LIBERTAS MATHEMATICA, VOL XXI (2001)

DYNAMIC PROGRAMMING BASED SUPERVISED LEARNING METHOD FOR NEURAL NETWORKS

ANGELICA CALU

Abstract. New hybrid methods for solving the multilayer perceptron optimization problem are proposed, which use the computation capabilities of Bellman's dynamic programming (DP) method. To solve the neural network optimization problem, we consider differently the case of output neurons from those of hidden neurons. For the neurons of the output layer we apply the conventional DP and for the hidden neurons we apply a garadient method, e.g. Schiffmann's RPORP algorithm. Computer simulation performed on a big medical classification problem shows that our DP-based method performs better that the RPROP method.

1 Introduction

Problems involving supervised learning or "learning with a teacher" have attracted great attention in the past decades and many learning algorithms were proposed. The error back-propagation method of Rumelhart et al. [6] is the well-known method for adapting the weights of a multilayer perceptron (MLP). Although it is simple to compute locally, which offers advantages to implement it on parallel computers or to realize it hardware in VLSI architectures, it has the disadvantages to be computationally expensive or to get stuck in local minima. Succeeding methods ([3], [10], [13], [5], [4], [12], [14], [15]) have brought improvements in diminishing these disadvantages, thought there is a great demand on finding new learning methods.

As an unsupervised learning paradigm, where learning is performed without a teacher, reinforcement learning is also playing an important role in the current research. The adaptive critic algorithms, which are most closely related to the theory of optimal control, are based on dynamic programming (DP) method of Bellamn. DP is formulated for sequential decision problems, where a set of decisions are to be performed in stages to minimize a total cost over the number of stages. DP provides a computational technique based on the use of *principle of optimality*, where each decision is computed separately, tacking into account the decisions, which were made before, through the returned result or the optimal cost till to that decision stage.

For a discrete-time dynamic system evolving from state $z_n \in R^p, n = 1, 2, \ldots, N, N \in N$, under the influence of the control $u_n \in R^q$, to the next state

$$z_{n+1} = f(z_n, u_n) \tag{1.1}$$

where $f : R^{p+q} \mapsto R^p$ is the transfer function of the system, and incuring a cost $g(z_n, u_n)$, the *functional equation* (Bellman's equation), which results from applying the principle of optimality, is:

$$S_n(z_n) = \min_{u_n \in R^q} [g(z_n, u_n) + S_{n+1}(z_{n+1})] \tag{1.2}$$

where S_n is the minimal cost of the system starting from the state z_n or the *cost-to-go* of the state z_n.

There are two approaches for solving the DP functional equation (1.2), depending on the type of the treated problem. For problems with finite horizon (N is finite), the optimal controls u_n are found through a recursive approach. For problems with infinite horizon ($N = \infty$) or when the number of states and controls is very large the computational requirements of DP are overwhelming (the so-called *"curse of dimensionality"*) and a suboptimal solution is required. This is done through successive approximations of the cost-to-go function $S_n(z_n)$. In this regard, the backgammon playing program of Tesauro [11] shows apkrecisble results using Sutton's Temporal Differences [9] method, which is an approximate DP method. Another possible approach is through the use of neural networks, which are known to be universal approximators, i.e. given a data set of input-output values, they are matched to a nonlinear mapping using a training algorithm. This approximation methodologie of DP is named "neuro-dynamic programping" [1].

Thus, the use of neural networks gives improvements to the DP-based learning algorithms. However, the success depends on how good the neural networks have been trained, i.e how good they do approximate.

In this article we show new attempts, in which gradient-based training algorithms could be improved through the use of DP. To solve the neural network optimization problem, we consider differently the case of output neurons from those of hidden neurons. For the neurons of the output layer we apply the conventional DP and for the hidden neurons we apply a method based on gradient approach. Proceeding in this way we show that the new method converges faster than the gradient-based method.

