Initialization and optimization of multilayered perceptrons

Włodzisław Duch, Rafał Adamczak, and Norbert Jankowski
Department of Computer Methods, Nicholas Copernicus University,
Grudziądzka 5, 87-100 Toruń, Poland.
E-mail:duch,raad,norbert@phys.uni.torun.pl

Abstract

Despite all the progress in neural networks field the technology is brittle and sometimes difficult to apply. Good initialization of adaptive parameters in neural networks and optimization of architecture are the key factor to create robust neural networks. Methods of initialization of MLPs are reviewed and new methods based on clusterization techniques are suggested. Penalty term added to the error function leads to optimized, small and accurate networks.

I. Introduction

FINDING global minimum of a nonlinear function with many parameters is an NP-hard problem [1]. Learning in neural networks is most frequently based on minimization of a cost function. Good initialization of adaptive parameters may enable finding solutions in complex, real-world problems and may significantly decrease learning time. Subsequent optimization should lead to compact networks capable of good generalization.

In this paper methods of initialization and optimization of the multi-layer perceptrons (MLPs) used for classification problems are investigated. In MLP networks sigmoidal transfer functions provide hyperplanes, dividing the input space into classification regions. Coefficients of these hyperplanes – called activation weights – are usually the only adaptive parameters of such networks. Initialization procedures should propose architecture (the number of nodes and connections), weights (biases are also counted as weights), and in some cases also the slopes of sigmoidal functions. Initial structure of MLP networks is frequently optimized during learning, either using genetic or other global minimization methods, or enforcing skeletonization of networks using penalty terms in the cost functions.

In the next section methods of initialization of MLPs are discussed and new methods, allowing to solve some classification problems without tedious optimization, proposed. In the third section optimization of MLP architecture based on new form of error function is described and results compared with other methods. Short discussion closes this paper.

II. Initialization of MLP weights

Although MLP networks have important advantages they are less robust and require much longer training than RBF (Radial Basis Functions) networks [1]. The main reason for long training times is the lack of proper initialization methods. In a long series of computer experiments Schmidhuber and Hochreiter [2] observed that repeating random initialization ("guessing" the weights) many times is the fastest way to convergence. Wrong initialization may create network of sigmoidal functions dividing the input space into areas where the network

Submitted to the Third Conference on Neural Networks and Their Applications, Kule, Poland 14-18.10.1997

function gives constant inputs for all training data, making gradient learning procedures useless. If some weights overlap too much (scalar product $\mathbf{W} \cdot \mathbf{W}'/|\mathbf{W}||\mathbf{W}'|$ is close to 1) the number of effective hyperplanes is reduced.

Random weight initialization is still the most popular method. Bottou [3] recommended values in the $\pm a/\sqrt{n_{inp}}$ range, where n_{inp} is the number of inputs the neuron receives and a is determined by the maximum curvature of the sigmoid ($a = 2.38$ for unipolar sigmoid). Several random initialization schemes have recently been compared by Thimm and Fiesler [3] using a very large number of computer experiments. The best initial weight variance is determined by the dataset, but differences for small deviations are not significant and weights in the range ± 0.77 give the best mean performance. A few authors proposed initialization methods which are not based on random weights. In classification problems clusterization techniques are better suited to determine initial weights. Initialization of MLPs by prototypes has been developed by Denoeux and Lengellé [4] and Weymaere and Martens [5] but is still used quite rarely.

For a center of an input data cluster $\mathbf{C}$ laying on the unit sphere the activation is $\mathbf{W} \cdot \mathbf{C}$. The largest activation is obtained when the weights $\mathbf{W}$ point in the same direction as $\mathbf{C}$. The sigmoidal function $\sigma(\mathbf{C} \cdot \mathbf{X} - \theta) = (1 + \exp(s(-\mathbf{C} \cdot \mathbf{X} + \theta)))^{-1}$, where s determines the slope, has the largest gradient in the direction of $\mathbf{W} = \mathbf{C}$. Its value is equal to $\sigma(0) = 0.5$ at a θ distance from the origin of the coordinate system. Since the $\mathbf{C}$ vector is normalized $\theta = 1$ places the contours for 0.5 value tangentially to the unit hypersphere. Contours for lower values $\sigma(\mathbf{C} \cdot \mathbf{X} - \theta) < 0.5$ cut segments of the hypersphere in which the value of $\sigma(\mathbf{C} \cdot \mathbf{X} - \theta)$ is constant. Activation of the sigmoid for normalized inputs $\sigma(I_{max} - d(\mathbf{C}, \mathbf{X})$ depends on the distance between $\mathbf{C}$ and $\mathbf{X}$. This suggests the following prescription for the initialization of the first layer:

