred, giving the user an idea how far is the test case from the decision border (this is sometimes taken as an estimation of the probability of classification). To prevent an excessive weight growth either the training must be stopped early or a regularization term [26] should be added to the error function (for complex data this may be useful), or the parameters max_*w* and max_*d* (defined in section 3.3) must be set to limit maximum values of weights. VSS decreases the step size as a result of tighter curvature of the error surface ravine rather than gradient value. Obviously VSS will stop when there is no difference between the error values in two successive training cycles.

As the training approaches the final stage, the changes of direction are usually slow if no regularization term is added to the error function. If the regularization term (proportional to the sum of the square of the weight values) is added, the error surface in the areas where the weight vector reaches optimal length resembles a paraboloid, preventing further weight growth, but allowing for some small fluctuations of the weight direction.

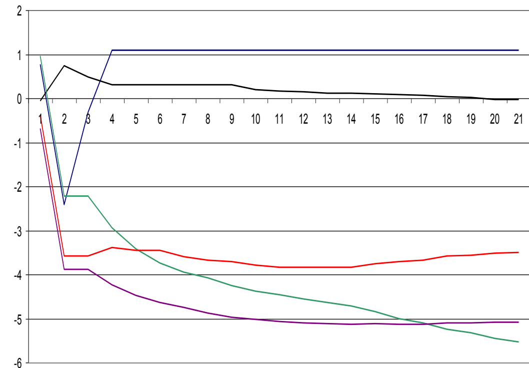

Fig. 8. Hidden layer weight values for Iris (4-4-3) trained with VSS (vertical axis: weight values, horizontal axis: epoch number).

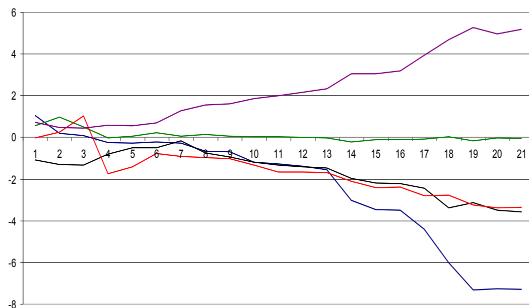

Fig. 9. Hidden layer weight values for Iris (4-4-3) trained with LM (vertical axis: weight values, horizontal axis: epoch number).

5. Experimental Results

In this section VSS performance is compared with the performance of three well known neural learning algorithms, Rprop, SCG and LM. These algorithms were chosen because they are most effective and widely used for neural network training.

Numerical experiments with the VSS algorithm have been made on some well-known benchmark dataset from the UCI learning repository, and the 3-bit parity data. The UCI datasets and their detailed description can be found in [39]. The five benchmark datasets used for our tests have also been used in many studies [40]. They range from very simple data, such as Iris (4 continuous features, 3 classes, 150 vectors), to data of moderate size (WBC, Wisconsin Breast Cancer, with 10 discrete features, 2 classes and 699 cases), and to datasets that are challenging in different ways. The Mushrooms dataset contains descriptions of 8124 samples of edible and inedible mushrooms with 22 symbolic attributes changed to 125 logical features. The Thyroid data contains three classes, with diagnosis based on the 15 binary and 6

continuous features, for 3772 training cases (screening tests for thyroid problems), and 3428 cases given as test data. The training Shuttle dataset contained 43500 vectors and the test set 14500 vectors, each with 9 attributes, describing events from 7 categories. State-of-the-art results for these datasets may be found in [23]. The *n*-bit parity problems are in general difficult for MLP networks, therefore the 3-bit parity problem was also included in the comparisons.

The binary features in Mushrooms and 3-bit parity were represented by 0 and 1. Before training all data was normalized to zero mean and unit standard deviation for each feature:

$$x \leftarrow \frac{x - \bar{x}}{\sigma} \quad (7)$$

For each training algorithm 20 experiments were conducted with each dataset. The network was tested either on a separate test data (Thyroid, Shuttle), or using the 10-fold crossvalidation (Iris, WBC, Mushroom). A vector was considered to be classified correctly if its corresponding output neuron signal was larger than the other neuron signals, and larger than 0.5. All training algorithms were run with their default parameters, the same for each dataset. Table 2 shows a summary of results for which the training accuracy was used as a stopping criterion (*%trn*), which on average corresponded to the given test accuracy (*%test*).

VSS calculations have been performed using the program developed by one of us (MK), written in Borland Delphi. The Matlab Neural Network Toolbox (written by H. Demuth and M. Hagen) was used for Rprop, SCG and LM calculations.

