12-1991

An Efficient Neural Algorithm for the Multiclass Problem

Rangachari Anand
Syracuse University

Kishan Mehrotra
Syracuse University, mehrotra@syr.edu

Chilukuri K. Mohan
Syracuse University, ckmohan@syr.edu

Sanjay Ranka
Syracuse University

SU-CIS-91-40

An Efficient Neural Algorithm for the Multiclass Problem

R. Anand, K. Mehrotra, C. Mohan, S. Ranka
December, 1991

School of Computer and Information Science
Syracuse University
Suite 4-116, Center for Science and Technology
Syracuse, New York 13244-4100

An Efficient Neural Algorithm for the Multiclass Problem

Rangachari Anand, Kishan Mehrotra, Chilukuri K. Mohan,
and Sanjay Ranka
4-116 Center for Science and Technology,
School of Computer and Information Science,
Syracuse University, Syracuse, NY 13244-4100
(e-mail: kishan@top.cis.syr.edu)

December 11, 1991

Abstract

One connectionist approach to the classification problem, which has gained popularity in recent years, is the use of backpropagation-trained feed-forward neural networks. In practice, however, we find that the rate of convergence of net output error is especially low when training networks for multi-class problems. In this paper, we show that while backpropagation will reduce the Euclidean distance between the actual and desired output vectors, the difference between some of the *components* of these vectors will actually increase in the first iteration. Furthermore, the magnitudes of subsequent weight changes in each iteration are very small, so that many iterations are required to compensate for the increased error in some components in the initial iterations. We describe a modular network architecture to improve the rate of learning for such classification problems. Our basic approach is to reduce a K-class problem to set of K two-class problems with a separately trained network for each of the K problems. We also present the results from several experiments comparing our new algorithm and approach with standard backpropagation, and find that speedups of about one order of magnitude can be obtained.

Key words: Backpropagation, Modular Networks, Classification Problems, Multi-class problems, Feedforward Networks.

1 Introduction

Classification, the assignment of an object to one of a number of predetermined groups, is of fundamental importance in a number of areas ranging from image and speech recognition to the social sciences. Consequently, a number of statistical classification techniques have been developed. These are based primarily on Bayes's rule.

In the classification problem, we assume that a pattern can belong to exactly one of K classes. We are provided with a training set, T, consisting of sample patterns which are representative of all classes along with class membership information for each pattern. Using the training set, we deduce rules for membership in each class and create a classifier, which can then be used to assign other patterns to their respective classes according to these rules.

One connectionist approach to the classification problem, which has gained popularity in recent years, is the use of backpropagation-trained feed-forward neural networks [10]. Backpropagation is based on the method of steepest descent [5], and is one of the most widely used training algorithms for feed-forward neural networks. Since these networks can be taught arbitrary non-linear mappings, it is relatively straightforward to use them for pattern classification tasks [4].

A feed-forward neural network computes a real output vector when presented with a real input vector. The output of the network may be controlled by varying parameters called weights. The training set for a K-class problem consists of a set of pattern vectors representative of each class along with their desired output vectors. When training a network with backpropagation, we start by assigning random values to the weights. In each step of training, the backpropagation algorithm prescribes changes to the weights designed to reduce the magnitude of the difference between the actual output vector and the desired output vector.

When training a network with backpropagation for a two-class problem in which the numbers of exemplars for the two classes differ greatly (i.e., the training set is *imbalanced*), we have observed that the rate of convergence of net output error is especially low. In [1] we have described a modified version of backpropagation which is faster than standard backpropagation for two-class problems with imbalanced training sets by as much as one order of magnitude. In this paper, we consider K-class problems, where $K > 2$, and the training sets for all or most classes are

approximately equal in size. We consider the error vector obtained by considering the errors associated with K output nodes of the neural network and show that while backpropagation will reduce the Euclidean distance between the actual and desired output vectors, the difference between some of the *components* of these vectors will actually increase in the first iteration. Furthermore, the magnitudes of the subsequent weight changes in each iteration are very small. Hence, many iterations are required to compensate for the increased error in some components in the initial iterations.

We propose a modular network architecture to improve the rate of learning for such classification problems. In this architecture, each module is a single-output network which determines whether a pattern belongs to a particular class, thereby reducing a K-class problem to a set of K two-class problems. A module for class $\mathcal{C}_k$ is trained to distinguish between patterns belonging to classes $\mathcal{C}_k$ and its complement $\overline{\mathcal{C}_k}$. If there are approximately equal numbers of exemplars for each of the K classes, there will be many more exemplars for class $\overline{\mathcal{C}_k}$ than for class $\mathcal{C}_k$. This corresponds to the two class problem ($\mathcal{C}_k$ vs. $\overline{\mathcal{C}_k}$) in which the training sets are naturally imbalanced and the modified algorithm in [1] applies.

The modular approach yields a good speedup in training times in several ways:

1. The sum of the numbers of iterations needed to train the individual modules in the modular approach is less than the number of iterations needed to train a nonmodular network for the same task.

2. The time taken for one iteration, when training a module of a modular network, is less than the time taken for one iteration when training the equivalent non-modular network. This is because the modules in a modular network are generally smaller than the equivalent nonmodular network.