2 Gradient Based Method

Given $(x_p, d_p), x_p \in R^M, d_p \in R^N, p = 1, 2, \ldots, P$ a set of input-output patterns, the training of a MLP is a nonlinear optimization problem in which a cost or error function

$$E(w) = \sum_{p=1}^{P} \sum_{n=1}^{N} [d_{np} - y_{np}]^2 \tag{2.1}$$

has to be minimized with respect to the connections' weights w of the neural network, where d_{np} and y_{np} are the desired and actual values of each output neuron $n = 1, \ldots, N$ for the p-th training vector. For this purpose the synaptic weights are adjusted so as to make the actual output of the network move closer to the desired output. This is the popular *back-propagation algorithm*, whose update rule is:

$$w(i+1) = w(i) + \eta \Delta w(i) \tag{2.2}$$

where η, a positive constant, is the *learning-rate parameter* and $\Delta w(i)$ is the adjustment applied to the weight $w(i)$ at iteration i:

$$\Delta w(i) = -\left.\frac{\partial E}{\partial w}\right|_i \tag{2.3}$$

The success of the back-propagation algorithm depends on the choice of the learning parameter η. If η is small, the gradient descent can be very slow and if η is to large it can oscillate. There are many algorithms dealing with this problem, whereby RPROP of Schiffmann et.al [7] showed very good results regarding the speed of convergence and the generalization capabilities. It is a local learning rate adaptation algorithm, i.e. it uses independent update step sizes η_{ij} for each connection between neuron i of a layer and neuron j of the next layer of the network. The step sizes and the weights of the synaptic connections are adapted depending on the sign of the actual and the last derivative of the error function with respect to the weight parameters. The RPROP adaptation algorithm is presented below:

1. Initialize $\eta_{ij}(0)$.
2. Adapt the step sizes:

$$\eta_{ij}(n) = \begin{cases} u\eta_{ij}(n-1), & if \left.\frac{\partial E}{\partial w_{ij}}\right|_n \left.\frac{\partial E}{\partial w_{ij}}\right|_{n-1} > 0 \\ d\eta_{ij}(n-1), & if \left.\frac{\partial E}{\partial w_{ij}}\right|_n \left.\frac{\partial E}{\partial w_{ij}}\right|_{n-1} < 0 \end{cases} \tag{2.4}$$

   To avoid oscillation and arithmetic underflow of floating point values, the step sizes are bound by upper and lower limits:

$$\eta_{ij}(n) = \begin{cases} \eta_{max}, & if \quad \eta_{ij} \geq \eta_{max} \\ \eta_{min}, & if \quad \eta_{ij} \leq \eta_{min} \end{cases} \tag{2.5}$$

3. Update the weights:

$$w_{ij}(n+1) = \begin{cases} w_{ij}(n) - \eta_{ij}(n), & if \left.\frac{\partial E}{\partial w_{ij}}\right|_n > 0 \\ w_{ij}(n) + \eta_{ij}(n), & if \left.\frac{\partial E}{\partial w_{ij}}\right|_n < 0 \\ w_{ij}(n), & if \left.\frac{\partial E}{\partial w_{ij}}\right|_n = 0 \end{cases} \tag{2.6}$$

   The parameters u, d, η_{min} and η_{max} have constant values and their choice depends on the problem to be solved.

3 DP for solving supervised learning problem

To apply the Bellman's principle of optimality for minimizing the cost function (2.1), we have to define the problem in terms of DP, i.e. to decompose it into small problems, each

consisting of a single decision variable (weight) and a state variable. The state variable has the role to maintain the values of the decision variables taken before in the minimization process.

Without restricting the generalization we consider a neural network with just an output neuron and we write the formula of the cost function (2.1) in an enhanced form as

$$E(w) = \sum_{p=1}^{P} \left[d_p - \varphi(\sum_{l=0}^{L} h_l w_l) \right]^2 \tag{3.1}$$

where $\varphi : R \mapsto [0,1], \varphi(x) = \dfrac{1}{1+e^{-x}}$ is the activation function of the neuron, $w_l, l = 0, \ldots, L$ the weights of the connection to the output neuron, $w_0 = \theta$ the threshold of the output neuron, $h_l, l = 0, \ldots, L, h_0 = -1$ the inputs to the output neuron, which are also the outputs of the neurons in the last hidden layer. If we take a look on the cost function (3.1) we observe that it couldn't be decomposed, since it is not separable. Without radical modifications of the cost function (3.1) we can define a new one, which is separable in the weights of the output neurons. Instead of the problem:

