STUDIA INFORMATICA 2005
Volume 26 Number 1 (62)

Mirosław KORDOS
Politechnika Śląska, Instytut Informatyki

SEARCH-BASED APPROACH TO MULTILAYER PERCEPTRON TRAINING

Summary. The paper presents two search-based algorithms for MLP training; numerical gradient and variable step search algorithm. The advantages of the methods comparing to analytical gradient-based algorithms include low memory requirements, the algorithm simplicity and determining more optimal next step direction by direct access to the influence of hidden layer weights on network error.

Keywords: MLP, neural networks, search algorithms

METODY OPARTE NA POSZUKIWANIU W UCZENIU PERCEPTRONA WIELOWARSTWOWEGO

Streszczenie. Artykuł przedstawia dwie metody uczenia sieci MLP oparte na przeszukiwaniu: gradient numeryczny i metodę zmiennego kroku przeszukiwania (VSS). Zaletą tych metod są małe wymagania pamięciowe, prosta budowa algorytmu oraz dokładniejsze określenie kierunku następnego kroku przez bezpośrednie określanie wpływu wartości wag warstwy ukrytej na błąd sieci.

Słowa kluczowe: MLP, sieci neuronowe, algorytmy przeszukiwań

1. Introduction

Multilayer perceptrons (MLP) are usually trained using either analytical gradient-based algorithms with error backpropagation or global optimization methods. The most popular methods from the first group are: standard backpropagation [1], RPROP [2], Quickprop [3], Quasi-Newton [4], Levenberg-Marquardt [5] and conjugate gradients [6]. Also recursive least squares (RLS) methods [7] can be counted to that group, though they use different learning mechanism. The second group is represented by: genetic algorithms [8], simulated annealing

[9] and its variant Alopex [10], particle swarm optimization [11], novel [12], taboo search [13] and Linear Least Square Simplex [14].

The paper introduces two simple search-based methods that can be used for MLP training. Chapter 2 describes numerical gradient and chapter 3 variable step search algorithm. Both algorithms were designed with the help of some techniques for visualization of learning processes in MLP networks [15]. The chosen visualization techniques along with conclusions drawn from them are presented in chapter 2 and 3. Search-based methods were also successfully applied for logical rule extraction from special structures of MLP networks [16], this topic is however out of scope of the paper.

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

Chapter 4 compares the main features of analytical gradient-based, search-based and global optimization MLP learning algorithms. Experimental results are presented in chapter 5 and the final conclusions in chapter 6.

2. Numerical Gradient

The first version of numerical gradient (NG) for MLP training was first proposed in [17]. The current version of NG, described below allows for better estimation of the next step direction *dS* and thus for better convergence and shorter training times.

NG 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 a single weight *w* and the change of the error *dE(w)* is calculated as

$$dE(w) = [E(w) - E(w + dw)] \tag{1}$$

Various error measures can be used with NG and VSS, nevertheless in this paper we consider only MSE (mean square error).

The MLP error surface changes slower in the areas that are located further from its centre and therefore are reached by the learning trajectory at the final stage of the training, but mostly output layer weights contribute to slower changes (Fig. 1).

However, it would not be optimal to search for the minimum along the gradient direction. We tried to assess the statistically optimal search direction components *dS(w)* as a function of three variables: the network layer, the gradient value *dE(w)* and the epoch number:

(2)

The analysis of error surfaces supported by experiments showed that good results are obtained when the search component *S(w)* in the direction of the weight *w* is calculated as:

(3)

where the coefficients *c* and *α* optimize the line search direction. *c* is a variable and it equals 1 for the output layer and for the hidden layer it changes gradually from about 8÷12 at the initial phase of the training down to 1 at the final stage of the training. *α* is a constant from *(0;1)* and frequently α=0.5 gives the best results.