3. Since the modules can be trained independently, we can train them in parallel. In our implementation, we have used a simple distributed batch queueing system to train the modules in parallel on a cluster of high-speed workstations.

There has been considerable interest in developing modular neural networks. Studies of human and other brains suggest that there is considerable specialization in different parts of the brain. Minsky [6] describes a model of the human brain viewed as a collection of interacting modules called *agents*. While each agent is capable only

of performing simple actions, the agents collectively behave in an intelligent manner. The approach of this paper differs from such models in that modularity is planned by the network-builder, attempting an optimal use of resources, whereas that is not possible in a completely self-organizing network.

Rueckl et al. [11] have studied the problem of analyzing images in which one of a number of known objects could occur anywhere. The goal was to train a network to identify the location and also recognize the object in the image. They found that training time was shorter when two separate networks were used for this task. Jacobs et al. [3] have studied the problem of training a modular network so that the modules learn to specialize in different tasks. In their architecture, the modules compete for the right to learn particular patterns. They claim that such a system is more robust and will generalize better. However such a system is likely to use far more resources than a system in which modules co-operate on related tasks.

The problem of modularity has been approached from a more pragmatic viewpoint in the area of speech recognition. Waibel et al. [15] have devised a technique called "connectionist glue" by means of which it is possible to train networks for different tasks and then connect them together. In this manner, networks can be built in an incremental manner. However, in this technique, emphasis is on reducing the complexity of the problem by partitioning rather than finding a solution that addresses all of the training set collectively.

The rest of this paper is organized as follows. We present an analysis of the reason for the observed slow rate of convergence of standard backpropagation in section 2. In section 3, we describe a simple modular network architecture which overcomes these difficulties. Each module in such a network is trained with a modified version of backpropagation, which we have described in a previous paper [1]. In section 4 we consider three examples to illustrate the improvement achieved due to modularity and in section 5 we make some concluding remarks.

2 Analysis of backpropagation

In this section, we analyze the reasons for the poor rate of convergence of error when training nonmodular networks for classification problems. Overall, the goal of training is to reduce the error for all outputs for all exemplars. Our analysis shows that the

overall weight changes for the output layer weights are negative in the first iteration of the backpropogation training process. Magnitudes of subsequent changes are also small. Hence, the rate of convergence of error is very slow.

2.1 Definitions

In order to explain the reasons for the observed phenomenon, it is necessary to recall some of the well-known properties of feed-forward networks. In this section, we define these concepts and introduce necessary notation.

Network architecture: A schematic diagram of a feed-forward network is shown in figure 1. The nodes in the network are organized in the form of layers. There are no interconnections among nodes in the same layer. The output of each node in a layer feeds into all nodes in the next layer through weighted connections. No computation is performed by the input layer: it merely receives the input pattern and distributes the components to the first hidden layer. In this paper we analyze networks with one hidden layer.

The number of nodes in the output layer depends on the class membership representation used. We have chosen a discrete class membership representation for simplicity. In this representation we use one node in the output layer for each class – i.e., there are K nodes in the output layer. Although other representations are possible, we have found experimentally that training is more accurate when this representation is used.

Notation: A feed-forward network with one hidden layer (HL) is shown in figure 2. There are $I+1$ nodes in the input layer for input patterns of length I; the additional node represents the bias, θ, in the function $1/(1+e^{-(\mathbf{w}\cdot\mathbf{x}+\theta)})$ computed at each node. HL contains $L+1$ nodes including a node for the bias term. There are K nodes in the output layer.

The exemplars of class C_k form the set

$$T_k = \{(\mathbf{x}^{(j,k)}, \mathbf{t}^{(j,k)}) : \quad j = 1, \ldots, n_k, \quad k = 1, \ldots, K\}.$$

For a K-class problem, the training set T is $T_1 \cup \ldots \cup T_K$. Throughout this paper the ranges of j and k are: $j = 1, \ldots, n_k$ and $k = 1, \ldots, K$. The input vector for the jth exemplar of the kth class is $\mathbf{x}^{(j,k)} = (x_1^{(j,k)}, \ldots, x_{I+1}^{(j,k)})$, the target vector is $\mathbf{t}^{(j,k)} =$

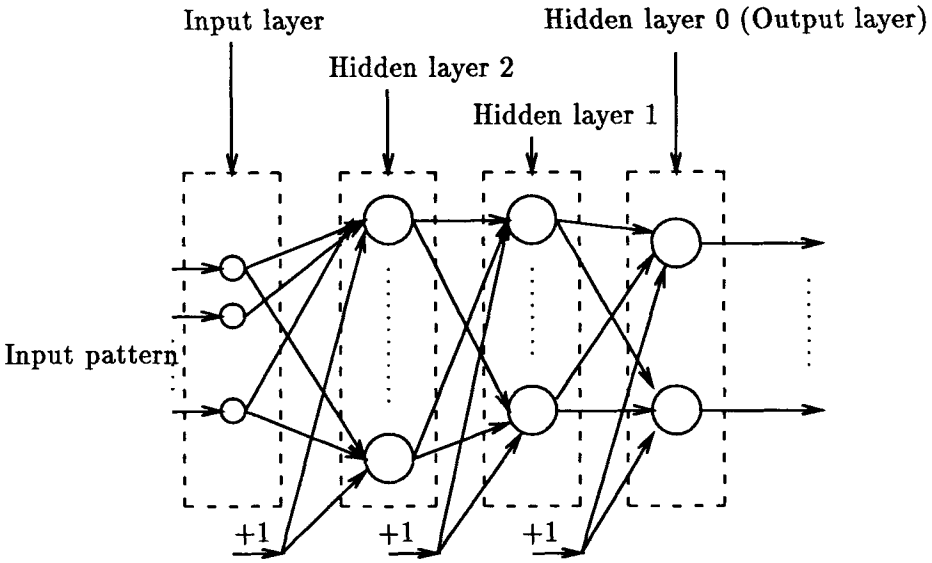

Figure 1: A multilayer feed-forward network for a K-class problem.

