Constructive Neural Networks as Estimators of Bayesian Discriminant Functions[1]

Dit-Yan Yeung

Department of Computer Science

Hong Kong University of Science and Technology

Clear Water Bay, Kowloon

Hong Kong

17 February 1992

Abstract

Of crucial importance to the successful use of artificial neural networks for pattern classification problems is how the appropriate network size can be automatically determined. We address this issue by formulating the process as an automatic search in the space of functions that corresponds to a subclass of multilayer feedforward networks. Learning is thus a dynamic network construction process which involves adjusting both the network weights and the topology. Adding new hidden units corresponds to extracting higher-level features from the original input features for reducing the residual classification errors. It can be shown that the resultant network approximates a Bayesian classifier that implements the Bayesian decision rule for classification. This paper also reports the empirical results of several pattern classification experiments.

Keywords: pattern classification, discriminant functions, Bayesian classifiers, feedforward networks, constructive networks, transfer of learning.

[1]The research work reported here has been partially supported by a research grant (HKUST 15/91) from the Hong Kong Research Grants Council. An earlier version of this paper was presented at the *Eighth International Workshop on Machine Learning* held June 27–29, 1991 in Evanston, Illinois, U.S.A.

1 Introduction

1.1 Pattern Classification

The problem of concern in this paper is *pattern classification* by *supervised learning.* In particular, we consider the use of *artificial neural networks* for pattern classification problems. Informally, the general classification problem can be stated as follows. Given a finite training set of patterns with known class membership, the problem is to construct an appropriate representation or classification model which correctly classifies the training patterns to any specified degree of accuracy. The generalization capability of the learned classification model is measured by the classification performance of the model in a separate testing set. Since the problem involves generalizing the classification model from the training patterns to the testing patterns, it can be considered as an *inductive inference* process.

1.2 Neural Network Approach

Many neural network models have been proposed for pattern classification problems. Among them, the class of *multilayer feedforward networks* [16] is perhaps the most popular one. The degenerate case in this class of networks is a single *perceptron* [15], which is basically a threshold logic unit. Suppose each training or testing pattern is described as a feature vector of real-valued attributes and its corresponding classification. It is well known that a perceptron partitions the feature space into two subspaces by a hyperplane. If the training patterns belong to two linearly separable classes, the perceptron convergence theorem guarantees that a perceptron can be trained to classify all the training patterns correctly using the *perceptron learning algorithm* [15]. A more general definition of perceptron does not require the transfer function to be of the threshold type. If a continuous sigmoid transfer function such as the hyperbolic tangent is used in place of the discontinuous threshold function, a learning algorithm similar to the perceptron learning algorithm can be used [18]. It is often referred to as the *least mean square* (LMS) *algorithm.*

In reality, however, many classification problems do not satisfy the linear separability assumption. Nonlinear decision surfaces are often needed to form decision regions in the feature space. The corresponding networks for representing these classification models are feedforward networks with at least one layer of hidden units between the input and output layers. These multilayer feedforward networks are generalized cases of simple networks with no hidden layers. They can also be considered as hierarchies of perceptrons and thus are sometimes referred to as *multilayer perceptrons*. A generalized form of the LMS algorithm called the *back-propagation* (BP) *learning algorithm* [16] has been proposed to train such networks if the hidden and output units all have differentiable transfer functions.[2]

Theoretical results exist for characterizing the representation or approximation capabilities of multilayer feedforward networks. For example, formal proofs were established to show that single-hidden-layer feedforward networks with sigmoid hidden units are capable of approximating any Borel measurable function from one finite-dimensional space to another to any specified degree of accuracy, provided that there are sufficiently many hidden units [9]. Similar results on the universal approximation capabilities of such networks can be found in [2]. However, these results assert no theoretical bounds on the number of hidden units required.

1.3 Research Issues

There are a number of open research issues related to this class of networks and the corresponding network learning problem. For the purposes of this paper, three of them are discussed here:

1. *Slow learning in deep networks.* The more hidden layers a network has, the slower will be the back-propagation learning process. This is due to the fact that the error signal is attenuated significantly each time it is propagated backward through a layer.

2. *Network size determination.* With a given classification problem at hand, the number of hidden units needed is usually unknown in advance.

3. *Learnability.* The theoretical results on the universal approximation capabilities of multilayer feedforward networks only provide existence proofs and thus leave open the question of whether the desired mappings can be learned efficiently using a given learning algorithm. In other words, the universality of approximation capabilities of multilayer feedforward networks by no means implies their learnability.

[2]Although linear transfer functions are obviously differentiable with constant first derivatives, the hidden units of multilayer feedforward networks always use nonlinear transfer functions because any such network with linear hidden units can always be reduced to one with no hidden layers for representing the same mapping.

This paper describes a method which, to a certain extent, addresses all these three issues.

The remainder of the paper is organized as follows. Section 2 gives a functional formulation of the particular subclass of multilayer feedforward networks considered in this paper, and shows that a network of the subclass approximates asymptotically a Bayesian classifier in the minimum mean squared-error sense. Based on the functional formulation, Section 3 describes how the networks can be constructed dynamically during learning and relates the construction process with a function space search strategy. Section 4 discusses three pattern classification domains and their experimental results. Concluding remarks and discussions on possible further extensions can be found in Section 5.

2 Single-Hidden-Layer Feedforward Networks

2.1 Functional Formulation

We consider in this paper a class of single-hidden-layer feedforward networks in which a typical network has direct connection between the input and output layers, as exemplified in Figure 1. Table 1 gives the definitions of symbols used in providing a functional formulation of the class of networks.

***** FIGURE 1 *****

***** TABLE 1 *****

Each input feature x_i is assigned an input unit which is connected to the kth output unit with connection weight w_{ik}. An extra unit (called the bias unit) with fixed value $x_0 = 1$ and the weights w_{0k}, $1 \leq k \leq m$, represent the biases of the m output units. Mathematically, each network with h hidden units corresponds to a mapping $\mathcal{N}_{nhm} : \Re^n \rightarrow \Re^m$ whose (vector) function value is used to specify the class to which a training or testing pattern belongs. For the degenerate extreme with $h = 0$, the mapping (denoted as $\mathcal{N}_{nm}$) can be defined as:

$$\mathcal{N}_{nm}(\mathbf{x}) = (f(\mathbf{x}^{n+1} \cdot \mathbf{w}_1^{n+1}), f(\mathbf{x}^{n+1} \cdot \mathbf{w}_2^{n+1}), ..., f(\mathbf{x}^{n+1} \cdot \mathbf{w}_m^{n+1}))^T.$$

Equivalently, we can replace f by a function $\tilde{f}_0 : \Re^{n+1} \rightarrow \Re$ which takes $\mathbf{w}_k^{n+1}$'s as the parameter vectors:

$$\mathcal{N}_{nm}(\mathbf{x}) = (\tilde{f}_0(\mathbf{x}^{n+1}; \mathbf{w}_1^{n+1}), \tilde{f}_0(\mathbf{x}^{n+1}; \mathbf{w}_2^{n+1}), ..., \tilde{f}_0(\mathbf{x}^{n+1}; \mathbf{w}_m^{n+1}))^T. \tag{1}$$

For the general case with $h > 0$, the output value of the jth hidden unit is $g(\mathbf{x}^{n+1} \cdot \mathbf{v}_j)$ or $\tilde{g}(\mathbf{x}^{n+1}; \mathbf{v}_j)$. We define the augmented feature vector $\mathbf{x}^{n+h+1}$ as:

$$\mathbf{x}^{n+h+1} = (1, x_1, ..., x_n, \tilde{g}(\mathbf{x}^{n+1}; \mathbf{v}_1), ..., \tilde{g}(\mathbf{x}^{n+1}; \mathbf{v}_h))^T.$$

The mapping $\mathcal{N}_{nhm}$ corresponding to the network can thus be defined as:

$$\mathcal{N}_{nhm}(\mathbf{x}) = (f(\mathbf{x}^{n+h+1} \cdot \mathbf{w}_1^{n+h+1}), f(\mathbf{x}^{n+h+1} \cdot \mathbf{w}_2^{n+h+1}), ..., f(\mathbf{x}^{n+h+1} \cdot \mathbf{w}_m^{n+h+1}))^T$$

or

$$\mathcal{N}_{nhm}(\mathbf{x}) = (\tilde{f}_h(\mathbf{x}^{n+h+1}; \mathbf{w}_1^{n+h+1}), \tilde{f}_h(\mathbf{x}^{n+h+1}; \mathbf{w}_2^{n+h+1}), ..., \tilde{f}_h(\mathbf{x}^{n+h+1}; \mathbf{w}_m^{n+h+1}))^T. \tag{2}$$

Comparing (1) with (2), it can be seen that the general case ($h > 0$) is very similar in form to the degenerate case ($h = 0$), except that the original input feature space is augmented to a higher-dimensional space as seen by the output units.

From Equation (1), learning $\mathcal{N}_{nm}$ reduces to learning the parameter vectors $\mathbf{w}_k^{n+1}$'s. Obviously this can be done by using a simple network learning algorithm such as the LMS algorithm. Equation (2) shows that $\mathcal{N}_{nhm}$ can also be learned in a similar way by learning the parameter vectors $\mathbf{w}_k^{n+h+1}$'s, provided that the parameter vectors $\mathbf{v}_j$'s are known and fixed when the unknown parameter vectors $\mathbf{w}_k^{n+h+1}$'s are learned. To achieve this, we can keep the parameter vectors $\mathbf{v}_j$'s non-adjustable once they are determined and introduced into the network. Thus there is only one layer of adjustable weights ($\mathbf{w}_k^{n+h+1}$'s) and so no back-propagation of errors is needed. The important issue of how to determine the parameter vectors $\mathbf{v}_j$'s during the dynamic network construction process will be the concern of the next section.

Allowing as few as only one layer of adjustable weights at each learning stage is a simple yet rather effective technique for speeding up network learning. Let us mention here some previous attempts which are also based on this very idea and at the same time bear resemblance in their formulation with the model discussed above. [14] discusses the use of Φ functions for classifying patterns. A Φ function is a linear combination of a set of component functions each of which is a linearly independent, real-valued function of the input features. However, the exact forms of the component functions are not fixed by the model. Our model corresponds roughly to a specific family of Φ functions with the input features themselves and the h functions $\tilde{g}$'s constituting the component functions. Another approach is referred to as functional-link networks [10]. This approach is very similar to the use of Φ functions. The functional links map the feature space to a usually higher-dimensional space with the net effect of enhancing the input representation without introducing any intrinsically new *ad hoc* information. The functional links can be high-order transforms of the input features but their exact forms should be custom-made for each individual problem. [13] describes a successful application of artificial neural networks to the control of robot manipulators. To learn the mapping for inverse dynamics, a simple learning algorithm is used to learn the weights for combining a set of subsystems linearly. The subsystems form a mapping which transforms the input state variables for better representation. The subsystems or their underlying functions can be determined from the manipulator dynamics equation whose form is assumed to be known *a priori*. There are certainly other examples which also take advantage of using only a single layer of adjustable weights at a time.