1. Pre-process all input data: $k = 1..n$ vectors, with $i = 1..N$ components, $\mathbf{X}^{(k)}$ is the vector number k, $\mathbf{X}_i$ is the vector of i-th components of all data vectors.
   (a) Find minimum, maximum and middle values: $X_i^{min} = \min_k X_i^{(k)}$; $X_i^{max} = \max_k X_i^{(k)}$; $\bar{X}_i = (X_i^{max} - X_i^{min})/2$
   (b) Shift vectors to the middle values of each component: $\mathbf{X} \leftarrow \mathbf{X} - \mathbf{X}^{min} - \bar{\mathbf{X}}$.
   (c) Rescale components of vectors to ± 1 sector: $\mathbf{X}_i \leftarrow \mathbf{X}_i / \bar{\mathbf{X}}$.
   (d) Find the biggest norm $||X||_{max}$ and renormalize all vectors $\mathbf{X} \leftarrow \sqrt{2}\mathbf{X}/||X||_{max}$.
   (e) Normalize vectors adding an extra dimension $\mathbf{X} \leftarrow (\mathbf{X}, X_r)$, where: $X_r = \text{sign}(1 - ||X||^2)\sqrt{|1 - ||X||^2|} \in [-1, +1]$.
2. Cluster analysis: find means of the normalized data clusters (using dendrograms or other clusterization methods).
3. Choose weights (in N+1 dimensional space) equal to the center of clusters $\mathbf{C}$ and the biases equal to $+1$.

Now test the network on all training data. Increasing the norm of the weights or changing biases shifts the area where the sigmoid function has large values towards the outside of the hypersphere, thus reducing the slice where the value of the transfer function is large. The range of the sigmoid should extend to the vectors on the border of a cluster. For a vector $\mathbf{B}$ laying at the border a good estimation for the bias is given by the scalar product $\mathbf{C} \cdot \mathbf{B}$. Since the front of the sigmoid is perpendicular to the vector $\mathbf{C}$ and should be in the plane $\mathbf{X} - \mathbf{B}$ perpendicular to $\mathbf{C}$, therefore $\mathbf{C} \cdot (\mathbf{X} - \mathbf{B}) = \mathbf{C} \cdot \mathbf{X} - \mathbf{C} \cdot \mathbf{B} = \mathbf{0}$, hence the estimation. To avoid strong overlaps between classes one should start with relatively small biases and optimize the slope

(this is one-parameter optimization since other parameters are kept fixed) to obtain smooth initial approximation. Increasing the slope s of overlapping nodes also helps to reduce the area on the hypersphere where sigmoids are active.

In the XOR case the input vectors for class = T are $(0,1),(1,0)$ and for the class = F are $(0,0),(1,1)$. The mean for each feature is 0.5 and after shifting and renormalizing the vectors are $\mathbf{C}_1 = (-1,+1)/\sqrt{2}$, $\mathbf{C}_2 = (+1,-1)/\sqrt{2}$ for class T and $(-1,-1)/\sqrt{2}$, $(+1,+1)/\sqrt{2}$ for class F. Selecting one of the classes for output, for example class T, initial weights for the first neuron are given by $\mathbf{C}_1$ and for the second neuron by $\mathbf{C}_2$, while the hidden to output layer weights are all $+1$. This is the correct and the simplest solution for the XOR problem found without any optimization of the network! The case of 3 vectors representing clusters in two dimensions is presented in Fig. 1. Initial placements of sigmoids generated by our algorithm in the original (here one-dimensional) input space is presented in the lower part of Fig. 1. Note that the middle sigmoid is localized (due to the normalization in the extended space), and that the values of different sigmoids cross in optimal boundary points between the prototypes.

