

Improving Backpropagation Learning with Feature Selection

Rudy Setiono

Huan Liu

Department of Information Systems and Computer Science

National University of Singapore

Kent Ridge, Singapore 0511

Republic of Singapore

Email:rudys,liuh@iscs.nus.sg

Abstract

There exist redundant, irrelevant and noisy data. Using proper data to train a network can speed up training, simplify the learned structure, and improve its performance. A two-phase training algorithm is proposed. In the first phase, the number of input units of the network is determined by using an information base method. Only those attributes that meet certain criteria for inclusion will be considered as the input to the network. In the second phase, the number of hidden units of the network is selected automatically based on the performance of the network on the training data. One hidden unit is added at a time only if it is necessary. The experimental results show that this new algorithm can achieve a faster learning time, a simpler network and an improved performance.

Keywords: Feedforward neural network, backpropagation, information theory, feature selection.

1 Introduction

We are concerned in this paper with the problem of distinguishing patterns from two disjoint sets in n -dimensional space. In recent years, much research have been done on algorithms that dynamically construct neural networks for solving this pattern classification problem. These algorithms include the dynamic node creation [1], the cascade correlation algorithm [2], the tiling algorithm [3], the self-organizing neural network [4], and the upstart algorithm [5]. These construction algorithms were designed to eliminate the need to determine the number of hidden units prior to training required by backpropagation learning. The aim of these algorithms is to use as few hidden units as possible to solve a given problem. A simpler network architecture with fewer hidden units reduces the computational costs, while a large network with many parameters may overfit the training data and gives poor predictive

accuracy on new data not in the training set. The issue of the optimal number of input units, which corresponds to the number of relevant attributes in the input data, however, has been largely left unaddressed.

In this paper, we propose a two-phase method for constructing a three layer feedforward neural network. In phase one, the number of units in the input layer is determined. This is done by employing an information-based method to choose the relevant attributes of the input data. These attributes are selected based on their expected information for classification of the input patterns. The advantage of using only relevant attributes in neural network training is twofold. It reduces the training time since fewer parameters are involved and it may be able to dilute the effect of noise in the data to improve the overall performance.

The final topology of the network is determined in the second phase of the algorithm where the network is constructed dynamically. The algorithm starts with one unit in the hidden layer. Additional units are added to the hidden layer one at a time to improve the accuracy of the network on the training data. Each time a new unit is added, the network is retrained by a variant of the quasi-Newton method, a fast converging algorithm for unconstrained optimization. The optimal weights obtained from the smaller network are retained as initial weights for training the new network with one additional hidden unit. When the network is trained to minimize the cross-entropy error measure and when there is no inconsistency in the data, it is always possible to construct a neural network that correctly classifies all the patterns used for training.

The organization of this paper is as follows. In Section 2, we describe how the relevant attributes of the data can be selected to solve the problem of classifying the patterns in the data set. In Section 3, we show how to construct the neural network by incrementing the number of units in the hidden layer. Experimental results are reported in Section 4. We have tested our proposed algorithm on two problems: the MONK's problems and the mushroom problem. The data for these problems are available via ftp to ics.uci.edu [6]. Finally in Section 5, a brief summary of the paper is given.

A brief word about our notation now. For a vector x in the n -dimensional real space \mathbb{R}^n , the norm $\|x\|$ denotes the Euclidean distance of x from the origin, that is, $\|x\| = (\sum_{i=1}^n x_i^2)^{\frac{1}{2}}$. For a matrix $A \in \mathbb{R}^{m \times n}$, A^T will denote the transpose of A . The superscript T is also used to denote the scalar product of two vectors in \mathbb{R}^n , that is $x^T y = \sum_{i=1}^n x_i y_i$. For a differentiable function $f(x)$, the gradient of $f(x)$ is denoted by $\nabla f(x)$.

2 Feature Selection Using Information Theory

2.1 Information Measures

Information measures are used to select attributes based on the training instances. More informative attributes for classification are chosen as *features* to be used as input in neural network training. Suppose a set \mathcal{S} of objects is divided into \mathcal{N}_c subsets, called classes. The expected information (or entropy) for classification is [7]

$$I(\mathcal{S}) = - \sum_{c=1}^{\mathcal{N}_c} \frac{n_c}{n} \cdot \log_2 \frac{n_c}{n}; \quad I(\mathcal{S}_{ik}) = - \sum_{c=1}^{\mathcal{N}_c} \frac{n_{ikc}}{n_{ik}} \cdot \log_2 \frac{n_{ikc}}{n_{ik}},$$

where n_c is the number of objects belonging to class \mathcal{C}_c , and n is the total number of objects in \mathcal{S} . Similarly suppose \mathcal{S}_{ik} is a subset of \mathcal{S} in which the i -th attribute A_i of all objects has its k -th value. $I(\mathcal{S}_{ik})$ is the expected information contributed by \mathcal{S}_{ik} for classification, where n_{ik} is the number of objects for which A_i takes its k -th value, and among these n_{ik} objects, n_{ikc} belong to class \mathcal{C}_c . Therefore, for attribute A_i the expected information is

$$\mathcal{E}_i = \sum_{k=1}^{\mathcal{N}_i} \frac{n_{ik}}{n} \cdot I(\mathcal{S}_{ik}); \quad \mathcal{G}_i = I(\mathcal{S}) - \mathcal{E}_i, \mathcal{G}'_i = \frac{\mathcal{G}_i}{I_i}, \quad I_i = - \sum_{k=1}^{\mathcal{N}_i} \frac{n_{ik}}{n} \cdot \log_2 \frac{n_{ik}}{n} \quad (1)$$

where \mathcal{N}_i is the number of values A_i can take. \mathcal{G}_i is the *first order information gain* contributed by A_i , a measure of A_i 's contribution to classification. Because the gain \mathcal{G}_i defined above is related to the number of values \mathcal{N}_i that an attribute A_i can take, i.e., A_i tends to have a large \mathcal{G}_i if \mathcal{N}_i is large, we also use normalized gains to measure the contribution of each attribute. \mathcal{G}'_i is the normalized gain.

