J. O. Moody and P. J. Antsaklis, "Neural Networks Construction and Rapid Learning for System Identification," P roc o f t he 8 th I EEE I nternational S ymposium o n I ntelligent C ontro l , pp. 475-480, Chicago, IL, August 25-27, 1993.

Proceedings of the 1993 International Symposium on Intelligent Control Chicago, Illinois, USA - August 1993

Neural Network Construction and Rapid Learning for System Identification

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

Abstract

A very large number of potential applications for feedforward neural networks in the field of control theory have been proposed in recent years. The main method used for training such networks is gradient descent, such as the backpropagation algorithm, which is slow and often impractical for real time applications. In this paper a new learning algorithm is introduced and used to identify nonlinear functions. Its distinctive features are that it transforms the problem to a quadratic optimization problem that is solved by a number of linear equations and it constructs the appropriate network that will meet the specifications. The architecture and network weights produced by the algorithm are suitable for further on-line training by backpropagation since the initial conditions produced by the algorithm provide a small initial error. The quadratic optimization/dependence identification algorithm extends the results of quadratic optimization single layer network training and significantly speeds up learning in feedforward multilayer neural networks compared to standard backpropagation.

1 Introduction

In recent years, multilayer feedforward neural networks have been the subject of a great amount of research by control engineers. The ability to learn arbitrary nonlinear functions through neural learning, as well as the potential for extremely fast parallel computations, brought about numerous application ideas in the fields of nonlinear, adaptive and intelligent control.

The type of neural network most commonly used in control systems is the *feedforward multilayer neural network*, where no information is fed back during operation. There is however feedback information available during training. Supervised learning methods, where the neural network is trained to learn a sequence of input/output patterns, are typicaly used.

One property of multilayer neural networks central to most applications in control is that of function approximation. It has been shown [5] that these networks can approximate, arbitrarily well, any continuous function. To model the input/output behavior of a dynamical system, the neural network is trained using input/output data and the weights of the neural network are adjusted using some training algorithm, most often gradient descent. Modeling system behavior via multilayer sigmoidal neural networks has been studied by a number of researchers; see among others Bhat et al. [3], Hou and Antsaklis [8], and Narendra and Parthasarathy [10].

In general there are potential applications of neural networks at all levels of hierarchical intelligent controllers that provide higher degrees of autonomy to systems. Neural networks are useful at the lowest execution level where the conventional control algorithms are implemented via hardware and software, through the coordination level, to the highest organization level, where decisions are being made based on possibly uncertain and/or incomplete information. An extended discussion of these issues can be found in [1].

The main tool for training multilayer neural networks is gradient descent, such as the backpropagation algorithm developed by Rumelhart [11]. 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

0-7803-1206-6/93/$3.00 ©1993 IEEE

J. O. Moody and P. J. Antsaklis, "Neural Networks Construction and Rapid Learning for System Identification," P roc o f t he 8 th I EEE I nternational S ymposium o n I ntelligent C ontro l , pp. 475-480, Chicago, IL, August 25-27, 1993.

architecture, but using random initial conditions and guessing at the network architecture usually leads to a slow and ponderous training process. Backpropagation requires that the nonlinear neural activation functions be restricted to those with continuous derivatives. The designer must specify the number of network layers and the number of neurons in the "hidden layers" when using basic backpropagation.

The dependence identification algorithm is proposed in this paper. Section 2 presents the method of quadratic optimization, a major tool used by the dependence identification algorithm presented in section 3. Section 4 gives examples showing comparisons of neural network construction using dependence identification versus backpropagation training. Concluding remarks appear in section 5.

2 Quadratic Optimization

Many methods exist for training single layer neural networks including steepest descent, least mean squares, and the "perceptron training algorithm" [7]. Sartori and Antsaklis propose a method known as quadratic optimization [12] which transforms the nonlinear training cost function of a single layer network into a quadratic one. Suppose a neural 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 following matrices can then be constructed:

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

where U is the matrix of input patterns and D is the matrix of corresponding desired output patterns. The output $Y \in \mathbb{R}^{p \times n}$ of a single layer neural network to the input U is then

$$Y = \Phi(UW)$$

where $W \in \mathbb{R}^{m \times n}$ is the neural weight matrix and $\Phi : \mathbb{R}^{p \times n} \rightarrow \mathbb{R}^{p \times n}$ is the nonlinear neural activation function, the same function $\phi : \mathbb{R} \rightarrow \mathbb{R}$ is performed on each of the $p \times n$ elements of UW in order to form Y.