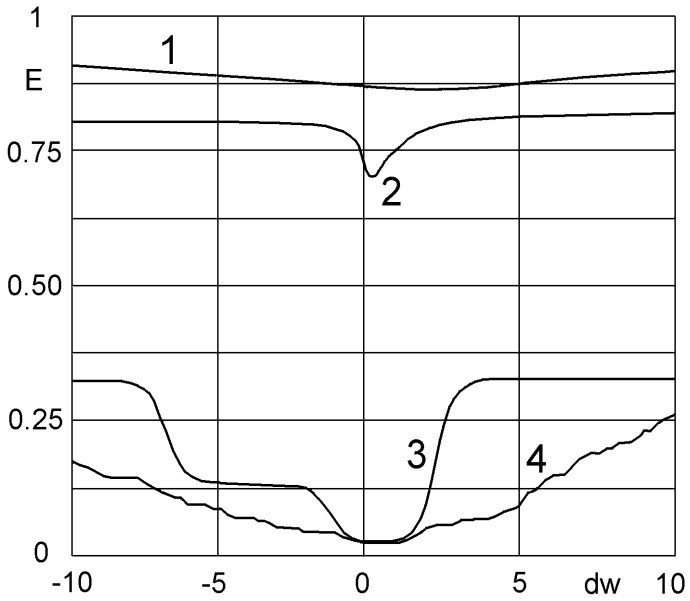

Fig. 1. Typical error surface sections in the direction of: hidden weight at the beginning of the training (1), output weight at the beginning of the training (2), output weight at the end of the training (3), hidden weight at the end of the training (4)

Rys. 1. Typowe przekroje powierzchni błędu w kierunku: wag warstwy ukrytej na początku uczenia (1), wag warstwy wyjściowej na początku uczenia (2), wag warstwy wyjściowej pod koniec uczenia (3), wag warstwy ukrytej pod koniec uczenia (4)

Neither the gradient direction (as in backpropagation) nor the direction dependent only on the sign of a derivative (as in RPROP) is the optimal direction. Levenberg-Marquardt (LM) algorithm on the one hand finds the optimal direction very well, but on the other hand its computational effort grows rapidly with network and dataset size. In practice it limits the use of LM algorithm to network of less than several hundred weights and less than several thousand training vectors (see chapter 5). The method used here aims at joining the low computational cost of one training epoch of first order methods (e.g. backpropagation, RPROP) with the low number of required training epochs of second order methods (e.g. LM).

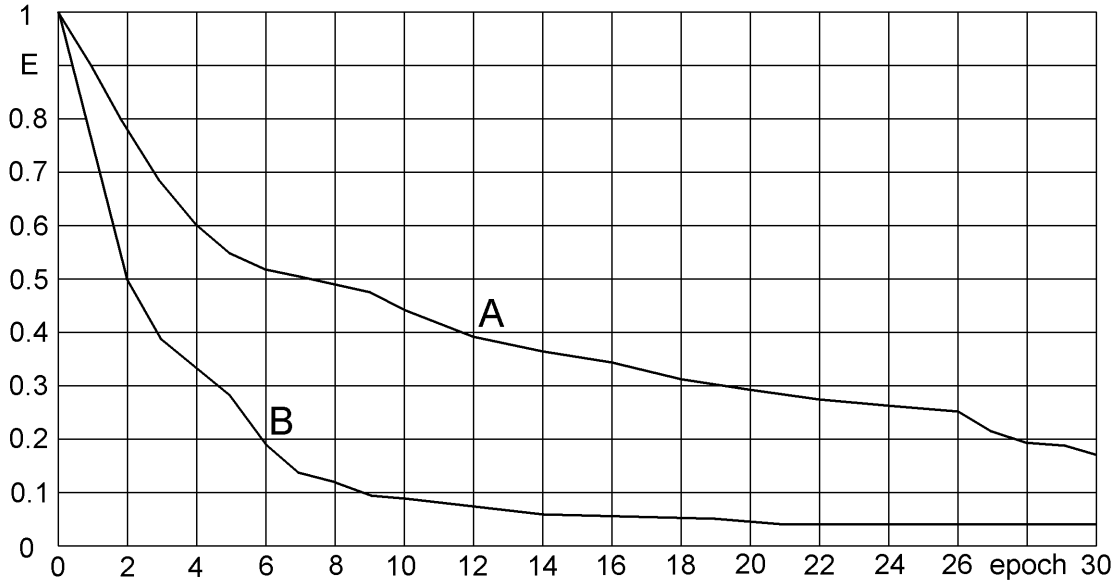

Fig. 2. MSE of *iris* (4-4-3) trained with NG; line A: steps in gradient directions, line B: steps in the optimized direction given by formula (3)

Rys. 2. Błąd średniokwadratowy podczas uczenia sieci o strukturze (4-4-3) na zbiorze *iris*; lina A: poruszanie się w kierunku gradientu, linia B: poruszanie się w kierunku zoptymalizowanym, określonym wzorem (3)