$$\min_{w_h, w_0, \ldots, w_L} \sum_{p=1}^{P} [d_p - y_p]^2 \tag{3.2}$$

where w_h and $w_0, \ldots, w_L$ are the weights of the hidden layers and output layer, respectivly, we consider the following problem:

$$\min_{w_h, w_0, \ldots, w_L} \sum_{l=0}^{L} \Phi_l(w_h, w_l) \tag{3.3}$$

where

$$\begin{aligned}
\Phi_l(w_h, w_l) &= [d - \tilde{\varphi}(v_l)]^2 \\
&= \left[d - \tilde{\varphi} \left(\sum_{\substack{j=0 \\ j \neq l}}^{L} h_j \tilde{w}_j + h_l w_l \right) \right]^2 \\
d &= (d_1, \ldots, d_P)^T \in R^P \\
l &= 0, \ldots, L \\
h_l &= (h_{l1}, \ldots, h_{lP})^T \in R^P \\
h_0 &= (-1, \ldots, -1)^T \in R^P \\
\tilde{\varphi} &: R^P \mapsto R^P \\
\forall v &= (v_1, \ldots, v_P)^T \in R^P, \\
\tilde{\varphi}(v) &= (\varphi(v_1), \ldots, \varphi(v_P))^T
\end{aligned}$$

For clarity sake, we have noted in the above formula the fixed value weights as $\tilde{w}$ to distinguish them from the unknown weights w. Applying the principle of conditional optimization ([8]) we can decompose the problem (3.3) in two problems:

$$\min_{w_h, w_0, \ldots, w_L} \sum_{l=0}^{L} \Phi_l(w_h, w_l) = \tag{3.4}$$
$$\min_{w_h} \left\{ \min_{w_0, \ldots, w_L} \sum_{l=0}^{L} \Phi_l(w_h, w_l) \right\}$$

where we see that the function to be optimized with respect to $w_0, \ldots, w_L$ is separable.

First, we have to solve the problem:

$$\min_{w_0, \ldots, w_L} \sum_{l=0}^{L} \Phi_l(w_l) \tag{3.5}$$

where we have dropped the variable w_h since for this minimization it is considered to have an optimal value. For the solution of this problem we will resort to the DP method. To bring the problem in a suitable form we introduce the state variable z_l and the following state transformation function:

$$\begin{aligned} &\forall l = 0, \ldots, L \\ &z_l = (z_{l1}, \ldots, z_{lp})^T \in R^P \\ &z_{l+1} = z_l - h_l w_l \\ &z_{L+1} = 0 \end{aligned} \tag{3.6}$$

Thereupon the DP optimalty equation would take the form:

$$\begin{aligned} &\forall l = 0, \ldots, L-1 \\ &S_l(z_l) = \min_{w_l} \left\{ \left[d - \tilde{\varphi} \left(\sum_{\substack{j=0 \\ j \neq l}}^{L} h_j \tilde{w}_j + h_l w_l \right) \right]^2 \right. \\ &\qquad \left. + S_{l+1}(z_{i+1}) \right\} \\ &S_L(z_L) = \min_{w_L} \left[d - \tilde{\varphi} \left(\sum_{j=0}^{L-1} h_j \tilde{w}_j + h_L w_L \right) \right]^2 \end{aligned} \tag{3.7}$$

An analytical solution of the above recursive equation will give the following update formulae for the weights and the optimal returns in each stage:

$$\begin{aligned} &\forall l = 0, \ldots, L \\ &w_l = \tilde{w}_l + \frac{h_l^T \left(z_l - \sum_{j=l}^{L} h_j \tilde{w}_j \right)}{(L - l + 1) h_l^T h_l} \end{aligned} \tag{3.8}$$

$$S_l(z_l) = (L-l+1)\left[d - \tilde{\varphi}\left(\frac{z_l + (L-l)\sum\limits_{l=0}^{L} h_l\tilde{w}_l - \sum\limits_{j=1}^{l-1} h_j\tilde{w}_j}{L-l+1}\right)\right]^2$$

which depends on the stage state z_l. This causes, in the adaptation process, the weights to be changed in the order prescribed by the state transformation (3.6), hence from $l = 0$ to L. The optimal value of the problem (3.5) is:

$$S_0(z_0) = (L+1)\left[d - \tilde{\varphi}\left(\frac{z_0 + L\sum\limits_{l=0}^{L} h_l\tilde{w}_l}{L+1}\right)\right]^2 \tag{3.9}$$

which is a function of the start state z_0. From the state transformation (3.6) we get $z_0 = \sum\limits_{l=0}^{L} h_l w_l$ and since this is the optimal potential to the output neuron its value must be $\tilde{\varphi}^{-1}(d)$.