2.2 Approximating Bayesian Discriminant Functions

We now show that learning in the class of single-hidden-layer networks described above results in network classifiers that approximate the Bayesian discriminant functions. Figure 2(a) gives the functional decomposition of a network classifier. The mapping $\mathcal{N}_{nhm}$ maps $\mathbf{x}$ to the network output vector $\mathbf{y} = (y_1, y_2, ..., y_m)^T$. The parameter matrices $\mathbf{V}$ and $\mathbf{W}$ of $\mathcal{N}_{nhm}$ are defined in terms of the weight vectors $\mathbf{v}_j$'s and $\mathbf{w}_k^{n+h+1}$'s, respectively. $\mathbf{V}$ is an $(n+1) \times h$ matrix whose column vectors are $\mathbf{v}_j$'s, whereas $\mathbf{W}$ is an $(n+h+1) \times m$ matrix whose column vectors are $\mathbf{w}_k^{n+h+1}$'s. With the network output vector, the decision vector $\mathbf{d} = (d_1, d_2, ..., d_m)^T$ is determined by applying the maximum selector mapping $\mathcal{M}$, such that:

$$d_k = \begin{cases} 1 & y_k > y_{k'}, \forall k' \neq k \\ 0 & \text{otherwise} \end{cases}$$

The decision vector can be considered to be the estimated value of the desired classification vector $\mathbf{c} = (c_1, c_2, ..., c_m)^T = \mathcal{F}(\mathbf{x})$, where $\mathcal{F}$ is the desired mapping the network is supposed to learn. Figure 2(b) shows the functional decomposition of the corresponding Bayesian classifier. The m

Bayesian discriminant functions β_k's are defined as:

$$\beta_k(\mathbf{x}) = P(\omega_k \,|\, \mathbf{x})$$

i.e., the *a posteriori* probability of ω_k given $\mathbf{x}$. The corresponding decision vector $\mathbf{d}^*$ gives the estimated value of the desired classification vector.

***** FIGURE 2 *****

Table 2 gives the definitions of symbols used in establising the asymptotic equivalence result. We define the sample total error function $\varepsilon_{\mathcal{S}}$ for $\mathcal{S}$ in the form of a quadratic error function commonly used in learning algorithms such as the BP learning algorithm:

$$\varepsilon_{\mathcal{S}}(\mathbf{V},\mathbf{W}) = \sum_{s\in\mathcal{S}} \sum_{k'=1}^{m} (y_{k'}(\mathbf{x}^s;\mathbf{V},\mathbf{w}_{k'}^{n+h+1}) - c_{k'}(\mathbf{x}^s))^2.$$

This error function can be rewritten as:

$$\begin{aligned}
\varepsilon_{\mathcal{S}}(\mathbf{V},\mathbf{W}) &= \sum_{k=1}^{m} \sum_{s\in\mathcal{S}_k} \sum_{k'=1}^{m} (y_{k'}(\mathbf{x}^s;\mathbf{V},\mathbf{w}_{k'}^{n+h+1}) - c_{k'}(\mathbf{x}^s))^2 \\
&= \sum_{k=1}^{m} \sum_{s\in\mathcal{S}_k} [(y_{k}(\mathbf{x}^s;\mathbf{V},\mathbf{w}_{k}^{n+h+1}) - c_{k}(\mathbf{x}^s))^2 + \sum_{k'=1,k'\neq k}^{m} (y_{k'}(\mathbf{x}^s;\mathbf{V},\mathbf{w}_{k'}^{n+h+1}) - c_{k'}(\mathbf{x}^s))^2].
\end{aligned}$$

Since

$$c_k(\mathbf{x}^s) = \begin{cases} 1 & s \in \mathcal{S}_k \\ 0 & \text{otherwise,} \end{cases}$$

we have:

$$\varepsilon_{\mathcal{S}}(\mathbf{V},\mathbf{W}) = \sum_{k=1}^{m} \sum_{s\in\mathcal{S}_k} [(y_{k}(\mathbf{x}^s;\mathbf{V},\mathbf{w}_{k}^{n+h+1}) - 1)^2 + \sum_{k'=1,k'\neq k}^{m} (y_{k'}(\mathbf{x}^s;\mathbf{V},\mathbf{w}_{k'}^{n+h+1}))^2]. \tag{3}$$

***** TABLE 2 *****

As the size of $\mathcal{S}$ increases, the population average error function $\overline{\varepsilon}$ can be defined as:

$$\overline{\varepsilon}(\mathbf{V},\mathbf{W}) = \lim_{|\mathcal{S}|\to\infty} \frac{1}{|\mathcal{S}|} \varepsilon_{\mathcal{S}}(\mathbf{V},\mathbf{W}). \tag{4}$$

Substituting (3) into (4), we have:

$\overline{\varepsilon}(\mathbf{V},\mathbf{W})$

$$
\begin{aligned}
&= \lim_{|\mathcal{S}|\to\infty} \sum_{k=1}^{m} \sum_{s\in\mathcal{S}_k} \frac{|\mathcal{S}_k|}{|\mathcal{S}|} \frac{1}{|\mathcal{S}_k|} [(y_k(\mathbf{x}^s; \mathbf{V}, \mathbf{w}_k^{n+h+1}) - 1)^2 + \sum_{k'=1,k'\neq k}^{m} (y_{k'}(\mathbf{x}^s; \mathbf{V}, \mathbf{w}_{k'}^{n+h+1}))^2] \\
&= \sum_{k=1}^{m} P(\omega_k) \int_{\Omega_{\mathbf{X}}} [(y_k(\mathbf{x}; \mathbf{V}, \mathbf{w}_k^{n+h+1}) - 1)^2 + \sum_{k'=1,k'\neq k}^{m} (y_{k'}(\mathbf{x}; \mathbf{V}, \mathbf{w}_{k'}^{n+h+1}))^2] \, \rho(\mathbf{x} \,|\, \omega_k) \, d\mathbf{x} \\
&= \sum_{k=1}^{m} \int_{\Omega_{\mathbf{X}}} [(y_k(\mathbf{x}; \mathbf{V}, \mathbf{w}_k^{n+h+1}) - 1)^2 + \sum_{k'=1,k'\neq k}^{m} (y_{k'}(\mathbf{x}; \mathbf{V}, \mathbf{w}_{k'}^{n+h+1}))^2] \, P(\omega_k \,|\, \mathbf{x}) \, \rho(\mathbf{x}) \, d\mathbf{x} \\
&= \sum_{k=1}^{m} \int_{\Omega_{\mathbf{X}}} [(y_k(\mathbf{x}; \mathbf{V}, \mathbf{w}_k^{n+h+1}) - 1)^2 P(\omega_k \,|\, \mathbf{x}) + \sum_{k'=1,k'\neq k}^{m} (y_k(\mathbf{x}; \mathbf{V}, \mathbf{w}_k^{n+h+1}))^2 P(\omega_{k'} \,|\, \mathbf{x})] \, \rho(\mathbf{x}) \, d\mathbf{x} \\
&= \sum_{k=1}^{m} \int_{\Omega_{\mathbf{X}}} [(y_k(\mathbf{x}; \mathbf{V}, \mathbf{w}_k^{n+h+1}))^2 \sum_{k'=1}^{m} P(\omega_{k'} \,|\, \mathbf{x}) - 2\, y_k(\mathbf{x}; \mathbf{V}, \mathbf{w}_k^{n+h+1}) \, \beta_k(\mathbf{x}) + \beta_k(\mathbf{x})] \, \rho(\mathbf{x}) \, d\mathbf{x} \\
&= \sum_{k=1}^{m} \int_{\Omega_{\mathbf{X}}} [(y_k(\mathbf{x}; \mathbf{V}, \mathbf{w}_k^{n+h+1}) - \beta_k(\mathbf{x}))^2 + \beta_k(\mathbf{x})(1 - \beta_k(\mathbf{x}))] \, \rho(\mathbf{x}) \, d\mathbf{x}.
\end{aligned}
$$

We define

$$\overline{\delta}(\mathbf{V}, \mathbf{W}) = \sum_{k=1}^{m} \int_{\Omega_{\mathbf{X}}} (y_k(\mathbf{x}; \mathbf{V}, \mathbf{w}_k^{n+h+1}) - \beta_k(\mathbf{x}))^2 \rho(\mathbf{x}) \, d\mathbf{x}.$$

Thus

$$\overline{\varepsilon}(\mathbf{V}, \mathbf{W}) = \overline{\delta}(\mathbf{V}, \mathbf{W}) + \sum_{k=1}^{m} \int_{\Omega_{\mathbf{X}}} \beta_k(\mathbf{x})(1 - \beta_k(\mathbf{x})) \rho(\mathbf{x}) \, d\mathbf{x}. \tag{5}$$

Suppose our network learning algorithm minimizes the sample total error function $\varepsilon_{\mathcal{S}}$ with respect to $\mathbf{W}$. The population average error function $\overline{\varepsilon}$ will also be minimized if the training set $\mathcal{S}$ is large enough. Since the second term of the right hand side of Equation (5) is independent of $\mathbf{W}$, minimizing $\overline{\varepsilon}$ implies minimizing $\overline{\delta}$ as well. In other words, the (error-minimization) learning algorithm works in such a way that the network output values approximate the Bayesian discriminant functions in the minimum mean squared-error sense. It should be noted, however, that the network learning algorithm only minimizes $\varepsilon_{\mathcal{S}}$ in the local sense. Furthermore, a network should possess sufficient hidden units in order to give satisfactory approximation. It is for this very reason that the whole issue of determining network size is central to the research described here.

3 Constructive Networks

3.1 Network Construction as Function Space Search

The class of single-hidden-layer networks described above can be formulated as an infinite family of functions $\mathcal{N} = \{\mathcal{N}_{nhm} \,|\, n \geq 1, h \geq 0, m \geq 2\}$. For the subfamily with fixed values of n and m, each function in it has an index equal to the number of hidden units in the corresponding network.[3] It is obvious that $\mathcal{N}_{nim}$ is a specialization of $\mathcal{N}_{njm}$ for $i < j$. The process of determining the appropriate network size for approximating the desired mapping corresponds to searching through the corresponding subfamily for the appropriate function. Among all possible search strategies, sequential search is probably the simplest one. One possibility is to guess an initial index to start with and then decrement, if necessary, the index by one at a time until the appropriate function is reached. In general, however, there is no *a priori* knowledge about what the appropriate index is and so providing an initial guess can be difficult. If the initial guess is too large compared with the appropriate index, much learning time will be spent on networks larger than necessary and hence learning is rather inefficient. On the other hand, if the initial guess is too small, sequential search will surely miss the appropriate function.

Incremental sequential search is preferable for several reasons:

1. We can always start with $\mathcal{N}_{nm}$ without having to provide an initial guess, unless *a priori* knowledge is available to suggest a larger index.
2. Most of the learning efforts will be spent on small networks in which fast learning is possible.
3. Previous learning efforts spent on smaller networks can facilitate the subsequent learning of larger networks.

With transfer of prior learning [7, 3], we can avoid training large networks directly from scratch. Hopefully the desired mapping will become more reachable. Incremental sequential search can be considered as fine-tuning an existing model to better fit the reality, which is an important characteristic of human learning. In what follows, we will describe one way of implementing the incremental sequential search strategy.

We first describe the network construction process here in high-level terms, so that the general structure of the algorithm can be understood more easily. Some parts that are less obvious and hence require more detailed discussions will be left to the next subsection.