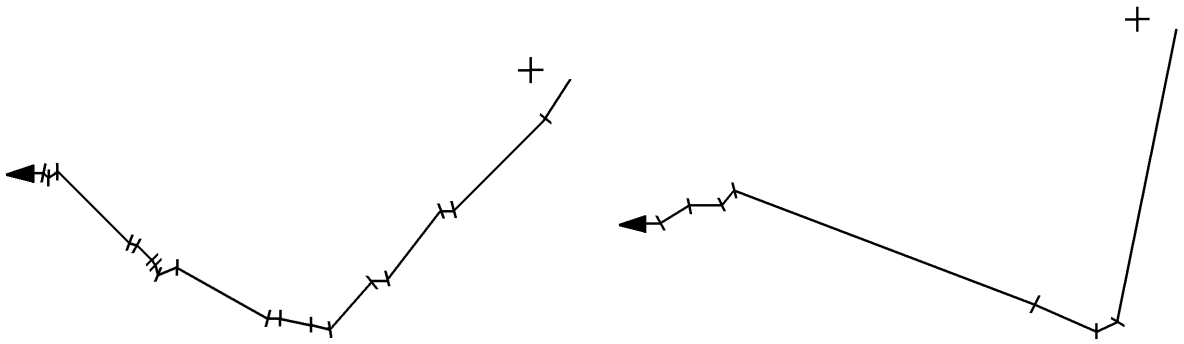

Fig. 3. Projection of *iris* (4-4-3) learning trajectory trained with NG in the first and second PCA direction: left-steps in the gradient direction, right-steps in the optimized direction. The cross shows the zero point in the weight space.The crosswise lines separate training cycles

Rys. 3. Projekcja trajektorii uczenia w pierszym i drugim kierunku PCA. Sieć (4-4-3) uczona gradientem numerycznym na zbiorze *iris*; po lewej - poruszanie się w kierunku gradientu, po prawej – poruszanie się w kierunku zoptymalizowanym. Krzyżykiem zaznaczono punkt zerowy w przestrzenii wag. Kreski ukośne oddzielają poszczególne cykle treningowe

Since only one weight is changed at a time while searching for the gradient direction, the signals need not 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 weight change. The remaining sums of 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 need not be calculated again if a weight of another output neuron is changed. The signal table significantly shortens training times, especially for bigger networks. For a network structure 125-8-2 the acceleration is about 30 times.

The minimum along the search direction can be located with many line search methods (e.g. parabolic interpolation or Brent's search).

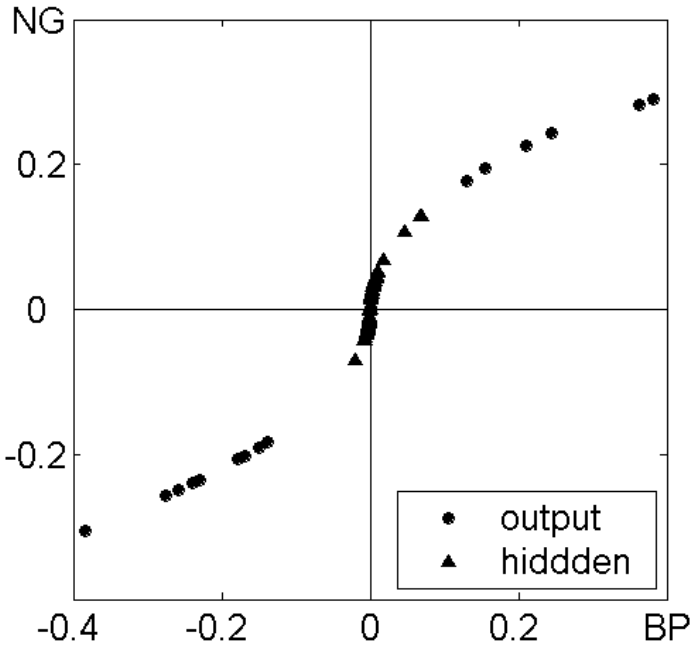

Fig. 4. Comparison of numerically and analytically determined gradient components in particular weight directions for *thyroid* (21-4-3) dataset

Rys. 4. Porównanie numerycznie i analitecznie wyznaczonych składowych gradientu w kierunku poszczególnych wag dla zbioru danych *thyroid* (21-4-3)

It is interesting to compare gradient direction determined analytically by backpropagation algorithm (the same direction is used by all algorithms that use the backpropagation mechanism to determine gradient) to gradient direction determined numerically (not the search direction given by (3)). To obtain pure gradient direction we use $\alpha=1$ and $c=1$ in (3) for every training cycle and $dw<0.05$ (any $dw<0.05$ gives practically the same gradient direction as $dw=0.05$). In the paper we use sigmoids with slopes equal one as neural transfer functions if not explicitly stated others.