Having solved the problem (3.5), we further proceed to minimize it with respect to the weights of the hidden layers w_h, in accordance with (3.4):

$$\min_{w_h}(L+1)\left[d - \tilde{\varphi}\left(\frac{\tilde{\varphi}^{-1}(d) + L\sum\limits_{l=0}^{L} h_l\tilde{w}_l}{L+1}\right)\right]^2 \tag{3.10}$$

using a gradient method like that presented in the section 2.

In the following we present the algorithm described so far:

1. Initialize the weights of the network. Set the iteration number $i = 0$.
2. Propagate all the patterns (x_p, y_p), $p = 1, \ldots, P$ through the network and calculate the output.
3. Adapt the weights of the output neurons:
   $for \quad n = 1 \quad to \quad N$
   $z_{0n} = \tilde{\varphi}^{-1}(d_n)$
   $for \quad l = 0 \quad to \quad L$
   $$w_l(i+1) = w_l(i) + \frac{h_l^T\left(z_{ln} - \sum\limits_{j=l}^{L} h_j w_j(i)\right)}{(L-l+1)(h_l^T h_l)}$$
   $z_{l+1,n} = z_{ln} - h_l w_l(i+1)$
   $end \quad for$
   $end \quad for$
4. Calculate the new error at the output neurons and propagate it back to the input.

5. Adapt the weights of the hidden layers, using a gradient algorithms, like that presented in section 2:

   $w_h(i+1) = w_h(i) + \eta \Delta w_h(i)$

6. Calculate the new error E at the output neurons. If $E \leq \varepsilon$ stop, where ε is a sufficiently small error energy threshold, otherwise set i = i+1 and go to step 2.

4 Results

Previous results on our DP-based algorithm were presented in [2], where the method was combined with three gradient descent algorithms (back-propagation with momentum [6], delta-bar-delta [3] and conjugate gradient method) and compared with them. We showed that our algorithm outperforms the considered gradient algorithms in the speed of convergence and the generalization capabilities. Further, to show the performance of the DP-based learning algorithm, we have tested it in combination with Schiffmann's RPROP algorithm [7] (noted as DPRPROP) on a very hard practical classification problem and compared it with RPROP algorithm, who was founded to be the fastest training algorithm tested on the same problem. The problem is from the medical domain, consisting of a big data set, which are mesurements of the thyroid gland. Each mesurements vector of the data set consists of 21 values (15 binary and 6 analog values) and is assigned to a class from three classification classes, which correspond to the hyper-, hypo- and normal function of the thyroid gland. The training set consists of 3772 learning examples and the testing set of 3428 patterns.

The architecture of the fully interconnected network consists of three layers with 21 inputs, 10 hidden neurons and 3 output neurons. The weights are initialized randomly with values from the interval $[-1, 1]$. The parameters of the RPROP algorithm was set to $u = 0.4$, $d = 0.9$, $\eta_{max} = 50$, $\eta_{min} = 0.000001$. The patterns of the training set are presented sequentially to the input layer of the network at each epoch. The weights are adapted at the end of each epoch. Each algorithm has a training period of 5000 epochs. Because 92 percent of the patients are not hyperthyroid, a good classifier must be significant better that 92 percent. Table 1 shows the training and test results for three different initial values for the learn parameters $\eta_{ij}(0)$.