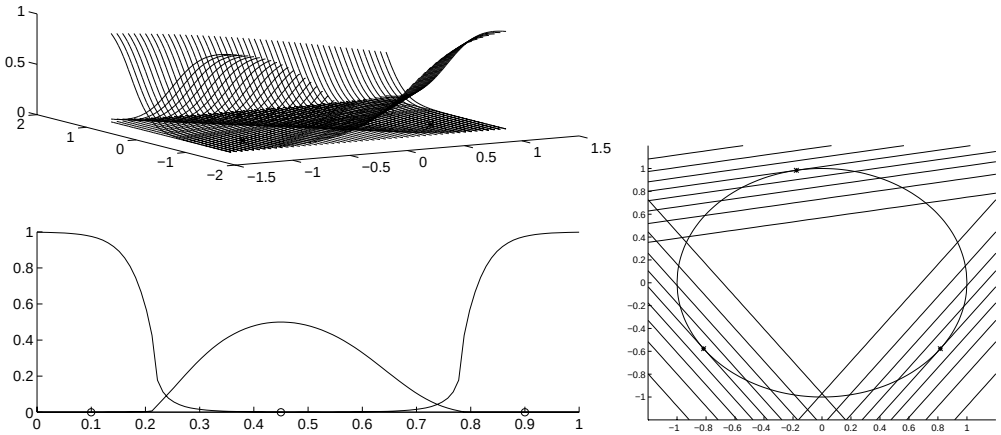

Fig. 1. Initialization of 3 neurons for 3 clusters in one-dimensional case. Top: values of sigmoidal functions in the two-dimensional space after renormalization of input vectors; bottom – sigmoids after transformation back to the original space; right – contours of sigmoidal functions, positions of prototypes are marked with asterisks.

The output layer weights and biases are initialized to unit values. If vectors of different classes are close to each other or the topology of clusters is rather complex a second hidden layer may be helpful. This layer serves primarily to create more localized areas of high output values of the network. Combination of two sigmoids $\sigma(\mathbf{W}\cdot\mathbf{X}-\theta)-\sigma(\mathbf{W}\cdot\mathbf{X}-\theta')$ defines a “window" of non-zero output. If the data is normalized on a sphere the difference defines a complex bent shape of non-zero output and in the space of unnormalized components it represents the difference of two localized shapes. The structure of the network is thus following: in the first layer pairs of neurons with weights initialized to $+\mathbf{C}, -\theta$ and $-\mathbf{C}, +\theta$ are defined, i.e. realizing sigmoidal functions of the form: $\sigma_+(\mathbf{C}\cdot\mathbf{X}-\theta)$ and $\sigma_-(-\mathbf{C}\cdot\mathbf{X}+\theta)$. Second layer has half the number of neurons of the first layer, each connected to two neurons of the pair with weights $+1$ and -1 and biases $+1$, while the output connections have $+1$ value. Full description of data clusters by window-type units may also be attempted. A combination

of two sigmoids per one dimension, or $2N$ sigmoids, is needed to describe cuboidal clusters. They can be represented either in the original input space or in the (N+1)-dimensional space, where cylindrical coordinate system is recommended. In the input space in the simplest case (no rotations) the cluster **C** should be contained within the cuboid:

$$\prod_{i=1}^{N} \left[\sigma(X_i - C_i + \theta_i/2) - \sigma(X_i - C_i - \theta_i/2)\right] \tag{1}$$

where θ_i represents dispersion of the cluster in i-th dimension. This cuboid is approximated by a sum of $2N$ sigmoids per cluster, with biases $-C_i + \theta_i/2$ and $-C_i - \theta_i/2$. The first hidden layer is therefore composed of $2N$ neurons per cluster, with half of the input-hidden layer weights equal to $+1$ and the other half -1 Without rotation only one input per neurons is left – the rotated cuboid case has been solved in Duch, Adamczak, Jankowski (this volume) and requires all N inputs to each node. The number of units in the second layer is equal to the number of clusters, all hidden-hidden layer weights are $+1$, and biases are equal to N. Finally the output layer weights and biases are all $+1$. A single sigmoid separating two clusters may lead to the same classification error as $4N$ sigmoids describing two clusters. Therefore after initial network structure is created it is systematically pruned checking the influence of each connection on the classification error. Large networks are created only during the intermediate steps of initialization procedure but final networks may be rather small.

