VARIABLE STEP SEARCH ALGORITHM FOR MLP TRAINING

Mirosław Kordos

Faculty of Automatic Control,
Electronics and Computer Science
The Silesian University of Technology
Gliwice, Poland

Włodzisław Duch

Department of Informatics
Nicolaus Copernicus University,Toruń, Poland
and School of Computer Engineering
Nanyang Technological University, Singapore
www.phys.uni.torun.pl/~duch

ABSTRACT
The variable step search algorithm is based on a simple search procedure that changes one network parameter at a time. Visualization of learning trajectories and MLP error surfaces is used for the algorithm design and optimization. The algorithm is compared to three other MLP training algorithms: Levenberg-Marquardt, scaled conjugate gradient, and training based on numerical gradient.

KEY WORDS
neural networks, MLP, search algorithms, learning trajectory

1. Introduction

The search-based algorithms for multi-layer perceptron (MLP) training are based on the idea of changing network parameters (weight and biases) and checking the influence of such change on the mean-square error (MSE), or another error measure. In contrast to backpropagation and other training algorithms that use analytical gradients search based algorithms impose no restrictions on transfer functions, error functions or neural connection structures. In particular transfer functions do not have to be differentiable.

Many variants of search algorithms are possible. One of the simplest from the search-based family called variable step search algorithm (VSS) is presented here. It changes one weight at a time and roughly searches for a minimum of the error along the weight direction (bias is considered here to be on of the weights). VSS algorithm does not calculate gradients, but use certain properties of MLP error surface that impose similar changes on the same weight in two successive training cycles.

In the paper we consider only standard 3-layer MLP networks with sigmoidal transfer functions in hidden and output layer trained for classification tasks using MSE calculated on the entire training set as performance measure. The network structure is fixed during the training.

Many papers compare new algorithms with standard gradient backpropagation. Instead we compare VSS not with the algorithm that was developed as first but with algorithms that are considerd to be most effective as Levenberg-Marquardt algorithm and scaled conjugate gradient.

2. Training Algorithms

2.1 Variable Step Search Algorithm (VSS)

The simplest search-based algorithm works in the following way: in one training cycle the value of *dw* is added to or subtracted from a single weight w. If the error decreases after the change then the change is kept, otherwise it is rejected. Then *dw* is added or subtracted from the next weight and again the error is calculated, until the changes of all weights are examined. dW maybe decreased each training cycle. The simplest search-based algorithm was also used for rule extraction from MLP network with not fully connected layers [1].

VSS is a modified version of that algorithm, in which *dw* is not constant, but dynamically adjusted independently for each weight during a rough minimization in each weight direction. VSS was designed taking the advantage of MLP error surface properties that its steepness in different directions varies ranks of orders, and the ravines in which the MLP learning trajectories lay are usually curves, slowly changing their direction [2-5]. Basic on the properties we can expect that optimal *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 differ ranks of order.

In each training cycle *i* the first guess of $dw_i[w]$ for a given weight *w* might be the value $dw_{i-1}[w]$ that the weights changes about in the previous training cycle. However the detailed experimental analysis of the algorithm behaviour lead to the conclusion that for most cases the least number of calculations is obtained when the first guess is $dw_i=(0.33\div0.37)dw_{i-1}$, in spite that statistically the ratio dwi/dw_{i-1} is close to 1.

Fig. 1. shows a diagram for determining *dw* of a single weight in one training cycle. Before the training starts the weights are initialized with random values from the interval (-1;1). (Initializing all hidden layer weights with zero values and setting large *d0* is a good way for feature reduction in the first training cycle. The larger *d0* (0.5, 1, 2) the more features are eliminated from further training. After the first training cycle all hidden weights that are still zero are pruned and d0 is set to a smaller value.)