Several values determining algorithm efficiency are considered here: the number of training cycles (N) required to achieve the desired accuracy, the percentage of the algorithm runs that converge to such solutions (C_R), the approximate memory requirements, and the total computational cost. Comprehensive comparison of various properties of different algorithm is a very complex and difficult problem. The number of training epochs can be easily compared, but there is no simple way of comparing other performance parameters. The number of training epochs or the number of times the error is calculated can be quite misleading. For example, in the LM algorithm calculation of the error is only a small fraction of the overall cost of calculations, while in VSS, using the signal table, calculations of the partial errors consumes almost all time.

The training times between Matlab implementation of Rprop, SCG and LM algorithms and our implementation of the VSS algorithm in Delphi are not easy to compare; for example, operations on big arrays, done in LM and other algorithms, are performed much faster in Matlab, while operations on scalar variables are faster in Delphi. To make the comparison more software and platform independent the algorithm speed s has been expressed as the ratio of the training time to the time of a single propagation of the training set through the network, increasing the number of vectors 100-fold. Only the 3-bit parity dataset was too small for such estimation. Using VSS for small datasets this ratio for the Iris data is $s \approx 0.8$ and for the WBC data $s \approx 0.4$, showing that VSS was about 5-times faster than the three algorithms used for comparison. For the Mushroom data $s \approx 0.7$ and for the Thyroid $s \approx 3.7$, showing that the speed of VSS, SCG and Rprop were of the same order, while LM was about 5 times slower.

Implementations of all algorithms use 8-byte floating point representation of numbers, therefore increase of memory requirements by the programs after the initialization of the network may be compared. For the Iris, Breast and the 3-bit parity it was below the accuracy of measurement for all the algorithms. For the Mushroom data it was 40MB for Rprop and SCG, 240MB for LM and 0.4MB for VSS. For the Thyroid it was 1MB for Rprop and SCG, 30MB for LM and 0.2MB for VSS.

Table. 2. Comparison of the VSS, RPROP, LM and SCG algorithms. N is the number of training cycles (N) required to achieve the desired training accuracy % *trn*, t is defined in Eq. (8), C_R is the percentage of the algorithm runs that converge to such solutions.

Algorithm	data set	Iris	WBC	Mushroom	Thyroid	Shuttle	3bit parity
	network	4-4-3	10-4-2	125-4-3	21-4-3	9-6-7	3-3-2
	% trn	97.3	97.0	99.8	98.4	99.2	100
	% tst	96.0	96.0	99.7	98.0	99.0	100
Rprop	N	104	89	15	87	15	131
	σ	18	66	3.0	42	4.8	65
	t	110	50	41	65	18	74
	C_R	100	100	100	85	80	50
SCG	N	54	38	45	186	46	104
	σ	20	28	19	91	16	87
	t	56	21	48	91	40	51
	C_R	90	60	100	75	60	80
LM	N	20	15	6.0	43	15	27
	σ	12	8.0	3.7	27	7.5	17
	t	29	26	17	44	44	32
	C_R	80	85	90	60	60	75
VSS	N	3.5	1.6	2.0	10	6.0	3.1
	σ	1.0	0.4	0.5	2.4	2.0	0.6
	C_R	100	100	100	95	95	95

Relative time and memory values are not reported in Tab. 2 because they obviously depend on a particular software implementation of a given algorithm, but they give an idea of what relative speeds and memory requirements may be expected. It is clear that VSS may easily be used to handle much bigger problems than Mushroom or Thyroid. Estimation of the computational complexity of VSS algorithm is shown in Table 1.

Only VSS and LM algorithms were able to find the optimal solutions with the training accuracy frequently higher than the required minimum, as shown in Tab. 2. However, LM frequently did not converge to the solution and the training had to be repeated with new random weights. Nevertheless, solutions with such low error on the training

set usually have higher errors on the test set. Since the task of neural networks is not to learn the training data points but the underlying data model in order to ensure good generalization, this aspect will not be analyzed further.

The C_R parameter in Table 2 gives the percentage of the algorithm runs that converged to the desired solution within 250 epochs for LM and VSS and within 1000 epochs for Rprop and SCG. VSS had always the highest rate of converged runs and the lowest variance of the results.