Increasing the dimension is equivalent to addition of one circular unit to the MLP. In the one-dimensional case there are two weights, (W_x, W_r), and if the W_r weight is zero than $\sigma(W_x\, x)$ is monotonic as normal sigmoidal function, while if W_x is zero $\sigma(W_r\sqrt{1-x^2})$ has maximum around $x = 0$. Ridella et.al. [6] used circular units in their Circular Backpropagation Networks. Projection of data on a sphere has been used previously in the projection network by Wilensky and Manaukian [7] and by Jaeger and Wilamowski [8], where a transformation from N-dimensional to $2N$-dimensional input space is recommended as simpler to realize in hardware. In this case each X_i component is complemented with $\sqrt{r^2 - X_i^2}$ and this function is approximated with a sigmoidal function $r\tanh[\pi(1 - X/r)]$.

III. Optimization of network architecture

Initialization methods described in previous section lead to specific network structures that may be further optimized. In particular most networks are too complex, use too many features and therefore are not capable of good generalization. The simplest approach to network optimization is based on penalty terms added to the standard error function. Penalty terms allow for pruning the network. Two most common terms are the quadratic penalty and rational penalty terms [1]. Recently these terms were analyzed in details and used in combination [9]. Many other penalty terms and weight pruning techniques were developed, some of them quite sophisticated [10]. All these approaches encourage decay of weights. However, optimal solution may require a few large weights. If these weights are distributed around $a > 0$ and $b < 0$ than it should be more appropriate to add the following penalty term:

$$E(W) = E_0(W) + \frac{\lambda_1}{2}\sum_{i,j} W_{ij}^2 + \frac{\lambda_2}{2}\sum_{i,j} W_{ij}^2 (W_{ij} - a)^2 (W_{ij} + b)^2 \tag{2}$$

$E_0(W)$ is the standard quadratic error measure, the second term with λ_1 leads to large number of zero weights, i.e. elimination of irrelevant features, and the third term vanishes for weights equal 0, a or $-b$. Similarly as in the weight pruning technique case in the backpropagation algorithm these extra terms lead to the additional change of weights:

$$\Delta W_{ij} = \lambda_1 W_{ij} + \lambda_2 W_{ij}(W_{ij} - a)(W_{ij} + b)(3W_{ij}^2 + 2W_{ij}(b - a) - ab) \tag{3}$$

where λ_1 and λ_2 scale the relative importance of auxiliary conditions. This form of error function has two advantages: independent parameters control enforcing of the 0 and a, b weights, and an interpretation of this function from the Bayesian point of view [11] is straightforward. It defines our prior knowledge about the probability distribution $P(W|M)$ of the weights in our model M. Optimal value of a, b parameters are found iteratively, starting from $a = b = 1$ values:

$$a = +\sum_{ij} W_{ij}^3 (W_{ij} + b)^2 / \sum_{ij} W_{ij}^2 (W_{ij} + b)^2 \tag{4}$$
$$b = -\sum_{ij} W_{ij}^3 (W_{ij} - a)^2 / \sum_{ij} W_{ij}^2 (W_{ij} - a)^2$$

We have noticed that the accuracy of logical rules [12] extracted from networks using the penalty function with $a = b = 1$ significantly exceeded accuracies obtained from other MLP networks, even after careful optimization has been performed. A good comparison may be done for the hypothyroid dataset [13]. Two types of the disease, primary hypothyroid and compensated hypothyroid, are diagnosed and differentiated from normal (no hypothyroid) cases using the results of 22 medical tests. Thus the problem has 3 classes and 22 attributes, 3772 cases for training and 3428 cases for testing, with about 10% of values missing (typical for medical data). This data was used by Schiffman et.al. [14] in optimization of about 15 MLPs trained with different variants of backpropagation and cascade correlation algorithms. In addition tedious genetic optimization has been performed on many network architectures. The best results of this study [14] are reported in the Table I. Our results are far from being optimal since they were obtained from standard MLP network randomly initialized, with the penalty term containing unoptimized $a = b = 1$ values. Nevertheless they are superior even in comparison with the networks optimized by genetic algorithms.