The main distortion caused by analytical gradient is that small gradient values (frequently hidden weight components at the beginning of the training) are interpreted as smaller than they really are and big ones as still bigger.

That results firstly in too long training times and secondly in falling in spurious local minima. With numerically determined gradient directions, the convergence requires fewer epochs (NG with directional minimization compared to BP with directional minimization and NG with constant step compared to BP with constant step). Moreover, BP much more frequently does not converge at all. The effect is known as falling in spurious local minima, which are only in the backpropagation-estimated gradient direction, but it does not exist in the real gradient direction. In our experiments the networks were trained with BP. When they stacked in a "local minimum", then from the same point in the weight space the trainings continued with NG frequently managed to leave the apparent minimum and finally converge. The momentum term can be implemented with NG in an analogue way as in BP and gives similar effects.

Calculating the sigmoid value is the most time consuming part of network training. Since NG and VSS do not relay on analytical gradient, the transfer functions do not have to be differentiable and computationally cheap staircase transfer functions (with 20 or more stairs) can be used, thus reducing training times by 20-40% without compromising accuracy.

3. Variable Step Search algorithm

Variable Step Search algorithm for MLP training was first proposed in [18]. 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 then the change is kept, otherwise it is rejected. Then dw is added to or subtracted from the next weight and again the error is calculated, until the changes of all weights are examined.

VSS is a modified version of the simplest search-based algorithm, in which dw is not constant, but dynamically adjusted independently for each weight during a rough error 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 directions. Basing on the properties we 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 differ ranks of order.

In each training cycle i the first guess of $dw(w,i)$ for a given weight w might be the value $dw(w,i\text{-}1)$ that the weight changed about in the previous training cycle. However, the detailed

experimental analysis of the algorithm behaviour leads to the conclusion that in most cases the least amount of calculations is obtained when the first guess is $dw(w,i)=(0.3\div0.4)*dw(w,i\text{-}1)$, though statistically the ratio $dw(w,i)/dw(w,i\text{-}1)$ is close to 1.

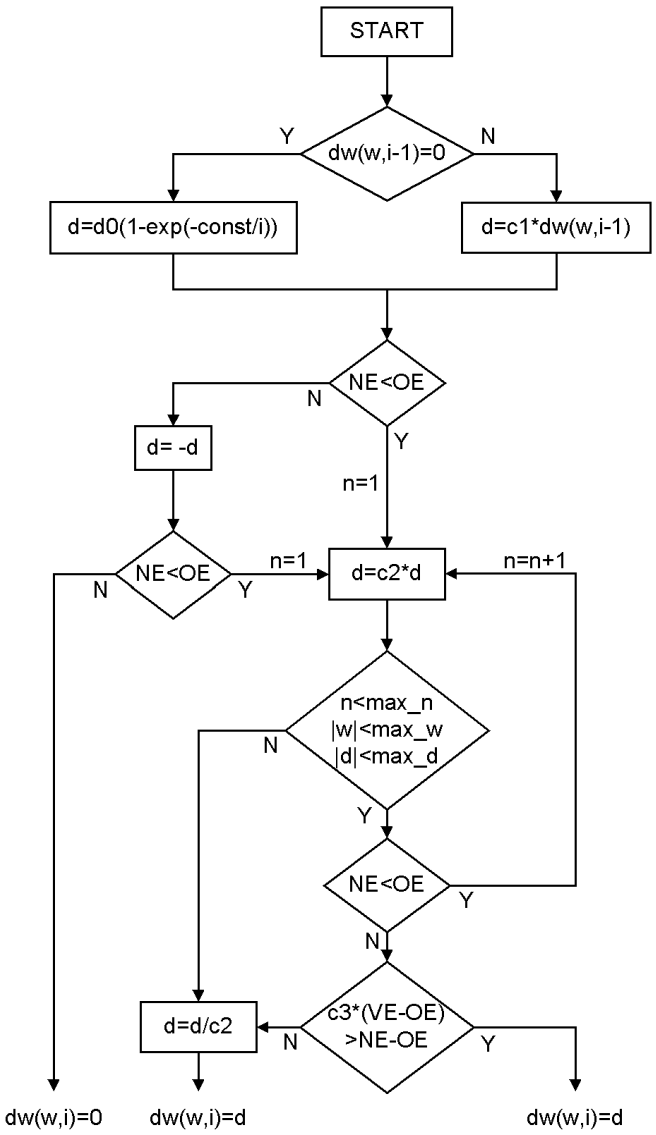

Fig. 5. Determining a single weight value in one training cycle of the Variable Step Search Algorithm