If \mathcal{G}_i is greater than an appropriate threshold, attribute A_i can be chosen. If \mathcal{G}_i is too small, A_i may or may not be used, because the information gain may be contributed not only by A_i itself but also by some noise or a deviation of the sample set from the true distribution of \mathcal{S} . In this case, second or higher order information gains may help. The definitions of second or higher order information gains can be given in a way similar to that of first order information gains. For example, the *second order information gain* of a pair of attributes A_i and A_j can be defined as follows:

$$\mathcal{G}_{ij} = I(\mathcal{S}) - \mathcal{E}_{ij}, \quad \mathcal{E}_{ij} = \sum_{k=1}^{\mathcal{N}_i} \sum_{l=1}^{\mathcal{N}_j} \frac{n_{ijkl}}{n} \cdot I(\mathcal{S}_{ijkl}); \quad I(\mathcal{S}_{ijkl}) = - \sum_{c=1}^{\mathcal{N}_c} \frac{n_{ijklc}}{n_{ijkl}} \cdot \log_2 \frac{n_{ijklc}}{n_{ijkl}}, \quad (2)$$

where n_{ijkl} is the number of objects in \mathcal{S} whose i -th and j -th attributes take their k -th and l -th values, respectively; and n_{ijklc} is the number of objects belonging to class \mathcal{C}_c whose i -th

and j -th attributes take their k -th and l -th values, respectively. The normalized second order gain can be defined accordingly.

$$\mathcal{G}'_{ij} = \frac{\mathcal{G}_{ij}}{I_{ij}}, \quad I_{ij} = - \sum_{k=1}^{\mathcal{N}_i} \sum_{l=1}^{\mathcal{N}_j} \frac{n_{ijkl}}{n} \cdot \log_2 \frac{n_{ijkl}}{n}. \quad (3)$$

Sometimes, we need to consider both the absolute higher order gains and the higher order gains related to the corresponding lower order gains. For example, the second order relative gain of attribute pair A_i and A_j related to the first order gain \mathcal{G}_i is $\mathcal{G}_{ij,i} = \mathcal{G}_{ij} - \mathcal{G}_i$ and the corresponding normalized relative gain $\mathcal{G}'_{ij,i} = \mathcal{G}'_{ij} - \mathcal{G}'_i$. The relative gains are useful when both \mathcal{G}_i and \mathcal{G}_{ij} are greater than the corresponding thresholds, and we would like to know whether the contribution of A_j is significant to the classification. We show below how these thresholds are determined and how attributes are selected.

2.2 Feature Selection

We discuss heuristic rules here to select attributes related to the underlying classification. An attribute A_i is called *first order selectable*, if its gain $\mathcal{G}'_i > \mathcal{G}'^{(1)}$, where $\mathcal{G}'^{(1)}$ is the average of all normalized first order gains. If an attribute is first order selectable, its contribution to the classification is above average and thus should not be ignored. If an attribute is not first order selectable, it can still be selected because of the joint contribution of the attribute and other attribute(s). An attribute A_i is called *second order selectable* if one of the following conditions is satisfied:

1. There is another attribute A_j ; A_i and A_j are not first order selectable but both unnormalized and normalized second order gains are such that $\mathcal{G}_{ij} > \mathcal{G}^{(2)}$ and $\mathcal{G}'_{ij} > \mathcal{G}'^{(2)}$. Here $\mathcal{G}^{(2)}$ and $\mathcal{G}'^{(2)}$ are the average of all unnormalized and normalized second order gains excluding those gains for which both attributes are first order selectable.
2. A_j is first order selectable, but A_i is not, and $\mathcal{G}_{ij,j} > \mathcal{G}^{(2,1)}$, where $\mathcal{G}^{(2,1)}$ is the average of relative unnormalized second order gains of all pairs of the attributes that are related to those first order selectable attributes.

If an attribute is neither first nor second order selectable, we call it *second order rejectable*. A k th order rejectable can be defined similarly. However, our experiments show that normally second order gains are sufficient to determine features. Before we give exam-

ples how this method works, we discuss the algorithm for training and constructing a three layer feedforward neural network next.

3 Training and constructing feedforward neural network

3.1 A neural network construction algorithm

One of the drawbacks of the traditional backpropagation method is the need to determine the number of units in the hidden layer prior to training. To overcome this difficulty, many algorithms that construct a network dynamically have been proposed.

The algorithm which generates a single hidden layer feedforward network that we have recently proposed [8] can be outlined as follows

Feedforward neural network construction algorithm

1. Let $h = 1$ be the initial number of hidden units in the network. Set all initial weights in the network randomly.
2. Find a point that minimizes an error function.
3. If this solution results in a network that correctly classifies all the input patterns, then stop.
4. Add one unit to the hidden layer and select initial weights for the arcs connecting this new node with the input units and the output unit. Set $h = h + 1$ and go to Step 2.

The error function is normally defined as the sum of the squared errors (see Figure 1):

$$E(w, v) = \frac{1}{2} \sum_{i=1}^k (S^i - t^i)^2, \quad (4)$$

where

- k is the number of patterns.
- $t^i = 0$ or 1 target value for x^i .

- S^i is the output of the network.

$$S^i = f \left(\sum_{j=1}^h f \left((x^i)^T w^j \right) v^j \right) \quad (5)$$

- x^i is an n -dimensional input pattern, $i = 1, 2, \dots, k$.
- w^j is an n -dimensional vector of weights for the arcs connecting the input layer and the j -th hidden unit, $j = 1, 2 \dots h$.
- v^j is a real valued weight for the arc connecting the j -th hidden unit and the output unit.

