Weight Update Sequence in MLP Networks

Mirosław Kordos[1], Andrzej Rusiecki[2], Tomasz Kamiński[1], and Krzysztof Greń[1]

[1] University of Bielsko-Biala, Department of Mathematics and Computer Science, Bielsko-Biała, Willowa 2, Poland
mkordos@ath.bielsko.pl

[2] Wroclaw University of Technology, Institute of Computer Engineering, Control and Robotics, Wrocław, Wybrzeże Wyspiańskiego 27, Poland
andrzej.rusiecki@pwr.edu.pl

Abstract. The advantages of Variable Step Search algorithm - a simple local search-based method of MLP training is that it does not require differentiable error functions, has better convergence properties than backpropagation and lower memory requirements and computational cost than global optimization and second order methods. However, in some applications, the issue of training time reduction becomes very important. In this paper we evaluate several approaches to achieve this reduction.

1 Introduction

The most popular methods used to train MLPs are analytical gradient-based algorithms using error backpropagation (BP). These algorithms include standard gradient descent, resilient backpropagation (RPROP), Quickprop, Levenberg-Marquardt algorithm , several versions of conjugate gradients or the scale conjugate gradients methods [2]. Regular gradient-based algorithms are usually able to obtain satisfiable solutions, which are not necessarily global optima.

Another group of methods consists of global optimization techniques such as genetic algorithms [3], simulated annealing [4] and its variants, particle swarm optimization or tabu search [5]. These global methods are computationally expensive (especially with evolutionary approach) and also do not guarantee better network performance.

In this paper, we consider the third group of methods, namely algorithms based on local search techniques [1,7], in particular Variable Step Search Algorithm (VSS) first described in [9] and then successfully applied in [10,11]. The idea of the method was based on inspection of the learning process, which helped in formulating basic rules to change one weight at time. The VSS supports non-differentiable transfer and error functions, what was crucial in our applications [10,12]. Also minimizing the network training time was crucial and therefore we especially address this issue in this paper.

The main focus of this work is the discussion of how to reduce the computational cost of the method. In the next sections we describe the basic VSS algorithm and its novel variants with various weight update schema including, among others, updating only selected weights, discarding weights for which the update did not reduce the error, and combination of these approaches. Section 4 presents experimental results for 5 classification and 5 regression benchmark tasks, where network performances for the

E. Corchado et al. (Eds.): IDEAL 2014, LNCS 8669, pp. 266–274, 2014.

tested methods and several levels of computational efforts are compared. Based on the simulation results, general conclusions are formulated in the last section.

2 The VSS Algorithm

Because of the need for MLP learning algorithms that work with non-differentiable error functions, and a high computational cost of the known global search methods that can address this issue, we began experimenting with determining the gradient direction numerically instead of analytically. Thus we changed each weight a little and measured how it affected the network error. Then the weight was restored to its original value and the next weight was examined. The gradient direction we obtained was very close to the gradient direction obtained by the backpropagation algorithm, but the computational cost was even higher, due to the need of determining the network error as many times in one epoch as the number of weights. However, we noticed that if changing one weight by dw caused the error decrease and we did not restore the weight to its original value but try to change the next weight in that new point, we obtained much better and faster convergence of the algorithm (the convergence abilities were comparable to those of the Levendberg-Maguardt (LM) algorithm [9]). That is the basic idea of the VSS algorithm. Additional advantage of the VSS algorithm is that it can be applied with any feedforward network structure. It can be directly used to train deep neural architectures [6], eg. with 10 hidden layers (in contrary to backpropagation) and after introducing some modifications, which are described later, it is suitable for big networks, when the training time increases much slower than that of the LM algorithm.

Because of these advantages, especially that we have to use non-differentiable error functions, we recently used the VSS algorithm for hundreds of thousands of extensive tests [9]. Then the problem of decreasing the training time became crucial, and even as little time reduction as 20% was noticeable.

3 Reduction of Computational Cost

To reduce the training time we introduced the following enhancements: individually adjusted changes of each weight, remembering signals in a table and, recently, optimization of weight probing points. The paper focuses on the third enhancement, but before discussing it we shortly introduce the idea of the first and second one, to present a full picture.

