J. O. Moody and P. J. Antsaklis, "The Dependence Identification Neural Network Construction Algorithm," 1 994 I nternational C o nference o n N eural N etwork s , Vol VII, pp 4799-4804, Olrando, FL, June 26-July 2, 1994.

The Dependence Identification Neural Network Construction Algorithm

John O. Moody and Panos J. Antsaklis
Department of Electrical Engineering
University of Notre Dame
Notre Dame, IN 46556

Abstract

An algorithm for constructing and training multilayer neural networks, dependence identification, is presented in this paper. Its distinctive features are that (i) it transforms the training problem into a set of quadratic optimization problems that are solved by a number of linear equations and (ii) it constructs an appropriate network to meet the training specifications. The architecture and network weights produced by the algorithm can also be used as initial conditions for further on-line training by backpropagation or a similar iterative gradient descent training algorithm if necessary. In addition to constructing an appropriate network based on training data, the dependence identification algorithm significantly speeds up learning in feedforward multilayer neural networks compared to standard backpropagation.

1 Introduction

The main tool for training multilayer neural networks is gradient descent, as used by the backpropagation (BP) algorithm developed by Rumelhart [7]. Gradient descent algorithms are susceptible to local minima, sensitive to initial conditions, and slow to converge. Gradient descent can work quite well with the appropriate set of of initial conditions and with a proper network architecture, but using random initial conditions and guessing at the network architecture usually leads to a slow and ponderous training process. The designer must specify the number of network layers and the number of neurons in the "hidden layers" when using basic backpropagation. A method of quickly specifying a network architecture and set of intitial weight values for possible further training through gradient descent would be extremely valuable.

The dependence identification algorithm (see also [4] and [5]) is proposed in section 2 of this paper. Section 3 gives examples of neural network construction, and concluding remarks appear in section 4.

2 Dependence Identification

This section details a method called *dependence identification* (DI) for constructing *multi-layer* neural networks. The method is based on the ability to easily solve single layer neural network training problems. Many methods exist [3] for training single layer neural networks including steepest descent, least mean squares, and the "perceptron training algorithm" (for single layer classification problems). In [8], Sartori and Antsaklis proposed quadratic optimization which transforms the nonlinear training cost function of a single layer network into a quadratic one. This method reduces the problem to solving a set of linear equations and is the method used in the examples of section 3.

The network is to be constructed and trained to approximate a function with m inputs and n outputs. The training set consists of p input/output patterns. The training patterns are stored in the following matrices:

$$U \in \mathbb{R}^{p \times m} \qquad D \in \mathbb{R}^{p \times n}$$

where U is the matrix of input patterns and D is matrix of desired output patterns. The network is constructed one layer at a time, using the rules for single layer network construction. The desired hidden layer outputs are equal to portions of the actual desired outputs. The hidden layer activations (outputs) for a layer are used as the input matrix U for the next layer.

2.1 The Algorithm

To construct the network, first attempt to create a single layer neural network using an appropriate single layer training algorithm. Compute the error of the single layer network. If the error is acceptable (less than a given tolerance) then training is complete and the solution is a single layer network, otherwise create a layer of hidden neurons each of which gets a subset of the complete set of training patterns matched correctly, i.e.,

J. O. Moody and P. J. Antsaklis, "The Dependence Identification Neural Network Construction Algorithm," 1 994 I nternational C o nference o n N eural N etwork s , Vol VII, pp 4799-4804, Olrando, FL, June 26-July 2, 1994.

every pattern should be classified correctly by at least one hidden layer neuron. To do this choose an $m \times m$ portion of U (it is assumed that there are more training patterns than there are inputs, i.e., $p > m$.), and the corresponding $m \times n$ portion of D. Use these patterns to solve a single layer neural network training problem as above. It is guaranteed that a unique zero error solution to this problem exists if the $m \times m$ portion of U is full rank; see [9]. Repeat this procedure until every pattern is matched by at least one hidden layer neuron. Patterns in U and D which are correctly matched by a single hidden layer neuron are linearly dependent after being operated on by the nonlinear neural activation function. This is the reason for the name dependence identification. The hidden layer outputs are then treated as inputs to form a new layer. Layers may be added to the network until a maximum number of layers has been added or the network error is within the desired tolerance.

