Variable Step Search MLP Training Method.

Mirosław Kordos[1] and Włodzisław Duch[2,3]

[1] Division of Biomedical Informatics, Cincinnati Children's Hospital Medical Center, Cincinnati, OH, USA (e-mail: kordos@phys.uni.torun.pl)
Dept. of Informatics, Nicolaus Copernicus University, Toruń, Poland
[2] School of Computer Engineering, Nanyang Technological University, Singapore
Google: Duch

Abstract. The MLP training process is analyzed and a variable step search-based algorithm (VSS) that does not require gradient information is introduced. This algorithm finds rough position of the minima in each single weight direction, and successively updates the weights. Only a small fragment of the network is analyzed for each update, making the method computationally efficient. The VSS algorithm is simpler to program than backpropagation, yet the quality of results and the speed of convergence are at the level of state-of-the-art Levenberg-Marquardt and scaled conjugate gradient algorithms.

1 Introduction.

Multilayer perceptrons (MLP) are usually trained using analytical gradient-based algorithms with error backpropagation. Some of the most popular methods that include the standard backpropagation (BP), RPROP, Quickprop, Levenberg-Marquardt (LM) [1] [2], and the scaled conjugate gradient (SCG) algorithm [3] [4]. Also many global optimization methods were used for neural network training [5]. However in spite of much higher computational cost they only seldom give better results for real-world problems [6].

The MLP training is a search for the minimum of the multivariate error function $E(\mathbf{W};\mathcal{D}) = \sum_X ||\mathbf{Y} - M(\mathbf{X};\mathbf{W})||$, known also as the error surface. The error surface is a function of the network mapping $M(\mathbf{X};\mathbf{W})$ parameterized by weights $\mathbf{W}$ (including biases as W_0 weights), the training dataset $(\mathbf{X},\mathbf{Y}) \in \mathcal{D}$, and the type of the error measure used (frequently square of the Euclidean norm). However, the error surface does not depend on the training algorithm. The learning processes are represented by trajectories $\mathbf{W}(t)$ that lie on the hyper-surface $E(\mathbf{W};\mathcal{D})$ in the weight space $\mathbf{W}$. Error surface can be visualized using projections of the original space on the PCA-directions [7]. The first two PCA components capture usually about 95-97% of the total weight variance during the training [8]. Although sophisticated learning techniques have been derived from mathematical analysis of the learning process [9] they are rarely useful in practice, while observations of weight changes in the PCA space lead to practical training heuristics. As a result of this analysis the Variable Step Search algorithm (VSS) for MLP training has been developed [10]. In contrast to most other neural training methods it is based neither on analytical gradient nor on global optimization but on the local search approach.

After an introduction of the VSS algorithm experimental results on several datasets are presented, comparing convergence properties, accuracy and complexity of calculations. 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 considered to be most effective for smaller networks; Levenberg-Marquardt algorithm and for bigger networks: scaled conjugate gradient. The networks considered here have standard 3-layer MLP architecture, and use sigmoidal transfer functions with unit slope, trained for data classification.

The final section of the paper contains conclusions and remarks on future work.

2 Variable Step Search algorithm

The analytical gradient-based algorithms have no direct access to the influence of hidden layer weights on the network error, instead they use the error backpropagation mechanism to assess the gradient component in each hidden weight direction. Well-known formulas for gradient of the output weights

$$\frac{\partial E(\mathbf{W})}{\partial w_k} = (M(\mathbf{X};\mathbf{W}) - Y)\,\frac{\partial M(\mathbf{X};\mathbf{W})}{\partial w_k} \tag{1}$$

are expressed using derivatives of the transfer functions and the errors made by the network, propagated to the input layer to calculate gradients for the remaining weights. In numerical gradient network training [10] a single weight w_k is subject to a small perturbation dw and the network error change as a response to that perturbation is used as a gradient component in w_k direction:

$$\frac{\partial E(\mathbf{W})}{\partial w_k} = \frac{E([w_1, ..., w_k + dw, ..., w_n]) - E(\mathbf{W})}{dw} \tag{2}$$

Analytical backpropagation makes small gradient values even smaller than those computed numerically, and the large ones larger, especially in larger networks trained with complex data. In empirical tests a finite step along numerical gradient led in most cases to a faster decrease of error than the same step along analytical gradient. The difference between analytical and numerical gradient tends to be stronger when directional error minimization is used with both algorithms. [10].