The neural network training problem involves making Y as close an approximation as possible to D. If this "closeness" is defined in the least mean squares sense, then the neural network training problem is to find a set of weights such that

$$\min_{w_{ij}} \sum_{k=1}^{p} \sum_{j=1}^{n} (y_j^k - d_j^k)^2$$

where $i = 1, \ldots, m$. The problem can also be expressed in the matrix case:

$$\min_{W} \text{ trace } ((Y - D)^T (Y - D)) =$$

$$\min_{W} \text{ trace } ((\Phi(UW) - D)^T (\Phi(UW) - D)) \quad (1)$$

In order to deal with the nonlinearity of equation (1) consider defining a new matrix $V \in \mathbb{R}^{p \times n}$ such that

$$\Phi(V) = D \quad (2)$$

If Φ is bijective then we can take $V = \Phi^{-1}(D)$, however it is not necessary that a one-to-one inverse of Φ exist. For example, the step function is surjective for the domain of $\mathbb{R}$ and the range $\{0, 1\}$ i.e., it has an infinite number of inputs which will evaluate to 1 and an infinite number of inputs which will evaluate to 0. V can still be defined by assigning some positive number to v_{ij} whenever d_{ij} is 1 and assigning some negative number to v_{ij} whenever d_{ij} is 0. The numbers assigned may be random or some prechosen constants. With the change of variables in effect, the single layer neural network training problem is transformed to

$$\min_{W} \text{trace } ((UW - V)^T (UW - V)) \quad (3)$$

which is equivalent to solving

$$U^T U W = U^T V$$

for W if $U^T U$ is positive definite (full rank).

Of course the real problem to be solved is the nonlinear problem given by equation (1). The relationship between the nonlinear and quadratic will not be discussed here except to mention that if a zero error solution to equation (1) exists, then there exists a V such that the same solution can be found by solving equation (3). When a zero error solution to (1) does not exist then the relationship between the two solutions is more subtle. A more complete discussion of the error relationship can be found in [12].

J. O. Moody and P. J. Antsaklis, "Neural Networks Construction and Rapid Learning for System Identification," P roc o f t he 8 th I EEE I nternational S ymposium o n I ntelligent C ontro l , pp. 475-480, Chicago, IL, August 25-27, 1993.

3 Dependence Identification

This section details a method called *dependence identification* (DI) for constructing *multilayer* neural networks, as opposed to the single layer construction method detailed in the previous section. 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 matrices U and D are the same as defined in section 2. 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.

To construct the network, first attempt to create a single layer neural network by solving equation (3) using the method of quadratic optimization. Compute the error of the single layer network. If the error is acceptable then training is done, otherwise create a layer of hidden neurons which get portions of the output matched correctly, i.e., 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 zero error solution to this problem exists if the $m \times m$ portion of U is full rank. Repeat this procedure until every pattern is matched by at least one hidden layer neuron. Patterns in U and the transformed variable V which are correctly matched by a single hidden layer neuron are linearly dependent, which is the origin of the term 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.

The algorithm is presented formally in Algorithm 1. The inputs to the algorithm include the training patterns U and D as well as the neural activation function ϕ, 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.

Algorithm 1 (Dependence Identification).

input $U \in \mathbb{R}^{p\times m}$, $D \in \mathbb{R}^{p\times n}$, $\phi : \mathbb{R} \to \mathbb{R}$, ϵ_N, ϵ_p, $h_{\max}$.
Create V such that $\Phi(V) = D$
Set $h = 1$
repeat
  Solve $\min_{W_h}$ trace $((UW_h - V)^T(UW_h - V))$, $W_h \in \mathbb{R}^{m\times n}$.
  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 $U_{\text{unmarked}} = U$, $V_{\text{unmarked}} = V$
    repeat
      Choose $u \subset U_{\text{unmarked}}$, $v \subset V_{\text{unmarked}}$ where $u \in \mathbb{R}^{m\times m}$, $v \in \mathbb{R}^{m\times n}$
      if less than m patterns are unmarked then augment u and v with previously marked patterns.
      Solve:
      $\min_g$ trace $((ug-v)^T(ug-v))$, $g \in \mathbb{R}^{m\times n}$.
      Add the solution g as new column(s) of G.
      $l \leftarrow l + n$
      Mark all patterns which do not 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 $(e \le \epsilon_N)$ or $(h = h_{\max})$
output W_i for $i = 1, \ldots, h$ and the error e

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, e.g. [2], have shown (with different degrees of ease and generality) that p neurons in the hidden layer suf-

J. O. Moody and P. J. Antsaklis, "Neural Networks Construction and Rapid Learning for System Identification," P roc o f t he 8 th I EEE I nternational S ymposium o n I ntelligent C ontro l , pp. 475-480, Chicago, IL, August 25-27, 1993.

fice to store p arbitrary patterns in a two layer network[1]. The dependence identification algorithm has an upper bound of ceiling$(\frac{p}{m})n + 1$ hidden layer neurons[2], 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.

