National Défense
Defence nationale

NEURAL NETWORKS FOR CLASSIFICATION OF RADAR SIGNALS

by

Christopher A.P. Carter and Nathalie Massé

DEFENCE RESEARCH ESTABLISHMENT OTTAWA
TECHNICAL NOTE 93-33

Canada

November 1993
Ottawa

National Defence Défense nationale

NEURAL NETWORKS FOR CLASSIFICATION OF RADAR SIGNALS

by

Christopher A.P. Carter
Radar Electronic Support Measures Section
Electronic Warfare Division

and

Nathalie Massé
Applied Silicon Inc. Canada

DEFENCE RESEARCH ESTABLISHMENT OTTAWA
TECHNICAL NOTE 93-33

PCN
041LB

November 1993
Ottawa

ABSTRACT

Radar Electronic Support Measures (ESM) systems are faced with increasingly dense and complex electromagnetic environments. Traditional algorithms for signal recognition and analysis are highly complex, computationally intensive, often rely on heuristics, and require humans to verify and validate the analysis. In this paper, the use of an alternative technique — artificial neural networks — to classify pulse-to-pulse signal modulation patterns is investigated. Neural networks are an attractive alternative because of their potential to solve difficult classification problems more effectively and more quickly than conventional techniques. Neural networks adapt to a problem by learning, even in the presence of noise or distortion in the input data, without the requirement for human programming. In the paper, the fundamentals of network construction, training, behaviour and methods to improve the training process and enhance a network's performance are discussed. The results and a description of the classification experiments are also presented.

RÉSUMÉ

L'environnement électromagnétique dans lequel doit opérer le système de Mesure de Soutien Electroniques (MSE) pour signaux radar est de plus en plus dense et complexe. Les algorithmes classiques de reconnaissance et de traitement de ces signaux sont complexes et heuristiques; ils requièrent en général de longs temps de calcul ainsi que l'intervention humaine dans les étapes d'analyse et de validation. Les réseaux de neurones artificiels sont une solution de rechange intéressante vu leur capacité à résoudre des problèmes complexes de classification de manière plus efficace et plus rapide que les algorithmes classiques. Leur capacité d'apprendre leur permet de s'adapter au problème même en environnement bruité ou pour une distortion des entrées et ce, sans l'intervention humaine. Dans ce rapport, les réseaux de neurones artificiels sont utilisés pour effectuer la classification de signaux selon le type de modulation de la fréquence des impulsions. Dans la première partie du rapport, la structure, les algorithmes d'apprentisage et le comportement d'un réseau sont présentés. On aborde également la question de l'accélération de la vitesse d'apprentissage. La deuxième partie décrit les résultats des classifications de signaux effectuées par réseau de neurones.

EXECUTIVE SUMMARY

The purpose of a radar Electronic Support Measures (ESM) system is to detect, determine signal parameter values, and identify radar emitters in the electromagnetic environment. Research initiated at the Defence Research Establishment Ottawa (DREO) has supported the development of hardware and software for the next generation naval ESM system. This project is called CANEWS 2.

CANEWS 2 incorporates a number of complex signal processing algorithms designed to classify radar signals. One of these determines the pulse-to-pulse modulation pattern of an agile signal. The objective of this paper is to investigate the technology of neural networks and their applicability to this signal classification problem.

The potential offered by artificial neural networks has sparked a great deal of interest in the research community. Although traditional algorithms have been very successful at tasks that are characterized by formal logical rules, they have had little success at tasks which are difficult to characterize this way. These are often tasks that the human brain performs well, such as pattern recognition and common-sense reasoning. Among the attractive features of neural networks are their ability to learn, even in the presence of noise or distortion in the input data, the ease of maintenance, high performance, and the potential to solve difficult classification problems more effectively than conventional techniques.