Dependence identification is presented formally in Algorithm 2.1. The inputs to the algorithm include the training patterns U and D as well as the neural activation function ϕ $\{\phi : \mathbb{R} \rightarrow \mathbb{R}\}$, the maximum number of layers $h_{\max}$, and two tolerances ϵ_N and ϵ_p. The acceptable error for the entire network is given by ϵ_N. The other tolerance, ϵ_p, is used to determine whether the output of the network for an individual pattern is within bounds. Larger values for ϵ_p give fewer hidden layer neurons as discussed in section 2.2. The algorithm uses the notation $\Phi(A)$ to mean that every element of the matrix A should be operated on by the nonlinear activation function ϕ.

Algorithm 2.1 (Dependence Identification).
input $U \in \mathbb{R}^{p \times m}$, $D \in \mathbb{R}^{p \times n}$, $\phi : \mathbb{R} \rightarrow \mathbb{R}$, ϵ_N, ϵ_p, $h_{\max}$.
Set $h = 1$
repeat
  Solve $\min_{W_h}$ trace $((\Phi(UW_h) - D)^T(\Phi(UW_h) - D))$, $W_h \in \mathbb{R}^{m \times n}$.
  Compute error:
  $e =$ trace $((\Phi(UW_h) - D)^T(\Phi(UW_h) - D))$
  if $(e > \epsilon_N)$ **and** $(h < h_{\max})$ **then**
    Set G = **empty matrix,** $l = 0$
    Set $U_{\text{unmarked}} = U$, $D_{\text{unmarked}} = D$
    repeat
      Choose $u \subset U_{\text{unmarked}}$, $d \subset D_{\text{unmarked}}$ **where** $u \in \mathbb{R}^{m \times m}$, $d \in \mathbb{R}^{m \times n}$
      if less than m **patterns are unmarked then augment** u **and** d **with previously marked patterns.**
      Solve:
      $\min_g$ trace $((\Phi(ug) - d)^T(\Phi(ug) - d))$, $g \in \mathbb{R}^{m \times n}$.
      Add the solution g **as new column(s) of** G.
      $l \leftarrow l + n$
      Mark all patterns which satisfy $U_{\text{unmarked}} g = V_{\text{unmarked}}$ **within the tolerance** ϵ_p.
    until all patterns have been marked.
    Find the hidden layer outputs:
    $H = \Phi(UG) \in \mathbb{R}^{p \times l}$.
    Add a zero column to G **with a single nonzero element corresponding to the constant input.**
    $W_h \leftarrow G$
    $U \leftarrow H$
    $m \leftarrow l$
    $h \leftarrow h + 1$
until $(e \leq \epsilon_N)$ **or** $(h = h_{\max})$
output W_i **for** $i = 1, \ldots, h$ **and the error** e

J. O. Moody and P. J. Antsaklis, "The Dependence Identification Neural Network Construction Algorithm," 1 994 I nternational C o nference o n N eural N etwork s , Vol VII, pp 4799-4804, Olrando, FL, June 26-July 2, 1994.

2.2 Hidden Layer Neurons

The dependence identification algorithm constructs a network which solves the *single layer problem* within the desired accuracy, if such a solution actually exists. The *number of layers* created can be bounded with the parameter $h_{\max}$. The *number of hidden layer neurons* depends on the number of layers and the desired accuracy. Several authors have shown (with different degrees of ease and generality) that p neurons in the hidden layer suffice to store p arbitrary patterns[1] in a two layer network; see [1], [6], and [9] for constructive proofs.

The bound on the number of hidden layer neurons created by dependence identification is proportional to the bound of p described above. Assume that the $m \times m$ portions of U are full rank and that the number of network layers is limited to two (one hidden layer). The U matrix has p rows, so it will be broken into at most ceiling($\frac{p}{m}$) linearly dependent groups[2] in order to form the hidden layer neurons. This bound is achieved when every group is made of at most m patterns. In practice the number of groups can be much smaller than ceiling($\frac{p}{m}$) since larger groups of linearly dependent patterns may be found. Every linearly dependent groups yields n hidden layer neurons (one for each output). An extra neuron which has a constant output is then added to the hidden layer in order to help form constant offsets and/or thresholds for the output layer. Thus the dependence identification algorithm has an upper bound of ceiling($\frac{p}{m}$)$n+1$ hidden layer neurons for a two network and assuming the $m \times m$ portions of U are full rank. The dependence identification algorithm assumes that one input is a constant used to form thresholds for the first layer's neurons. Thus for a SISO network $m = 2$ and $n = 1$ and the bound on the number of hidden layer neurons created by dependence identification is approximately one half the bound of p.