In the first training cycle $d=d0=0.25$. Since $dw_0[w]=0$, for each weight *w* in the first training cycle the first guess

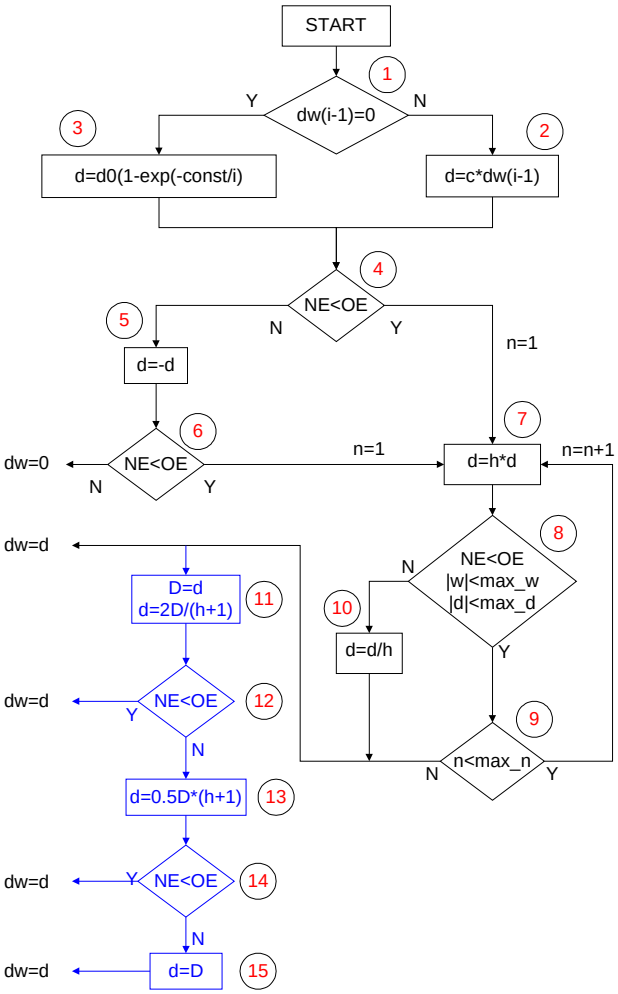

Fig. 1. Determining a single weight value in one training cycle of the Variable Step Search Algorithm

is $dw_1[w]=d0$. The ravine on the error surface is narrow close to the algorithm starting point. Thus setting $d0>0.5$ frequently causes that the trajectory cannot well fit into the ravine bottom and some weights oscillate while others do not change at all during some initial training cycles what results in slowing down the training.

The VSS algorithm follows (Fig. 1):

for *i=1* to *NumOfTrainingCycles* do begin
for *w=1* to *NumOfWeights* do begin

1: If $dw_{i-1}[w]=0$ then goto 3.
2: The value of $c*dw_{i-1}[w]$ (experimentally determined optimal c=0.33÷0.37) is added to the weight *w*.
3: If the weight *w* did not change in the previous training cycle try to add (or subtract) to it a smaller value $d=d0*(1-exp(-const/i))$. where *i* is the training cycle number. Optionally the weights can be frozen.
4: If the new error *NE* after the change is smaller than old error *OE* before the change then the direction of the change is correct, goto 7.
5: otherwise change the direction of search $d=-d$.
6: If the new error *NE* after the change is not smaller than old error *OE* before the change then do not change the weight, take the next weight and goto 1.
7: Search for an approximate minimum along this direction; set $d=h*d$ (experimentally determined optimal h=2.0÷2.5).
8: If the new error *NE* after the change is smaller the than old error *OE* before the change, $|w|<max\_w$ and $|dw|<max\_dw$ then goto 9 else goto 10 (maximal acceptable values for a single weight *max_w* and for a single weight change *max_dw* given in order to prevent excessive weight growth, the values are optional and can be set to infinity).
9: If $n<max\_n$ goto 7. *max_n* is given to prevent the loop through points 7-9 from being executed too many times. (experimentally determined optimal $max\_I=4$ when $h=2$)
10: set $d=d/h$.
11-15: Optional steps. The error value is calculated in two additional points that are the geometric mean of *d* and *d/h* and of *d* and *d*h*. The best value of the two points and d becomes *dW*. In most cases points 11-15 do not need to be used. h=2 assures a maximum difference between $dw_i[w]$ and the distance from w_{i-1} to the error minimum in the direction *w* of 50%, what in most cases providing the fastest network learning. Finding minima with greater precision could also be done by setting *h* to a smaller value but that slows down the training more

if error<desired_error then exit;
end;
end;

On average the error is calculated about 3 times while determining a single weight value in one training cycle. Since only one weight is changed at a time, the signals do not need to be propagated through the entire network to calculate the error, but only through the fragment of the network in which the signals are different before and after the change. The remaining signals incoming to all neurons of hidden and output layers are remembered for each training vector in an array called “signal table”. The signals must be propagated through the entire network only once at the beginning of the training thus filling in the signal table. 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 the number of hidden and output neurons. After a single weight is changed only the appropriate entries in signal table are updated. Also the MSE error of each output neuron is remembered and do not need to be calculated again if a weight of another output neuron is changed. Using the signal table causes a significant speedup of the training, especially for bigger networks. For the network 125-8-2 the speed up is 25 times and for the network 4-4-2 two times.