Individually Adjusted Weight Changes. The algorithm starts with random weights and the initial $dw(i) = 0.1$ for each weight $w(i)$ in the output layer and $dw(i) = 0.3$ in the hidden layer. Then it individually adjust $dw(i)$ for each weights; $dw(i)$ becomes the value by which the weight changed in the current epoch. When the next epoch begins, the initial guess of the optimal change of each weight is $d1 \cdot dw(i)$, because as the experiments showed, the weights tend to keep the approximate proportion between their changes in two consecutive epochs. For the same reason when we have to reverse the direction of changing a weight, we decrease the step, multiplying the previous step by $0.5 \cdot d1 \cdot dw(i)$ instead $d1 \cdot dw(i)$.

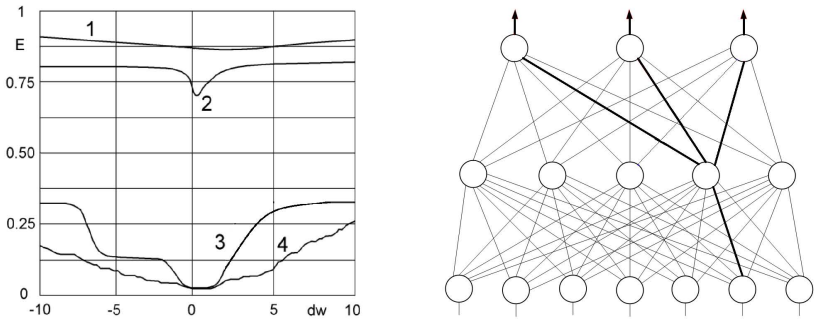

Fig. 1. Left: Typical error surface cross-sections in the direction of: (1) hidden weight at the beginning of the training; (2) output weight at the beginning of the training, (3) output weight at the end of the training, (4) hidden weight at the end of the training. Right: The idea of Signal Table (When the weight between the 4th hidden and 6th input neuron is modified, the signals are propagated only through the connections shown in bold).

Signal Table. As we change only one weight, there is no reason to propagate the signals through the entire network every time. That must be done only once, at the beginning of the training, to fill the "signal table" - an array, which stores the input and the output signals for each neuron for each training vector. Then, when we modify a single weight, we update only the portion of the signal table, which was affected by this change (the bold lines in figure 1-right in case of modifying a hidden layer weight and only one input and output of a single neuron in case of modifying an output layer weight). That allows reducing the computational cost by up to 98%, depending on the network size and structure.

Optimization of Weight Probing Points. The questions are: what is the optimal sequence of particular weight evaluation, in what direction should the search go and what is the optimal precision of the search? Finding the minimum on the error surface in a particular weight direction requires probing the error value in several points. The more points we use, we are able to find the minimum more precisely but on the other hand the computational cost of such a search is higher. There are some methods known from numerical analysis, such as the three-point formula [8], however the problem of minimizing the computational cost while modifying one weight at a time is so specific, that the general methods cannot be successfully applied to it. For that reason, in this paper, we evaluate different strategies (see Algorithms 1 and 2), where the assessment criterion is the network error (MSE) after a given training time expressed by the computational effort (ce):

1. d1 - probing the error value in one point in given direction
2. d1d2 - probing the error value in two points in given direction
3. d1d2d3 - probing the error value in three point in given direction
4. parabola - probing the error in two points and going to the parabola vertex

We tested all the four strategies with "superweights" ("s" in tables 1-3), with weight freezing ("f") and with "superweights" and weight freezing ("sf"). That gave together