The standard t-test for the statistical significance of the difference between the numbers of training cycles was used:

$$t = \frac{\overline{N}_{VSS} - \overline{N}_X}{\sqrt{\frac{\sigma_{VSS}{}^2}{n_{VSS}} + \frac{\sigma_{VSS}^2}{n_X}}} \quad (8)$$

For n_{VSS}=n_X=20 VSS training will require fewer training cycles than training with algorithm X with probability 0.999 if t is greater than 3.55; this was true in all cases (Table. 2). Although in the distribution of the number of training cycles the skewness is usually greater than one, the t values were significantly greater than 3.5, justifying the use of the t-test.

The evolution of MSE error and classification accuracy during the VSS training is shown in Fig.7.

6. Discussion and conclusions

Most of the MLP training algorithms used in practical applications are based on analytical gradient techniques and the backpropagation of error computational scheme. Stochastic search algorithms, based on simulated annealing or evolutionary approaches are more costly and do not seem to be competitive comparing to the multistart gradient-based methods [18], although there are indications that on more complex data results may be different [9]-[11].

A new class of neural training algorithms based on systematic rather than stochastic search has been introduced here. Systematic search techniques have always been popular in artificial intelligence [41], but are neglected in the neural network research. Not much is known about the relative merits of these methods in comparison to widely used stochastic, evolutionary, swarm, ant and other algorithms. Very few attempts to use systematic search techniques have been made so far. Numerical evaluation of gradients in neural network training has been used in [24],[25],[42], and in the extraction of logical rules from data [22],[23] beam search techniques and updating the pairs of weights has been used. In this paper one of the simplest variants of systematic search algorithms has been explored, based on the single weight update.

Analysis of the learning trajectories using the first two principal components in the weight space to visualize MLP error surfaces did not show local minima in "craters" (see more examples in [24],[37]), except the one created by regularization term. The main problem of neural training seems thus not to be the local minima, but rather finding narrow ravines on the landscape of the error function that lead to flat valleys where optimal solutions are found (this is the reason why many starting points followed by short training may be more effective than long training), and getting stuck on the highly situated plateaus. Algorithms based on analytical gradients sometimes cannot precisely determine optimal direction for the next step and may behave as if they were in a local minimum. For that reason it is worthwhile to develop an MLP training algorithm that does not use the gradient information to determine direction and is not so expensive as stochastic or evolutionary algorithms. VSS may get stuck only in those cases when an unfortunate random initialization will lead it away from a good solution, to a point attractor on a highly situated ravine.

Analysis of learning trajectories helped formulate the variable step size training algorithm based on a sequence of single-weight updates, as it is done in the first iteration of Powell's quadratically convergent minimization algorithm [33]. Numerous improvements of the efficiency of the VSS algorithm have been proposed, the most important being the signal table that allows for efficient updates of the neuron activations. Although the VSS algorithm uses some heuristic functions and constants (as most analytical gradient algorithms also do [26],[31]) their values are kept fixed and need not be adjusted by the user.

The VSS algorithm has many advantages. First, the method is quite simple to program, even with all heuristics described in this paper. It does not require calculation of matrices, derivatives, derivation of complex formulas and careful organization of information flow in the backward step. This implies greater modularity of the software, for example the ability to change error functions without re-writing the program, or using cross-entropy error function or arbitrary powers of error. There are also no restrictions on the type of neural functions that can be used – the discontinuous staircase functions may easily be replaced by discrete approximation to transfer functions of any shape [43]. This is very important because some of the functions suitable for neural training lead to much faster convergence on difficult problems [44], but their implementation in the backpropagation networks require rather tedious changes in many parts of the program. Implementation of heterogeneous functions in a single neural network using analytical approach is particularly difficult [45],[46]. Implementing such functions with the VSS algorithm requires very little changes to calculate activations and approximate neural output functions, thus allowing for rapid development of programs for any type of feedforward network (including arbitrary radial basis function networks [26]), making this approach ideal for experimentation.

It is rather surprising that in empirical tests VSS algorithm performed so well, in most cases even better than well established Rprop, SCG and LM algorithms, converging frequently to good solutions in very few epochs.

Most algorithms manipulate only the batch size (the number of vectors presented to the network before the weights are updated) and change all the weights at once. Updating the error function many times in each epoch seems to be a unique feature of the VSS algorithm. The micro iterations that change only a single weight at a time allow for more precise exploration of the error surface. The same is true for iterative solutions to eigenproblems when updates are obtained after multiplication of a single row of diagonalized matrix by approximated eigenvector instead of the whole matrix-vector product [33].