The error surface of networks with hidden layers has a starfish structure [11]. Near the middle (W=0) all weight configurations have moderately high error values. Radiating out from the center the valleys get deeper as they go out, but asymptotically level out. In the best valleys, the error is exactly or asymptotically zero, other valleys have higher floors. Both global and local minima rarely create craters but frequently ravines reaching their minimum in infinity (without penalty term for big weights added to the error function). This corresponds to the infinite growth of (usually output layer) weights when continuing the training enough long. Though local minima in finite points can exist [12], in practice flat areas and saddle points have much more important influence on the training performance [13],[14]. Analytical gradient algorithms may get trapped in the pleteaus because gradients on some parts of error surfaces may become very

small [15, 16]. In many cases a downward path exists from such points and a finite step numerical gradient training starting from such points reaches final convergence.

The error surface near the end of the training becomes almost flat, slowing significantly the gradient-based training. More detailed analysis of this problem shows that mostly the output layer weights contribute to the slower learning in the final stage. Statistical relations between the size of the gradient component dEw in the weight w direction and the distance mw from the current point $\mathbf{W}$ to the minimum of the error function show that the optimal component $dS(w) \approx mw$, expect for very large mw values, where it should be limited. This leads to a heuristic formula:

$$dS(w) = (1 + a \cdot \exp(-b \cdot epoch)) \cdot \text{sign}(dE(w)) \cdot dE_1^{\alpha} \tag{3}$$

where $dE_1 = |dE(w)|$ for $-dE_1 \leq dE(w) \leq dE_1$ and $dE_1 = dE_{\max}$ otherwise, α is a constant from $(0, 1)$ interval (frequently $\alpha = 0.5$ gives the best results). The parameter values for a and b can be typically: $a = 0$ for the output layer; $a \in [10, 20]$ for the hidden layer; $b \in [0.10, 0.20]$, and $dE_1 \in [5, 10]\sigma$, where σ is the standard deviation of gradient components in a given training cycle. Using this approximation Eq. (3) leads to an average reduction of the number of epochs required for convergence by 40-60%.

In the Numerical Gradient neural training algorithm [10] all weight changes are examined in the same point on the error surface and then a single step is made in the calculated direction. In VSS, sequential line searches are made in the direction of each individual weight (orthogonal directions). Once an approximate minimum is found in a given weight direction, the step is immediately made to that minimum and the possible changes of next weights are already examined in the new point. This is the main difference between VSS and numerical gradient algorithms. This difference causes VSS to be much more effective than numerical gradient. Using numerical approach such micro-iteration updates may be efficiently computed. In networks with many parameters, situations where this approach does not find the proper direction to the error minimum may theoretically happen, but they were never observed in our experiments.

Since only one weight is changed at a time, the input signals need to be propagated only through the fragment of the network in which the signals change as a result of the weight update. The remaining signals incoming to all neurons of hidden and output layers are remembered for each training vector in an array called “signal table”. All training data must be propagated through the entire network only once at the beginning of the training, thus filling in the signal table entries. 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 number of hidden and output neurons, respectively. After a single weight is changed only the appropriate entries in the signal table are updated. Calculating the value of sigmoidal transfer functions is the most time consuming part of the network training. Since VSS algorithm does not relay on analytical gradient the transfer functions do not have to be differentiable and an array implementing a staircase approximation to the transfer function (with at least 20 elements) was used, without compromising accuracy. Using this approximation together with the signal table increases efficiency of the algorithm for larger networks by more than two orders of magnitude. Without the signal table the staircase transfer function would reduce the calculations only by a few percent, because the signal table reduces the addition and multiplication operations sev-

eral orders of magnitude, reducing also at most several times the number of calculations of sigmoid function values.

The VSS algorithm takes advantage of the MLP error surface properties. The steepness of 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 Fig. (1) [7]. Therefore one can expect that an 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 have values that differ by orders of magnitude (3). Higher PCA components have significant values only at the beginning of the training, because at that stage training algorithms have to choose the proper direction. As the training approaches the final stage, the direction changes are usually slow. In each training cycle i the first guess of the optimal change of a given weight $dw(i)$ might be the previous $dw(i-1)$ value. A detailed experimental analysis of the line search behavior leads to the conclusion that in most cases the lowest number of calculations is obtained when a smaller value $dw(i) = 0.35\ dw(i-1)$ is taken, although statistically the ratio $dw(i)/dw(i-1)$ is close to 1. This is related to the way minima are calculated along each weight direction. Surprisingly, high precision calculations (e.g. by repeated parabolic interpolations) increase the number of the training cycles (the error surface is not always convex), therefore only a rough estimation of the step size dw in each direction is made.