Algorithm 1. VSS-d1, VSS-d1d2 and VSS-d1d2d3 algorithm

for $n = 1$ to numberOfEpochs **do**
  for $i = 1$ to numberOfWeights **do**
    $E_0 = E_{min}$
    $w(i,n) = w(i,n-1) + d1 \cdot dw(i)$
    if $E < E_{min}$ **then**
      $E_{min} = E$
      if $d2$ **then**
        $w(i,n) = w(i,n) + d2 \cdot dw(i)$
        if $E < E_{min}$ **then**
          $E_{min} = E$
          if $d3$ **then**
            $w(i,n) = w(i,n) + d3 \cdot dw(i)$
            if $E < E_{min}$ **then**
              $E_{min} = E$
            else
              $w(i,n) = w(i,n) + d2 \cdot dw(i)$
            end if
          end if
        else
          $w(i,n) = w(i,n-1) + d1 \cdot dw(i)$
        end if
      end if
    else
      $w(i,n) = w(i,n-1) - 0.5 \cdot d1 \cdot dw(i)$
      if $E < E_{min}$ **then**
        $E_{min} = E$
        if $d2$ **then**
          $w(i,n) = w(i,n) - 0.5 \cdot d2 \cdot dw(i)$
          if $E < E_{min}$ **then**
            $E_{min} = E$
            if $d3$ **then**
              $w(i,n) = w(i,n) - 0.5 \cdot d3 \cdot dw(i)$
              if $E < E_{min}$ **then**
                $E_{min} = E$
              else
                $w(i,n) = w(i,n) - 0.5 \cdot d2 \cdot dw(i)$
              end if
            end if
          else
            $w(i,n) = w(i,n-1) - 0.5 \cdot d1 \cdot dw(i)$
          end if
        end if
      end if
    end if
    if $E_0 > E_{min}$ **then**
      $dw(i) = w(i,n) - w(i,n-1)$
    else
      $w(i,n) = w(i,n-1)$
      $dw(i) = 0.5 \cdot dw(i)$
    end if
  end for
end for

16 combinations. By superweights we mean the 25% of weights, which when changed resulted in the greatest error reduction. Then the superweights are once again adjusted in the same training cycle before next training cycle begins. There are usually also several weights, which when changed did not cause the error decrease. They can get frozen for the next training cycle, because it is likely that changing them in the next cycle will reduce the error only to a minimal extend (or not at all) and thus the computational effort can be better used for examining other weights in the next training cycle. The experimentally determined optimal $dw1$, $dw1$, $dw3$ values were close to 1.5 and that values we used.

Algorithm 2. VSS-parabola algorithm

for $n = 1$ to numberOfEpochs **do**
  for $i = 1$ to numberOfWeights **do**
    $E_0 = E$
    $w_0 = w(i, n)$
    $w_1 = w(i, n) = w(i, n-1) + d1 \cdot dw(i)$
    if $E_1 = E < E_{min}$ **then**
      $w_2 = w(i, n) = w(i, n) + d2 \cdot dw(i)$
    else
      $w_2 = w(i, n) = w(i, n) - 0.5 \cdot \ d1 \cdot dw(i)$
    end if
    $E_2 = E$
    $w(i, n) = vertexOfParabola(w_0, E_0, w_1, E_1, w_2, E_2)$
    if $parabolaIsConcave$ **and** $E < E_{min}$ **then**
      $dw(i) = w(i, n) - w(i, n-1)$
    else
      $w(i, n) = min_E(w_0, w_1, w_2)$
      $dw(i) = w(i, n) - w(i, n-1)$
    end if
  end for
end for

4 Experimental Evaluation

In the experiments we used the following datasets for classification tasks: Iris (4 attributes / 150 vectors / 3 classes), Ionosphere (34/351/2), Climate Simulation Changes (19/540/2), Image Segmentation (19/1050/7), Glass (10/214/6) and the following for regression tasks: Steel (13 attributes / 960 vectors), Yacht Hydrodynamics (7/308), Building (15/1052), Concrete Compression Strength (8/1030), Crime and Communities (8/318) All of them but Steel and Building come from [13]. To perform the experiments we created the software in C#. The datasets and the source code are available from [14].