There may be situations where dependence identification determines a number of neurons and/or layers that may not be physically realizable due to hardware or software constraints. The number of layers may be limited by the parameter $h_{\max}$ in Algorithm 2.1, and the number of hidden layer neurons may be decreased by increasing the tolerance ϵ_p. This value is used to determine whether a pattern should be considered part of a linearly dependent set, i.e., if a pattern subset u, d and associate weight g satisfies $\Phi(ug) = d$, then another pattern $\hat{u}, \hat{d}$ is considered "linearly dependent" if the elements of $|\Phi(\hat{u}g) - \hat{d}|$ are all less than ϵ_p. Linear dependence is written in quotes here because the nonlinear activation function is being used to determine a pattern match. An example of hidden layer neuron reduction is given in section 3.2. Further discussion and implementation concerns regarding dependence identification can be found in [4].

3 Examples

Programs are written in C to perform backpropagation (BP) and dependence identification (DI) and are run on Sun SPARCstation 2 workstations. The single layer training problems are solved with quadratic optimization [8] and the block conjugate residual algorithm [2], [4]. Section 3.1 shows a comparison between networks constructed with dependence identification to networks trained through trial and error and backpropagation. Section 3.2 shows how the number of hidden layer neurons may be reduced by increasing the DI tolerance value ϵ_p. The overall error of the network is kept down, while the tolerance is increased, by recomputing the weights after each linearly dependent set of training patterns is found. Section 3.3 shows how dependence identification may be successfully used to construct a network architecture and set of initial conditions for further training by backpropagation.

3.1 Function Approximation Examples

Dependence identification has been tested on several multi-input/multi-output functions with the number of training patterns ranging as high as 2000. The functions were also trained using backpropagation with random initial conditions.

The activation function for the networks created by backpropagation (BP) and dependence identification (DI) for all three functions is $\phi(x) = \tanh(x)$. The training results are summarized in table 1. The column

[1] The bound of p assumes that one input to the network is constant, if the neural thresholds are handled differently then the bound is $p - 1$.

[2] The notation ceiling(x) means the smallest integer greater than or equal to x.

J. O. Moody and P. J. Antsaklis, "The Dependence Identification Neural Network Construction Algorithm," 1 994 I nternational C o nference o n N eural N etwork s , Vol VII, pp 4799-4804, Olrando, FL, June 26-July 2, 1994.

for network architecture has entries of the form $m - h - n$ where m is the number of network inputs, n is the number of outputs, and h is the number of hidden layer neurons.

	Training Method	Network Architecture	Square Error (Training)	Square Error (Test Set)	Time to Solution (seconds)
1 input/1 output Function	DI	2 – 9 – 1	0.2923	1.8023	0.0167
	BP	2 – 10 – 1	0.9993	3.8570	274.8
2 input/2 output Function	DI	3 – 103 – 2	0.3547	1.5508	15.14
	BP	3 – 103 – 2	0.7902	2.1196	4429.25
3 input/1 output Function	DI	4 – 286 – 1	0.4495	0.1221	291.13
	BP	4 – 50 – 1	1.2661	0.2475	3026.93

Table 1: Comparative results of network training methods.

The process of determining BP learning parameters, the number of hidden layer neurons, and the stopping requirement for backpropagation is a tedious trial and error procedure. The time required to actually determine an appropriate set of parameters is not included as part of the solution time for backpropagation. The times in table 1 are actual CPU times spent performing network construction and training. These times illustrate that even when the learning rates and network architecture are known, dependence identification computes an entire network with similar error in much less time than is required by the gradient descent performed by backpropagation. However the most important contribution of dependence identification is that the search for the appropriate parameters is eliminated and an appropriate network architecture is an output of the algorithm, not an input.