Construct_Network($\mathcal{S}$; H, ψ, $E_{\mathbf{W}}$, $\mathcal{E}_{\mathbf{W}}$, $\mathcal{T}_{\mathbf{W}}$, $E_{\mathbf{V}}$, $\mathcal{E}_{\mathbf{V}}$, $\mathcal{T}_{\mathbf{V}}$)

[3] Formally speaking, each element $\mathcal{N}_{nhm}$ in the subfamily represents a parameterized family of functions in itself. For simplicity, we still refer to it as a function with the assumption that no confusion will be caused.

```
Input:
    $\mathcal{S}$        training set
User-specified parameters:
    $H$        maximum number of hidden units allowed
    $\psi$        threshold percentage classification error as success criterion
    $E_{\mathbf{W}}$      maximum number of learning epochs for $\mathbf{W}$ allowed for each distinct value $h$
    $\mathcal{E}_{\mathbf{W}}$      threshold percentage change in residual error when training $\mathbf{W}$ for each
              distinct value $h$
    $\mathcal{T}_{\mathbf{W}}$      maximum number of consecutive learning epochs for $\mathbf{W}$ with no significant
              change in residual error that can be tolerated for each distinct value $h$
    $E_{\mathbf{V}}$, $\mathcal{E}_{\mathbf{V}}$, and $\mathcal{T}_{\mathbf{V}}$ will be defined in Add_Hidden_Unit later
Output:
    $\mathcal{N}_{nhm}$    network with $h \leq H$ hidden units for approximating the desired mapping
{
  Initialize current network $\mathcal{N}_{nhm}$ as $\mathcal{N}_{nm}$;   (i.e., $h = 0$)
  For $i = 1$ to $E_{\mathbf{W}}$ {
      Train weight matrix $\mathbf{W}$ of $\mathcal{N}_{nm}$ for 1 epoch using $\mathcal{S}$;
      If percentage of incorrectly learned training patterns $< \psi$,
          Exit with success;
      If percentage change in residual error $< \mathcal{E}_{\mathbf{W}}$ in each of the last $\mathcal{T}_{\mathbf{W}}$ epochs,
          Exit "For" loop;
  }
  While $h < H$ {
      Add_Hidden_Unit($\mathcal{N}_{nhm}$; $E_{\mathbf{V}}$, $\mathcal{E}_{\mathbf{V}}$, $\mathcal{T}_{\mathbf{V}}$);   (i.e., increment $h$ by 1)
      For $i = 1$ to $E_{\mathbf{W}}$ {
          Train weight matrix $\mathbf{W}$ of $\mathcal{N}_{nhm}$ for 1 epoch using $\mathcal{S}$;
          If percentage of incorrectly learned training patterns $< \psi$,
              Exit with success;
          If percentage change in residual error $< \mathcal{E}_{\mathbf{W}}$ in each of the last $\mathcal{T}_{\mathbf{W}}$ epochs,
              Exit "For" loop;
      }
```

```
        }
        Exit with failure;
    }
```

Let us use an example to illustrate the basic ideas. Suppose Figure 3(a) is the network $\mathcal{N}_{223}$ after its weight matrix $\mathbf{W}$ is trained to completion without arriving at an acceptably small residual error. In order to further reduce the error, the network grows in size by introducing a new hidden unit. As shown in Figure 3(b), the newly added unit is initially unconnected with the output units. The weight vector $\mathbf{v}_3$ is modified with the aim of reducing the residual error when the weight matrix $\mathbf{W}$ is trained again later. (Details of the learning algorithm involved will be presented in the next subsection.) Since the new hidden unit is not connected to the output units when the weight vector $\mathbf{v}_3$ is trained, the residual error will not change at this stage. The objective of training $\mathbf{v}_3$ is solely to maximize, hopefully, the *future* contribution of the hidden unit. After the training of $\mathbf{v}_3$ has been completed, connection weights are established between the newly added hidden unit and the output units, as illustrated in Figure 3(c). This is followed by training the new (augmented) weight matrix $\mathbf{W}$.

***** FIGURE 3 *****

With a specified level of required accuracy, we can test a trained network with $\mathcal{S}$ to see if the residual error is acceptably small and hence the termination criterion has been satisfied. If necessary, we can also define additional termination criteria such as the performance of using a trained network to classify the testing patterns.

3.2 Learning Algorithms

We now describe the learning algorithms for the weight matrices $\mathbf{W}$ and $\mathbf{V}$. Learning of $\mathbf{W}$ for either $\mathcal{N}_{nm}$ or $\mathcal{N}_{nhm}$ involves essentially only one layer of adjustable weights, and thus any gradient-descent learning algorithm like the LMS algorithm can be used. For our purposes, however, the *quickprop algorithm* [4] is used instead to speed up the learning process by virtue of the second-order information of the weight space. Appendix A provides more details about the algorithm.

The procedure of training the weight vector $\mathbf{v}_h$ of the newly added (hth) hidden unit requires more detailed discussions. Similar to the training of $\mathbf{W}$, it can be considered as an optimization problem. The weight values are adjusted to optimize an objective function which is related to the approximation accuracy of $\mathcal{N}_{nhm}$. The method to be described below is similar to that proposed by [5].

To give a better mapping $\mathcal{N}_{nhm}$, we optimize the degree of association between the output z_h of the hth hidden unit and the residual error of $\mathcal{N}_{n,h-1,m}$. We define the residual error e_k of the kth output unit as:

$$e_k = (y_k - \tilde{c}_k)\, f'(t_k)$$

where t_k is the total input of the kth output unit, f' is the first derivative of f, and $\tilde{c}_k = 2\,c_k - 1$.[4] The objective function J_h for the hth hidden unit is defined as the sum of the absolute values of the covariances $\mathrm{cov}(z_h, e_k)$ between z_h and e_k over the set $\mathcal{O}$ of output units:

$$J_h = \sum_{k \in \mathcal{O}} |\mathrm{cov}(z_h, e_k)| = \sum_{k \in \mathcal{O}} \mathrm{sgn}(\mathrm{cov}(z_h, e_k))\, \mathrm{cov}(z_h, e_k)$$

where $\mathrm{sgn} : \Re \to \{-1, 1\}$ is the sign function.

The following algorithm summarizes the procedure of adding a new hidden unit and training its input weight vector:

```
Add_Hidden_Unit($\mathcal{N}_{nhm}$; $E_{\mathbf{V}}$, $\mathcal{E}_{\mathbf{V}}$, $\mathcal{T}_{\mathbf{V}}$)
    Input:
        $\mathcal{N}_{nhm}$
    User-specified parameters:
        $E_{\mathbf{V}}$    maximum number of learning epochs for $\mathbf{V}$ allowed before a new hidden unit
               is added
        $\mathcal{E}_{\mathbf{V}}$    threshold percentage change in $J$ when training $\mathbf{V}$ before a new hidden unit
               is added
        $\mathcal{T}_{\mathbf{V}}$    maximum number of consecutive learning epochs for $\mathbf{V}$ with no significant
               change in $J$ that can be tolerated before a new hidden unit is added
    Output:
        $\mathcal{N}_{nhm}$ with $h$ incremented by 1
{
    Establish weights between input units and new hidden unit;
```

[4] $\tilde{c}_k$ is obtained by (linearly) mapping the desired classification c_k from $\{0, 1\}$ to $\{-1, 1\}$.

```
    For $i = 1$ to $E_{\mathbf{V}}$ {
        Train weight vector $\mathbf{v}_{h+1}$ for 1 epoch using $\mathcal{S}$;
        If percentage change in $J < \mathcal{E}_{\mathbf{V}}$ in each of the last $\mathcal{T}_{\mathbf{V}}$ epochs,
            Exit "For" loop;
    }
    Establish weights between output units and new hidden unit;
    Increment $h$ by 1;
}
```

The covariance $\mathrm{cov}(z_h, e_k)$ can be defined as:

$$\mathrm{cov}(z_h, e_k) = \frac{1}{|\mathcal{S}|} \sum_{s \in \mathcal{S}} (z_h^s - \overline{z}_h)(e_k^s - \overline{e}_k) = \frac{1}{|\mathcal{S}|} \sum_{s \in \mathcal{S}} z_h^s \, (e_k^s - \overline{e}_k)$$

where $\overline{z}_h$ and $\overline{e}_k$ are the averages of z_h^s and e_k^s respectively over $\mathcal{S}$. The gradient component $\partial J_h / \partial v_{ih}$ of J_h with respect to v_{ih} can be expressed as:

$$\frac{\partial J_h}{\partial v_{ih}} = \frac{1}{|\mathcal{S}|} \sum_{k \in \mathcal{O}} \mathrm{sgn}(\mathrm{cov}(z_h, e_k)) \sum_{s \in \mathcal{S}} g'(t_h^s) \, x_i^s \, (e_k^s - \overline{e}_k)$$

where t_h is the total input of the hth hidden unit.

To maximize the objective function J_h, the weight updating rule $\Delta v_{ih} = \eta \, (\partial J_h / \partial v_{ih})$ can be used to learn the weight vector $\mathbf{v}_h$, where $\eta > 0$ is the learning rate parameter. This hill-climbing (gradient-ascent) procedure is very similar in principle to that for learning (gradient-descent) the output weight matrix $\mathbf{W}$. Equivalently, the quickprop algorithm can also be used. After the weight values v_{ih}'s are determined, they become fixed and will never change during the subsequent learning. The final output weight values of $\mathcal{N}_{n,h-1,m}$ become the initial output weight values of $\mathcal{N}_{nhm}$, with the addition of m new output weights between the newly added hidden unit and the m output units. This is how the transfer effects of prior learning are utilized in the incremental construction of large networks from smaller ones.

The network construction strategy described above was inspired by the *cascade-correlation networks* [5]. While cascade-correlation networks allow more than one hidden layer with only one unit per layer, our method constructs feedforward networks with at most one hidden layer. Simple complexity analysis shows that the number of weights grows as $O(h^2)$ for cascade-correlation networks but only as $O(h)$ for our networks.

4 Classification Experiments

4.1 Example Domains

In this section, we will discuss four of the pattern classification domains we have studied. They are referred to as Mushroom, Thyroid, Waveform, and Symmetry, respectively, and described briefly below.

4.1.1 Domain 1: Mushroom

This domain refers to the data set that contains descriptions of hypothetical samples corresponding to 23 species of gilled mushrooms in the Agaricus and Lepiota Family. It was obtained from the repository of machine learning databases at the University of California, Irvine, U.S.A., which have been widely used by the machine learning community as benchmarks for comparing different learning algorithms. Subsets of the data set were randomly selected for use in our experiments. Each species in the data set is classified as definitely edible, definitely poisonous, or of unknown edibility and not recommended. The latter category is combined with the poisonous one, thus resulting in two classes. All the 22 feature attributes are nominal-valued. Table 3 gives more details about the feature attributes.

***** TABLE 3 *****

4.1.2 Domain 2: Thyroid

This domain refers to the data set that contains thyroid disease records provided by the Garavan Institute of Sydney, Australia. Like the Mushroom data set, this was also obtained from the University of California, Irvine. Subsets of the data set were randomly selected for use in our experiments. Each record (pattern) consists of a number of continuous-valued, binary-valued, and nominal-valued feature attributes which correspond to the symptoms and personal particulars of a patient, and a class which indicates whether or not the patient has thyroid disease. Details of the feature attributes can be found in Table 4.

***** TABLE 4 *****

4.1.3 Domain 3: Waveform

This problem is adapted from the waveform recognition problem as defined in [1, pp.49–55]. Each class is generated from a random convex combination of two of the three base waves sampled at intervals with Gaussian noise added. Mathematically, the three base waves are represented as $(s_i(1), s_i(2), ..., s_i(21))$, $1 \leq i \leq 3$, where:

$$s_1(i) = \begin{cases} -\cos((i-1)\pi/6) & 1 \leq i \leq 13 \\ -1 & 14 \leq i \leq 21 \end{cases}$$

$$s_2(i) = \begin{cases} -1 & 1 \leq i \leq 4 \\ -\cos((i-5)\pi/6) & 5 \leq i \leq 17 \\ -1 & 18 \leq i \leq 21 \end{cases}$$

$$s_3(i) = \begin{cases} -1 & 1 \leq i \leq 8 \\ -\cos((i-9)\pi/6) & 9 \leq i \leq 21. \end{cases}$$

A waveform $(x_1, x_2, ..., x_{21})$ is generated from the base waves as follows:

$$x_i = \begin{cases} \mu\, s_1(i) + (1-\mu)s_2(i) + \xi_i & \text{Class 1} \\ \mu\, s_2(i) + (1-\mu)s_3(i) + \xi_i & \text{Class 2} \\ \mu\, s_1(i) + (1-\mu)s_3(i) + \xi_i & \text{Class 3} \end{cases}$$

where μ is a random variable with uniform distribution on the interval $[0,1]$ and ξ_i's are independent random variables with normal distribution $N(0, 0.01)$. Additionally, another problem instance is defined by introducing 19 noise attributes which are simply independent random variables with distribution $N(0, 0.01)$.