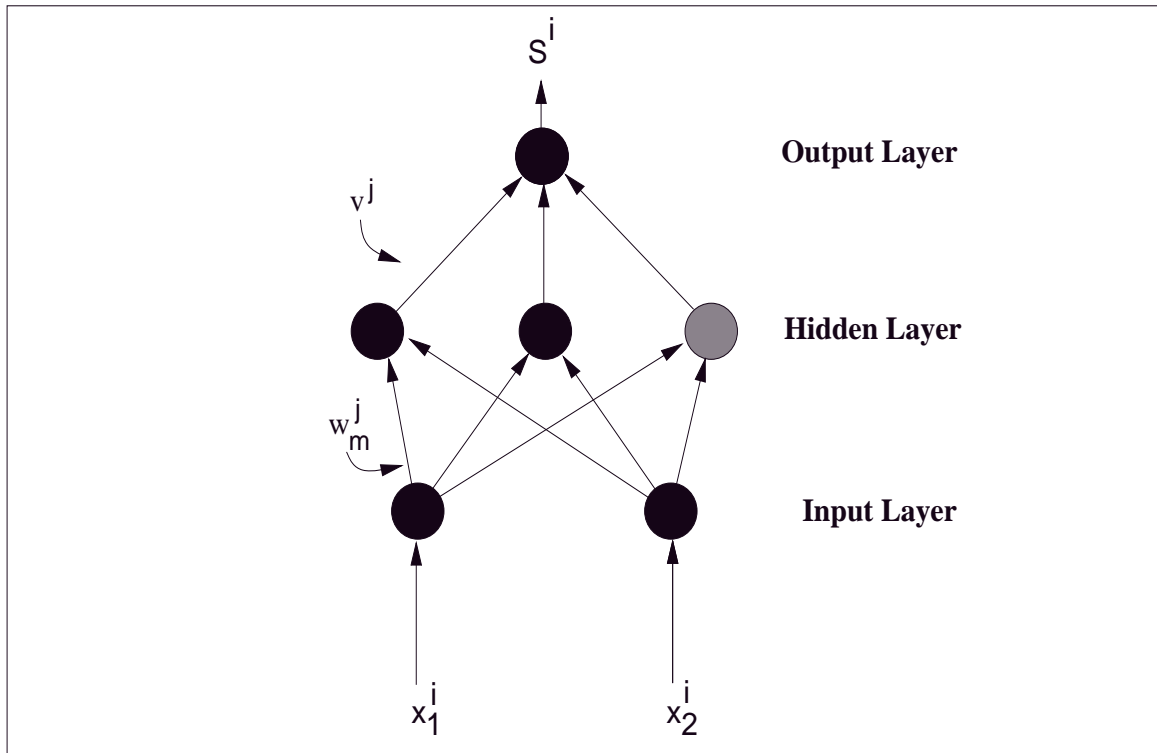


Figure 1: Feedforward neural network with 3 hidden units. When a third hidden unit is added, the optimal weights obtained from the original network with two hidden units are used as *initial* weights for retraining the new network. The initial value for w^3 is chosen randomly and v^3 is initially set to 0.

The activation function $f(y)$ is usually the sigmoid function $f(y) = 1/(1 + e^{-y})$ or the hyperbolic tangent function $f(y) = (e^y - e^{-y})/(e^y + e^{-y})$. For all the results reported in this

paper, we have used the hyperbolic tangent function as the activation function at the hidden units and the sigmoid function at the output unit.

To improve the convergence in neural network training, it has been suggested by several authors [9] [10] that the cross-entropy error function be used

$$F(w, v) = - \sum_{i=1}^k \left(t^i \log S^i + (1 - t^i) \log(1 - S^i) \right). \quad (6)$$

We use this cross-entropy function in conjunction with our construction algorithm. Given two disjoint sets of patterns, it has been shown [11] that using this cross-entropy measure of the errors, it is always possible to construct a neural network such that all the patterns are correctly classified. Note that throughout this paper, a pattern will be considered to be correctly classified if the following condition is satisfied

$$e^i = |S^i - t^i| \leq 0.45.$$

3.2 Quasi-Newton method for neural network training

The main difference between our neural network construction algorithm and the Dynamic Node Creation [1] is in the training of the growing network. We use a variant of the quasi-Newton method which is considerably faster than the gradient descent method. This method is described in [12] and can be outlined as follows.

SR1/BFGS Algorithm for minimizing $F(z)$

Step 0. Initialization.

Choose any z^1 as a starting point. Let $H^1 = I$, set $k = 1$. Let $\epsilon > 0$ be a small terminating scalar.

Step 1. Iterative Step.

If $\|\nabla F(z^k)\| \leq \epsilon \max\{1, \|z^k\|\}$ then Stop.

Otherwise

1. Compute the search direction:

$$d^k = -H^k \nabla F(z^k).$$

2. Calculate a step length $\lambda^k > 0$ and compute

$$z^{k+1} = z^k + \lambda^k d^k.$$

3. Let

$$\begin{aligned}\delta^k &= z^{k+1} - z^k \\ \gamma^k &= \nabla f(z^{k+1}) - \nabla f(z^k).\end{aligned}$$

4. If $(\delta^k - H^k \gamma^k)^T \gamma^k > 0$, update the matrix H by the SR1 update:

$$H^{k+1} = H^k + \frac{(\delta^k - H^k \gamma^k) (\delta^k - H^k \gamma^k)^T}{(\delta^k - H^k \gamma^k)^T \gamma^k}. \quad (7)$$

Otherwise by the BFGS update:

$$H^{k+1} = \left(I - \frac{\delta^k (\gamma^k)^T}{(\delta^k)^T \gamma^k} \right) H^k \left(I - \frac{\delta^k (\gamma^k)^T}{(\delta^k)^T \gamma^k} \right)^T + \frac{\delta^k (\delta^k)^T}{(\delta^k)^T \gamma^k}, \quad (8)$$

5. Set $k = k + 1$ and repeat Step 1.

The matrix H^k in the algorithm above is an approximation to the inverse of the Hessian matrix of the function $F(z)$ at z^k . The SR1 update (7) is called a Symmetric Rank-one update, because the H^k is updated by a rank one matrix, while the BFGS update (8) was independently proposed by Broyden, Fletcher, Goldfarb, and Shanno in 1970 [13, 14, 15, 16].