2.2 Numerical Gradient (NG)

NG algorithm developed by us [2] has many common features with the Variable Step Search Algorithm. The training algorithm consists of two stages: finding the gradient direction and finding the minimal error along this direction.

To find the gradient direction, a constant value dw is added to each weight and the *E[w+dw]* calculated. If it decreases, then the component of the gradient direction V[w] is calculated as:

$$V[w] = \eta\,[(E[w] - E[w+dw]) / E[w]]$$

The same value Δw is subtracted from the weight w and the E[w-Δw] calculated. If it decreases more then it decreased previously, then the gradient component should be:

$$V[w] = \eta\,[(E[w] - E[w-dw]) / E[w]]$$

If changing of the weight does not change the error than V[w]= 0. The values V[w] are proportional to the decrease of error as a result of weight perturbation with Δw.

Making one step dw along the gradient direction $\mathbf{V}$ moves from the starting point $D = dw\ ||\boldsymbol{V}||$ units, where

$$\|\mathbf{V}\| = \left(\sum_{w} \mathbf{V}[w]^2 \right)^{1/2}$$

The starting point of an iteration is in the minimum of the previous iteration, D is also the distance between two successive starting points. The bigger $\|\boldsymbol{V}\|$ is the faster are the error changes when moving along the gradient direction (the error surface is steeper). Adding momentum term usually makes the training faster.

2.3 Levenberg-Marqurdt Algorithm (LM)

LM algorithm is one of the most efficient training algorithms for smaller networks. It uses gradient descent incorporating curvature of the error surface (Newton's method) [3]. Combining these two approaches the following update rule is obtained:

$$w_{i+1} = w_i - (H + \lambda I)^{-1} \nabla E(w_i)$$

where $H = \nabla^2 E(w_i)$ is the Hessian matrix and $\nabla E(w_i)$ is the Jacobian matrix. Replacing the identity matrix with the Hessian diagonal increases the step in the direction of small gradient minimizing the trajectory oscillations [4]. Thus we get the final Levenberg-Marquardt update rule:

$$w_{i+1} = w_i - (H + \lambda \mathrm{diag}[H])^{-1} \nabla E(w_i)$$

λ is dynamically decreased by an order of magnitude if the error decreases. If the error increases λ is increased by an order of magnitude, the learning trajectory returns to the previous point and the step is repeated.

The main disadvantage of LM algorithm is its high memory requirement. The size of the Jacobian is $N_V N_O N_W$ and that of Hessian is N_w^2. In practice even for modest networks it becomes a problem, for example for the satellite image database with 27 hidden neurons (see [5]) the Jacobian alone requires 248 MB memory using double precision (8 Byte) representation. By comparison VSS algorithm requires only 1.26 MB for the same network architecture. Storing the training set in memory takes 1.3 MB of RAM.

To reduce the memory requirements the Jacobian may be divided into several parts and Hessian calculated by summing partial results, but this adds a significant computational overhead.

2.4 Scaled Conjugate Gradient (SCG)

SCG algorithm is considered to be the quickest one among the well known algorithms for larger networks.

With initial gradient g_0 and initial vector d_0=-g_0 the conjugate gradient method recursively constructs two vector sequences:

$$g_{i+1} = \nabla E(w_{i+1}) \quad \text{and} \quad d_{i+1} = g_{i+1} + \gamma_i d_i$$

where g is gradient direction and d is called conjugate direction. We proceed from w_i along the direction d_i to the minimum of E at w_{i+1} through line minimization and then set g_{i+1} at the minimum. [6]

Since conjugate gradient methods do not compute any matrices they scale well with the network size.

The conjugate direction d minimizes trajectory oscillations and allows longer steps, which leads to a faster convergence than steepest descent directions, although the error function decreases most rapidly in the steepest descent directions. A similar affect is caused by momentum added to numerical gradient or to backpropagation.