$w_0 \in [-1,1]$	$\eta_{ij}(0)$	Training Set		Test Set	
		Error	Recog. rate (%)	Error	Repog. rate (%)
DPRPROP	6.001	29.36	99.62	108.23	97.75
	0.03	35.32	99.57	134.9	97.95
	0.9	28.54	99.62	104.89	97.4
RPROP	0.041	43.97	99.21	124.49	96.75
	0.41	45.14	29.33	938.55	97.63
	0.1	47.6	99.31	117.98	97.72

Table 1. Simulation results for the thyroid classification problem

We see that the best case by the training for DPRPROP is for $\eta_{ij}(0) = 0.1$ and for RPROP $\eta_{ij}(0) = 0.01$. Figure 1 shows the convergence behaviour for these two cases.

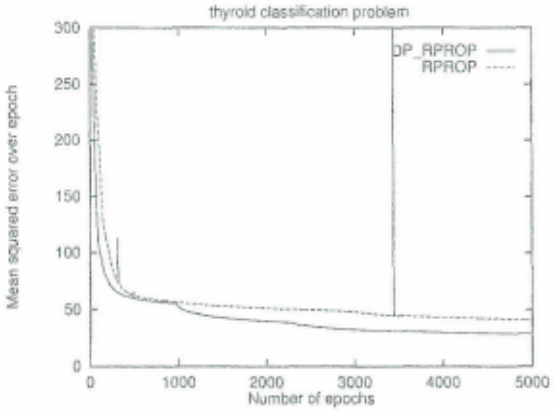

Fig. 1. Convergence behavior for the thyroid classificatioy problem

5 Discussion

We have presented in this paper a new hybrid method for training a multilayer perceptron by applying dynamic programming and we saw that it performed better than the RPROP gradient-based method. The difficult point in developing the new method was in finding a decomposition scheme for the optimization problem, since the cost function is not decomposable into the variable weights. This was the limitation why we have applied dynamic programming just for the weights of the output neurons. We have combined our approach with the a gradient method, but there are certain other optimization methods worth studying.

References

1. D.P. Bersekas and J.N. Tsitsiklis. *Neuro-Dynamic Programming.* Athena Scientific, 1996.
2. A. Calu. Improved convergence through dynamic programming approach. *Cybernetic & Systems: An International Journal, submitted,* 2000.
3. R.A. Jacobs. Increased rates of convergence through learning rate adaptation. *Neural Networks,* 1:295–307, 1998.
4. S. McLoone, M.D. Brown, and G. Irwin. A hybrid linear/nonlinear training algorithm for feedforward neural networks. *IEEE Transactions on Neural Networks,* 9(4):669–684, 1998.
5. M.F. Møller. A scaled conjugate gradient algorithm for fast supervised learning. *Neural Networks,* 6:525–533, 1993.

6. D.E. Rumelhart, G.E. Hinton, and R.J Williams. Learning representations by error propagation. In *Parallel Distributed Processing: Explorations in the Microstructure of Cognition.* D.E.Rumelhart and J.L.McClelland, editors, MIT Press, 1986.
7. W. Schiffmann, M. Joost, and R. Werner. Comparison of optimized backpropagation algorithms. *Proceeding of European Symposium on Artificial Neural Networks, Brussels,* pages 97–104, 1993.
8. M. Sniedovich. *Dynamic Programming.* Marcel Dekker, Inc., New York, 1992.
9. R.S. Sutton. Learning to predict by the methods of temporal differences. *Machine Learning,* 3:9–44, 1988.
10. N. Tepedelenlioglu, A. Rezgui, R. Scalero, and R. Rosario. Fast algorithms for training multilayer perceptrons. In *Neural and Intelligent Systems Integration,* chapter 4. Branko Souček and the IRIS Group, 1991.
11. G.J. Tesauro. Practical issues in temporal difference learning. *Machine Learning,* 8:257–277, 1992.
12. P.P. Van der Smagt. Minimization methods for training feedforward neural networks. *Neural Networks,* 7:1–11, 1994.
13. A. Van Ooyen and B. Nienhuis. Improving the convergence of the back-propagation algorithm. *Neural Networks,* 5:465–471, 1992.
14. B. Verma. Fast training of multilayer perceptrons. *IEEE Transactions on Neural Networks,* 8(6):1314–1320, 1997.
15. G.J. Wang and C.C. Chen. A fast multilayer neural-network training algorithm based on the layer-by-layer optimizing procedures. *IEEE Transactions on Neural Networks,* 7(3):768–775, 1996.