To make the comparison easy, each dataset was standardized before the training (inputs in classification tasks and input and output in regression tasks). For classification

Table 1. MSE on training set after ce=1000

method	iris	ion.	clim.	img.	glass	steel	yacht	bld.	conc.	crime	**av-c**	**av-r**
d1	0.764	1.372	0.594	7.033	5.285	0.313	0.321	0.186	0.859	0.327	0.625	0.417
d2	0.800	1.461	1.483	7.248	5.571	0.380	0.311	0.206	0.861	0.336	0.741	0.436
d3	0.854	1.355	1.428	7.339	5.684	0.387	0.296	0.219	0.865	0.359	0.734	0.441
par.	0.984	1.392	1.414	7.420	5.778	0.411	0.358	0.231	0.900	0.361	0.751	0.480
d1,s	0.779	1.343	0.450	7.152	5.238	0.315	0.293	0.179	0.847	0.320	0.610	0.401
d2,s	0.784	1.416	1.441	7.289	5.413	0.397	0.280	0.204	0.857	0.330	0.727	0.425
d3,s	0.890	1.396	1.497	7.297	5.707	0.397	0.276	0.220	0.867	0.359	0.747	0.432
par,s	0.924	1.452	1.371	7.425	5.792	0.455	0.329	0.218	0.882	0.369	0.749	0.472
d1,f	0.784	1.336	0.583	6.910	5.342	0.309	0.306	0.178	0.856	0.322	0.620	0.405
d2,f	0.784	1.560	1.591	7.519	5.471	0.371	0.305	0.194	0.862	0.333	0.765	0.427
d3,f	0.803	1.452	1.431	7.520	5.781	0.387	0.284	0.204	0.861	0.362	0.749	0.431
par,f	0.940	1.419	1.480	7.199	5.694	0.455	0.342	0.255	0.875	0.347	0.748	0.487
d1,sf	0.786	1.365	0.438	7.208	5.240	0.304	0.293	0.176	0.848	0.320	**0.613**	**0.397**
d2,sf	0.809	1.384	1.445	7.073	5.558	0.358	0.282	0.187	0.846	0.325	0.724	0.409
d3,sf	0.829	1.414	1.586	7.418	5.598	0.398	0.281	0.199	0.858	0.357	0.754	0.429
par,sf	0.862	1.396	1.273	7.392	5.627	0.452	0.330	0.239	0.873	0.363	0.723	0.478

we used neurons with hyperbolic tangent transfer functions. The number of output neurons was equal to the number of classes and we trained the network so the signal of the neuron corresponding to the current class should be greater than 0.995 (if the signal was greater than 0.995 we assumed that the error made by this neuron is zero, the purpose of that was to prevent unnecessary growth of output layer weights) and the signals of the remaining output neurons should be smaller than -0.995. For regression tasks the output neuron had linear transfer function. For each of the 10 dataset and each of the 16 combinations of parameters we trained the network 100 times starting from random weight values. The average values of the 100 trainings are presented in tables 1-3. The standard deviations were of similar order for each dataset: about σ=0.33 for ce=1000, σ=0.10 for ce=2000 and σ=0.02 for ce=4000, where ce is the computational effort, which is proportional to the training time. The training time could not be reliably measured directly in many cases, because it was frequently only a fraction of second. Thus ce was calculated in the following way: When the network training starts ce=0. For changing each weight of an output neuron:

$$ce = ce + 1 + x, \tag{1}$$

For changing each weight of a hidden neuron:

$$ce = ce + 1 + L2 * (L1 + 1) * 0.2 + L2 * 0.8 + x; \tag{2}$$

where $L1$ is the number of neurons in the hidden layer and $L2$ is the number of neurons in the output layer. $L2 * (L1 + 1)$ is the number of signals that must be recalculated in the output layer (for additions and subtractions the cost is 0.2) and $L2$ is the number of transfer functions that must be calculated in the output layer (for calculating hyperbolic tangent the cost is 0.8). For the current layer the cost is $0.2 + 0.8 = 1$. x is the cost the constant operations independent of the weight location within the network structure.