VSS is able to find very good solutions and has very low memory requirements, making it suitable for large scale applications. This algorithm can be used as a reference for more sophisticated and computationally costly methods using stochastic or evolutionary search techniques. There is also plenty of room for improvement of different aspects of this algorithm, for example adding additional directions in the search process. Other algorithms that belong to this family, based on more sophisticated search techniques, should also be developed.

References

[1] D.E. Rumelhart, G.E. Hinton and R.J. Williams, "Learning Internal Representations by Error Propagation". In Parallel Data Processing, Vol.1, Chapter 8, the M.I.T. Press, Cambridge, 1986, pp. 318-362.

[2] M. Riedmiller and H. Braun, "RPROP – a fast adaptive learning algorithm", Technical Report, University Karlsruhe, 1992.

[3] S.E. Fahlman, "Faster Learning Variations of Backpropagation: an empirical study", Connectionist Models Summer School, Morgan Kaufmann, pp. 38-51, 1998.

[4] C. Igel. M. Husken, "Empirical Evaluation of the Improved Rprop Learning Algorithm", Neurocomputing, vol. 50, pp. 105-123, 2003.

[5] A.D. Anastasiadis, G.D. Magoulas, M.N. Vrahatis, "New Globally Convergent Training Scheme Based on the Resilient Propagation Algorithm", vol. 64, pp. 253-270, 2005.

[6] D. Marquard, "An Algorithm for Least-squares Estimation of Nonlinear Parameters", SIAM J. Appl. Math., vol.11, pp. 431-441, 1963.

[7] N.N.R. Ranga Suri, D. Deodhare, P. Nagabhushan, "Parallel Levenberg-Marquardt-Based Neural Network Training on Linux Clusters - A Case Study", Proc. 3rd Indian Conf. on Computer Vision, Graphics & Image Processing, Ahmadabad, India 2002.

[8] M.F. Möller, "A Scaled Conjugate Gradient Algorithm for Fast Supervised Learning", Neural Networks, vol. 6, pp. 525-533, 1993.

[9] M.D. Ritchie, B.C. White, J.S. Parker, L.W. Hahn, J.H. Moore, "Optimization of neural network architecture using genetic programming improves detection and modeling of gene-gene interactions in studies of human diseases". BMC Bioinformatics 4: 28, 2003.

[10] R.S. Sexton, R.E. Dorsey, N.A. Sikander, "Simultaneous optimization of neural network function and architecture algorithm". Decision Support Systems, vol. 36(3), pp. 283-296, 2004.

[11] N. Garcia-Pedrajas, D. Ortiz-Boyer, C. Hervas-Martinez, "An alternative approach for neural network evolution with a genetic algorithm: crossover by combinatorial optimization". Neural Networks, vol. 19(4), pp. 514-528, 2006.

[12] J. Engel, "Teaching Feed-forward Neural Networks by Simulated Annealing", Complex Systems vol. 2, pp. 641-648, 1988.

[13] K.P. Unnikrishnan, and K.P. Venugopal, "Alopex: A Correlation-Based Learning Algorithm for Feed-Forward and Recurrent Neural Networks", Neural Computations, 6, pp. 469-490, 1994.

[14] V. F. Koosh, "Analog Computation and Learning in VLSI", PhD Thesis, Caltech, Pasadena, CA, 2001.

[15] R. Battiti and G. Tecchiolli, "Training Neural Nets with the Reactive Tabu Search", IEEE Trans. on Neural Networks, vol.6, pp. 1185-1200, 1995.

[16] W. Duch, J. Korczak, Optimization and global minimization methods suitable for neural networks, Technical Report 1/99, Nicolaus Copernicus University, http://citeseer.ist.psu.edu/duch98optimization.html

[17] L. Hamm and B. Wade Brorsen, "Global Optimization Methods", The 2002 International Conference on Machine Learning and Applications (ICMLA'02), Monte Carlo Resort, Las Vegas, Nevada, USA, June 2002.

[18] D. Saad (ed.), "On-Line Learning in Neural Networks", Cambridge, UK: Cambridge University Press 1998.

[19] J.R. Quinlan, R.M. Cameron-Jones, "Oversearching and layered search in empirical learning". Proc. of 14th Int. Joint Conference on Artificial Intelligence (IJCAI-95), pp. 1019-1024, Montreal, Canada, 1995.

[20] E. Aarts , J.K. Lenstra, "Local Search in Combinatorial Optimization", John Wiley & Sons, Inc., New York, NY, 1997.