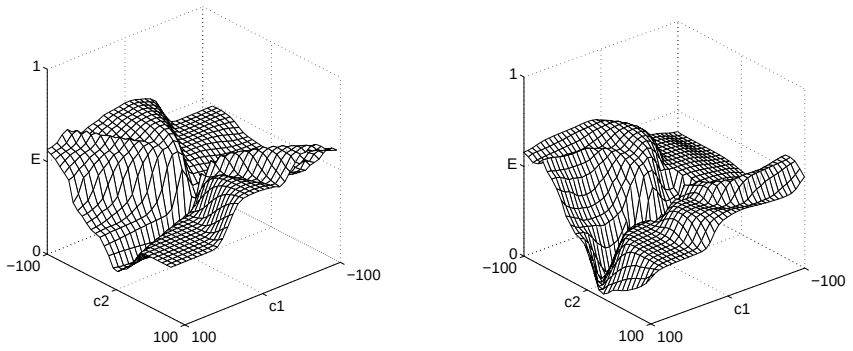

Fig. 1. Left: MLP error surface displayed in the first and second PCA direction (network structure 4-4-3, trained on the Iris dataset). Right: How the same error surface could look like using variable projection directions in attempt to display the error surface properties more faithfully.

Before the training starts the weights are initialized with random values taken from $(-1, +1)$ interval. Since $dw(0) = 0$, for each weight w in the first training cycle the first guess is $dw(1) = d0 \in [0.2, 0.3]$. The error surface ravine that leads to a good solution is narrow close to the starting point in the weight space, therefore $d0$ must be sufficiently small to keep the trajectory within the ravine. Although VSS algorithm has several heuristic parameters to speed up the calculations they are kept fixed in normal runs.

The number of epochs needed for VSS convergence is very small. In contrast to the standard batch-like training each epoch here consists of N_w microiterations (N_w is the number of weights). Fig. (2) shows the accuracy, MSE error and the total weight norm growth during the training of an MLP network with 4-4-3 structure on the Iris data. The error reaches minimum value already after two epochs. Analyzing directions of the weight changes in the first two PCA directions shows that in each iteration correct direction of the error function ravine is quickly found and maintained [10]. Although in the second-order algorithms (such as LM) the hidden layer weights grow quicker than in the first order ones, since the step component in a given weight direction is there approximately proportional to the ratio of the first to the second derivative, they still tend to be underestimated. VSS on the contrary does not estimate any weight changes but directly searches for the optimal changes. 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.

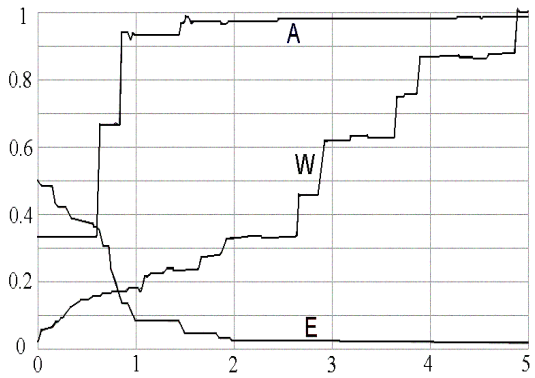

Fig. 2. MSE error (E), classification accuracy on the training set (A) and a normalized weight vector length $||W(i)||/||W(5)||$ (W) during the first 5 training cycles for the Iris (4-4-3) network.

VSS does not decrease the step when the gradient decreases, because it does not rely on the gradient information, but rather on the learning history contained in the trajectory, which frequently displays a repetitive pattern (3). Due to this the final part of the network training is relatively fast and the number of epochs needed for convergence is quite small, for simple data it can be as low as 2-3, and for complex data it may reach 20-30 epochs. But on the other hand it may lead to large weight values, making the network "overconfident" in its decisions and focused more on the error minimization then large margin generalization. To prevent the excessive weight growth either the training must be stopped early or a regularization term should be added to the error function (in complex cases this can give better results [3]). The purpose of the constants N_{max}, W_{max} and dW_{max} used in the pseudocode below is to limit weight growth. Alternatively the standard quadratic or Weigend [17] regularization term can be added to the error function. Without weight regularization VSS decreases the step when the

curvature of the ravine gets tighter and stops when the numerical gradient reaches zero values.