Scaled conjugate gradient algorithm is a version of conjugate gradient that avoids the time-consuming line search along conjugate directions. As a Levenberg-Marquardt algorithm, it introduces a scalar λ to regulate the indefiniteness of $H = \nabla^2 E$. The scaling procedure is described in details in [7].

3. Learning Trajectories

3.1 Visualization of Learning Trajectories and Error Surface

MLP error surface E(W)=$\sum$x$\|$Y-$M(\boldsymbol{X};\boldsymbol{W})\|$ is defined in the weight space $\boldsymbol{W}$ for a given training data $\boldsymbol{X}$, desired output vector $\boldsymbol{Y}$ and structure of network mapping $M(\boldsymbol{X};\boldsymbol{W})$. An MLP training process can be defined as searching for a minimum on the error surface, where the learning process creates a trajectory on that surface.

PCA (Principal Component Analysis) as a natural choice for visualizing high dimensional data and was previously used for three-dimensional visualization of learning trajectories [8][9][11] and MLP error surfaces [10][11]. PCA is performed by singular value

decomposition on the covariance matrix of weights taken from each training cycle.

The vertical axis in Fig.3. shows MSE error. Horizontal axes show distances in the weight space in c_1 and c_2 PCA directions corresponding to the first and second eigenvector of the weight covariance matrix. The view of error surface in PCA projections depends only on dataset and on network structure, but does not depend almost at all on the training algorithm so it is shown only once. Instead the trajectories stronly depend on the training algorithm.

All sample visualizations use the same network with 4 inputs, 4 hidden and 3 output neurons trained with iris dataset.

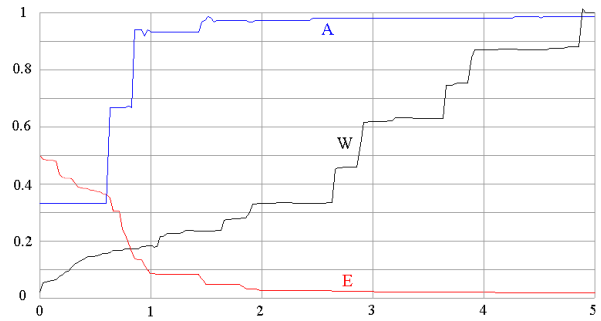

Fig. 2. MSE error (E), classification accuracy on training set (A) and $||W_i||/||W_5||$ (W) during 5 training cycles of IRIS (4-4-3). After three training cycles the training has converged and should be stopped

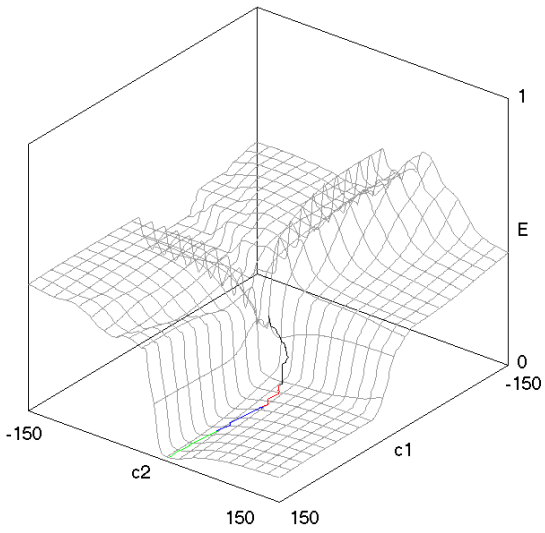

Fig. 3. Projection of IRIS (4-4-3) error surface trained with VSS and learning trajectory in the first and second PCA direction. The trajectory color changes every training cycle

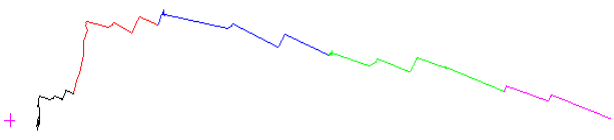

Fig. 4. Projection of IRIS (4-4-3) learning trajectory trained with VSS in the first and second PCA direction. The cross shows the zero point in the weight space