$(t_1^{(j,k)}, \ldots, t_K^{(j,k)})$, the output vector of the hidden layer is $\mathbf{y}^{(j,k)} = (y_1^{(j,k)}, \ldots, y_{L+1}^{(j,k)})$, and finally, the network output is the K-dimensional vector $\mathbf{z}^{(j,k)} = (z_1^{(j,k)}, \ldots, z_K^{(j,k)})$. In the above vectors $x_{I+1}^{(j,k)} \equiv 1$, $y_{L+1}^{(j,k)} \equiv 1$, and the elements of $\mathbf{t}^{(j,k)}$ satisfy

$$\begin{aligned} t_k^{(j,k)} &= 1 - \epsilon \\ t_\ell^{(j,k)} &= \epsilon, \text{ for } \ell \neq k, \end{aligned}$$

where ϵ is a small positive real number. The hidden nodes outputs $\{y_1^{(j,k)}, \ldots, y_L^{(j,k)}\}$ are computed as $y_s^{(j,k)} = \frac{e^{\mathbf{x}^{(j,k)}\mathbf{w}_s}}{1+e^{\mathbf{x}^{(j,k)}\mathbf{w}_s}}$, for $s = 1, \ldots, L$ and similarly the network outputs $\{z_1^{(j,k)}, \ldots, z_K^{(j,k)}\}$ are $z_s^{(j,k)} = \frac{e^{\mathbf{y}^{(j,k)}\nu_s}}{1+e^{\mathbf{y}^{(j,k)}\nu_s}}$, for $s = 1, \ldots, K$. Due to the nature of the sigmoid function, $\frac{e^u}{(1+e^u)}$, the values $y_s^{(j,k)}$ and $z_s^{(j,k)}$ are always positive and in the range $(0, 1)$ for all values of s.

In this network, the weight assigned to the link from the rth node of the input layer to the sth node of the HL is denoted by $w_{s,r}$. The vector of weights on the links from the input layer to the sth node in HL is denoted by

$$\mathbf{w}_s = (w_{s,1}, \ldots w_{s,I+1}),$$

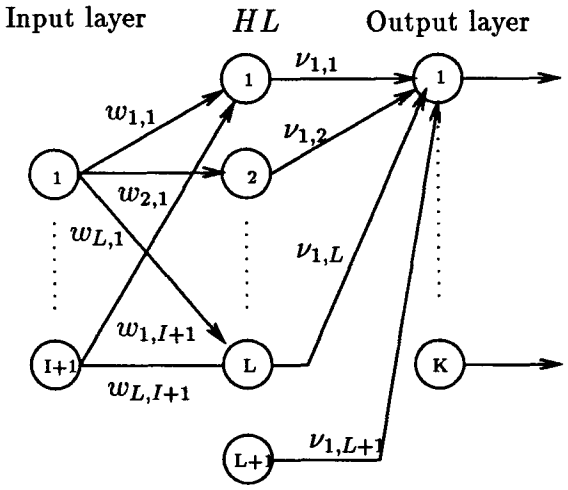

Figure 2: Notation for identifying nodes and weights in a network.

and we refer to all weights between the input layer and HL collectively as

$$\mathbf{w} = (\mathbf{w}_1, \ldots, \mathbf{w}_L).$$

The weight of the link from the rth node of the hidden layer to the sth output node is denoted by $\nu_{s,r}$, weights on links from nodes in the hidden layer to the sth node of the output layer are given by ν_s, i.e., $\nu_s = (\nu_{s,1}, \ldots, \nu_{s,L+1})$ and we refer to all the weights on links between HL and the output layer as $\boldsymbol{\nu}$. Thus

$$\boldsymbol{\nu} = (\nu_1, \ldots, \nu_K).$$

Finally, all weights of the network are denoted by $\mathbf{W}$, i.e., $\mathbf{W} = (\boldsymbol{\nu}, \mathbf{w})$.

Gradients: The net error for the training set, $E(\mathbf{W})$, can be written in terms of the sum of $K \times K$ components, each component representing the net error associated with one network output for one class of patterns. That is,

$$E(\mathbf{W}) = \sum_{k=1}^{K} \sum_{\ell=1}^{K} E_{(k,\ell)}(\mathbf{W}),$$

where

$$E_{(k,\ell)}(\mathbf{W}) = \sum_{j=1}^{n_k} \left(t_\ell^{(j,k)} - z_\ell^{(j,k)} \right)^2$$

represents error associated with ℓth output node for patterns of class k.