The algorithm's solution is based on solving a succession of *systems of linear equations.* Krylov subspace methods, like a block form of the conjugate residual algorithm [6], are recommended since they are iterative methods for solving linear equations, working toward minimizing a quadratic error function instead of attempting to solve the problem exactly in a single complicated step. These methods are guaranteed to converge in a number of steps equal to the order of the system, assuming that the problem is sufficiently well conditioned [9]. The speed of these algorithms is a key to the overall speed of dependence identification.

The dependence identification algorithm can not only be used to construct continuous sigmoidal networks but can also be used to construct networks that use discontinuous switching functions. The discontinuities of the these functions prevent them from being used with standard gradient descent training.

Dependence identification is, as presented, a batch training process, which might make it inappropriate for certain on-line training tasks. However dependence identification is very useful for creating an initial network that may be further trained on-line using some sort of gradient descent that responds to each new training pattern as it is presented to the network. The following section shows the superiority of developing this initial network with dependence identification as opposed to starting from scratch with backpropagation.

[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.

4 Examples

Dependence identification has been tested on several multi-input/multi-output functions with the number of training patterns ranging from 4 to 2000. The functions were also trained using backpropagation with random initial conditions. It was discovered that dependence identification could achieve a smaller error than backpropagation while performing the job 10 to 1600 times faster.

Two examples are covered in this paper. The examples include the approximation of a 2 input/2 output function and a 3 input/1 output function. The 2 input/2 output function is given by

$$\begin{aligned} d_1 &= \frac{1}{6}(5\sin(u_1) + \cos(u_2)) \\ d_2 &= \frac{1}{5}(3\cos(u_1) + 2\sin(u_2)) \\ u_1, u_2 &\in [0, 2\pi] \end{aligned} \tag{4}$$

The 3 input/1 output function is given by

$$\begin{aligned} d &= \frac{1}{10}(e^{u_1} + u_2 u_3 \cos(u_1 u_2) + u_1 u_3) \\ u_1 &\in [0, 1] \\ u_2, u_3 &\in [-2, 2] \end{aligned} \tag{5}$$

The activation function for the networks created by backpropagation (BP) and dependence identification (DI) is $\phi(x) = \tanh(x)$. A training set for function (4) is created by generating 200 uniformly distributed random values of u_1 and u_2 and calculating the associated values of d_1 and d_2. The training set for function (5) contains 2000 patterns. The learning parameters for back-propagation are $\eta = 0.005$ and $\alpha = 0.1$, where η is the learning step size along the cost gradient and α is the momentum factor. The BP iterations are halted[3] when the overall square error is less than 0.8 for function (4) and after 1 million BP iterations for function (5). The programs for both quadratic optimization and

[3]The process of determining η, α, the number of hidden layer neurons and the stopping requirement for backpropagation is an exhausting trial and error procedure that involves modifying the network architecture and working from high learning rates to lower ones. The time required to actually determine an appropriate set of parameters is not included in the reported times for solutions using backpropagation.

J. O. Moody and P. J. Antsaklis, "Neural Networks Construction and Rapid Learning for System Identification," P roc o f t he 8 th I EEE I nternational S ymposium o n I ntelligent C ontro l , pp. 475-480, Chicago, IL, August 25-27, 1993.

	Training Method	Network Architecture	Square Error (Training)	Square Error (Test Set)	Time to Solution (seconds)
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.

backpropagation are written in C and are run on Sun SPARCstation 2 computers. The results are summarized in table 1. The training times given in the table are the actual CPU times used to calculate the solutions. Note that dependence identification derived a lower error solution almost 300 times faster than back-propagation for function (4) and 10 times faster for function (5), despite the fact that DI must deal with very large matrices in order to construct the network for function (5). The number of hidden layer neurons used by BP was kept low while approximating function (5) due to the excessively long training times required.

Figures 1a and 1b show the results of testing the approximations of function (4) with u_1 and u_2 set to parametrized functions of t, which takes on 100 evenly spaced values within the input range. Figures 1c and 1d show the results of testing the approximations of function (5). Table 1 shows that dependence identification performs better on the test sets as well as the training sets.

5 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. The new method is faster than the gradient descent of backpropagation and is much more systematic since the actual network architecture is determined by the algorithm. Dependence identification relaxes the constraint that neural activation functions be continuously differentiable, 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 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 neural control applications.