3.2 Increased Tolerance and Recomputed Weights

Hardware and/or memory restrictions may limit the number of hidden layers or hidden layer neurons that may be used in a particular neural network. Section 2.2 mentions that it is possible to decrease the number of hidden neurons created by the dependence identification algorithm by increasing the tolerance, ϵ_p, used to check whether or not a pattern should be included in a set of "linearly dependent" patterns. Unfortunately as the parameter ϵ_p is increased and the number of hidden layer neurons decreases, the overall error of the network tends to increase. The increase in network error can be combated by recomputing the weight matrix g (see Algorithm 2.1) for the entire set of linearly dependent patterns after this set has been found. Figure 1a shows how the number of hidden layer neurons decreases as the tolerance ϵ_p is increased from 0.05 to 5.00. The training patterns are for 1 input / 1 output function approximation example from section 3.1. The network is constrained to have three layers[3] ($h_{\max} = 3$). Note that in this example the dependence identification algorithm determines linear dependence on the basis of the quadratic error cost used by quadratic optimization [8] which allows the error tolerance to be increased past $\epsilon_p = 2$ up to $\epsilon_p = 5$. Figure 1a shows that *the number of hidden layer neurons can be successfully decreased with and without the recomputation of the weights.* The reason the number of hidden layer neurons is different when the weights are recomputed is that a three layer network is being constructed. The inputs produce a set number of neurons in the first hidden layer dependent on ϵ_p only, not on whether the weights are recomputed. The recomputation of weights produces different hidden layer activations (outputs) than those produced when the weights are not recomputed. Thus the *second hidden layer may have a different number of neurons when the weights are recomputed* since the inputs to the second hidden layer are different.

Figure 1b shows how the overall error of the network varies as the tolerance is increased. The network error is the final value of e in Algorithm 2.1. The figure shows how *recomputing the weight matrix helps to keep the overall error of the network from blowing up.* It also shows that there is no simple relationship between the overall error and the tolerance ϵ_p. For example, when ϵ_p increases from 3 to 4, the overall error when the weights are not recomputed actually decreases from about 9.2 to 3.2. However the trend is that the recomputation keeps the error down. The average error for the fourteen different tolerances used in the

[3]The networks in section 3.1 have two layers

J. O. Moody and P. J. Antsaklis, "The Dependence Identification Neural Network Construction Algorithm," 1 994 I nternational C o nference o n N eural N etwork s , Vol VII, pp 4799-4804, Olrando, FL, June 26-July 2, 1994.

a) Decrease of hidden layer neurons. b) Overall network error.

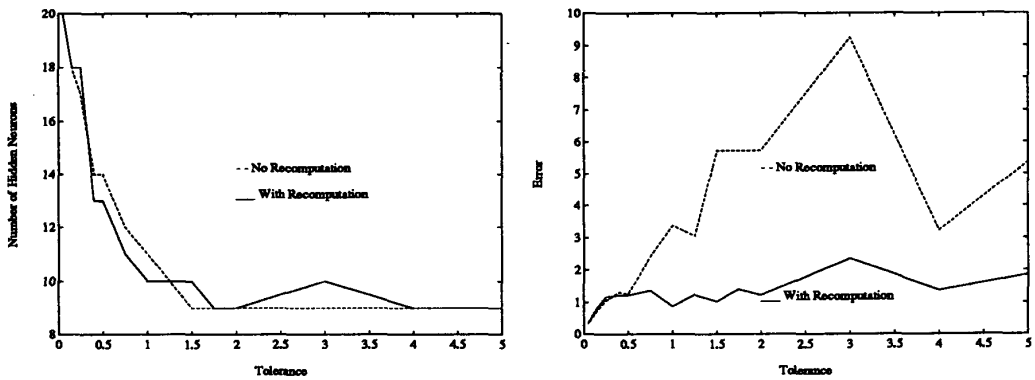

Figure 1: Effects of increasing tolerance ϵ_p.

figure is 3.47 when the weights are not recomputed and 1.25 when they are. It is possible that an error of 1.25 may be higher than the degree of approximation accuracy required for a particular problem. The next section shows how dependence identification can be used to identify an appropriate network architecture and set of initial conditions that will allow backpropagation to quickly decrease the overall network error.

3.3 Dependence Identification for Backpropagation Initial Conditions

Backpropagation is extremely sensitive to initial conditions and there are some problems that are inherently very difficult for backpropagation to solve due to a large number of local minima and relatively flat sections of the cost function. One of these difficult problems is the parity checker [10]. A parity checker receives a number of boolean inputs. In this example, the boolean values are 1 and -1. The parity checker has one boolean output which should be 1 when there is an odd number of 1's at the input and -1 when there are an even number of 1's at the input.