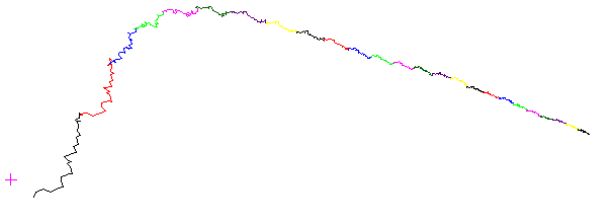

Fig. 5. Projection of IRIS (4-4-3) learning trajectory trained with the simplest search algorithm changing one weight at a time. (VSS version going always through point 3 instead of point 2 with max_n=1, see Fig.1) in the first and second PCA direction

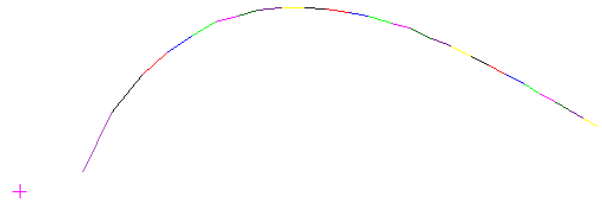

Fig. 6. Projection of IRIS (4-4-3) learning trajectory trained with NG in the first and second PCA direction

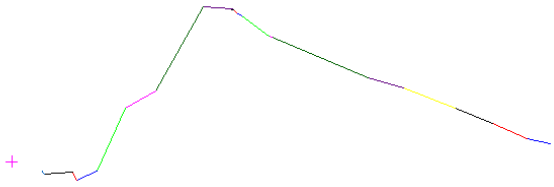

Fig. 7. Projection of IRIS (4-4-3) learning trajectory trained with LM in the first and second PCA direction

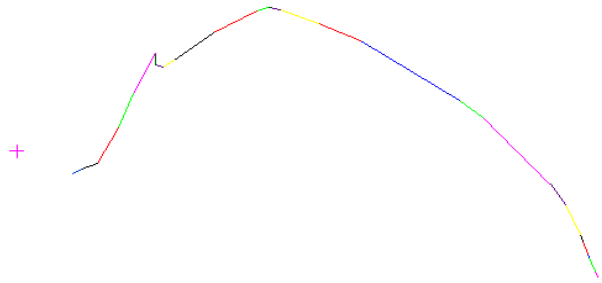

Fig. 8. Projection of IRIS (4-4-3) learning trajectory trained with SCG in the first and second PCA direction

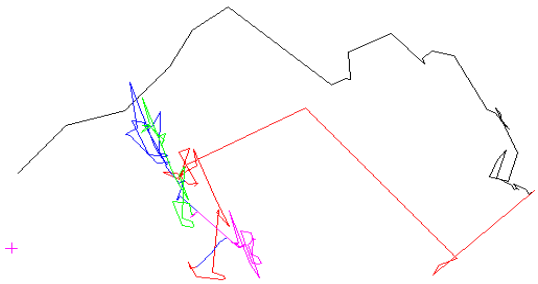

Fig. 9. Projection of IRIS (4-4-3) learning trajectory trained with VSS in the third and fourth PCA direction. Higher PCA components have significant values only at the beginning of the training, what is clear, because VSS algorithm frequently changes the direction at the beginning of the training and weights grow quicker in the next cycles

3.2 Using Error Surface and Learning Trajectory Properties to Enhance Algorithms Performance

The similarity between all trajectories presented in Fig. 4-8 is obvious; they create similar arcs following the shape of iris error surface ravine Also the difference between VSS and other algorithms learning trajectories is clearly visible. Using gradient-based information makes the training dependent on a factor that vanishes as the training progresses, so gradient-based algorithms have a tendency to decrease their learning steps as gradient decreases and thus slowing down the training even more. (LM and SCG additionally make smaller steps when the trajectory is turning more and bigger when it goes through relatively straight parts of the error surface ravine – effect of using second order information, which is proportional to the curvature).

VSS increases the step when the gradient decreases and that makes the algorithm effective. The increase of VSS step is caused by the fact that VSS does not rely on gradient information, but rather on the learning history. The learning history creates a trajectory. When performing PCA on the points through which the trajectory goes, then usually about 95% of total variance is captured by the first and second PCA direction. So figures 3-8 reflect the learning trajectory properties quite well. The trajectories show some regularities, not only for iris but also for all datasets. VSS uses the regularities to make the training more effective.