Table 2. MSE on training set after ce=2000

method	iris	ion.	clim.	img.	glass	steel	yacht	bld.	conc.	crime	**av-c**	**av-r**
d1	0.613	0.603	0.370	6.124	2.587	0.248	0.185	0.136	0.802	0.295	0.399	0.317
d2	0.564	0.638	0.389	6.489	2.647	0.289	0.165	0.150	0.803	0.296	0.414	0.323
d3	0.583	0.920	0.365	6.726	2.774	0.321	0.149	0.148	0.810	0.316	0.452	0.327
par.	0.690	0.955	0.385	6.730	2.708	0.340	0.186	0.187	0.845	0.343	0.463	0.370
d1,s	0.545	0.585	0.347	6.303	2.413	0.217	0.152	0.122	0.796	0.293	0.390	0.291
d2,s	0.474	0.628	0.372	6.571	2.522	0.264	0.143	0.128	0.804	0.296	0.403	0.301
d3,s	0.517	0.937	0.373	6.666	2.840	0.286	0.139	0.146	0.811	0.312	0.451	0.315
par,s	0.658	0.985	0.387	6.809	2.757	0.454	0.249	0.179	0.834	0.339	0.468	0.413
d1,f	0.588	0.564	0.363	6.121	2.457	0.231	0.162	0.130	0.797	0.291	0.389	0.301
d2,f	0.529	0.658	0.364	6.796	2.607	0.292	0.145	0.143	0.804	0.293	0.419	0.313
d3,f	0.546	0.982	0.333	6.868	2.855	0.325	0.130	0.148	0.804	0.317	0.459	0.320
par,f	0.723	0.967	0.385	6.529	2.768	0.359	0.190	0.195	0.825	0.303	0.462	0.369
d1,sf	0.521	0.556	0.338	6.269	2.410	0.195	0.155	0.115	0.797	0.290	**0.384**	**0.285**
d2,sf	0.491	0.656	0.347	6.361	2.626	0.246	0.141	0.117	0.799	0.295	0.402	0.291
d3,sf	0.519	0.918	0.337	6.763	2.758	0.272	0.145	0.140	0.807	0.316	0.445	0.312
par,sf	0.710	0.927	0.376	6.747	2.766	0.364	0.230	0.189	0.844	0.325	0.463	0.390

Table 3. MSE on training set after ce=4000

method	iris	ion.	clim.	img.	glass	steel	yacht	bld.	conc.	crime	**av-c**	**av-r**
d1	0.222	0.470	0.279	2.769	2.207	0.134	0.074	0.073	0.758	0.273	0.242	0.216
d2	0.213	0.467	0.281	2.893	2.301	0.179	0.053	0.079	0.761	0.274	0.248	0.220
d3	0.223	0.456	0.274	3.780	2.289	0.195	0.047	0.082	0.765	0.278	0.272	0.223
par.	0.343	0.509	0.378	3.261	2.490	0.219	0.072	0.095	0.776	0.289	0.288	0.246
d1,s	0.219	0.461	0.264	2.435	2.157	0.122	0.050	0.071	0.754	0.272	0.229	0.203
d2,s	0.207	0.462	0.263	2.742	2.222	0.164	0.050	0.076	0.762	0.275	0.239	0.214
d3,s	0.206	0.451	0.280	3.509	2.257	0.191	0.045	0.077	0.768	0.277	0.262	0.219
par,s	0.383	0.462	0.363	3.101	2.389	0.277	0.067	0.111	0.807	0.297	0.276	0.267
d1,f	0.222	0.432	0.275	2.621	2.216	0.124	0.073	0.071	0.758	0.272	0.234	0.213
d2,f	0.209	0.431	0.293	2.890	2.276	0.171	0.053	0.077	0.764	0.273	0.245	0.217
d3,f	0.202	0.398	0.264	3.679	2.236	0.197	0.042	0.079	0.763	0.280	0.259	0.220
par,f	0.332	0.479	0.359	3.246	2.560	0.234	0.097	0.106	0.817	0.287	0.284	0.267
d1,sf	0.220	0.432	0.229	2.348	2.147	0.113	0.049	0.067	0.750	0.271	**0.219**	**0.196**
d2,sf	0.219	0.440	0.248	2.749	2.279	0.155	0.045	0.071	0.761	0.276	0.238	0.209
d3,sf	0.232	0.398	0.259	3.478	2.232	0.183	0.045	0.077	0.765	0.281	0.255	0.218
par,sf	0.278	0.470	0.358	3.146	2.371	0.251	0.095	0.093	0.777	0.286	0.270	0.260

$x = 0.3$ for determining parabola vertex, $x = 0.2$ for selecting the superweights and $x = 0.1$ for all other methods.