Rys. 5. Określanie wartości pojedynczej wagi w jednym cyklu uczenia metodą Zmiennego Kroku Poszukiwania (VSS)

Fig. 5 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 before the weight change is applied and *NE* (New Error) is the MSE error after the weight change is applied. *VE* is the last error before *OE*, i.e. $NE=error(n)$, $OE=error(n\text{-}1)$, $VE=error(n\text{-}2)$. The parameters *max_n* (maximal number of iterations), *max_w* (maximal absolute value of a weight) and *max_d* (maximal change of a weight in one training cycle) are given to prevent the weight from excessive

growth. The parameters are optional and can be set to infinity. Experimentally determined optimal values of other parameters are: $c1=0.30\div0.40$, $c2=1.7\div3.0$, $c3=0.1\div0.4$, however the algorithm is not very sensitive to the parameter values.

Before the training starts, the weights are initialized with random values from the interval (-1;1). In the first training cycle $d=d0=0.2\div0.3$. Since $dw(w,0)=0$, for each weight w in the first training cycle the first guess is $dw(w,1)=d0$. Since the ravine on the error surface is narrow close to the algorithm starting point, $d0$ must be enough small to keep the trajectory within the ravine.

On average the error is calculated about 3 times while determining a single weight value in one training cycle. Many attempts were made to use the statistical distributions of weight changes, parabolic interpolation and PCA/ICA/kernel PCA/Principal Curves directions to speed up the network training, but so far they were successful only for some datasets. As a universal approach, the version of VSS algorithm presented here is the best solution found so far.

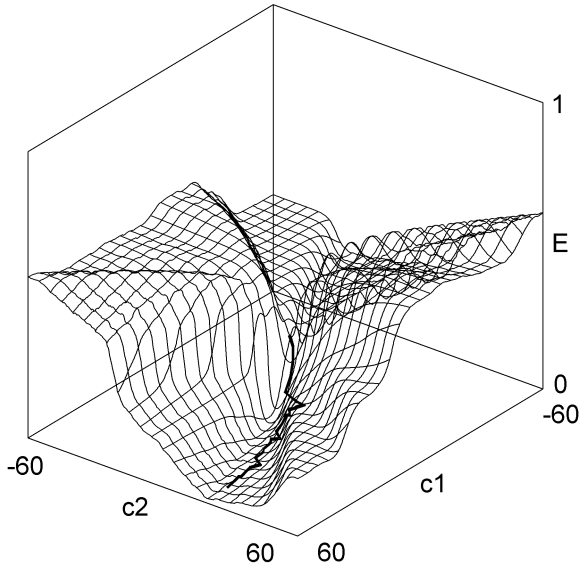

Fig. 6. PCA-based projection of *iris* (4-4-3) error surface with visible VSS learning trajectory

Rys. 6. Projekcja w dwóch kierunkach PCA powierzchni błędu zbioru *iris* (4-4-3) z widoczną trajektorią uczenia metodą VSS

The first and second PCA directions typically contain together over 95% of the learning trajectory variance. Therefore the trajectory projections in Fig. 6 and Fig. 7-left reflect the learning trajectory properties very well. Higher PCA components have significant influence only at the beginning of the training, where the trajectory changes its direction quickly (Fig. 7 – right).

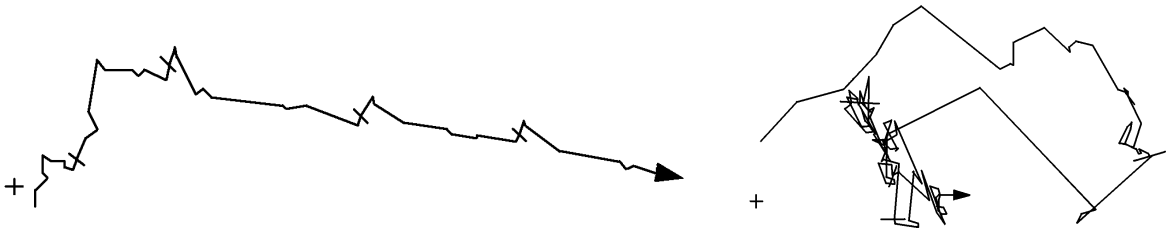

Fig. 7. Projection of *iris* (4-4-3) learning trajectory trained with VSS; left: in the first and second PCA direction, right: in the third and fourth PCA direction. The cross shows the zero point in the weight space. The crosswise lines separate training cycles