4.1.4 Domain 4: Symmetry

This domain considers $N \times N$ binary input patterns with a horizontal, vertical, or diagonal axis of mirror symmetry (Figure 4). The problem is to learn to correctly classify the patterns into three classes according to the axis of symmetry, as described in [17]. For every axis of symmetry, there are a total of $2^{N^2/2}$ possible input patterns. Patterns were generated randomly for both the training and testing sets but those with more than one axis of symmetry were excluded.

***** FIGURE 4 *****

4.2 Specifications of Domain Problem Instances

Table 5 summarizes the specifications of the problem instances. We distinguish between the number of feature attributes of a problem instance and the number of input units in the corresponding network. While a continuous-valued or binary-valued attribute is simply represented by a single input unit, a nominal-valued attribute can correspond to more than one unit. The number of units used is equal to the number of possible values the attribute can have. For the number of output units, however, it is always equal to the number of classes of the problem instance.

***** TABLE 5 *****

4.3 Experiments and Results

For each problem instance, we performed 100 trials to average out the variations due to random initialization of weights. We used the same set of user-specified parameter values ($H = 20$, $\psi = 0$, $E_{\mathbf{W}} = 50$, $\mathcal{E}_{\mathbf{W}} = 1\%$, $\mathcal{T}_{\mathbf{W}} = 8$, $E_{\mathbf{V}} = 50$, $\mathcal{E}_{\mathbf{V}} = 3\%$, $\mathcal{T}_{\mathbf{V}} = 8$) for all the problem instances.[5] The empirical results are summarized in Table 6. The three cost measures for characterizing the computational requirements are the number of hidden units, number of weight parameters, and total number of epochs spent on learning both the input and output weights. The percentage of training patterns correctly learned and the percentage of testing patterns correctly classified by the trained network can be used as performance measures.

***** TABLE 6 *****

For all the problem instances, a 100% accuracy can almost always be obtained for the training patterns. The percentage accuracy for testing is slightly lower for the Mushroom, Thyroid, and Waveform domains but is more than 30% lower for Symmetry. The Mushroom domain turns out to be linearly separable, and hence requires no hidden units at all. Overfitting of the training patterns has occurred for all the problem instances, but is especially significant for the Symmetry

[5]Except for H which can be chosen to determine the maximum network size and ψ which can be specified to determine the acceptance criterion of the user, all the other six parameters used their default values, which are results of previous experience with various problem domains. We always use these default values for any new problem given, thus avoiding the trouble of adjusting parameter values which makes portability to new problem domains a lot more difficult.

problem. It should be noted that the training sets for the Symmetry domain, though with 3000 patterns each already, only represent a very small portion of the feature space (about 1% for the 6×6 problem and 0.00007% for the 8×8 problem). The training patterns used may not represent all the discriminating features that are needed for satisfactory classification of the testing patterns. However, the generalization results are in fact not bad, considering that only about 33% accuracy would be expected for "zero-knowledge" random guess.

Typical learning curves for the four problem domains are shown in Figures 5–8. Each figure shows the percentage classification errors for both the training and testing data during the learning process. For the three domains that required hidden units in the experiments, the place where each new hidden unit is added is indicated by a dotted vertical line in the graph. Notice that there is always a flat region in the learning curve before a new unit is added. Each flat region corresponds to learning the input weights of a hidden unit with the output weights held constant. Upon adding a new unit, the curve usually shows significant improvement in classification accuracy of the training data especially during the early learning stage, although some local fluctuations are not uncommon.

***** FIGURE 5 *****

***** FIGURE 6 *****

***** FIGURE 7 *****

***** FIGURE 8 *****

One might ask whether hidden units are really necessary to the Thyroid, Waveform, and Symmetry domains. Are they in fact linearly separable like the Mushroom domain? To answer this question, we performed a set of control experiments on the three domains using networks with no hidden units ($\mathcal{N}_{nm}$). This can easily be done by changing the default values of some user-specified parameters ($H = 0$, $E_{\mathbf{W}} = 1000$, $\mathcal{T}_{\mathbf{W}} = 1000$). The results are summarized in Table 7. We compared their performance with the learning curves in Figures 6–8. To assess the learning and, more importantly, generalization capabilities, it is the classification performance of the testing data that really counts. For this reason, Figures 9–11 show the learning curves of the testing data trained under different conditions. As we can see, the introduction of hidden units does not help much for the Thyroid domain, but is significant for Waveform and especially useful for Symmetry.

***** TABLE 7 *****

***** FIGURE 9 *****

***** FIGURE 10 *****

***** FIGURE 11 *****

Both the Mushroom and Thyroid domains ended up requiring no hidden units in the experiments, although hidden units were actually introduced for the Thyroid domain trained with default settings of the user-specified parameters. Notice from Figure 9 that before the first hidden unit is added into the network, the percentage error is already rather low at about 5%.[6] To avoid introducing hidden units which do not help significantly in subsequent learning, the absolute level of the residual error could be useful as an additional criterion to determine when to stop learning. The issue of termination criteria in general, however, requires further work to be done.

5 Discussions

This paper describes an approach to the dynamic construction of artificial neural networks for supervised learning. With this approach, learning starts with a simple network with no hidden units in which fast learning is possible. New units are then added to the network one at a time until satisfactory approximation of the desired mapping is reached. The fast learning phase corresponds to quickly finding a crude model so that the approximate solution is in the right ballpark. This is followed by fine-tuning the model to give better solutions. Transfer effects of prior learning experience can thus be utilized in shaping subsequent learning.

While a traditional pattern recognition system consists of separate subsystems for feature extraction and classification, the network approach considered in this paper performs both functions as the network learns and grows in size. More precisely, new hidden units correspond to higher-level features extracted from the training patterns which are themselves expressed in lower-level features. This in turn modifies the classifier so as to provide a more accurate approximation of the desired mapping and hence reduce the residual classification error. Of course, the patterns can be expressed in high-level features if they are available. This should make the learning problem easier

[6]It is not always possible to achieve 100% classification accuracy with no overgeneralization, because the different classes may be overlapping in the feature space and hence inherently non-separable.

and the resultant network is expected to be smaller in size.

There have been several recent attempts to construct networks by starting with a small network which gradually grows to an appropriate size at the same time when learning of weights occurs. [8, pp.158–162] gives a brief review of some of these models. While the model presented in this paper was inspired by the cascade-correlation algorithm (as mentioned in Section 3.2), it bears little resemblance with most other models proposed so far. Models such as [12, 6, 11] are basically networks of threshold logic units for learning Boolean functions. They all assume that the training set is consistent in the sense that the same input will never correspond to different outputs. Our model, however, makes use of sigmoidal units for approximating Bayesian classifiers with Bayesian *a posteriori* probabilities. There is no need to assume the consistency of training patterns since the objective of the decision-theoretic classifier is to minimize classification errors. As far as the network construction process is concerned, our model is somewhat similar to [11] in that the resultant network has only one hidden layer with its units added one at a time. Up to now, there is no conclusive (theoretical or empirical) result to show which model performs best for a given problem. Extensive comparative studies of the different models on a wide range of pattern classification problems are thus needed in the future.

Although single-hidden-layer feedforward networks are general enough in representing or approximating continuous functions to any specified degree of accuracy, more hidden layers may still be used in practice to reduce the total number of hidden units used. We suggest here one possible extension of our current approach. When a new unit is added to a network, the unit can be added to an existing hidden layer or to a newly created one. A candidate hidden unit is temporarily added to an existing layer. The network attempts to find a set of input weight values which gives the candidate unit a reasonably high degree of association with the residual error. If the network fails to find such an acceptable set after a specified number of trials, it will remove the candidate unit from the layer and put it in a new layer. In general, a network can have different numbers of units in different hidden layers. As a comparison, [5] allows no more than one unit per hidden layer.

It should be noted that the objective function J_h defined in Section 3 is by no means the only possible measure for optimization. Future work includes investigating other measures of degree of association between the output of a hidden unit and the residual error of the network. Furthermore, the issues of overgeneralization and termination criteria as mentioned in Section 4.3 will also be addressed in our future study.

6 Acknowledgment

The author would like to thank the anonymous reviewers for their suggestions that led to improvement in presentation of the paper.

References

[1] L. Breiman, J.H. Friedman, R.A. Olshen and C.J. Stone, *Classification and Regression Trees.* Wadsworth International, Belmont, California (1984).

[2] G. Cybenko, Approximation by superpositions of a sigmoid function, *Mathematics of Control, Signals and Systems* **2**, 303–314 (1989).

[3] H.C. Ellis, *The Transfer of Learning.* Macmillan, New York (1965).

[4] S.E. Fahlman, An empirical study of learning speed in back-propagation networks. Technical Report CMU-CS-88-162, School of Computer Science, Carnegie Mellon University (1988).

[5] S.E. Fahlman and C. Lebiere, The cascade-correlation learning architecture, in *Advances in Neural Information Processing Systems 2*, D.S. Touretzky (editor). Morgan Kaufmann, San Mateo, California, 524–532 (1990).

[6] M. Frean, The upstart algorithm: A method for constructing and training feedforward neural networks, *Neural Computation* **2**, 198–209 (1990).

[7] R.F. Grose and R.C. Birney (editors), *Transfer of Learning.* Van Nostrand, Princeton, New Jersey (1963).

[8] J. Hertz, A. Krogh and R.G. Palmer, *Introduction to the Theory of Neural Computation.* Addison-Wesley, Redwood City, California (1991).

[9] K. Hornik, M. Stinchcombe and H. White, Multilayer feedforward networks are universal approximators, *Neural Networks* **2**, 359–366 (1989).

[10] M.S. Klassen and Y.H. Pao, Characteristics of the functional-link net: A higher order delta rule net, in *Proceedings of the IEEE International Conference on Neural Networks*, San Diego, California (1988).

[11] M. Marchand, M. Golea and P. Rujár, A convergence theorem for sequential learning in two-layer perceptrons, *Europhysics Letters* **11**, 487–492 (1990).

[12] M. Mézard and J.P. Nadal, Learning in feedforward layered networks: The tiling algorithm, *Journal of Physics A* **22**, 2191–2204 (1989).

[13] H. Miyamoto, M. Kawato, T. Setoyama and R. Suzuki, Feedback-error-learning neural network for trajectory control of a robotic manipulator, *Neural Networks* **1**, 251–265 (1988).

[14] N.J. Nilsson, *Learning Machines: Foundations of Trainable Pattern Classifying Systems.* McGraw-Hill, New York (1965).

[15] F. Rosenblatt, *Principles of Neurodynamics: Perceptrons and the Theory of Brain Mechanisms.* Spartan, New York (1962).

[16] D.E. Rumelhart, G.E. Hinton and R.J. Williams, Learning internal representations by error propagation, in *Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Volume 1*, D.E. Rumelhart and J.L. McClelland (editors). MIT Press, Cambridge, Massachusetts, 318–364 (1986).