In each iteration of the standard backpropagation algorithm we compute $\nabla E(\mathbf{W})$, the gradient vector of the error surface. Since net error decreases most rapidly in the direction exactly *opposite* to that of the gradient vector, the weights are changed in the direction of $-\nabla E(\mathbf{W})$. Backpropagation is summarized in the following equation:

$$\mathbf{W}(m+1) = \mathbf{W}(m) - \lambda \nabla E(\mathbf{W}(m)),$$

where $\mathbf{W}(m)$ is the weight vector of the network at the beginning of the mth iteration, λ, a positive constant, is the learning rate, and $\lambda \nabla E(\mathbf{W}(m))$ is the change in weights.

Weight change computation:

In the backpropagation algorithm, all weight changes consist of a product of the error signal for a node and the output of another node. The weight change in $\nu_{s,r}$ due to the (j,k)th exemplar is given by:

$$\begin{aligned} \Delta\nu_{s,r}^{(j,k)} &= \lambda \times \text{Error signal of output node } s \times \text{Output of } r\text{th node of } HL \\ &= \lambda \left((t_s^{(j,k)} - z_s^{(j,k)}) z_s^{(j,k)} (1 - z_s^{(j,k)}) \right) \left(y_r^{(j,k)} \right) \end{aligned} \tag{1}$$

for $s = 1, \ldots, K$. Overall change in $\nu_{s,r}$ due to exemplars of the kth class is obtained by adding the right hand side of the above equation over $j = 1, \ldots, n_k$ and finally over all exemplars by adding for $k = 1, \ldots, K$. Similarly, the weight change in $w_{s,r}$ due to the (j,k)th exemplar is given by:

$$\begin{aligned} \Delta w_{s,r}^{(j,k)} &= \lambda \times \text{Error signal of } s\text{th node of } HL \times \text{Output of } r\text{th input node} \\ &= \lambda \sum_{i=1}^{K} \left((t_i^{(j,k)} - z_i^{(j,k)}) z_i^{(j,k)} (1 - z_i^{(j,k)}) \nu_{i,s} \right) y_s^{(j,k)} (1 - y_s^{(j,k)}) x_r^{(j,k)} \end{aligned} \tag{2}$$

for $s = 1, \ldots, L$; $r = 1, \ldots, I+1$. This expression is summed over $j = 1, \ldots, n_k$ to get the contribution due to exemplars of the kth class and again over $k = 1, \ldots, K$ to get the overall change in the weight.

The contribution of the (j,k)th exemplar to the gradient vector, $\nabla E_{(j,k)}(\mathbf{W})$ is:

$$\nabla E_{(j,k)}(\mathbf{W}) = (\Delta \boldsymbol{\nu}^{(j,k)}, \Delta \mathbf{w}^{(j,k)}) \tag{3}$$

where $\Delta \boldsymbol{\nu}^{(j,k)} = (\Delta \boldsymbol{\nu}_1^{(j,k)}, \ldots, \Delta \boldsymbol{\nu}_K^{(j,k)})$ and similarly $\Delta \mathbf{w}^{(j,k)} = (\Delta \mathbf{w}_1^{(j,k)}, \ldots, \Delta \mathbf{w}_L^{(j,k)})$.

Finally, the gradient vector $\nabla E_k(\mathbf{W})$ is defined as follows:

$$\nabla E_k(\mathbf{W}) = \sum_{j=1}^{n_k} \nabla E_{(j,k)}(\mathbf{W}), \quad \text{for } k = 1, \ldots, K. \tag{4}$$

2.2 Analysis of weight changes

In this section, we examine the expected values of the weight changes in the first iteration of the backpropagation algorithm. These weight changes are given by equations (1) and (2) above.

Since we need to compute expected values of complicated functions of random variables, we follow the general procedure outlined below to find approximations of these (expected) values. In the following discussion '$\approx$' is used to indicate approximate equality.

Let g be some twice differentiable function of random variable u. Suppose that we wish to obtain $\mathcal{E}(g(u))$, the expected value of $g(u)$ and it is difficult to obtain. Then, using the Taylor's series expansion of $g(u)$ with respect to u upto three terms, about $\mathcal{E}(u) = \mu$, and taking the expected value of the expansion we get,

$$\begin{aligned}\mathcal{E}(g(u)) &\approx \mathcal{E}\left(g(\mu) + g'(\mu)\cdot(u-\mu) + \frac{1}{2}(u-\mu)\cdot g''(\mu)\cdot(u-\mu)\right) \\ &= \begin{cases} g(\mu) + \frac{1}{2}g''(\mu)\,\mathcal{E}\left((u-\mu)^2\right) & \text{if } u \text{ is a scalar} \\ g(\mu) + \frac{1}{2}\sum_i\sum_j g''_{ij}(\mu)\,\mathcal{E}\left(((u_i-\mu_i)(u_j-\mu_j)\right) & \text{if } u \text{ is a vector,} \end{cases}\end{aligned}$$

where $g''_{ij}(\mu)$ denotes the second derivative of $g(u)$ with respect to the ith and jth components of u. In particular, if u's are uniform random variables between -1 and +1, all statistically independent of each other, then $\mathcal{E}((u_i-\mu_i)(u_j-\mu_j)) = 0$ for $i \neq j$ and $\frac{1}{3}$ for $i = j$. Thus, in this particular case,