Symbols used in the VSS pseudo code: i - epoch number, n - number of iteration during the line search for a minimum in a single weight direction, $dw(i,n)$ - change of weight w after n-th iteration in i-th epoch relative to its previous value. Typical values of constants: $c1 = 0.33, c2 = 2, c3 = 0.3, d1 = 0.05$ for $i > 1$ and 0.2 for $i = 1, N_{max} = 4, W_{max} = 100, dW_{max} = 30$.

Algorithm 1. Variable Step Search

for $i = 1$ **to** *NumberOfEpochs* **do**
begin
  for $w = 1$ **to** *NumberOfWeights* **do**
  begin
    if $dw(i-1) = 0$ **then**
      $dw(i) = d1 * sign(w)$
    else
      $dw(i) = c1 * dw(i-1)$
    if $E(w + dw(i)) >= E(w)$ **then**
      $dw(i) = -dw(i)$ /*change the search direction*/
    if $E(w + dw(i)) >= E(w)$ **then**
    begin
      $dw(i) = 0$
      next w
    end
    for $n = 1$ **to** N_{max} **do**
    begin
      $dw(i) = c2 * dw(i)$ /* search for a minimum along this direction */
      if $|w| > W_{max}$ **or** $|dw| > dW_{max}$
      or $E(w + dw(i,n)) >= E(w + dw(i,n-1))$ **then**
        break
    end
    if $c3(E(w + dw(i,n-2)) - E(w + dw(i,n-1))$
      $> E(w + dw(i,n)) - E(w + dw(i,n-1))$ **then**
      $dw(i) = dw(i,n)$ /*accept that point in spite that the error in the previous point was a bit lower. It is likely to bring gain in the next training cycle */
    else
      $dw(i) = dw(i,n-1)/c2$ /* return to the previous point */
  end
end

The complexity of the above algorithm look like linear, but the complexity of calculating the error grows approximately like log N_w (it would grow linearly without using the signal table). Thus the total complexity of VSS is $O(N_w log N_w)$.

It is likely that a more efficient method exists that would reduce the average number of error calculations (that is about 3 times per one weight update) without increasing

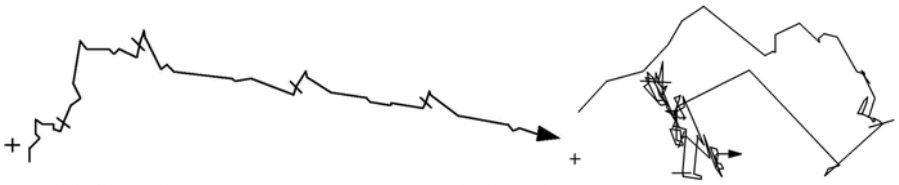

Fig. 3. Projection of MLP iris (4-4-3) learning trajectory trained with VSS; left: in the first and second PCA direction: right: int the third and fourth PCA direction. The cross shows the zero point in the weight space. The crosswise lines separate training cycles. Network structure 4-4-3, trained on the Iris dataset)

the total number of training epochs. It is possible that the optimal sequence has not been found so far, but it is quite likely that order of the parameter updates has only a weak influence on convergence. Therefore the weights are changed either in 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. After detecting that changing a given weight does not significantly reduce the error, the weight is frozen or pruned.

3 Experimental Results

Numerical experiments were made on several well-known benchmark dataset from the UCI learning repository [18]. For each training algorithm about 20 experiments were made with every dataset. The network was tested either on a separate test data (Thyroid, Shuttle) or using the 10-fold crossvalidation (Iris, WBC – Wisconsin Breast Cancer, Mushroom). A vector was considered to be classified correctly if its corresponding output neuron activation was larger then the other neuron signals and larger than 0.5. All training algorithms were run with the their default parameters, the same for each dataset.

Three measures to determine algorithm efficiency have been considered: the total computational complexity C_t required to achieve the desired effect, the quality of the solution the algorithm can find (% accuracy on the test set), and the percentage of the algorithm runs that converge to the desired solution (CR).