References

[1] Antsaklis, P. J., "Neural Networks for the Intelligent Control of High Autonomy Systems", *Mathematical Studies of Neural Networks*, J.G. Taylor, Ed., Elsevier, 1993. To Appear. Also Tech. Report of the ISIS (Interdisciplinary Studies of Intelligent Systems) Group, No. ISIS-92-01, Univ. of Notre Dame, September 1992.

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

[3] Bhat N.V., Minderman P., McAvoy, Wang N., "Modeling Chemical Process Systems via Neural Computation," *IEEE Control Systems Magazine*, vol. 10, no. 3, April 1990.

[4] Chester D. (1990), "Why Two Hidden Layers are Better than One," *Proc Int. Joint Conf. on Neural Networks*, IEEE Publications, pp. 265–268, 1990.

[5] Cybenko, G., "Continuous Value Neural Networks with Two Hidden Layers are Sufficient," *Math. Contr. Signals & Systems*, vol 2, pp. 303–314, 1989.

J. O. Moody and P. J. Antsaklis, "Neural Networks Construction and Rapid Learning for System Identification," P roc o f t he 8 th I EEE I nternational S ymposium o n I ntelligent C ontro l , pp. 475-480, Chicago, IL, August 25-27, 1993.

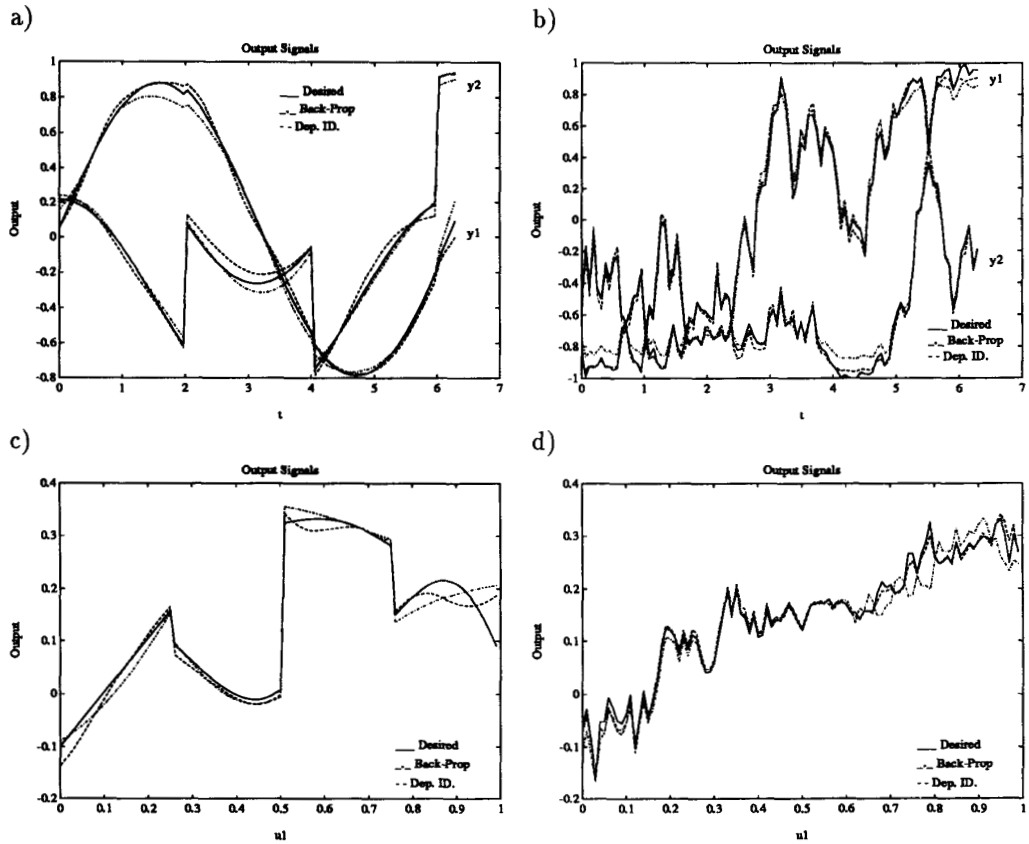

Figure 1: Testing results of the network approximations.

[6] 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.

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

[8] Hou Z., Antsaklis P. J., *Analysis of Auto Powertrain Dynamics and Modeling using Neural Networks*, MSc Thesis, Department of Electrical Engineering, University of Notre Dame, May 1992.

[9] Kincaid, D. and Cheney, W., *Numerical Analysis*, Brooks/Cole Publishing Company, 1991.

[10] Narendra K.S., Parthasarathy K., "Identification and Control of Dynamical Systems Using Neural Networks," *IEEE Transactions on Neural Networks*, vol. 1, no 1, pp. 4–27, March 1990.

[11] 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.

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