$$\mathcal{E}(g(u)) = g(0) + \frac{1}{6}\sum_i g''_{i,i}(0). \tag{5}$$

Several expected values arise in our analysis. To simplify the presentation we use $\mathcal{E}(\cdot)$ to denote the expectation of a certain quantity of interest with respect to all weights $\mathbf{W}$, and $\mathcal{E}_{\boldsymbol{\nu}}(\cdot)$ denotes the conditional expectation with respect to $\boldsymbol{\nu}$, while $\mathbf{w}$ remains fixed. Recall that initially all weights are assigned random values between -1 and +1, from a uniform distribution. In particular, initially $\mathcal{E}(\nu_{(r,s)}) = 0$, and $\mathcal{E}(\nu^2_{(r,s)}) = \frac{1}{3}$. Similar results also hold for each $w_{(r,s)}$.

Proposition 1 *For any weight $\nu_{r,s}$, associated with a link from the hidden layer to the output layer, the conditional expected weight change satisfies:*

$$\mathcal{E}_{\boldsymbol{\nu}}\left(\Delta\nu^{(j,k)}_{r,s}\right) \approx \lambda y^{(j,k)}_s \frac{(2t^{(j,k)}_r - 1)}{96}\left(11 - \sum_{i=1}^{L}\left(y^{(j,k)}_i\right)^2\right)$$

in the first iteration of backpropagation.

Proof:

The proof is obtained as an application of equation (5) when $g(u)$ is replaced by $\Delta\nu_{r,s}^{(j,k)}$. Recall that the expectation is taken with respect to $\boldsymbol{\nu}$ only; consequently, $y_s^{(j,k)}$'s are constants and in particular $y_{L+1}^{(j,k)} = 1$. Details are straightforward. □

Proposition 2

$$\begin{aligned}
\mathcal{E}_{\mathbf{w}}\left(y_s^{(j,k)}\right) &\approx \frac{1}{2} \\
\mathcal{E}_{\mathbf{w}}\left(\left(y_s^{(j,k)}\right)^2\right) &\approx \frac{1}{4} + \frac{\|\mathbf{x}^{(j,k)}\|^2}{48} \\
\mathcal{E}_{\mathbf{w}}\left(\left(y_s^{(j,k)}\right)^3\right) &\approx \frac{1}{8} + \frac{\|\mathbf{x}^{(j,k)}\|^2}{32}
\end{aligned}$$

in the first iteration of backpropagation.

Proof: The proof is obtained by applying equation (5) with $g(u)$ identified as $y_s^{(j,k)}$, $\left(y_s^{(j,k)}\right)^2$, and $\left(y_s^{(j,k)}\right)^3$ respectively. □

Proposition 3 *The unconditional expectation of the weight change in $\nu_{r,s}^{(j,k)}$ satisfies:*

$$\mathcal{E}(\Delta\nu_{s,r}^{(j,k)}) \approx \lambda(2t_r^{(j,k)} - 1)\left(\frac{11}{192} - \frac{L}{768} - \frac{L+2}{9204}\|\mathbf{x}^{(j,k)}\|^2\right)$$

in the first iteration of the backpropagation algorithm.

The expression for $\mathcal{E}_{\boldsymbol{\nu}}\ (\Delta\nu_{r,s}^{(j,k)})$ obtained from proposition 1 can be expanded as follows:

$$\mathcal{E}_{\boldsymbol{\nu}}\ (\Delta\nu_{r,s}^{(j,k)}) \approx \lambda\frac{(2t_r^{(j,k)} - 1)}{96}\left(11y_s^{(j,k)} - y_s^{(j,k)}\sum_{i=1}^{L}(y_s^{(j,k)})^2[i \neq s] - (y_s^{(j,k)})^3\right)$$

where $[i \neq s] = 1$ if $i = s$ and 0 otherwise. We then apply the results obtained from proposition 2. □

Proposition 4 *For each weight $w_{s,r}^{(j,k)}$ between an input node and a hidden layer node, the expected value is*

$$\mathcal{E}_{\boldsymbol{\nu}}\ (\Delta w_{s,r}^{(j,k)}) \approx -\lambda \frac{K}{48} \left((y_r^{(j,k)})^2 (1 - y_r^{(j,k)}) x_s^{(j,k)} \right)$$

in the first iteration of backpropagation.

Proof: When we take the conditional expectation of the expression in equation (2) with respect to $\boldsymbol{\nu}$, the product $y_s^{(j,k)}(1 - y_s^{(j,k)})x_s^{(j,k)}$ is a constant. Use of equation (5), with $g(u)$ replaced by $(t_i^{(j,k)} - z_i^{(j,k)})z_i^{(j,k)}(1 - z_i^{(j,k)})\nu_{i,s}$, gives

$$\mathcal{E}(t_i^{(j,k)} - z_i^{(j,k)})z_i^{(j,k)}(1 - z_i^{(j,k)})\nu_{i,s} \approx -\frac{1}{48} y_s^{(j,k)}$$

for $i = 1, \ldots, K$ and $s = 1, \ldots, L+1$. The rest of the proof is straightforward. □

Proposition 5 *The unconditional expectation of the change in each weight, $w_{s,r}^{(j,k)}$, satisfies*

$$\mathcal{E}(\Delta w_{s,r}^{(j,k)}) \approx -\lambda \frac{K}{48} \left[\frac{1}{8} - \frac{1}{96} \parallel \mathbf{x}^{(j,k)} \parallel^2 \right] x_r^{(j,k)}$$

in the first iteration of backpropagation for $s = 1, \ldots, L$.