We also made another series of experiments in a 10-fold crossvalidation to find out how the weight update scheme used in the network training influences the final prediction ability. However, it turned out that the way the network reached the predefined MSE value (which was the stopping criteria) did not influenced the prediction accuracy.

The average value column (av-c) in tables 1-3 for classification tasks is the weighted average of the values for particular datasets weighted by the inverse of output neuron numbers - thus it represents the average MSE per one output neuron:

$$av - c = (iris/3 + ion/2 + clim/2 + img/7 + glass/6)/5. \tag{3}$$

For regression it is:

$$av - r = (steel + yacht + bld + 0.5 \cdot conc + crime)/5. \tag{4}$$

5 Conclusions

We presented some improvements of the VSS algorithm, which was our algorithm of choice in the cases that we had to use non-differentiable error functions for big datasets. The conclusions are as follows: it does not make sense to locate the minimum in each weight direction precisely, because the closer to the minimum we are, the error surface gets flatter and the less we can gain (see fig. 1 - left). It better pays off to use the computational effort to modify the subsequent weight. Because the learning trajectory changes its direction rather gradually than suddenly, if changing same weights values in one epoch was very successful it is a good idea to modify them once again before going to the next epoch. On the contrary, if changing some weights did not cause improvement, the weights can be frozen for the subsequent epoch and the computational power can be used to change the more promising weights. The optimal $d1$ value is about 1.5, allowing for gradual step increase as the error surface is getting flatter with the training progress. The additionally obtained time reduction is on average about 30%. Obviously, well written and optimized code is another field of improvement. The 30% may seem not much for one training of a small or medium size network. However, in the case when we have to conduct thousands of experiments, on real-world datasets, it can shorten our work by many hours.

References

1. Aarts, E., Lenstra, J.K.: Local Search in Combinatorial Optimization. John Wiley & Sons, Inc., New York (1997)
2. Du, K.-L., Swamy, M.-N.S.: Neural Networks and Statistical Learning. Springer (2013)
3. Garcia-Pedrajas, N., et al.: An alternative approach for neural network evolution with a genetic algorithm. Neural Networks 19(4), 514–528 (2006)
4. Engel, J.: Teaching Feed-forward Neural Networks by Simulated Annealing. Complex Systems 2, 641–648 (1988)
5. Battiti, R., Tecchiolli, G.: Training Neural Nets with the Reactive Tabu Search. IEEE Trans. on Neural Networks 6, 1185–1200 (1995)
6. Bengio, Y.: Learning Deep Architectures for AI. Foundations and Trends in Machine Learning 2(1), 1–127 (2009)
7. Beliakov, G., Kelarev, A., Yearwood, J.: Derivative-free optimization and neural networks for robust regression. Optimization 61(12), 1467–1490 (2012)

8. Burden, R.L., Douglas Faires, J.: Numerical Analysis, Cengage Learning (2010)
9. Kordos, M., Duch, W.: Variable Step Search Algorithm for Feedforward Networks. Neurocomputing 71(13-15), 2470–2480 (2008)
10. Kordos, M., Rusiecki, A.: Improving MLP Neural Network Performance by Noise Reduction. In: Dediu, A.-H., Martín-Vide, C., Truthe, B., Vega-Rodríguez, M.A. (eds.) TPNC 2013. LNCS, vol. 8273, pp. 133–144. Springer, Heidelberg (2013)
11. Rusiecki, A., Kordos, M., Kamiński, T., Greń, K.: Training Neural Networks on Noisy Data. In: Rutkowski, L., Korytkowski, M., Scherer, R., Tadeusiewicz, R., Zadeh, L.A., Zurada, J.M. (eds.) ICAISC 2014, Part I. LNCS, vol. 8467, pp. 131–142. Springer, Heidelberg (2014)
12. Rusiecki, A.: Robust learning algorithm based on LTA estimator. Neurocomputing 120, 624–632 (2013)
13. Merz, C., Murphy, P.: UCI repository of machine learning databases (2014), http://www.ics.uci.edu/mlearn/MLRepository.html
14. Source code and datasets used in the paper, http://www.kordos.com/software/ideal2014.zip