[17] T.J. Sejnowski and P.K. Kienker, Learning symmetry groups with hidden units: Beyond the perceptron, *Physica* **22D**, 260–275 (1986).

[18] B. Widrow and M.E. Hoff, Adaptive switching circuits, in *IRE WESCON Convention Record* **4**, 96–104 (1960).

A Quickprop Algorithm

The algorithm is heuristic in nature and based on two assumptions. First, the error curve for each weight can be approximated by a concave-upward parabola. Second, changes in the error curve of any weight are not affected by all the other weights that are changing concurrently. For each weight, the previous and current error gradients and the weight change are used to determine the form of the parabola. It is desired that the next weight change brings it directly to the valley (minimum) of this parabola.

At time step t, a weight and its gradient component are denoted as $w[t]$ and $S[t]$, respectively. Suppose the error curve is of the parabolic form:

$$E = a\,(w - b)^2 + c,$$

where $a > 0$, b, and c are constants. So,

$$S[t-1] = 2\,a\,(w[t-1] - b) \tag{6}$$

and

$$S[t] = 2\,a\,(w[t] - b). \tag{7}$$

Let $\Delta w[t-1] = w[t] - w[t-1]$ denotes the weight change from $w[t-1]$ to $w[t]$. The next weight change $\Delta w[t]$ is set to be equal to $b - w[t]$ so as to jump directly to the minimum point. Equations (6) and (7) can be combined and rewritten in terms of $\Delta w[t-1]$ and $\Delta w[t]$:

$$\Delta w[t] = \frac{S[t]}{S[t-1] - S[t]}\,\Delta w[t-1].$$

Since the actual form of the error curve might not be exactly parabolic, $w[t+1] = w[t] + \Delta w[t]$ is in general only an approximation of the optimal value. The above procedure continues iteratively until the error is acceptably small.

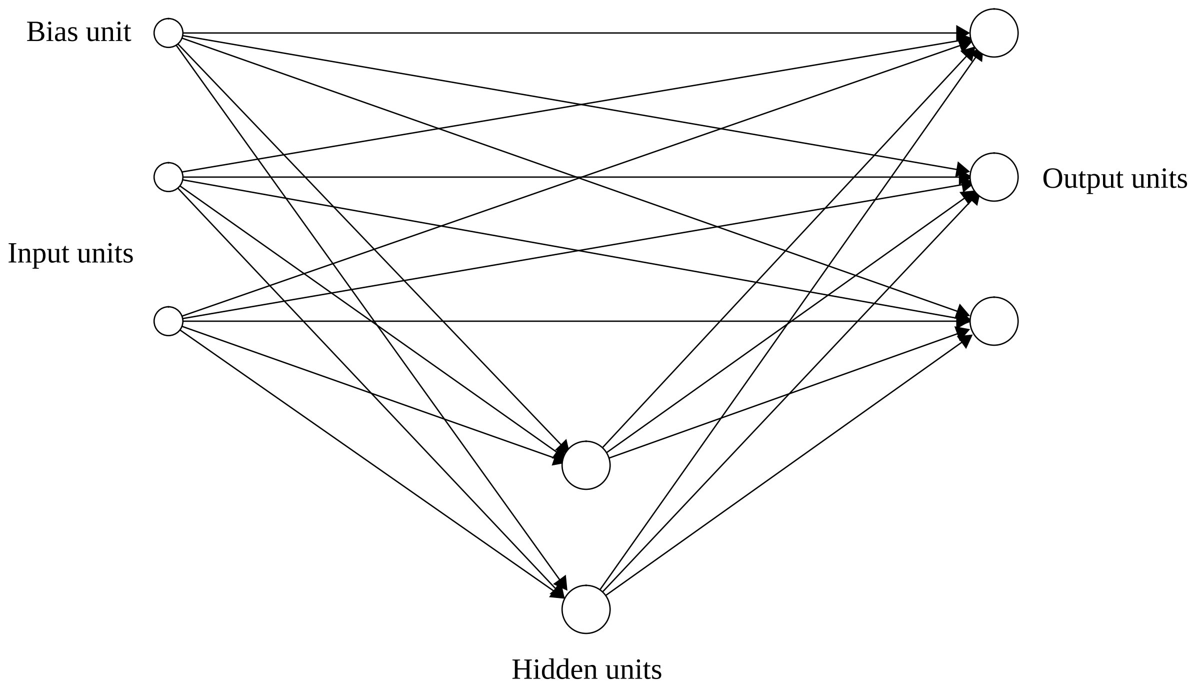


Figure 1: An example network with two input units, two hidden units, and three output units.

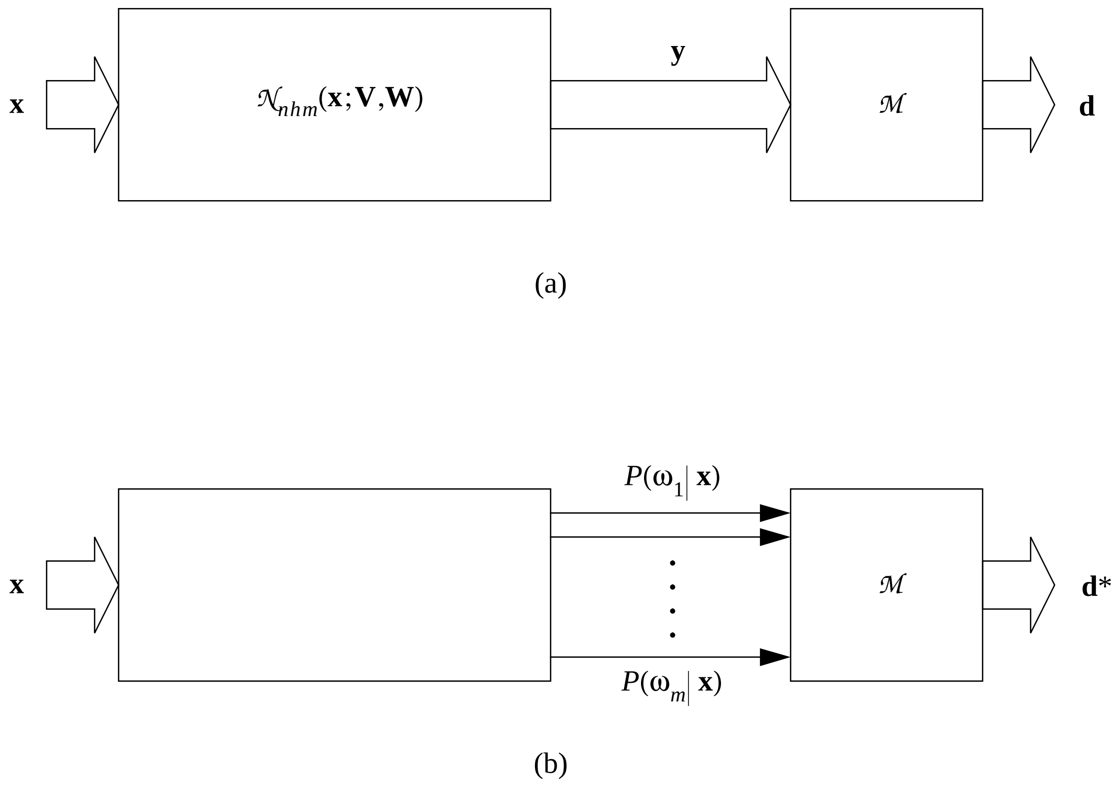
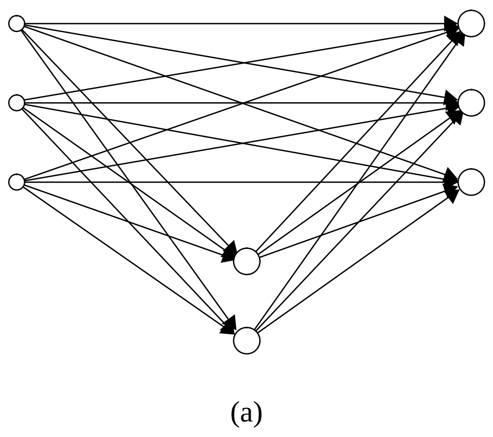
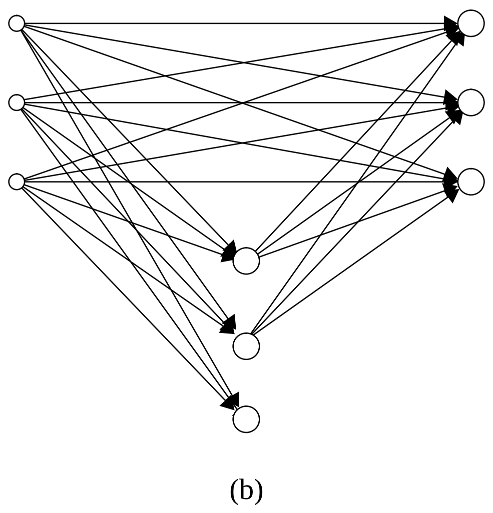


Figure 2: Functional decomposition of: (a) network classifier; (b) Bayesian classifier.





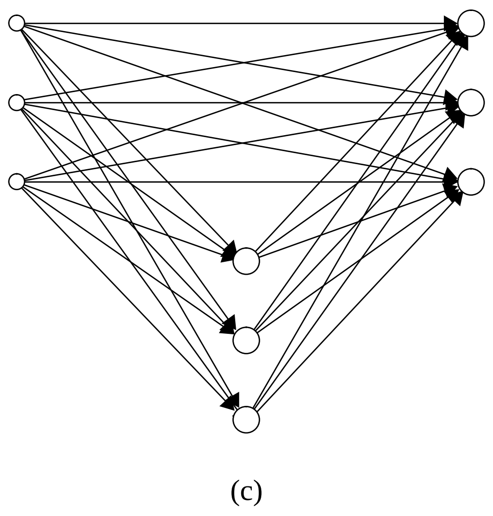
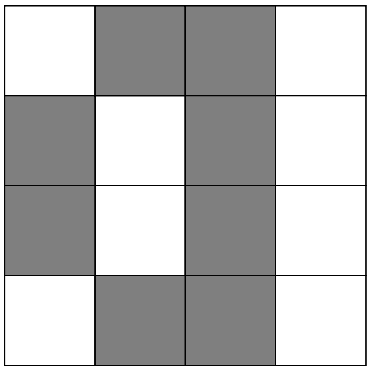
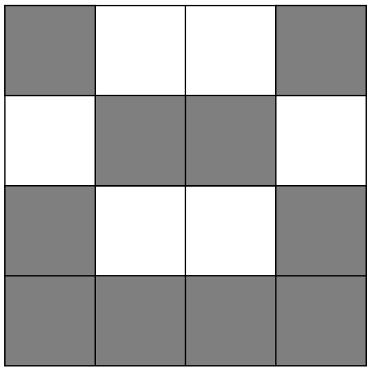


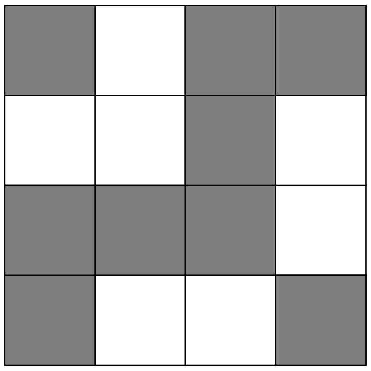
Figure 3: An example showing the addition of a new hidden unit.



(a)



(b)



(c)

Figure 4: Examples of the three types of mirror symmetry: (a) horizontal; (b) vertical; (c) diagonal.