Given a search direction d^k , an iterative one-dimensional optimization method can be applied to find a step length λ that solves the line search problem

$$\min_{\lambda > 0} F(z^k + \lambda d^k). \quad (9)$$

However, this procedure may require an excessive number of function and/or gradient evaluations. A step length $\lambda^k > 0$ is considered acceptable if it satisfies the following two conditions.

$$F(z^k + \lambda^k d^k) - F(z^k) \leq c_1 \lambda^k (d^k)^T \nabla F(z^k), \quad (10)$$

$$(d^k)^T \nabla F(z^k + \lambda^k d^k) \geq c_2 (d^k)^T \nabla F(z^k), \quad (11)$$

where c_1 and c_2 are two constants such that $0 < c_1 \leq c_2 < 1$ and $c_1 < 0.5$. The condition (10) is to ensure that the step length λ^k produces a sufficient decrease in the value of the function $F(z)$ at the new point z^{k+1} , while the condition (11) is to ensure that the step length is not too small. In our implementation, the values of c_1 and c_2 are 0.0001 and 0.9 respectively.

4 Experimental Results

Two sets of experiments are conducted. These are the MONK’s problems and the mushroom problem. They are selected because the data are available publicly. Many learning algorithms have been applied to solve these problems and the results can be used for comparison.

4.1 The MONK’s problems

We apply our algorithm to three benchmark MONK’s problems [17]. It is an artificial robot domain, in which robots are described by six different attributes:

A_1 : head_shape \in round, square, octagon;	A_2 : body_shape \in round, square, octagon;
A_3 : is_smiling \in yes, no;	A_4 : holding \in sword, balloon, flag;
A_5 : jacket_color \in red, yellow, green, blue;	A_6 : has_tie \in yes, no.

The learning tasks of the three MONK’s problems are of binary classification, each of them being given by the following logical description of a class.

- Problem M_1 : (head_shape = body_shape) or (jacket_color = red). From 432 possible examples, 124 were randomly selected for the training set.
- Problem M_2 : Exactly two of the six attributes have their first value. From 432 examples, 169 were selected randomly.
- Problem M_3 : (Jacket_color is green and holding a sword) or (jacket_color is not blue and body_shape is no octagon). From 432 examples, 122 were selected randomly and among them there were 5% misclassifications, ie. noise in the training set.

4.1.1 Feature selection

Using Equations 1-3, first and second order information gains are calculated. In Table 1, we list the first order information gains \mathcal{G} , normalized gains \mathcal{G}' and their averages for three MONK’s problems. First order selectables are: A_1 and A_5 for M_1 ; A_4 , A_5 and A_6 for M_2 ; and A_2 and A_5 for M_3 .

With respect to second order selectables, as mentioned above, we need to consider two cases:

Att.	M_1		M_2		M_3	
	\mathcal{G}	\mathcal{G}'	\mathcal{G}	\mathcal{G}'	\mathcal{G}	\mathcal{G}'
A_1	0.0753	0.0476	0.0038	0.0024	0.0071	0.0045
A_2	0.0058	0.0037	0.0025	0.0016	0.2937	0.1854
A_3	0.0047	0.0047	0.0011	0.0011	0.0008	0.0008
A_4	0.0263	0.0166	0.0157	0.0099	0.0029	0.0018
A_5	0.2870	0.1437	0.0173	0.0087	0.2559	0.1281
A_6	0.0008	0.0008	0.0062	0.0062	0.0071	0.0071
Ave	0.0667	0.0362	0.0077	0.0050	0.0946	0.0546

Table 1: The first order information gains \mathcal{G} and normalized gains \mathcal{G}' and their averages for three MONK's problems.

1. Neither of the two attributes under consideration is first order selectable – among the three problems, no extra attribute in addition to the first order selectable is found as second order selectable. The values of $\mathcal{G}^{(2)}$ and $\mathcal{G}'^{(2)}$ are 0.1711 and 0.0545 for M_1 , 0.0411 and 0.0144 for M_2 , 0.02292 and 0.0736 for M_3 .
2. One of the two attributes under consideration is first order selectable – A_2 is second order selectable with A_1 for M_1 ; A_1 with A_6 , A_2 with either A_5 or A_6 , A_3 with A_5 or A_6 for M_2 ; and A_4 with A_2 or A_5 for M_3 . The values of $\mathcal{G}^{(2,1)}$ are 0.0637 for M_1 , 0.0290 for M_2 , and 0.0588 for M_3 .

In summary, A_1 , A_2 and A_5 are selected for M_1 ; all six attributes are selected for M_2 ; and A_2 , A_4 and A_5 are selected for M_3 .

4.1.2 Neural network learning

One hundred neural networks, each with six attributes as input were constructed for problem M_2 . Problems M_1 and M_3 were also solved each 100 times with the full set of six attributes and another 100 times with only three selected attributes. The neural network construction algorithm was always started with one unit in the hidden layer and the initial weights were chosen randomly in the interval $[-1,1]$.

For problems M_1 and M_2 , the neural network construction algorithm was terminated only when there were sufficient hidden units in the network such that 100 % classification on the training data was achieved. For problem M_3 , since there is 5% noise in the data we terminated the algorithm as soon as 95 % of training data had been classified or when there was a maximum of three hidden units in the network. The results are summarized in Tables 2 - 6 below.