Rys. 7. Projekcja trajektorii uczenia zbioru *iris* (4-4-3) uczonego metodą VSS; po lewej w pierwszym i drugim kierunku PCA, po prawej: w trzecim i czwartym kierunku PCA. Krzyżykiem oznaczono punkt zerowy w przestrzeni wag. Kreski ukośne oddzielają poszczególne cykle treningowe

4. Comparison of Analytical Gradient-based, Global and Search-based MLP Training Methods

Analytical gradient-based methods use error backpropagation to calculate the partial error derivatives with respect to every weight. Some methods use not only the first but also the second partial derivatives. After the derivatives are calculated various methods use various weight update algorithms. The first order methods (standard backpropagation, RPOP, Quickprop) have low computational cost per training cycle, since after once propagating each signal forward through the network only the cumulative errors are once propagated backward and not much additional calculation is performed. But due to the fact that the next step direction determined by the methods cumulates the errors of two assumptions (firstly the backpropagated derivatives differ from the true derivatives and as a result so obtained gradient direction deflects from the true gradient direction and secondly neither the gradient direction not the direction given by RPROP is the optimal next step direction), they require many training cycles and frequently get stacked in spurious local minima. Second order methods, such as Levenberg-Marquard algorithm, find the next step direction in a much more optimal way, but they have greater computational cost per training cycle. Moreover, the cost grows rapidly with the network and dataset size. Thus, they perform well but only for small networks. All analytical gradient-based methods require differentiable transfer functions, what excludes using computationally cheap non-differentiable functions, such as staircase or linear saturated functions.

Global optimization methods, such as Alopex, tabu search, novel or evolutionary algorithms, require ranks of order more calculations than local methods. Since they are not starting point dependent, they have greater chance to find the optimal solution for difficult

problems, such as the artificial dataset of two spirals. The best solution found for two spiral problems with local methods required about 10 hidden neurons. Global methods solved the problem using only 5 hidden neurons but the training times were several hundreds to some thousand times longer than with local methods. However, for the real-world datasets the need for using global optimization methods is very rare. More complex search algorithms, such as beam search, usually do not perform better than using a single trajectory and since they are computationally more costly it is not advocated to use them, unless absolutely necessary.

The search-based methods perturb usualy one weight at a time and then calculate the error to assess the optimal change of the weight in the next step. Since it would require more calculations than in analytical gradient-based methods, to minimize the computational cost, the signals are propagated only through the recently modified fragment of the network. Other signals and errors are remembered in a special array called signal table. The first advantage of these methods over analytical gradient-based ones is that they have direct access to hidden layer weights influence on network error. That allows determining the next step direction more precisely. The next advantage is that they do not use analytical gradients thus the transfer functions do not have to be differentiable and computationally cheap transfer functions as staircase functions can be used.

There are several known improvements that can be used with many different algorithms, as weight pruning basing on various criteria, weight freezing, calculating the error not on the entire training set but only on a different part of it in each epoch, eliminating some vectors from the further training or replacing them with a single vector. The methods are not discussed here in detail since they are not specific only to search-based algorithms. Nevertheless it should be taken into consideration that implementing the methods can significantly accelerate the training processes. Moreover, weight pruning also frequently improves network generalization.

5. Experimental Results

The numerical experiments were conducted on some well known benchmark dataset from the UCI learning repository. The datasets and their detailed description can be found in [20]. For each training algorithm about 20 experiments were made with every dataset. The network was tested on test sets (*thyroid, shuttle*) or in 10-fold crossvalidation (*iris, Wisconsin breast, mushroom*). A vector was considered to be classified correctly if its corresponding output neuron signal was higher then other neurons signal and than 0.5. All training algorithms were run with the their default parameters, which were the same for each dataset.

Three values 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 us in Delphi. Matlab Neural Network Toolbox (written by H. Demuth and M. Hagen) was used for LM and SCG calculations. Since the times between Matlab and our program could not be directly compared, the computational complexity of the algorithms was assessed in the following way: first only the datasets were repeatedly propagated through the network with calculating the MSE error S_n times (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 required 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. A given algorithm computational complexity was calculated for given dataset and network structure per one training cycle as: $C_e=(T_t/T_n)/(S_t/S_n)$.