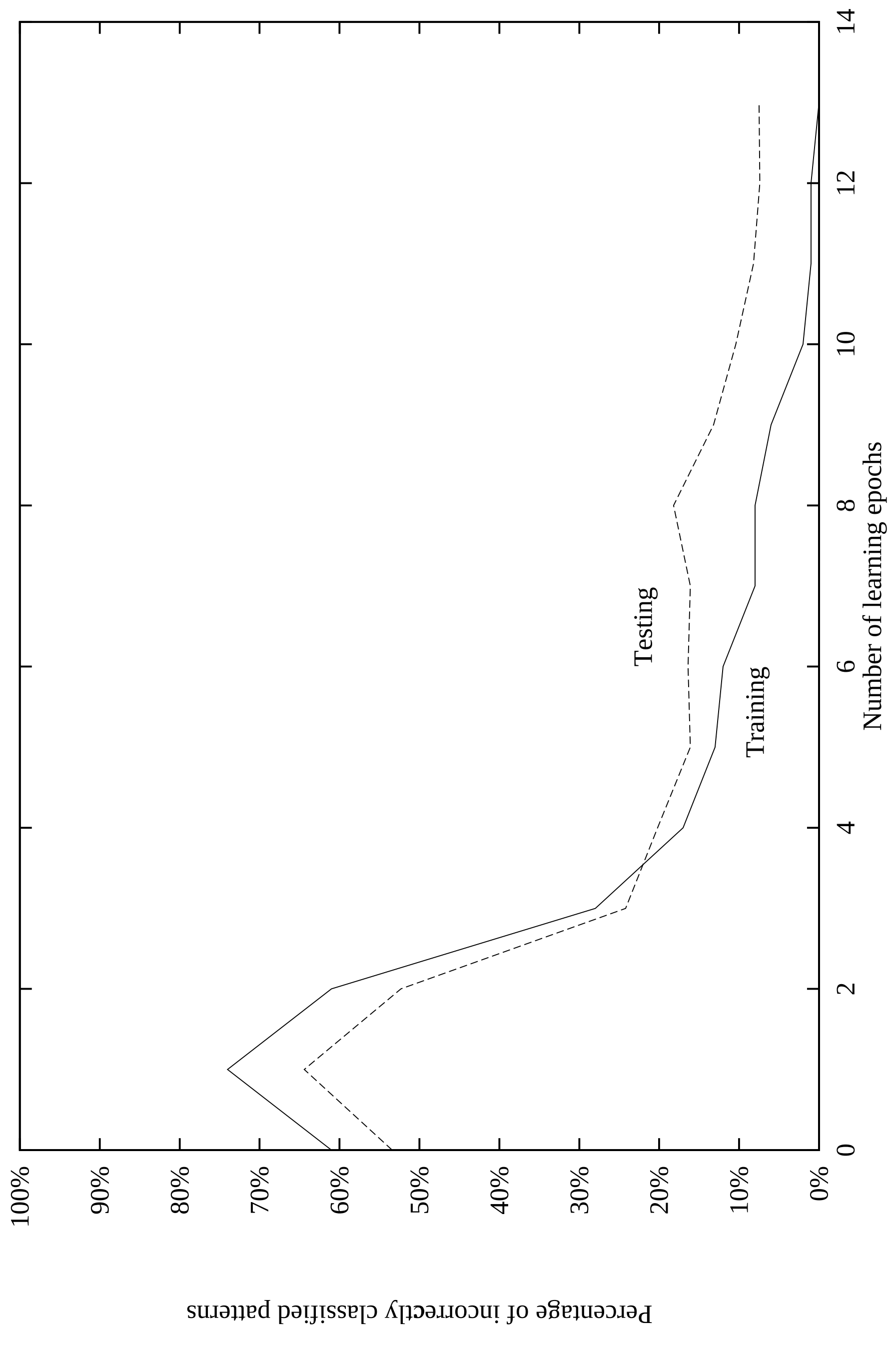


Figure 5: Learning performance for a typical run of the Mushroom domain using default settings for the user-specified parameters.

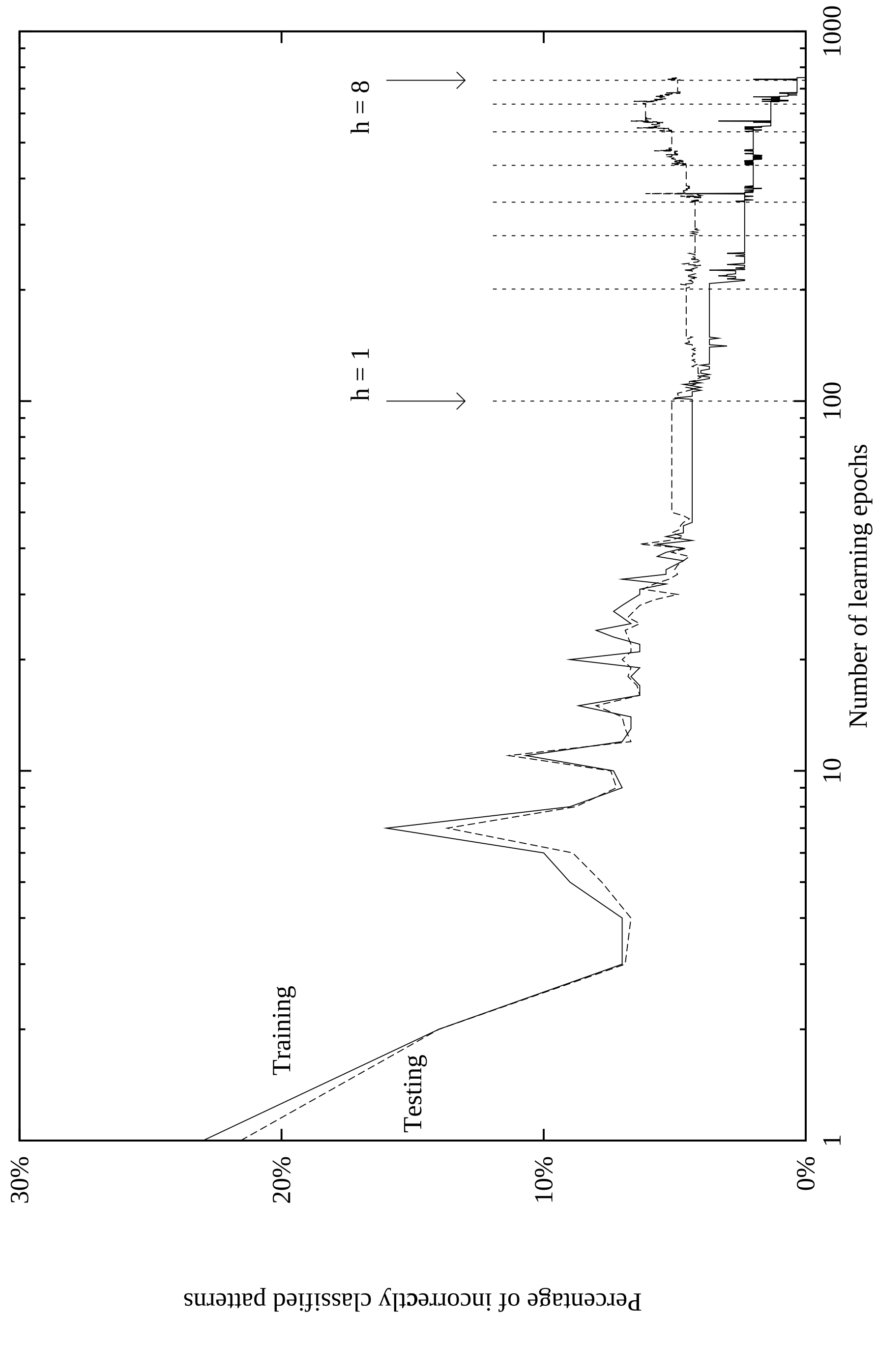


Figure 6: Learning performance for a typical run of the Thyroid domain using default settings for the user-specified parameters.

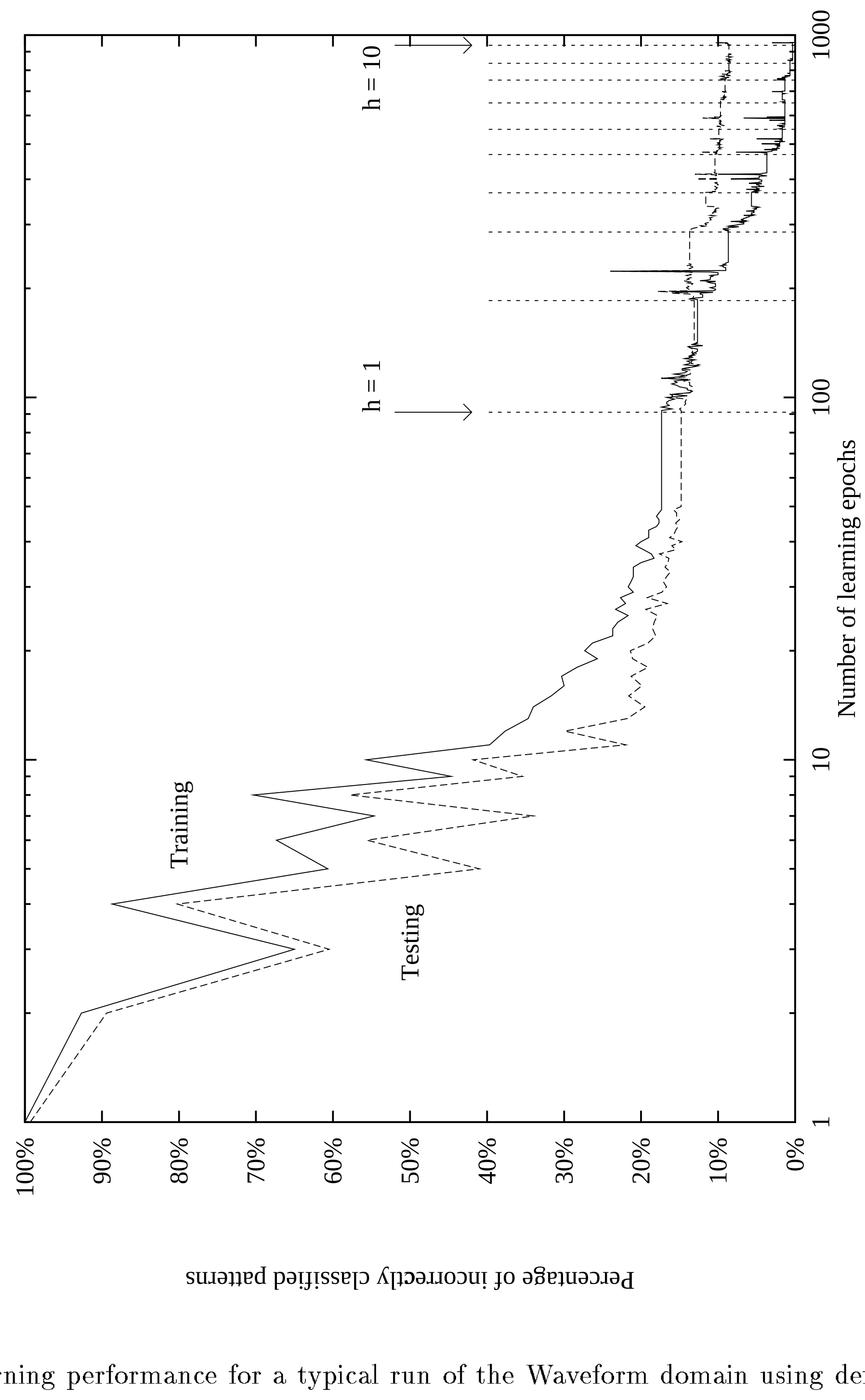


Figure 7: Learning performance for a typical run of the Waveform domain using default settings for the user-specified parameters.