Hidden Unit	Freq.	Iteration		Func. Eval		Acc. on train set (%)		Acc. on test set (%)	
		Ave.	Std. Dev.	Ave.	Std. Dev.	Ave.	Std. Dev.	Ave.	Std. Dev.
2	54	312	73	378	93	100.00	0.00	99.77	0.48
3	30	556	109	677	137	100.00	0.00	96.03	4.67
4	15	754	128	957	228	100.00	0.00	91.48	5.02
6	1	1672	-	2156	-	100.00	-	84.95	-

Table 2: Results from Problem M_1 , attributes used: $\{A_1, A_2, A_3, A_4, A_5, A_6\}$

Hidden Unit	Freq.	Iteration		Func. Eval		Acc. on train set (%)		Acc. on test set (%)	
		Ave.	Std. Dev.	Ave.	Std. Dev.	Ave.	Std. Dev.	Ave.	Std. Dev.
2	85	230	56	277	63	100.00	0.00	100.00	0.00
3	13	376	50	458	69	100.00	0.00	100.00	0.00
5	2	710	74	916	95	100.00	0.00	100.00	0.00

Table 3: Results from Problem M_1 , attributes used: $\{A_1, A_2, A_5\}$

Hidden Unit	Freq.	Iteration		Func. Eval		Acc. on train set (%)		Acc. on test set (%)	
		Ave.	Std. Dev.	Ave.	Std. Dev.	Ave.	Std. Dev.	Ave.	Std. Dev.
3	65	621	113	965	258	100.00	0.00	97.63	1.55
4	22	767	101	980	170	100.00	0.00	97.03	2.17
5	10	1162	665	1650	1071	100.00	0.00	94.14	3.22
6	3	1284	299	1554	337	100.00	0.00	94.60	3.01

Table 4: Results from Problem M_2 , attributes used: $\{A_1, A_2, A_3, A_4, A_5, A_6\}$

Hidden Unit	Freq.	Iteration		Func. Eval		Acc. on train set (%)		Acc. on test set (%)	
		Ave.	Std. Dev.	Ave.	Std. Dev.	Ave.	Std. Dev.	Ave.	Std. Dev.
1	65	119	63	150	80	96.00	0.67	95.32	2.13
2	29	556	228	767	361	98.28	1.62	92.96	1.02
3	6	697	120	902	157	96.86	2.04	92.40	3.36

Table 5: Results from Problem M_3 , attributes used: $\{A_1, A_2, A_3, A_4, A_5, A_6\}$

Hidden Unit	Freq.	Iteration		Func. Eval		Acc. on train set (%)		Acc. on test set (%)	
		Ave.	Std. Dev.	Ave.	Std. Dev.	Ave.	Std. Dev.	Ave.	Std. Dev.
1	53	92	58	118	74	95.08	0.00	100.00	0.00
2	7	276	77	371	97	95.08	0.00	100.00	0.00
3	40	697	178	894	226	93.60	0.95	97.08	1.77

Table 6: Results from Problem M_3 , attributes used: $\{A_2, A_4, A_5\}$

In the first column of these tables, we list the number of hidden units in the final network constructed by the neural network construction algorithm. When all attributes were used as input for problem M_1 , fifty four of the 100 runs stopped with a network having two hidden units, thirty with a network having 3 hidden units, fifteen with a network having 4 hidden units and one run stopped with a network having 6 hidden units. The average number of iterations and its standard deviation are given in columns 3 and 4. The number of iterations indicates the total number of times the weights are updated. In columns 5 and 6, the average number of total function evaluations and its standard deviation are shown. Note that the number of function evaluations also indicates the number of gradient evaluations, since in our implementation of the quasi-Newton method the gradient is always evaluated when the function value is computed. The number of function/gradient evaluations is larger than the number of iterations, because more than one function/gradient evaluation is sometimes needed to find a suitable step size. In the last four columns of the tables, the average and the standard deviation of the accuracy of the networks constructed on the training and test data are shown.

We can see from these tables that the algorithm was always successful in constructing a network with 100 % accuracy rate on the training data when there is no noise present in

the data, such as those for problems M_1 and M_2 . The number of hidden units required was relatively small. For problems M_1 and M_2 , the majority of the runs ended with the network having two or three hidden units. Networks with just one hidden unit were constructed for most of the runs for problem M_3 . Note that these networks had more than 95 % accuracy on the training data.

When all attributes were used as input, the accuracy of the constructed networks on the test data decreased as the number of hidden units increases. This emphasizes the importance of limiting the number of hidden units in the network.

Removing redundant attributes of problems M_1 and M_3 increased the predictive accuracy of the networks. When only selected attributes were used for problem M_1 , the predictive accuracy of the constructed network was always 100 % as depicted in Table 3. For problem M_3 , the accuracy on the training data actually decreased when three attributes were removed from the input data. However, the predictive accuracy was higher with exclusion of the redundant attributes (Figure 3). For sixty of the 100 runs which ended with one or two hidden units, the accuracy on the training data is 95.08 % and each of these sixty networks correctly classified all the test patterns. The 4.92 % error corresponds to the six noisy data in the training set which are classified wrongly. From this figure, it can be seen that networks with fewer hidden units generalize better.

The average number of function/gradient evaluations is less when only three attributes are used. For networks with two hidden units the decrease is more than 50 % (Figure 2). It is particularly interesting to note that a neural network with just *one* hidden unit is capable of discovering all the noisy patterns in the training set and achieves 100 % accuracy rate on the test data. One such network is depicted in Figure 4.