Not only *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 differ ranks of order but also some trends in weight changes may be observed.

Many attempts were made to use the statistical distributions of weight changes to speed up the algorithm but so far none of them was successful, and the version of VSS presented in that paper has the best performance. Also some attempts were made to extrapolate the trajectory in two or more PCA directions and then jump to that point thus omitting several training cycles. But due to the complex shape of the valleys on the error surface and to the fact that PCA/ICA/kernel PCA/Principal Curves or whatever directions are changing all the time during the training it is extremely difficult to guess proper point in a weight space several training cycles ahead. So far it was successful only for a few databases. Nevertheless the approaches seem very promising and worth further research.

4. Comparison of VSS, NG, LM and SCG

Three parameters determining algorithm efficiency are considered: the total computational complexity (C_t) required to achieve the desired effect, the quality of the solution the algorithm can find (%test) and the percentage of the algorithm runs that converge to the solution (CR).

VSS and NG calculations were done using a program developed by one of us (MK). Matlab Neural Network Toolbox (written by H. Demuth and M. Hagen) was used for LM and SCG calculations. Since the training times between Matlab and our program written in Delphi could not be directly compared, the computational complexity of the algorithms was assessed in the following way: first only the datasets were propagated through the network with calculating the MSE error S_n times. S_n was set to a value assuring the time of simulation S_t about 1 minute (in the case of Matlab it was done by modifying trainscg.m so that only sim() function was called within the plot). Then the algorithms were run the average number of training cycles require to converge T_n for mushroom, thyroid and shuttle and 1000 training cycles for iris and breast and the training time T_t was measured (the real training times for iris and breast were too short for reliable direct measurement). All on-screen display and additional options were switched off in both programs during the experiment. A given algorithm computational complexity per one training cycle was calculated for given dataset and network structure as: $C_e=(T_t/T_n)/(S_t/S_n)$.

For VSS C_e is from 20 for the smaller networks (iris, breast) to 100 for the bigger networks (mushroom). For LM C_e is between 8 for iris and 540 for mushroom and grows rapidly with network size. For SCG it does not depend much on network size being between usually between 4 and 8, however for very large datasets as shuttle with 43,500 vectors it falls down even below 2. The number of training cycles required to converge N_t was always the lowest for VSS and the highest for SCG.

The total computational complexity C_t shown in Table 1. is the parameter reflecting the algorithm speed. It express the ratio of the total training time to the time of propagating the dataset through the network once. C_t can be obtained by multiplying the per training cycle complexity C_e by the average number of training cycles N_t required to train the network: $C_t=C_eN_t$.

In all cases C_t for VSS was lower than that for LM. In most cases it was also lower than that of SCG, however for larger datasets that are relatively easy to train, as mushroom, the difference between C_t for VSS and for SCG vanishes.

VSS and LM were able to find the best solutions with %test frequently higher than shown in Table 1, which were in many cases out of reach for NG and SCG. However LM frequently did not converge to the solution and had to be repeated with other starting weights. The CR parameter in Table 1. expresses the convergence rate of algorithms, e.i. the percentage of the algorithm runs that converged to the desired solution within 5000 cycles.

For VSS and NG the minimum and maximum number of training cycles in that a given algorithm converges to a given solution differed less than 30% from the mean number N_t given in Table 1, while for LM the difference was often over 100%. VSS and NG algorithms had the smallest memory requirements. Though the performance of NG could be frequently improved by adding momentum, it still will be poorer than that of VSS. The main difference between the algorithms is that NG uses directly gradient information, while VSS does not.

For each training algorithm 10-20 experiments were made with every dataset. The network was tested on test sets (thyroid, shuttle) or in 10-fold crossvalidation (iris, breast, mushroom). A vector was considered to be classified correctly if its corresponding output neuron signal was higher then other neurons signals and than 0.5. All training algorithms were run with the same default parameters for each dataset.

It is clear that including additional techniques such as freezing or pruning the least significant weights, calculating error on a different part of vectors each training cycle instead of on the entire dataset or eliminating from further training vectors that give the least error [13] much shorter training times are possible with each of the examined algorithms, but till now it was not a matter of our research.