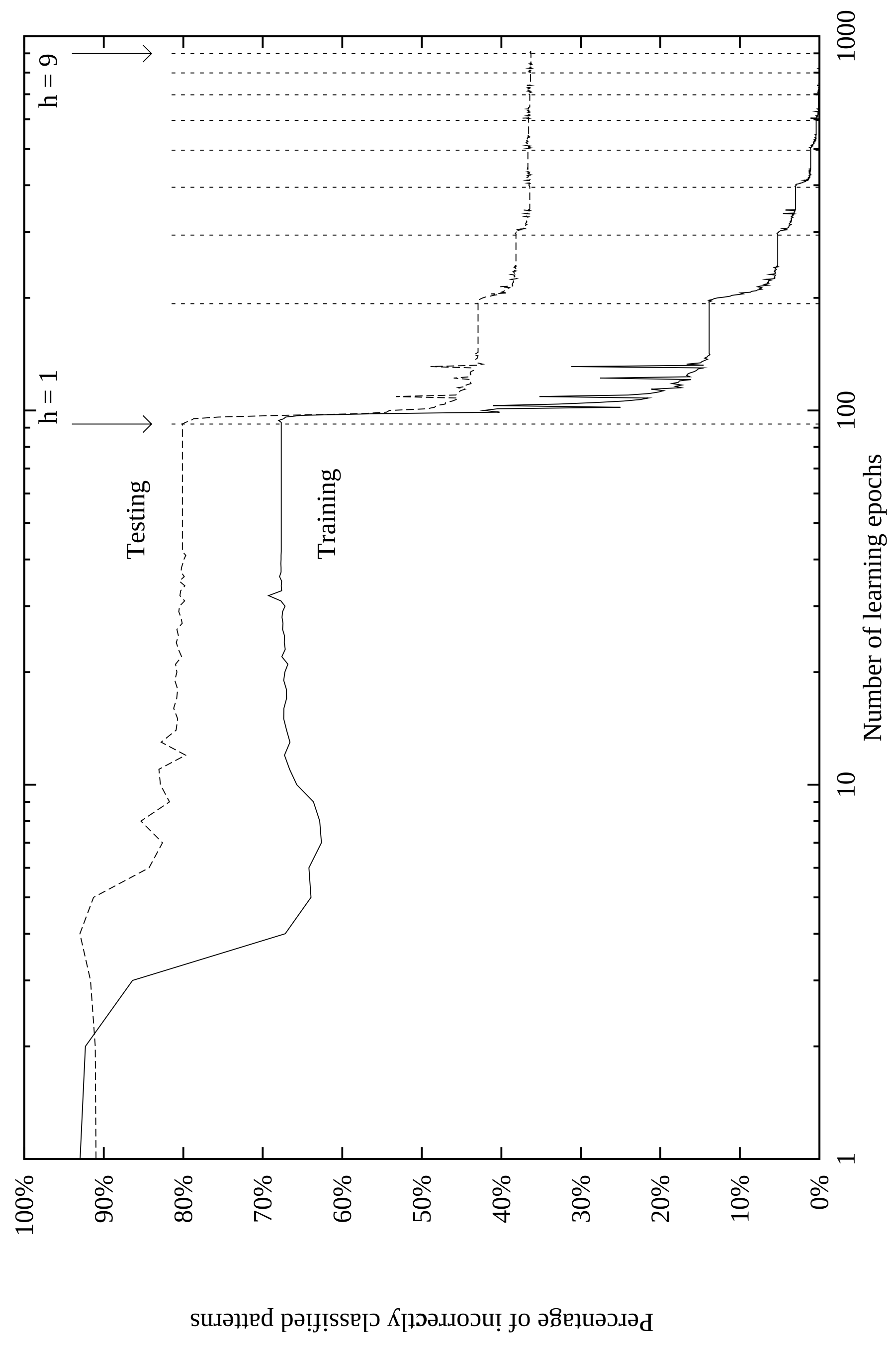


Figure 8: Learning performance for a typical run of the Symmetry domain using default settings for the user-specified parameters.

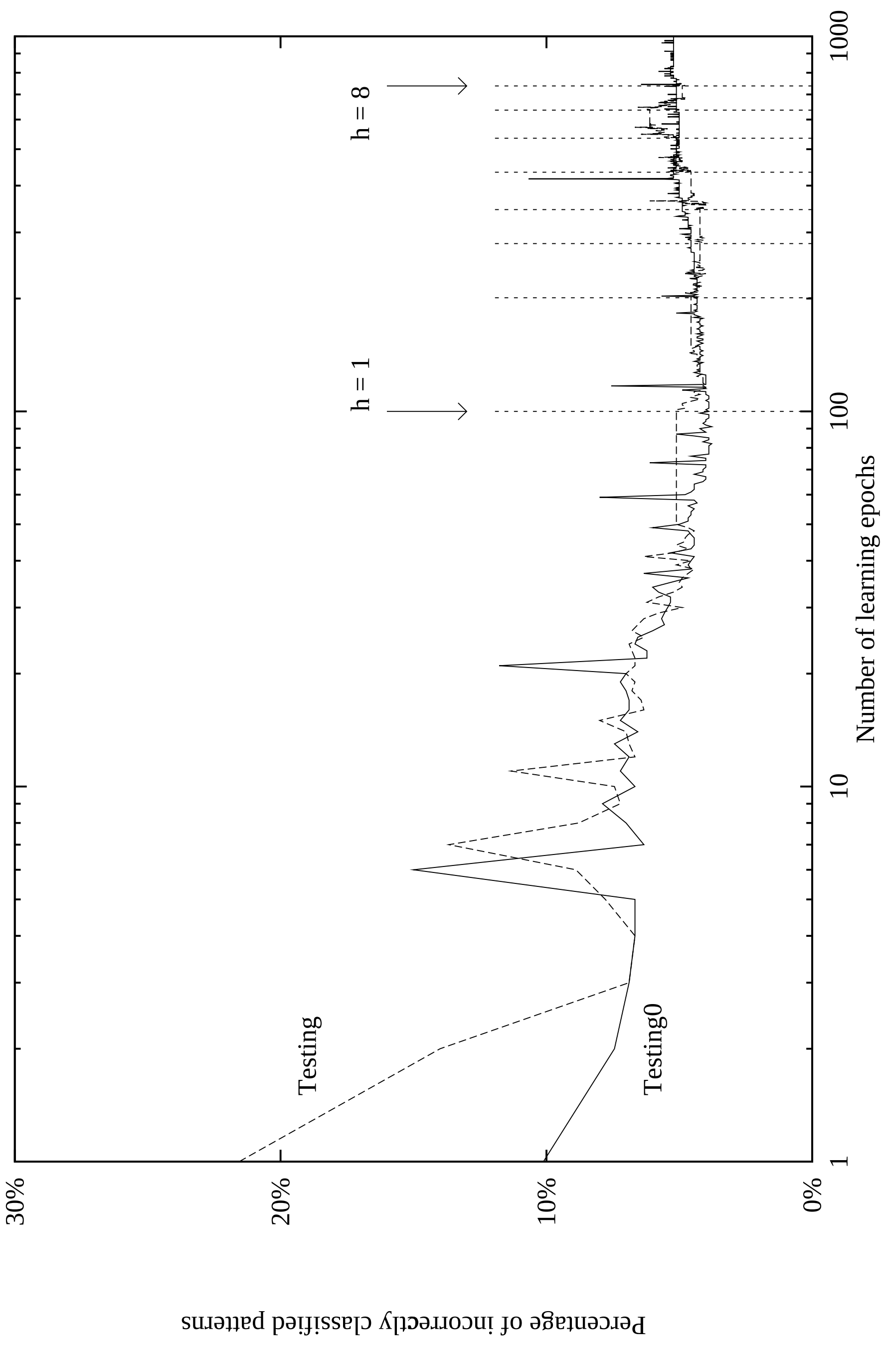


Figure 9: Comparing the "Testing" curve in Figure 6 (Thyroid domain) with one ("Testing0") obtained from training a network with no hidden units.

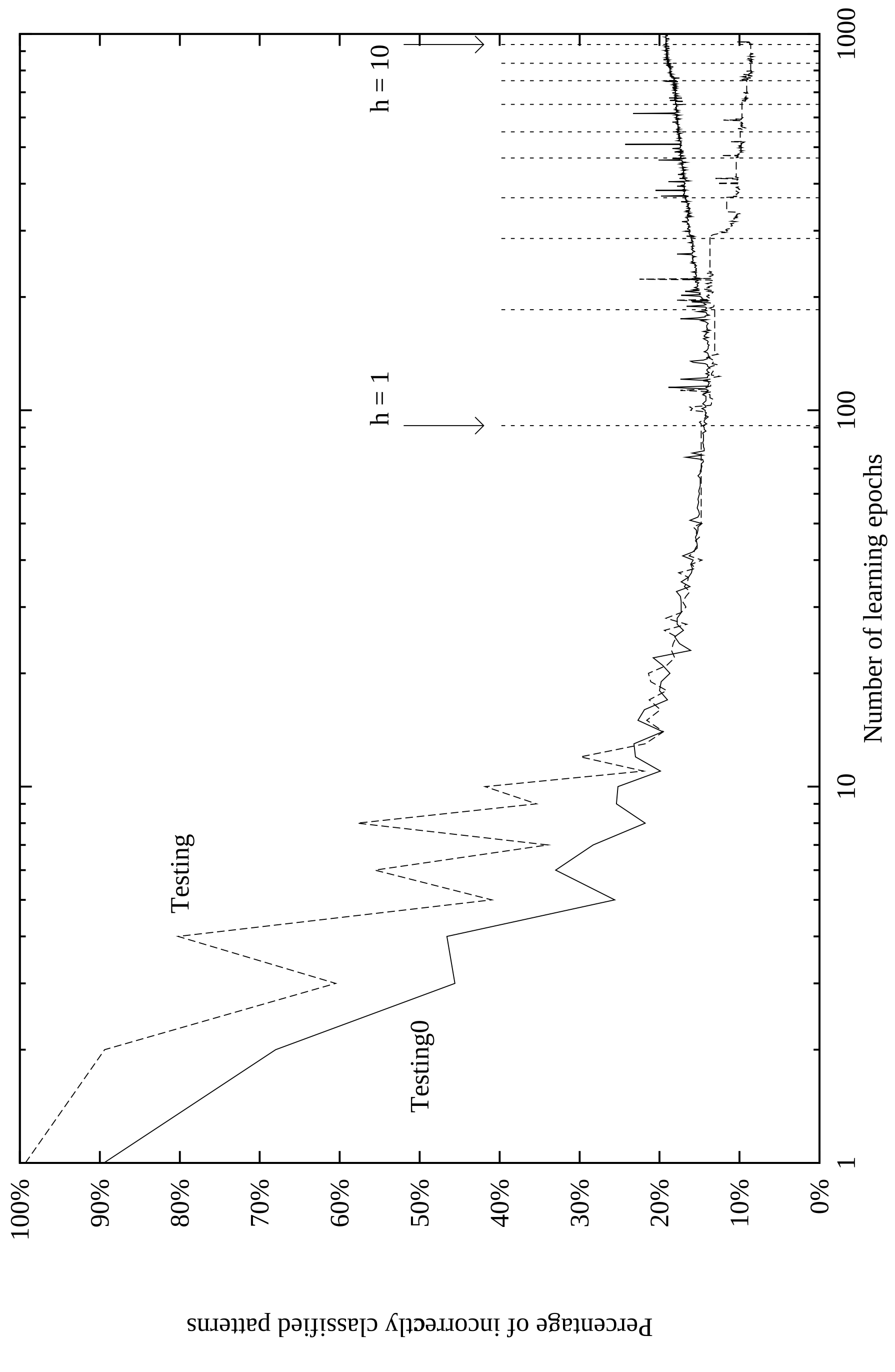


Figure 10: Comparing the "Testing" curve in Figure 7 (Waveform domain) with one ("Testing0") obtained from training a network with no hidden units.