VSS and NG calculations were run using a program developed by one of us (MK). The Matlab Neural Network Toolbox (written by H. Demuth and M. Hagen) was used for LM and SCG calculations. LM and SCG algorithms were chosen because they are recommended for smaller and larger networks respectively [3]. Moreover, without taking into consideration time and memory requirements, LM seems to be the best among gradient-based MLP training algorithms [2]. To make the training times of different implementations directly comparable the computational complexity of the algorithms was assessed in the following way: first, the datasets were repeatedly propagated through the network calculating the MSE error S_n times, which took S_t time units. For larger datasets (Mushroom, Thyroid, Shuttle) the algorithms were run the average number of the training cycles T_n required to reach their convergence, which took T_t time units. For smaller datasets (Iris and Breast) VSS and SCG were given 100-fold more input vectors (for LM small dataset complexity could not be measured). The algorithm com-

putational complexity was calculated for a given dataset and network structure per one training cycle as: $C_e = (T_t/T_n)/(S_t/S_n)$. For VSS C_e was from 15 for smaller networks (Iris, WBC) to 50 for the larger networks (Mushrooms). For the LM algorithm C_e was between 9 and 80, rapidly growing with the network size. For SCG C_e did not depend much on the network size, staying between 2 and 6. The number of the 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, reflects the overall algorithm speed. To provide hardware independent measure it is calculated as the ratio of the total training time to the time needed for a single feedforward propagation of the whole dataset through the network. C_t can be obtained by multiplying the per training cycle complexity C_e by the average number of training cycles N_t required for convergence, $C_t = C_e N_t$. It must be clearly stated that the C_t values cannot be compared with high precision and may significantly vary depending on a given algorithm implementation, nevertheless they give quite useful estimation of the training complexity dependence on the properties of the dataset and the network architecture.

In all cases C_t for VSS was lower than that for LM. In most cases it was also lower than that for SCG, however for larger datasets this difference tends to vanish. Only VSS and LM were able to find the best solutions with the test accuracy frequently higher than the required minimum shown in Table 1. However, LM frequently did not converge to the solution and the training had to be repeated with new random weights. The CR parameter in Table 1. gives the percentage of the algorithm runs that converged to the desired solution in 250 epochs (1000 for Thyroid with SCG).

	data set	Iris	WBC	Mushroom	Thyroid	Shuttle
	net	4-4-3	10-4-2	125-4-3	21-4-3	9-6-7
	min % test	96.0	96.0	99.7	98.0	99.0
SCG	N_t	54	38	45	186	46
	MB	-	0.4	40	1.0	20
	CR	90	60	100	75	60
	C_t	245	165	90	619	238
LM	N_t	20	15	6.0	43	15
	MB	-	1.5	240	30	1400
	CR	80	85	90	60	60
	C_t	-	-	566	1333	1280
VSS	N_t	3.5	1.5	2.0	10	6.0
	MB	-	-	0.4	0.2	1.6
	CR	100	100	100	95	100
	$C_{t(sigm)}$	112	64	160	697	457
	$C_{t(stair)}$	84	38	64	366	287

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

For VSS the minimum and the maximum number of training cycles required for convergence to the specified accuracy differed less than 30% from the mean number N_t given in Table 1, while for LM the difference was often over 100%. VSS algorithm had also the smallest memory requirements. Memory usage in megabytes (MB) for storing network parameters, without including memory for the data set, was calculated by subtracting the memory used by the program running the algorithm on a given dataset

from the memory used by the program with the given dataset loaded in memory and running the algorithm on the XOR dataset. Without the signal table the VSS memory requirements would significantly decrease but also the training times would dramatically increase.

4 Conclusions

Numerical search-based techniques, popular in artificial intelligence, have been almost completely neglected by the neural network community, although they provide a good basis for neural training algorithms. Global stochastic optimization models based on genetic algorithms are quite popular [5], [19], but systematic search techniques are almost never used. VSS algorithm is based on heuristics deduced from analysis of the standard MLP training trajectories observed on error surfaces projected on the first two PCA components, as well as observations of single weight changes.