Proof: The proof follows readily from proposition 4 and proposition 2. □

2.3 Analysis of changes in errors

From the results obtained above we infer below that the expected weight changes are negative. Since the node output functions $[(1 + e^{-u})^{-1}]$ are monotonically increasing, we expect that output values $z_s^{(j,k)}, s = 1, \ldots, K$ will decrease as a result of the changes in the first iteration. Now, since $t_k^{(j,k)} \approx 1$, we find that $E_{(k,k)}(\mathbf{W})$ values will increase, and since $t_s^{(j,k)} \approx 0$ for $s \neq k$, $E_{(k,l)}(\mathbf{W})$ values will decrease.

It remains to be shown that the prescribed weight changes for all weights in the network in the first iteration are expected to be negative, i.e., $\mathcal{E}(\Delta \boldsymbol{W}\ ) < 0$. In the case of the $\mathbf{w}$ weights, this is implied by proposition 5 because the leading term of the expected value is $-Kx_s^{(j,k)}/384$ which is clearly negative due to the reason that

all inputs $x_s^{(j,k)}$ are between 0 and 1. Only if I is very large or if all $\| \mathbf{x}^2 \|$ are near 1, this quantity will be positive. Moreover, these weights influence z_j's only through y's and therefore their influence on the network's outputs is of secondary significance.

In the case of the $\boldsymbol{\nu}$ weights, we see from proposition 3 that the sign of the expected weight change is largely determined by the expression $(2t_r^{(j,k)} - 1)$. Now, $(2t_r^{(j,k)} - 1) > 0$ for class $\mathcal{C}_r$ and $(2t_r^{(j,k)} - 1) < 0$ for the other $K - 1$ classes. Since K is generally large, it follows that the cumulative effect of all classes on the expected weight change will be negative. [This holds for $L < 44$ since the influence of the term containing $\|\mathbf{x}^{(j,k)}\|^2$ is small. Even if $L \geq 44$, a similar argument holds if the weight change is positive.]

Next, we consider the nature of weight changes in ν's in later iterations. For any weight $\nu_{r,s}$, we find that the weight change in the first step is positive when processing an exemplar belonging to class $\mathcal{C}_r$ and negative for exemplars belonging to all other classes. Since the error $E_{(r,r)}(\mathbf{W})$ increases after the first iteration, we expect that the magnitude of the positive weight changes to $\nu_{r,s}$ increases after the first iteration. On the other hand, we expect that the magnitude of the negative weight changes for $\nu_{r,s}$ will decrease since $E_{(k,r)}(\mathbf{W})$ $(k \neq r)$ decreases after the first iteration. In various experiments we have found that the positive and negative weight changes very nearly cancel each other after the first few iterations, resulting in very small changes in ν weights in later iterations. Since the magnitude of the net weight change applied in each iteration is small, the net error converges slowly. A plot showing prescribed positive and negative weight changes for an output weight in a typical problem is shown in figure 3.

3 Modular networks

In this section, we describe a modular network architecture for k-class problems which overcomes the problems with standard backpropagation discussed in the previous section. Our approach is to split a k-class problem into k two-class problems. A modular network is a collection of modules, each of which is a single-output feed-forward network which is used to distinguish one class of patterns from patterns belonging to the remaining classes. In other words, a module for class $\mathcal{C}_k$ is trained

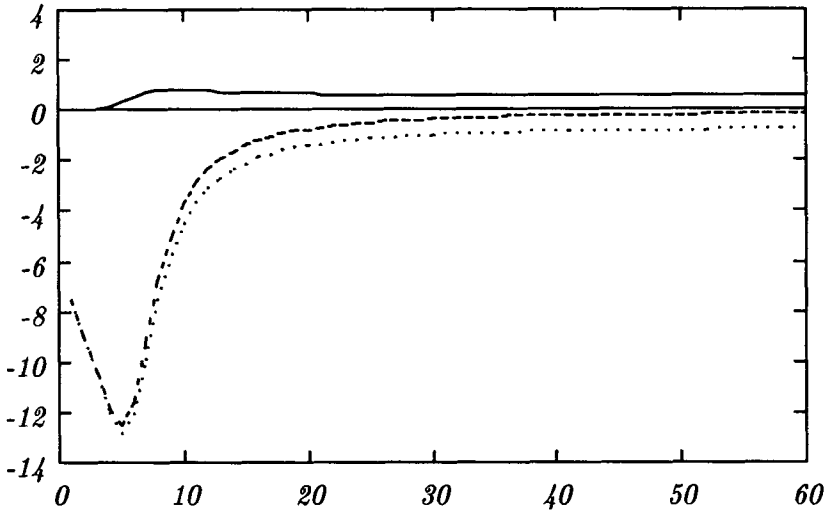

Figure 3: Positive (solid line), negative (dotted line), and net (dashed line) weight changes for an output layer weight in a typical K-class problem. The x-axis indicates the number of iterations.

to distinguish between patterns belonging to classes C_k and $\overline{C_k}$.