[21] J.C. Spall, "Introduction to Stochastic Search and Optimization". J. Wiley, Hoboken, NJ, 2003.

[22] W. Duch, K. Grąbczewski, "Searching for optimal MLP". 4th Conference on Neural Networks and Their Applications, Zakopane, Poland 1999, pp. 65-70.

[23] W. Duch, R. Setiono, J.M. Zurada, "Computational intelligence methods for understanding of data." Proc. of the IEEE vol. 92(5), pp. 771- 805, 2004.

[24] M. Kordos, "Search-based Algorithms for Multilayer Perceptrons", PhD Thesis, The Silesian University of Technology, Gliwice, Poland 2005, available at http://www.phys.uni.torun.pl/~kordos

[25] W. Duch, M. Kordos, "Multilayer Perceptron Trained with Numerical Gradient". Proc. of Int. Conf. on Artificial Neural Networks (ICANN), Istanbul, June 2003, pp. 106-109.

[26] S. Haykin, "Neural networks: a comprehensive foundations". New York: MacMillian Publishing 1994.

[27] K. W. Morton, D. F. Mayers, "Numerical Solution of Partial Differential Equations. An Introduction". Cambridge University Press, 2005.

[28] L.G.C. Hamey, "XOR has no local minima: A case study in neural network error surface analysis". Neural Networks, vol. 11(4), pp. 669-681, 1998.

[29] E.D. Sontag, H.J. Sussman, "Backpropagation Can Give Rise to Spurious Local Minima Even for Networks Without Hidden Layers", Complex Systems, vol. 3, pp. 91-106, 1989.

[30] F.M. Coetze, V.L. Stonick, "488 Solutions to the XOR Problem", Advances in Neural Information Processing Systems, vol. 9, pp. 410-416, Cambridge, MA, MIT Press, 1997.

[31] R. Hecht-Nielsen, "Neurocomputing", Adison-Wesley, Reading, MA, 1990.

[32] M. Lehr, "Scaled Stochastic Methods for Training Neural Networks", PhD Thesis, Stanford University, 1996.

[33] W. H. Press, S. A. Teukolsky, W. T. Vetterling, B. P. Flannery, "Numerical Recipes in C", Press Syndicate of The University of Cambridge, 1992.

[34] W. Duch, "Support Vector Neural Training". Lecture Notes in Computer Science, vol. 3697, 67-72, 2005.

[35] M. Gallagher, "Multi-layer Perceptron Error Surfaces: Visualization, Structure and Modeling", PhD Thesis, University of Queensland, 2000.

[36] M. Gallagher, T. Downs, *Visualization of Learning in Multi-layer Perceptron Networks using* PCA. IEEE Transactions on Systems, Man and Cybernetics-Part B: Cybernetics, vol. 33(1):28-34, 2003.

[37] M. Kordos and W. Duch, "A Survey of Factors Influencing MLP Error Surface", Control and Cybernetics, vol. 33(4), pp. 611-631, 2004.
[38] J. Denker et. al., "Large automatic learning, rule extraction and generalization", Complex Systems, 1:887-922, 1987
[39] C.J. Mertz, P.M. Murphy, UCI repository of machine learning databases, http://www.ics.uci.edu/~mlearn/MLRepository.html
[40] D. Michie, D.J. Spiegelhalter, C. C. Taylor, "Machine Learning, neural and statistical classification", Elis Horwood, London, 1994
[41] S. Russell, P. Norvig, Artificial Intelligence: A Modern Approach. Prentice Hall, 2nd ed, 2002.
[42] W. Duch, M. Kordos, "Search-based Training for Logical Rule Extraction by Multilayer Perceptron". Proc. of Int. Conf. on Artificial Neural Networks (ICANN), Istanbul, June 2003, pp. 86-89.
[43] W. Duch and N. Jankowski, "Survey of neural transfer functions", Neural Computing Surveys vol. 2, pp. 163-213, 1999.
[44] W. Duch, "Uncertainty of data, fuzzy membership functions, and multi-layer perceptrons". IEEE Transactions on Neural Networks vol. 16(1), pp. 10-23, 2005.
[45] W. Duch, K. Grąbczewski, "Heterogeneous adaptive systems". IEEE World Congress on Computational Intelligence, Honolulu, HI, pp. 524-529, 2002.
[46] N. Jankowski, W. Duch, "Optimal transfer function neural networks". 9th European Symposium on Artificial Neural Networks (ESANN), Brugge, Belgium. De-facto publications, pp. 101-106, 2001.