Many of these observations presented briefly in this paper (more details can be found in [10]) may be useful to improve traditional algorithms. For real-world datasets local minima on the error surface seem to be very rare. Local search algorithms based on analytical gradients that do not have direct access to the influence of hidden layer weights on the network error sometimes cannot determine the optimal next step direction and may fall in spurious local minima [12]. VSS does not fall in such minima and seldom requires repeated starts, getting stuck only in the case when there is really no way downwards from the starting point to one of the global minima.

State-of-the-art MLP training algorithms, such as the Levenberg-Marquardt or the SCG algorithm, are known to be highly effective. Surprisingly, results of numerical experiments (Tab. 1) show that MLP networks can be equally effectively trained with much simpler VSS algorithm. The simplicity of the VSS algorithm described in this paper makes the software implementation quite easy and requires almost no theoretical background to understand it. It runs fast, may use arbitrary transfer functions, can find very good solutions and has low memory requirements. So far its performance has been more than satisfactory. It is quite surprising that in empirical tests VSS tends to perform usually better then both scaled conjugate gradient and Levenberg-Marquardt algorithm, except for very large networks when the training times obtained with scaled conjugate gradient can be shorter than these with VSS. Combination of systematic search with large steps and traditional gradient-based techniques may be useful for MLP initialization if starting point far from small weights is required to find global optimum.

The algorithm was also modified to extract logical rules from data using the analysis of particular weight values. For that purpose a MLP2LN architecture was used [20], adding the first layer that discretizes the data (if needed) and assigning not only output but also hidden layer neurons to distinct classes [10].

Acknowledgement: WD is grateful for the support by the Polish Committee for Scientific Research, research grant 2005-2007.

References

1. D. Marquardt, “An algorithm for least-squares estimation of nonlinear parameters”, SIAM J. Appl. Math., 1963, Vol.11,pp.431-441.
2. N. N. R. Ranga Suri, D. Deodhare, P. Nagabhushan, “Parallel Levenberg-Marquardt-Based Neural Network Training on Linux Clusters - A Case Study”, Proc. 3^{rd} Indian Conf. on Computer Vision, Graphics & Image Processing, Ahmadabad, India, 2002.
3. S. Haykin, *Neural networks: a comprehensive foundations*, New York: MacMillian Publishing, 1994.
4. M. F. Moller, “A Scaled Conjugate Gradient Algorithm for Fast Supervised Learning”, Neural Networks, vol. 6, pp. 525-533, 1993.
5. W. Duch, J. Korczak, “Optimization and global minimization methods suitable for neural networks”, KMK UMK Technical Report 1/99
6. L. Hamm, B. Wade Brorsen, “Global Optimization Methods”, The 2002 International Conference on Machine Learning and Applications (ICMLA’02), June 2002, Las Vegas, Nevada, USA
7. M. Kordos, W. Duch, “A Survey of Factors Influencing MLP ErrorSurface”, Control and Cybernetics, vol. 33(4), pp. 611-631, 2004.
8. 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.
9. D. Saad (ed.), “On-Line Learning in Neural Networks”, Cambridge, UK: Cambridge University Press 1998.
10. M. Kordos, “Search-based Algorithms for Multilayer Perceptrons”,PhD Thesis, 2005, available from http://www.phys.uni.torun.pl/ ∼kordos
11. J. Denker et. al., “Large automatic learning, rule extraction and generalization”, Complex Systems, 1:887-922, 1987.
12. 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.
13. 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.
14. L. G. C. Hamey, “XOR Has No Local Minima: A case study in neural network error surface analysis”, Neural Networks, 11(4), pp. 669-681, 1998.
15. R. Hecht-Nielsen, “Neurocomputing”, Adison-Wesley, Reading, MA, 1990.
16. M. Lehr, “Scaled Stochastic Methods for Training Neural Networks”, PhD Thesis, Stanford University, 1996.
17. A. S. Weigend, D. E. Rumelhart, B. A. Huberman, “Generalization by Weight Elimination with Application in Forecasting”, Advances in Neural Information Processing Systems, San Mateo, CA, pp. 875-882, 1991.
18. C.J. Mertz, P. M. Murphy, UCI repository of machine learning databases www.ics.uci.edu/pub/machine-learning-databases
19. U. Seiffert, “Multiple Layer Perceptron Training Using Genetic Algorithm”, Proc. of the 9th European Symposium on Artificial Neural Networks ESANN 2001, Bruges, Belgium, April 25-27, pp. 159-164, 2001.
20. 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.