Modular and nonmodular networks are shown in figure 4.

The modules are trained independently in parallel. Similarly, the modules operate in parallel when classifying patterns. In the modular network, each module computes one element of the output vector. Each module has one hidden layer. The required number of nodes in the hidden layer is no larger than the number of nodes in an equivalent nonmodular network.

Since a training set for a K-class problem generally contains approximately equal numbers of exemplars for all classes, the training set for module k will contain many more exemplars for class $\overline{C_k}$ than for C_k . When training a network with backpropagation for such a two-class problem in which the numbers of exemplars for the two classes differ greatly (i.e., the training set is *imbalanced*), we have observed that the rate of convergence of net output error is especially low [1].

In an imbalanced training set, the class with more exemplars is called the *dominant* class while the other is called the *subordinate* class.

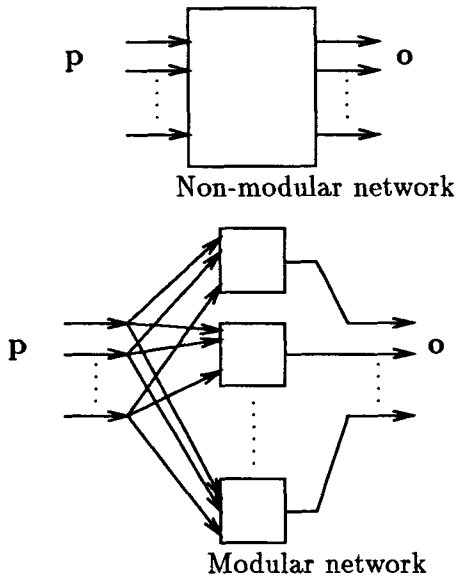

Figure 4: Modular architecture

In an earlier paper, we showed that the low rate of convergence of net error occurs because the negative gradient vector computed by backpropagation for an imbalanced training set does not initially decrease the error for the subordinate class. Consequently, in the initial iteration, the net error for the exemplars in the subordinate class *increases* significantly. The subsequent rate of convergence for the exemplars of the subordinate class is very low. To solve this problem, we suggested a modified algorithm for calculating a direction in weight-space which is downhill for both classes. Using this algorithm, we have been able to accelerate the rate of learning by one order of magnitude for two-class classification problems.

We find that the speedup in training time with our modified algorithm *improves* as the level of imbalance in the training set increases. Hence, in the context of modular networks, the advantage in training time enjoyed by the modular approach becomes increasingly significant as the number of classes increases.

4 Numerical results

We present a comparison of the training times and generalization abilities of modular and nonmodular networks for three different problems. In the first example, we analyze the well-known Fisher's Iris data set [2]. The second problem is in speech recognition while the third example is in character recognition.

4.1 Fisher's Iris data

Fisher's Iris data set contains 150 patterns belonging to three classes. There are 50 exemplars for each class and each input is a 4-dimensional real vector. The original patterns were translated and scaled such that each element of the input vector lies within the range $[0, 1]$. In the modular network and in each module 4 nodes were used in the hidden layer. The learning rate λ was set to 0.05 for each module and for the nonmodular network.

Training was stopped for the nonmodular network when two exemplars remained misclassified. In the case of the modular network, training of module 1 was stopped when all exemplars in its training set were correctly classified. In the case of modules 2 and 3, training was stopped when altogether only two exemplars remained misclassified. Reasons for stopping in the above manner are due to the well known property of the the Fisher's data that two exemplars cannot be correctly classified.

4.2 Speech recognition

The data used in this example is for a speech recognition problem and was obtained from the Univ. of California at Irvine (UCI) repository of machine learning databases and domain theories. The input patterns are 10 element real vectors representing vowel sounds which belong to one of 11 classes. While the training set contains 90 exemplars for each class, we used 45 exemplars from each class for training the networks and the remaining 45 to test for generalization ability. As in the previous examples, the patterns were translated and scaled so that each component lies within $[0, 1]$.

The nonmodular network, as well as each module in the modular network, contains 20 hidden nodes. The learning rate, λ, was set to 0.1 for both the modular and

nonmodular approach. When training the nonmodular network and when training each module of the modular network, training was stopped when the mean square error for each network output was reduced to 0.01.

4.3 Character recognition

This training set was also obtained from the UCI repository of machine learning databases and domain theories. This is a 26 class problem: the goal is to recognize digitized patterns. The input patterns are 16 element real vectors. Each element of the input vector is a numerical attribute computed from a pixel array containing the letters. The training set consists of 1000 exemplars with approximately 35 exemplars per class. The test set contains 4000 patterns. The patterns were translated and scaled such that each component of the input vector lies within $[0, 1]$.

The nonmodular network, as well as each module in the modular network, contains 15 hidden nodes. The learning rate, λ, was set to 0.01 when training the nonmodular network and 0.04 for the modular networks; these were the highest rates that permitted convergence (did not lead to oscillations).

We performed two comparative experiments with this training set. In the first experiment, training was stopped when the mean square error for each network output was reduced to 0.007 for each module and the nonmodular network. In the second experiment, the goal was to reduce the mean square error for each network output to 0.003. This was found to be possible only with the modular approach.