IV. Summary and discussion

Various methods of initialization of adaptive parameters in MLP networks have been briefly discussed. Initialization of MLPs is still done more often by randomizing weights [3], but initialization by prototypes based on initial clusterization should give much better results enabling solutions to complex, real life problems. Introduction of such methods of parameter initialization should allow for creation of neural systems requiring little optimization in further training stages.

Our optimization is done by adding penalty term to the error function, biasing the network weights towards 0, a and $-b$ values, where a, $-b$ parameters are optimized iteratively. Recently Refenes and Connor [15] proposed to use integer weights for regression networks. Their work has been motivated by the desire to recreate integer parameters in the time series models

TABLE I

Classification results for a number of well-optimized MLP networks applied to the thyroid dataset. Only the best results are shown here.

Method	Training set accuracy %	Test set accuracy %
BP+conjugate gradient	94.6	93.8
Best Backpropagation	99.1	97.6
RPROP	99.6	98.0
Quickprop	99.6	98.3
BP+ genetic optimization	99.4	98.4
Local adaptation rates	99.6	98.5
Cascade correlation	100.0	98.5
Our penalty term	99.5	99.1

producing the sample data. They define a prior probability as a product of exponential decay term and a sum of Gaussian terms centered at integers $-N, -N+1, \ldots N-1, N$. There is no particular reason why in general applications weights should assume integer values – our $a,\ b$ parameters are not integer. Initialization and optimization described in this paper can be done in a completely automatic way, contributing towards the goal of creation of robust neural systems.

Acknowledgments: Support by the Polish Committee for Scientific Research, grant 8T11F 00308, is gratefully acknowledged.

References

[1] C. Bishop, Neural networks for pattern recognition (Clarendon Press, Oxford 1995)
[2] J. Schmidhuber, S. Hochreiter, Guessing can outperform many long time lag algorithms. Technical Note, IDSIA-19-96
[3] G. Thimm, E. Fiesler, Higher order and multilayer perceptron initialization, *Trans. Neural Net.* 8 (1997) 349–359
[4] J. Denoeux. R. Lengelle, Initializing backpropagation networks with prototypes, *Neural Net.* 6 (1993) 351-363
[5] N. Weymaere, J.P. Martens, On the initialization and optimization of multilayer perceptrons, *Trans. Neural Net.* 5 (1994) 738-751
[6] S. Ridella, S. Rovetta, R. Zunino, Circular Backpropagation Networks for Classification, *Trans. Neural Net.* 8 (1997) 84–97
[7] G. Wilensky and N. Manaukian, The Projection Neural Network, *IJCNN'92*, Baltimore, 7-11.06.92, Vol. II, p.358-367
[8] B.M. Wilamowski, R.C. Jaeger, Implementation of RBF type netowrks by MLP networks, *ICNN'96*, Washington, DC, June 3-6, 1996, pp. 1670-1675.
[9] R. Setiono, A Penalty-Function Approach to Pruning Feedforward Neural Networks, *Neural Comp.* 9 (1997) 185-204
[10] S. Hochreiter, J. Schmidhuber, Flat minima, *Neural Comp.* 9 (1997) 1-42
[11] D.J. MacKay, A practical Bayesian framework for backpropagation networks, *Neural Comp.* 4 (1992) 448-472
[12] W. Duch, R. Adamczak, K. Grąbczewski, Extraction of crisp logical rules using constrained backpropagation networks. *ICANN'97*, Houston, 9-12.6.1997 (in print); Logical rules for classification of medical data using ontogenic neural algorithm, *EANN'97*, Stockholm, 16-18.06.1997 (in print)
[13] C.J. Mertz, P.M. Murphy, UCI repository of machine learning databases, http://www.ics.uci.edu/pub/machine-learning-databases.
[14] W. Schiffmann, M. Joost, R. Werner, Comparison of optimized backpropagation algorithms, *ESANN '93*, Brussels 1993, pp. 97-104; Synthesis and Performance Analysis of Multilayer Neural Network Architectures, Tech. Rep. 15/1992, available in `neuroprose` as schiff.gann.ps.Z
[15] A-P. Refenes, J.T. Connor, Biasing towards integer solutions, *ICONIP'96*, Hong Kong 1996, Vol.II, pp. 681-688