Due to the presence of noise, it is not possible to achieve 100 % accuracy on both the training data and test data for this problem. It is however still possible to construct a network that correctly classifies all the training data even when all attributes are included. In fact, 12 of the 100 runs terminated with networks having three or less hidden units that achieved 100 % accuracy on the training data. Of these 12 networks, the best predictive accuracy rate is 94.21 % (25 test data were incorrectly classified). The predictive accuracy of such networks can be expected to be lower than the figures in Table 5, because of the problem caused by overfitting of noisy data.

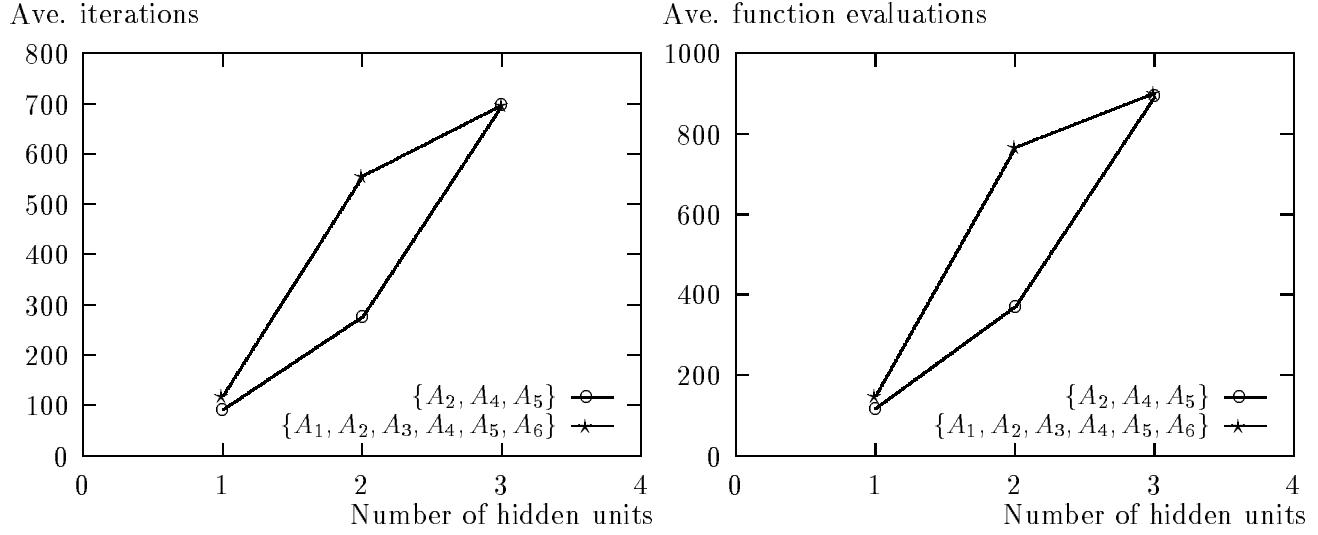


Figure 2: Comparison of the average number of iterations and function/gradient evaluations for problem M_3 with and without feature selection.

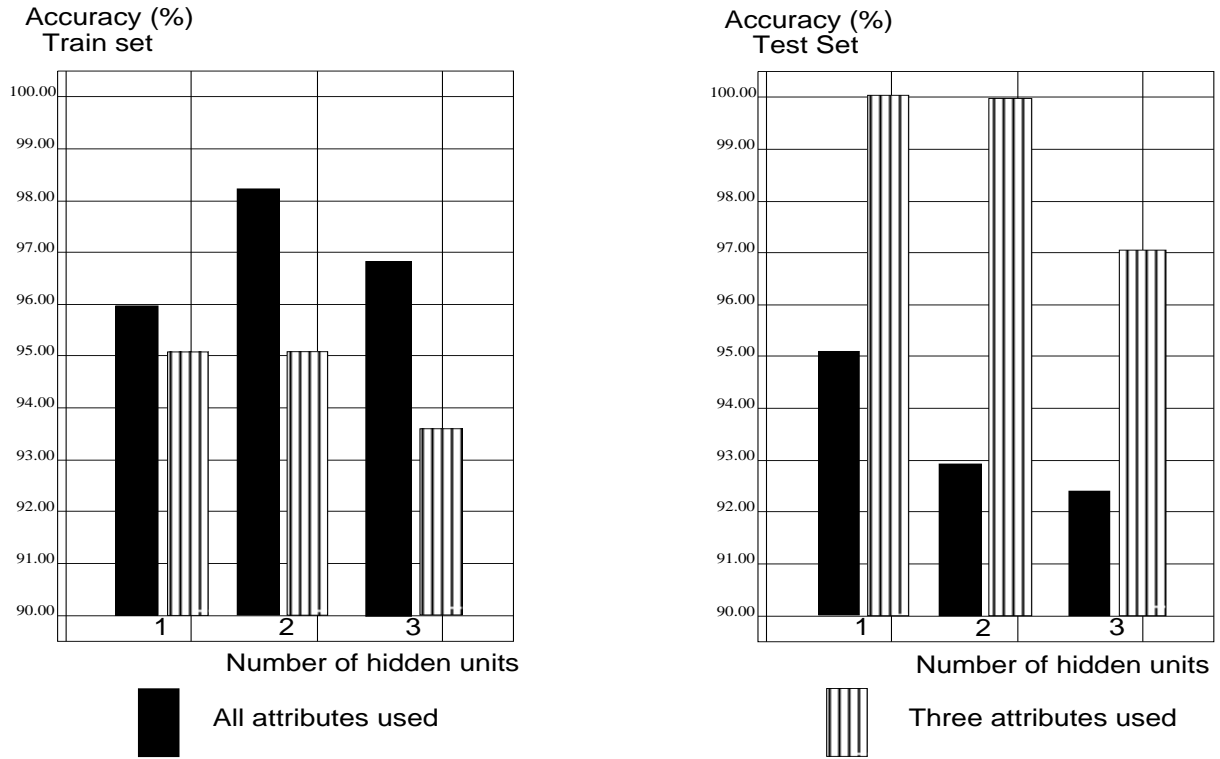


Figure 3: Comparison of the accuracy of the constructed networks for problem M_3 with and without feature selection. On the train set, the accuracy is higher with all six attributes than with only three attributes. However, it is the reverse for the test data. Also, networks with fewer hidden units have better generalization capability.