A six input parity checker is used in this problem. The number of network inputs is seven because one input is constant and is used to create biases. The number of training patterns is 64 which includes all possible combinations of six boolean inputs ($2^6 = 64$). Dependence identification is used to find an appropriate two layer network architecture of $7 - 7 - 1$. Networks are then trained using backpropagation with random initial conditions and with the initial conditions obtained from dependence identification. Table 2 shows the results. Patterns are considered to be identified correctly if the network has the proper sign, i.e, a desired output of 1 is classified correctly if the network output is positive and a desired output of -1 is classified correctly if the network output is negative. The entry for "% Error" is calculated as the number of missed patterns divided by 64.

The table shows that, by itself, dependence identification does not produce a very successful solution to the problem. However *the initial conditions it produces are quite good for improving the network with backpropagation*, bringing the error down from about 27% to just over 3%. When random initial conditions are used, *backpropagation is completely unsuccessful in learning the problem*, misclassifying more than half of the input space.

4 Conclusions

A new method of constructing feedforward multi-layer neural networks has been presented, and examples show that it works well for creating neural network approximations of continuous functions as well as providing a starting point for further backpropagation training. Dependence identification is a faster and more systematic method of developing initial network architectures than trial and error or pruning techniques used with gradient descent. Network size and construction time are conserved since dependence identifica-

J. O. Moody and P. J. Antsaklis, "The Dependence Identification Neural Network Construction Algorithm," 1 994 I nternational C o nference o n N eural N etwork s , Vol VII, pp 4799-4804, Olrando, FL, June 26-July 2, 1994.

Training Method	Correctly Classified Patterns	% Error
BP with initial conditions from DI	62	3.12%
Dependence Identification	47	26.56%
BP with random initial conditions	30	53.12%

Table 2: Results of parity checker training

tion builds a small network up instead of whittling a huge network down, and because it relies on the ability to quickly solve single layer training problems instead of ponderous gradient descents on entire multilayer networks.

Dependence identification relaxes the constraint that neural activation functions be continuously differentiable (assuming this is not a requirement of the particular single layer training method), and it determines the number of hidden layer neurons and layers as part of its operation. Dependence identification does not require trial and error with learning rates like backpropagation does. There may well be situations with specific applications where DI indicates a number of hidden layer neurons that can not be physically implemented (due to memory or hardware constraints). The number of hidden layer units can be decreased by increasing the tolerance ϵ_p in Algorithm 2.1. The number of layers can also be limited with the parameter $h_{\max}$. The speed of DI makes it appropriate for creating neural network architectures and initial weight values to be used in real applications.

References

[1] Baum E. B., "On the Capabilities of Multilayer Perceptrons," *J. Complexity*, vol 4, pp 193–215, 1988.

[2] Elman, H. C., "Iterative Methods for Large Sparse Nonsymmetric Systems of Linear Equations," Ph.D. thesis, Computer Science Dept., Yale University, New Haven, CT., 1982.

[3] Hertz, J., Krogh, A. and Palmer, R. G., *Introduction to the Theory of Neural Computation,* Addison-Wesley Publishing Company, 1991.

[4] Moody, J. O., "A New Method for Contructing and Training Multilayer Neural Networks," Master's thesis, Dept. of Electrical Engineering, University of Notre Dame, Notre Dame, IN., 1993.

[5] Moody, J. O., Antsaklis, P. J., "The Dependence Identification Neural Network Construction Algorithm," Technical report ISIS-93-005 of the ISIS Group at the University of Notre Dame, Notre Dame, IN., 1993.

[6] Nilsson N. J., *Learning Machines*, McGraw-Hill, 1965.

[7] Rumelhart, D. E., Hinton, G. E. and Williams, R. J., "Learning Internal Representations by Error Propagation," in Rumelhart, D. E. and McClelland, J. L., eds. *Parallel Distributed Processing: Explanations in the Microstructure of Cognition, vol 1: Foundation*, pp. 318–362, MIT Press, 1986.

[8] Sartori, M. A. and Antsaklis, P. J., "Neural Network Training via Quadratic Optimization", Proc of ISCAS, San Diego, CA, May 10–13, 1992.

[9] Sartori, M. A. and Antsaklis, P. J., "A Simple Method to Derive Bounds on the Size and to Train Multi-Layer Neural Networks", IEEE Trans. on Neural Networks, Vol 2, No 4, pp. 467–471, July 1991.

[10] Tesauro, G. and Janssens, B., "Scaling Relations in Backpropagation learning", *Complex Systems*, Vol 2 pp. 39–44, 1988.