The total computational complexity C_t shown in Table 1. reflects the algorithm speed. It expresses 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: $C_t=C_eN_t$. It is clear that C_t cannot be calculated very precisely and it will surely vary depending on a given algorithm implementation, nevertheless it provides quite useful outlook.

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 that are relatively easy to train, e.g. *mushroom*, the difference vanishes.

Only VSS and LM were able to find the best solutions with *%test* frequently higher than shown in Table 1. 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, i.e. 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 converged 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. The performance of NG was poorer than that of VSS. The main difference between the algorithms is that NG uses directly gradient information, while VSS does not.

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	3.0	-	100	53	11	-	100	175	20	-	80	223	118	-	90	948
4-4-3	98	4.3	-	100	76	18	-	100	286	22	-	80	246	157	-	90	1260
breast 10-4-2	97.5	1.0	-	100	30	8	-	100	224	20	1.5	100	109	147	0.4	60	271
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	0.4	100	206	-	0.4	0	-	6	240	90	3260	45	40	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	0.2	100	643	-	0.2	0	-	35	30	40	2300	-	1.0	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	1.6	100	564	58	1.6	90	1740	19	1400	60	1940	1620	20	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 was 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.

6. Conclusions

It is clear that search-based techniques, popular in artificial intelligence and neglected in neural networks can be the basis for 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. It is fast, can find very good solutions and has low memory requirements. It is also very simple to program because 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.

For real-world datasets local minima in craters are extremely rare on the error surface [19]. Local search algorithms based on analytical gradient that do not have direct access to the influence of hidden layer weights on network error cannot precisely determine the gradient direction and fall in spurious local minima. Neither NG nor VSS falls in spurious minima and

seldom requires multistart only in that case, when there is really no downwards way from the starting point to one of the global minima.

Although local optimization methods including search-based ones do not guarantee finding a global minimum for every problem, for prevailing number of real-world problems they are sufficient and it is rarely required to use global optimization methods, which on the one hand have greater chance to find the solution for complex problems but on the other hand require much higher computational effort.

REFERENCES

1. Rumelhart, Hinton G. E., Williams R. J.: Learning Internal Representations by Error Propagation. Parallel Data Processing, Vol.1, Chapter 8, the M.I.T. Press, Cambridge, 1986, pp. 318-362.
2. Riedmiller M., Braun H.: RPROP – a Fast Adaptive Learning Algorithm. Technical Report, University Karlsruhe, 1992.
3. Fahlman S. E.: Faster Learning Variations of Backpropagation: an Empirical Study. Connectionist Models Summer School, Morgan Kaufmann, 1998, pp. 38-51.
4. Battiti R.: First and second order methods for learning: Between steepest descent and Newton's method. Neural Computation, vol. 4, no. 2, 1992, pp. 141-166.
5. Marquardt D.: An Flgorithm for Least-squares Estimation of Nonlinear Parameters. SIAM J. Appl. Math., 1963, Vol.11, pp. 431-441.
6. Möller M. F.: A Scaled Conjugate Gradient Algorithm for Fast Supervised Learning. Neural Networks, vol. 6, 1993, pp. 525-533.
7. Bilski J., Rutkowski L.: Numerically Robust Learning Algorithms for Feed Forward Neural Networks. Th 6th International Conference on Neural Networks and Soft Computing, Zakopane, Poland, June 2002, pp. 149-154.
8. Montana D. J.: Neural Network Weight Selection Using Genetic Algorithms. Intelligent Hybrid Systems, Wiley, New York, 1995, pp. 85-104.
9. Engel J.: Teaching Feed-forward Neural Networks by Simulated Annealing. Complex Systems 2, 1988, pp. 641-648.
10. Unnikrishnan K. P., Venugopal K. P.: Alopex: A Correlation-Based Learning Algorithm for Feed-Forward and Recurrent Neural Networks. Neural Computations, 6, 1994, pp. 469-490.
11. Koosh V. F.: Analog Computation and Learning in VLSI. PhD Thesis, Caltech, Pasadena, CA, 2001.