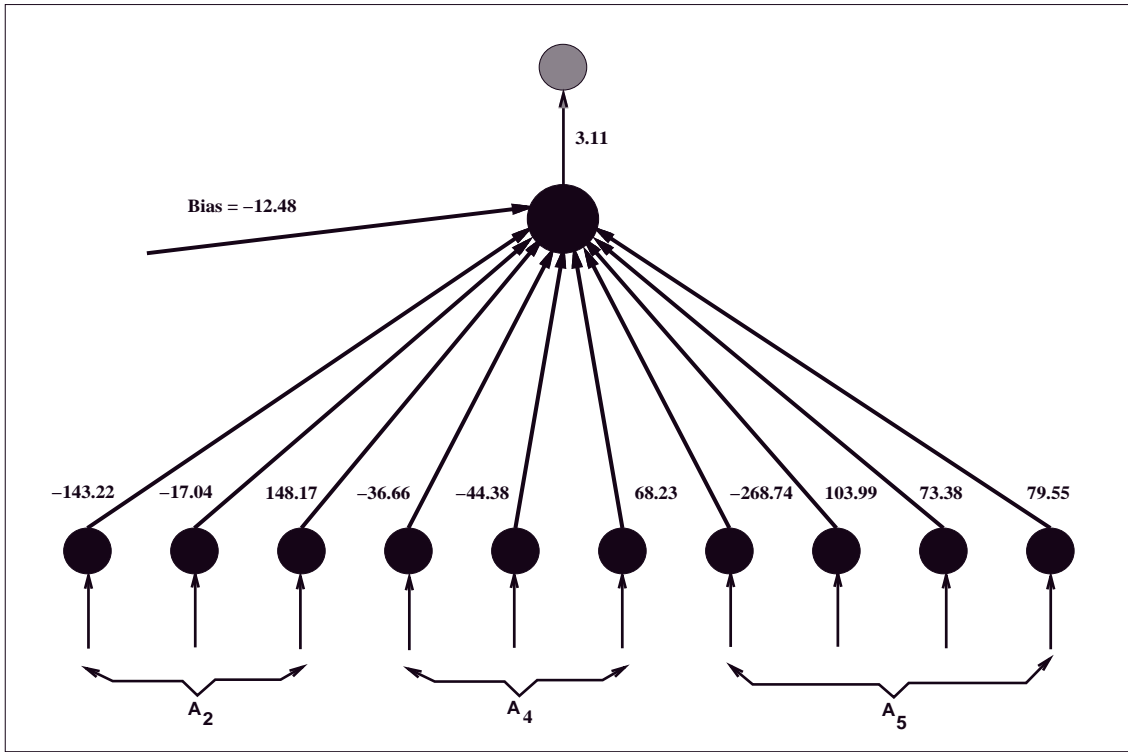


Figure 4: A network with one hidden unit that achieves 95.08 % accuracy on the training data and 100 % accuracy on the test data for problem M_3 .

4.1.3 Comparison with other works

The MONK's problems have been the subject of many studies on learning algorithms [17]. The best results reported have been obtained from backpropagation networks with weight decay (BPWD). We compare our *best* results, both without feature selection (WOFS) and with feature selection (WFS) in Table 7 below.

Method	M_1		M_2		M_3	
	Train (%)	Test (%)	Train (%)	Test (%)	Train (%)	Test (%)
BPWD	100	100	100	100	93.4	97.2
WOFS	100 (46)	100 (46)	100 (5)	100 (5)	95.08 (2)	100 (2)
WFS	100 (100)	100 (100)	100 (5)	100 (5)	95.08 (66)	100 (66)

Table 7: Comparison of BPWD, WOFS and WFS.

In the above table, the number in parentheses for WOFS and WFS indicates the number of runs (out of 100) that gave the best accuracy. For example, without feature selection, 46 of the networks constructed have 100 % accuracy on both the train set and the test set for problem M_1 . This number increases dramatically to 100 with only selected features used as input. The significance of feature selection is also highlighted by the results of problem M_3 where 66 runs ended with the best possible rate of accuracies on the training and test data.

4.2 The mushroom problem

The patterns for this problem correspond to species of mushroom in the Agaricus and Lepiota family. The data can be obtained from the University of California-Irvine repository [6]. It was reported that an accuracy rate of 95 % had been obtained [18],[19]. There are a total of 8124 patterns in the data set, one thousand of which are selected randomly to be used for training and the rest for testing. The problem is to classify each pattern as either edible or poisonous. Of the original 22 attributes describing each pattern, 19 were selected. Three attributes that were not selected are gill attachment, veil type and veil color.

For this problem, we also ran the neural network construction algorithm 100 times with all the attributes and another 100 times with only selected attributes as input. We ran the algorithm until 100 % accuracy rate on the training patterns had been obtained. The results are tabulated in Tables 8 and 9.

Hidden Unit	Freq.	Iteration		Func. Eval		Acc. on train set (%)		Acc. on test set (%)	
		Ave.	Std. Dev.	Ave.	Std. Dev.	Ave.	Std. Dev.	Ave.	Std. Dev.
1	2	325	2	421	54	100.00	0.00	98.37	0.87
2	89	623	138	784	190	100.00	0.00	98.18	0.36
3	9	908	148	1207	227	100.00	0.00	98.60	0.45

Table 8: Results from the mushroom problem with all 22 attributes used as input.