4.4 Results

A summary of results is shown in the table below. For each example, five separate training runs were performed, each with different random initial weights. The numbers in figure 5 are the average results from these five runs. All time measurements were in seconds, on an IBM RS6000/530 workstation.

	Max. No. Iterations Needed for any Module	Max. Time Needed to train any Module	Total No. of Iterations Needed	Total Time for Training	Fraction of Errors on Training Samples	Fraction of Errors on Test Samples
Fisher's Iris Data:						
Nonmodular	831	31.6	831	31.6	1.3%	-
Modular	260	6.25	457	11.0	1.3%	-
Speedup	3.20	5.06	1.8	2.87	-	-
Speech Recognition:						
Nonmodular	9938	8844.8	9938	8844.8	7.7%	44.7%
Modular	1765	600.2	9555	3248.6	5.5%	45.7%
Speedup	5.63	14.7	1.0	2.77	-	-
Char. Recog. target MSE 0.007:						
Nonmodular	7674	10130	7674	10130	14.0%	25.6%
Modular	554	399	5520	3974	10.4%	25.0%
Speedup	13.9	25.4	1.4	2.5	-	-
Char. Recog. target MSE 0.003:						
Nonmodular (no convergence)	-	-	-	-	-	-
Modular	1500	1080	16267	11712	3.4%	20.7%

Figure 5: Summary of performance comparison of nonmodular and modular networks.

5 Concluding remarks

Multi-class classification problems are common in real life, and are a prime candidate for neural network methods. However, the standard backpropagation algorithm is too slow to converge for such problems. Instead, a modular approach works best, where each module separately learns to 'recognize' each class. When there are many classes, the training sets are mostly imbalanced, i.e., there is a preponderance of training samples which are negative examples for each module. Hence standard backpropagation is again slow. Best results are achieved for modular networks on using a new training algorithm, in which imbalance of training samples is explicitly accounted for.

We have performed several experiments comparing our new algorithm and approach with standard backpropagation, and found that speedups of upto one order of magnitude can be obtained. The number of iterations is smaller, each iteration takes less time, there is much greater scope for parallelism, and in some cases (with low error tolerance), the new approach led to convergence whereas standard backpropagation completely failed to converge. The improvement in performance is more marked when the problem specifies a large number of classes. The method can be usefully applied to practical multi-class classification problems where backpropagation has so far been used, to improve speed and error tolerance.

References

[1] Anand, R., Mehrotra, K. G., Mohan, C. K., and Ranka, S., "An improved algorithm for neural network classification of imbalanced training sets", *Technical Report* Number SU-CIS-91-29, School of CIS, Syracuse University, Syracuse (New York), Aug. 1991.

[2] James, M. "Classification Algorithms", John Wiley and Sons, 1985.

[3] Jacobs, R. A., Jordan, M. I., and Barto, A. G., "Task decomposition through competition in a modular connectionist architecture: The what and where vision tasks", *COINS Technical Report 90-27*, University of Massachusetts, Amherst, 1990.

[4] Kohonen, T., Barna, G., and Chrisley, R. "Statistical Pattern Recognition with Neural Networks: Benchmarking Studies", *Proceedings of the International Conference on Neural Networks*, 1988, Vol-I, pp 61-68.

[5] Kowalik, J., and Osborne, M. R., "Methods for unconstrained optimization problems", American Elsevier Publishing Company Inc., 1968.

[6] Minsky, M., "The society of mind", Simon and Schuster, 1986.

[7] Ostrowski, A. M., "Solution of equations in Euclidean and Banach Spaces", Academic Press, 1973.

[8] Pierre, D. A., "Optimization theory with applications", John Wiley and Sons, Inc., 1969.

[9] Plaut, D. C., and Hinton, G. E., "Learning sets of filters using back-propagation", *Computer Speech and Language"*, Vol. 2, 1987, pp 35-61.

[10] Rumelhart, D. E., and McClelland, J. L. "Parallel Distributed Processing, Volume 1", MIT Press, 1987.

[11] Rueckl, J. G., Cave, K., R., and Kosslyn, S. M. "Why are "What" and "Where" processed by separate cortical visual systems? A computational investigation", *Journal of Cognitive Neuroscience*, Vol. 1, No. 2, 1989.

[12] Smieja, F. J., "Multiple network systems (Minos) modules: Task division and module discrimination", Proceedings of the 8th AISB conference on Artificial Intelligence, Leeds, April 1991. Also available from neuroprose repository.

[13] Sontag, E. D., and Sussmann, H. J. "Backpropagation separates when perceptrons do", *Proceedings of the International Conference on Neural Networks*, 1988, Vol-I, pp 639-642.

[14] Vogl, T. P., Mangis, J. K., Rigler, A. K., Zink, W. T., and Alkon, D. L., "Accelerating the Convergence of the Back-Propagation Method", *Biological Cybernetics*, Vol. 59, 1988, pp 257-263.

[15] Waibel, A., Sawai, H., and Shikano, K., "Modularity and Scaling in Large Phonemic Neural Networks", *IEEE Trans. on Accoustics, Speech and Signal Processing*, Vol. 37, No. 12, December 1989.