12. Shang Y., Wah B. W.: Global Optimization for Neural Network Training. IEEE Computer, 29, 1996, pp. 45-54.
13. Battiti R., Tecchiolli G.: Training Neural Nets with the Reactive Tabu Search, Transactions on Neural Networks, vol.6, 1995, , pp. 1185-1200.
14. Gupta H. V., Hsu K., Sorooshian S.: Superior Training of Artificial Neural Networks Using Weight-Space Partitioning. International Conference on Neural Networks. Houston, USA, 1997, pp. 1919-1923.
15. Gallagher M.: Multi-layer Perceptron Error Surfaces: Visualization, Structure and Modeling. PhD Thesis, University of Queensland, 2000.
16. Kordos M., Duch W.: Search-based Training for Logical Rule Extraction by Multilayer Perceptron. International Conference on Artificial Neural Networks (ICANN) Istanbul, Turkey, 2003, pp. 126-129.
17. Kordos M., Duch W.: Multilayer Perceptron Trained with Numerical Gradient. International Conference on Artificial Neural Networks (ICANN), Istanbul, Turkey, 2003, pp. 106-109.
18. Kordos M., Duch W.: Variable Step Search Algorithm for MLP Training. The 8th IASTED International Conference on Artificial Intelligence and Soft Computing, Marbella, Spain, 2004, pp. 215-220.
19. Kordos M., Duch W.: On Some Factors Influencing MLP Error Surface. The 7th International Conference on Artificial Intelligence and Soft Computing (ICAISC), Zakopane, Poland, 2004, pp. 217-222.
20. Mertz C. J., Murphy P.M.: UCI repository of machine learning databases, http://www.ics.uci.edu/~mlearn/MLRepository.html

Recenzent: Dr inż. Krzysztof Cyran

Wpłynęło do Redakcji 13 września 2004 r.

Omówienie

W artykule przestawiono dwie metody uczenia sieci MLP oparte na algorytmach poszukiwania: gradient numeryczny (NG) i metodę zmiennego kroku poszukiwania (VSS). NG oblicza gradient numeryczny w danym punkcie metodą perturbacji każdej wagi kolejno i badania zmienności błędu sieci w odpowiedzi na tę zmianę. Pokazano różnice między rzeczywistym kierunkiem gradientu a kierunkiem otrzymanym w wyniku propagacji

wstecznej błędu (rys. 4), oraz między kierunkiem gradientu a optymalnym kierunkiem następnego kroku, wzdłuż którego NG znajduje przybliżone minimum (rys. 1÷3). VSS nie korzysta z żadnej informacji o gradiencie (rys. 5), lecz wykorzystuje własność powierzchni błędu sieci MLP, polegającą na tym, że doliny, w których leżą trajektorie, są wolno zakręcającymi wąwozami, a zatem optymalna zmiana danej wagi w dwóch sąsiednich epokach różni się niewiele (rys. 6). Zostały pokazane projekcje w dwóch kierunkach PCA powierzchni błędu (rys. 6) oraz trajektorii różnych algorytmów dla przykładowego zbioru danych (rys. 3 i 7). Zarówno NG, jak i VSS zmieniają wagi tylko w nieznacznej części sieci na raz i tylko przez tę część są każdorazowo przepuszczane sygnały (to znacznie ogranicza nakład obliczeniowy). NG i VSS nie korzystają z pochodnych analitycznych, dlatego pozwalają na używanie tanich obliczeniowo nieróżniczkowalnych funkcji transferu neuronów, np. funkcji schodkowej, która przy odpowiedniej ilości schodków daje dokładności rozwiązań identyczne z funkcją sigmoidalną, natomiast skraca znacząco czas obliczeń. W eksperymentach przeprowadzonych na typowych testowych zbiorach danych z UCI Repository porównano złożoność obliczeniową, stabilność, wymagania pamięciowe i jakość osiąganych rozwiązań dla NG, VSS, algorytmu Levenberga-Marquardta oraz metody skalowanych gradientów sprzężonych (tab. 1). Dla większości zbiorów danych VSS osiągał najlepsze wyniki w każdym z tych kryteriów. Prowadzi to do wniosku, że zastąpienie metod gradientowych prostym algorytmem, który stara się przewidzieć optymalną zmianę wag w danej epoce na podstawie zmiany wag w poprzedniej epoce, jest w pełni uzasadnionym kierunkiem rozwoju metod uczenia sieci MLP.

Adres

Mirosław KORDOS: Politechnika Śląska, Instytut Informatyki, ul. Akademicka 16,
44-101 Gliwice, Polska, mkordos@acserwis.com.pl