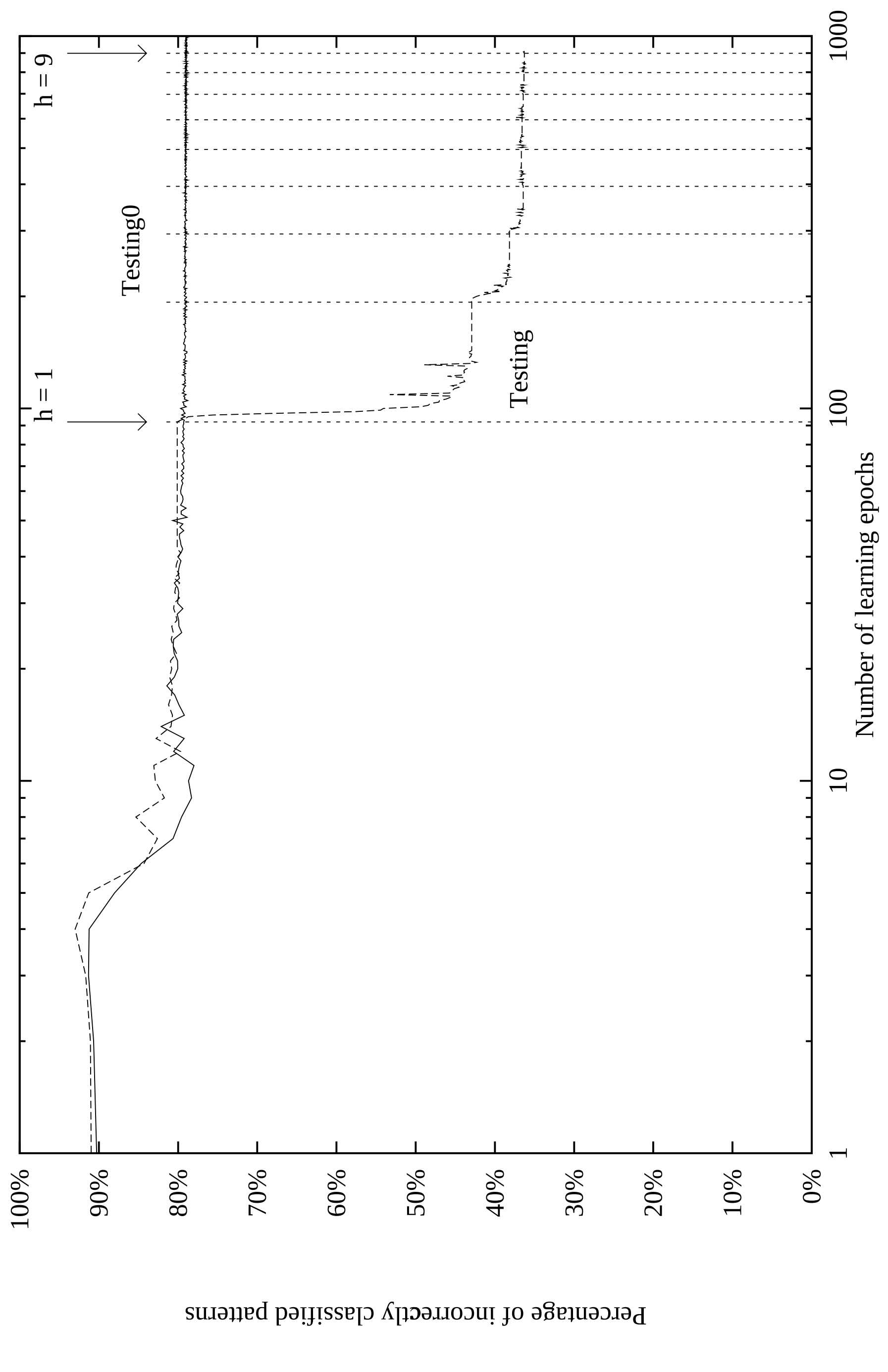


Figure 11: Comparing the “Testing” curve in Figure 8 (Symmetry domain) with one (“Testing0”) obtained from training a network with no hidden units.

Table 1: Definitions of symbols used in functional formulation.

Symbol	Definition	Remark
n	number of input units (number of input features)	$n \geq 1$
h	number of hidden units	$h \geq 0$
m	number of output units (number of output classes)	$m \geq 2$
i	index of the ith input unit ($1 \leq i \leq n$) or the bias unit ($i = 0$)	$0 \leq i \leq n$
j	index of the jth hidden unit	$1 \leq j \leq h$
k	index of the kth output unit	$1 \leq k \leq m$
$\mathbf{x}$	feature vector	$= (x_1, x_2, ..., x_n)^T$
$\mathbf{x}^{n+1}$	network input vector (augmented feature vector)	$= (x_0, x_1, ..., x_n)^T$
$\mathbf{v}_j$	weight vector of the jth hidden unit	$= (v_{0j}, v_{1j}, ..., v_{nj})^T$
$\mathbf{w}_k^{n+h+1}$	weight vector of the kth output unit	$= (w_{0k}, w_{1k}, ..., w_{n+h,k})^T$
$g(t)$	transfer function for hidden units	$= \tanh(t)$
$f(t)$	transfer function for output units	$= 1/(1 + e^{-t})$
$\tilde{g}(\mathbf{t}; \boldsymbol{\theta})$	parameterized version of g with parameter vector $\boldsymbol{\theta}$	$= g(\mathbf{t} \cdot \boldsymbol{\theta})$
$\tilde{f}(\mathbf{t}; \boldsymbol{\theta})$	parameterized version of f with parameter vector $\boldsymbol{\theta}$	$= f(\mathbf{t} \cdot \boldsymbol{\theta})$

Table 2: Definitions of symbols used in equivalence proof.

Symbol	Definition
$\Omega_{\mathbf{x}}$	feature space (space of all possible feature vectors)
$\mathcal{S}$	training set
$\|\mathcal{S}\|$	cardinality of $\mathcal{S}$
ω_k	event that the pattern belongs to the kth class
$\mathcal{S}_k$	subset of $\mathcal{S}$ with class membership ω_k
s	index of the sth training pattern
$P(\omega_k)$	*a priori* probability of ω_k
$P(\omega_k \mid \mathbf{x})$	*a posteriori* probability of ω_k given $\mathbf{x}$
$\rho(\mathbf{x})$	probability density function for $\mathbf{x}$
$\rho(\mathbf{x} \mid \omega_k)$	probability density function for $\mathbf{x}$ given ω_k
	(class-conditional probability density function for $\mathbf{x}$)
$\varepsilon_{\mathcal{S}}(\mathbf{V}, \mathbf{W})$	sample total error function for $\mathcal{S}$
$\overline{\varepsilon}(\mathbf{V}, \mathbf{W})$	population average error function

Table 3: Feature attributes and their number of attribute values for the Mushroom domain.

Feature attribute	Number of values
cap shape	6
cap surface	4
cap color	10
bruises?	2
odor	9
gill attachment	4
gill spacing	3
gill size	2
gill color	12
stalk shape	2
stalk root	6
stalk surface above ring	4
stalk surface below ring	4
stalk color above ring	9
stalk color below ring	9
veil type	2
veil color	4
ring number	3
ring type	8
spore print color	9
population	6
habitat	7
	Total = 125

Table 4: Feature attributes and their value types for the Thyroid domain.

Feature attribute	Value type
age	continuous
sex	nominal (2 values)
on thyroxine	binary
query on thyroxine	binary
on antithyroid medication	binary
sick	binary
pregnant	binary
thyroid surgery	binary
I131 treatment	binary
query on hypothyroid	binary
query on hyperthyroid	binary
lithium	binary
goitre	binary
tumor	binary
hypopituitary	binary
psych	binary
TSH measured	binary
TSH	continuous
T3 measured	binary
T3	continuous
TT4 measured	binary
TT4	continuous
T4U measured	binary
T4U	continuous
FTI measured	binary
FTI	continuous
TBG measured	binary
TBG	continuous
referral source	nominal (5 values)

Table 5: Specifications of the problem instances. For each training or testing set, the number of patterns in each class is shown in parentheses. (#F: number of features; #I: number of input units; #O: number of classes or number of output units; #Tr: number of training patterns; #Te: number of testing patterns)

Domain	Mushroom		Thyroid		Waveform		Symmetry	
Instance	1	2	1	2	1	2	1	2
#F	22	22	29	29	21	40	36	64
#I	125	125	33	33	21	40	36	64
#O	2	2	2	2	3	3	3	3
#Tr	300 (150; 150)	100 (50; 50)	500 (37; 463)	300 (17; 283)	300 (100; 100; 100)	300 (100; 100; 100)	3000 (1000; 1000; 1000)	3000 (1000; 1000; 1000)
#Te	1000 (500; 500)	1000 (500; 500)	900 (54; 846)	900 (54; 846)	1000 (333; 333; 334)	1000 (333; 333; 334)	3000 (1000; 1000; 1000)	3000 (1000; 1000; 1000)

Table 6: Empirical results of the problem instances (four domains) using default settings of the user-specified parameters. Each table entry has three numbers which correspond to the minimum, average, and maximum cost or performance measures among 100 trials. (#H: number of hidden units; #P: number of weight parameters; #E: number of learning epochs; %Tr: percentage accuracy for training; %Te: percentage accuracy for testing)

Domain		Mushroom		Thyroid		Waveform		Symmetry	
Instance		1	2	1	2	1	2	1	2
#H	min	0	0	6	5	9	6	6	6
	avg	0	0	12.32	12.65	13.98	10.42	8.59	8.92
	max	0	0	20	20	20	17	12	13
#P	min	252	252	284	248	291	387	351	603
	avg	252	252	511.52	523.40	415.50	581.48	454.60	801.56
	max	252	252	788	788	566	871	591	1079
#E	min	18	7	623	486	901	642	593	615
	avg	24.04	12.15	1088.58	1076.98	1311.86	1021.08	863.33	898.77
	max	32	19	1601	1691	1756	1549	1122	1295
%Tr	min	100	100	99.8	99.0	99.0	100	100	100
	avg	100	100	100	100	100	100	100	100
	max	100	100	100	100	100	100	100	100
%Te	min	97.1	88.9	94.4	91.0	89.3	88.0	64.2	60.7
	avg	98.6	93.1	95.6	94.1	92.0	90.2	65.4	63.4
	max	99.8	96.2	97.1	96.4	94.0	92.4	66.5	64.8

Table 7: Empirical results of the problem instances (three domains) using networks with no hidden units.

Domain		Thyroid		Waveform		Symmetry	
Instance		1	2	1	2	1	2
#H		0	0	0	0	0	0
#P		68	68	66	123	111	195
#E		1000	1000	1000	1000	1000	1000
%Tr	min	97.8	97.3	83.7	86.3	33.4	32.2
	avg	97.8	97.7	84.1	92.9	33.4	32.3
	max	98.0	97.7	84.7	94.3	33.4	32.3
%Te	min	96.3	94.4	81.6	77.6	9.8	20.9
	avg	96.5	94.7	81.9	80.7	9.8	21.0
	max	96.6	94.9	82.2	81.5	9.8	21.2