5. Conclusions

The proposed variable step search algorithm is fast, can find very good solutions, does not require multistart and has low memory requirements. It is also very simple to program since it does not require calculation of derivatives and matrices, therefore it is quite surprising that in empirical tests it usually outperforms both LM and SCG.

It is clear that search-based techniques, popular in artificial intelligence and completely neglected in neural networks (with an exception of Alopex algorithm [14] based on simulated annealing), may be the basis for competitive network training algorithms. They may be used for initialization and combined with traditional gradient-based techniques. But so far the performance of VSS as a standalone algorithm has been more than satisfactory.

References:

[1] W. Duch, M. Kordos, "Search-based Training for Logical Rule Extraction by Multilayer Perceptron", Int. Conf. on Artificial Neural Networks (ICANN) and Int. Conf. on Neural Information Processing (ICONIP), Istanbul, June 2003, pp. 86-89

[2] W. Duch, M. Kordos, "Multilayer Perceptron Trained with Numerical Gradient", Int. Conf. on Artificial Neural Networks (ICANN) and Int. Conf. on Neural Information Processing (ICONIP), Istanbul, June 2003, pp. 106-109

[3] Ananth Ranganathan, "The Levenberg-Marquardt Algorithm"

[4] D. Marquardt, "An algorithm for least-squares estimation of nonlinear parameters", SIAM J. Appl. Math., 1963, Vol.11,pp.431-441.

[5] N.N.R. Ranga Suri, Dipti Deodhare, P. Nagabhushan, "Parallel Levenberg-Marquardt-based Neural Network Training on Linux Cluster – A Case Study"

[6] Fang Wang et al. "Neural Network Structures and Training Algorithms for RF and Microwave Applications"

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

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

[9] M. Gallagher, T. Downs, "Visualization of Learning in Multi-layer Perceptron Networks using PCA", http://www.itee.uq.edu.au/~marcusg/ publications.html, 2003

[10] W. Duch, M. Kordos, "On Some Factors Influencing MLP Error Surface", 7th Int. Conf. on Artificial Intelligence and Soft Computing (ICAISC), Zakopane, Poland, June 2004 (in print)

[11] W. Duch, M. Kordos, "A Survey of Factors Influencing MLP Error Surface", Control and Cybernetics, 2004 (submitted)

[12] C.J. Mertz, P.M. Murphy, UCI repository of machine learning databases, http://www.ics.uci.edu/~mlearn/MLRepository.html

[13] W. Duch, "Support Vector Neural Training", IEEE Transactions on Neural Networks, 2004 (submitted)

[14] E. Micheli-Tzanakou (ed) "Supervised and Unsupervised Pattern Recognition: Feature Extraction and Computational Intelligence", CRC Press 2001

Table 1. Comparison of VSS, NG, LM and SCG algorithms.

dataset	%	VSS				NG				LM				SCG			
network	test	N_t	MB	CR	C_t	N_t	MB	CR	C_t	N_t	MB	CR	C_t	N_t	MB	CR	C_t
iris	96	2.3	-	100	41	14	-	100	223	20	-	80	223	118	-	90	948
4-4-3	98	3.5		100	63	22		100	350	22		80	246	157		90	1260
breast 10-4-2	98	1.0	-	100	30	10	-	100	280	20	1.5	100	109	438	0.4	10	807
mushroom	98	1.2	0.4	100	124	21	0.4	100	1070	4	240	90	2180	20	40	100	167
125-4-3	99.9	2.0		100	206	-		0	-	6		90	3260	45		100	377
thyroid	97	6.1	0.2	100	392	40	0.2	40	862	25	30	50	1640	103	1.0	70	581
21-4-3	98	10		100	643	-		0	-	35		40	2300	-		0	-
shuttle	98	4.5	1.6	100	423	34	1.6	90	1300	14	1400	60	1430	780	20	50	1480
10-6-7	99	6.0		100	564	58		90	1740	19		60	1940	1620		30	3080

N_t - number of training cycles

MB - memory usage in MB for storing network parameters, without including memory for the data set (calculated by subtracting the memory used by the program running the algorithm on given dataset from the memory used by the program with the given dataset loaded in memory and running the algorithm on xor dataset. Memory usage measured with Task Manager)

CR – convergence rate, percentage of training runs that converged to a given accuracy solution within 5000 training cycles

C_t - total computational complexity; the ratio of the total training time to the time of propagating the dataset through the network once