Hidden Unit	Freq.	Iteration		Func. Eval		Acc. on train set (%)		Acc. on test set (%)	
		Ave.	Std. Dev.	Ave.	Std. Dev.	Ave.	Std. Dev.	Ave.	Std. Dev.
2	90	628	134	776	174	100.00	0.00	98.15	0.33
3	10	900	70	1158	140	100.00	0.00	98.82	0.31

Table 9: Results from the mushroom problem with 19 selected attributes used as input.

From these tables, we observe that there is no significant difference in the performance of the neural network construction algorithm whether 22 or 19 attributes are used. In both cases, around 90 % of the runs terminated with a network having two hidden units with average accuracy of more than 98 % on the test data. Feature selection however, is still useful for this problem for the following two reasons:

1. When collecting new data, the values of three attributes of the mushroom -gill attachment, veil type and veil color- need not be determined.
2. The computation cost of a neural network algorithm depends not only on the number of iterations, but also on the number of weights in the network. Even though the number of iterations and the number of function/gradient evaluations decreased only slightly with 19 attributes used as input to the neural network, the difference in the computation cost is actually more significant.

5 Summary

We have presented a two-phase algorithm for pattern classification. In the first phase of the algorithm, the relevant attributes are determined based on the information they contain. Attributes with larger than average information gains will be used in the second phase of the algorithm. In the second phase, a feedforward neural network with a single hidden layer is constructed dynamically. This phase is always started with a single unit in the hidden layer. Hidden units are added to the hidden layer one at a time to increase the accuracy of the network on the training data. Having too many hidden units is undesirable. When more than the necessary number of hidden units are used, overfitting may deteriorate the generalization capability of the network. If there is no inconsistency in the data, it is always possible to construct a network with a sufficient number of hidden units such that all the training data is correctly classified. Our experiments on two sets of data suggest that the total number of hidden units required to completely classify patterns used in training is relatively small.

There are several advantages of excluding the irrelevant attributes: 1. a higher predictive accuracy on the test data is achieved. 2. it alleviates problems caused by the presence of noise in the data. 3. having fewer attributes may reduce the time to train and construct the neural network substantially. 4. attributes that do not contribute to the classification need not be determined when future data are collected.

References

- [1] T. Ash, "Dynamic node creation in backpropagation networks," *Connection Science*, vol.1, no. 4, pp. 365-375, 1989.
- [2] S.E. Fahlman and C. Lebiere, "The cascade-correlation learning architecture," in *Advances in Neural Information Processing Systems II*, edited by D. Touretzky, Morgan Kaufmann, San Mateo, CA., pp. 524-532, 1989.
- [3] M. Mezard and J.P. Nadal, "Learning in feedforward layered networks: The tiling algorithm," *Journal of Physics A*, vol. 22, no. 12, pp. 2191-2203, 1989.
- [4] M.F. Tenorio and W. Lee, "Self-organizing network for optimum supervised learning," *IEEE Transactions on Neural Networks*, vol.1, no. 1, pp. 100-110, 1990.
- [5] M. Frenn, "The upstart algorithm: a method for constructing and training feedforward neural networks," *Neural Computation*, vol. 2, no. 2, pp. 198-209, 1990.
- [6] P.M. Murphy and D.W. Aha, *UCI Repository of machine learning databases* [Machine-readable data repository]. Irvine, CA: University of California, Department of Information and Computer Science, 1992.
- [7] J.R. Quinlan, "Induction of decision trees," *Machine Learning*, vol. 1, no. 1, pp. 81-106, 1986.
- [8] R. Setiono and L.C.K. Hui, "Use of quasi-Newton method in a feedforward neural network construction algorithm," *IEEE Transactions on Neural Networks*, vol. 6, no. 1, pp. 273-277, Jan. 1995.
- [9] K.J. Lang and M.J. Witbrock, "Learning to tell two spirals apart," in *Proceedings of the 1988 Connectionist Model Summer School*, edited by D. Touretzky, G. Hinton, and T. Sejnowski, Morgan Kaufmann, San Mateo, CA., pp. 52-59, 1988.
- [10] A. van Ooyen and B. Nienhuis, "Improving the convergence of the backpropagation algorithm," *Neural Networks*, vol. 5, pp. 465-471, 1992.
- [11] R. Setiono, "A neural network construction algorithm which maximizes the likelihood function," *Connection Science*, vol. 7, no. 2, pp. 147-166, 1995.

- [12] P.K.H. Phua and R. Setiono, “Combined quasi-Newton updates for unconstrained optimization”, Dept. of Information Systems and Computer Science, National University of Singapore, Technical Report TR41/92, 1992.
- [13] C.G. Broyden, “The convergence of a class of double rank minimization, Algorithm 2, the new algorithm”, *Journal of the Institute of Mathematics and Applications*, no. 6, pp. 222-231, 1970.
- [14] R. Fletcher, “A new approach to variable metric algorithms”, *Computer Journal*, no. 13, pp. 317-322 (1970).
- [15] D. Goldfarb, “A family of variable metric algorithms derived by variational means”, *Mathematics of Computation*, no. 24, pp. 23-26, 1970.
- [16] D.F. Shanno, “Conditioning of quasi-Newton methods for function minimization”, *Mathematics of Computation*, no. 24, pp. 647-656, 1970.
- [17] S.B. Thrun et al., “The MONK’s Problems - A performance comparison of different learning algorithms”, Department of Computer Science, Carnegie Mellon University, CMU-CS-91-197, 1991.
- [18] J.S. Schlimmer, “Concept acquisition through representational adjustment,” Dept. of Information and Computer Science, University of California, Irvine, Technical Report 87-19, 1987.
- [19] W. Iba, J. Wogulis and P. Langley, “Trading off simplicity and coverage in incremental concept learning,” in *Proceedings of the 5th International Conference on Machine Learning*, Ann Arbor, Michigan, 1988, pp. 73-79.