The basic component of an artificial network is the artificial neuron. A network is created by connecting a number of neurons together. Each neuron contributes to the overall function performed by the network by taking values from the nodes on its input side and calculating a value which it passes on to the nodes on its output side. A special set of nodes are designated as initial input nodes; each node receives a single value from a vector representing the characteristics of an input to the network. These values are transformed as they propagate through the network — the final destination being a set of nodes designated as the network's output. The output vector produced by these nodes represents a function of the initial input vector.

This paper covers the fundamentals of neural network construction, training, and behaviour, then deals with problems pertaining to a particular type of network called the *back-propagation* network. Back-propagation networks are the most commonly used networks for practical applications. A number of pragmatic considerations in the design of such networks to improve their learning capacity and to enhance their performance are described. A set of neural networks to classify signal modulation patterns were initially designed to take the frequency values from a pulse train as input. By designing new networks, that took input values to which a Fourier transform had been applied, performance was greatly improved. The results obtained from these experiments have led the authors to conclude that neural networks are a promising technology in the area of radar signal classification.

TABLE OF CONTENTS

TABLE OF CONTENTS (cont'd)

LIST OF FIGURES

LIST OF TABLES

1.0 INTRODUCTION

The purpose of a radar Electronic Support Measures (ESM) system is to detect, determine signal parameter values, and identify radar emitters in the electromagnetic environment. Research initiated at the Defence Research Establishment Ottawa (DREO) has supported the development of hardware and software for the next generation naval ESM system. This project is called CANEWS 2.

CANEWS 2 incorporates a number of signal processing algorithms designed to classify radar signals. One of these determines the pulse-to-pulse modulation pattern of an agile signal. The goal of this paper is to investigate the technology of neural networks and their applicability to this signal classification problem.

1.1 Signal Classification in CANEWS 2

The approach taken by the CANEWS 2 software to identify radar emitters is to first deinterleave the detected signals into separate tracks. For an individual track, a classification is performed on each of the signal's parameters: pulse width, radio frequency of each pulse (RF), pulse repetition interval (PRI), and scan pattern. The classification process reveals structural and statistical attributes for each parameter. Identification then finds radars in an emitter library whose known characteristics match the observed characteristics on a parameter by parameter basis.

Of particular interest are the algorithms used to determine the structural and statistical attributes of the parameters. For each parameter, CANEWS 2 maintains a hierarchy of all of its possible structural characteristics. Entries at the same level in the taxonomy represent mutually exclusive classifications, entries below a particular entry represent specializations of that entry. For example, an RF signal is divided into two mutually exclusive classifications, either pulsed or continuous wave. Within the pulsed classification are two sub-classifications; the pulses either maintain a constant frequency or they vary from pulse to pulse. The variable classification in turn has two sub-classifications; the variation is either periodic or random; and of these structural possibilities, the signal could either cover an entire range of RF values in a continuous fashion, or merely cover a range at discrete points.

A set of statistical values is maintained for each structural classification. For example, in the case of a pulsed, variable, periodic, continuous RF signal, the set includes: the limits and the mean of the RF values observed, a description of the pulse-to-pulse modulation pattern (ramp, sinusoidal, triangular, ...), and the length of the period of the pattern. In the case of a random continuous RF signal, the set includes: the limits and mean as before, as well as a description of the probability distribution of the RF values (Gaussian, uniform, triangular, ...). The algorithms to compute these values, in particular the pulse-to-pulse modulation pattern and the probability distribution, have a high order of complexity. To determine the type of the modulation pattern, a match is made between a normalized set of the observed pulses and a sine wave to obtain a "goodness of fit" based on heuristics. This process is repeated for the other possible modulation patterns. The match with the highest goodness of fit determines the modulation type. The already high degree of complexity of this algorithm is only compounded when additional modulation patterns are included, as the order of complexity of the algorithm is linear with respect to the number of modulation types. This statement can also be made in reference to the determination of the probability distribution for the random classification.

The task of computing the pulse-to-pulse modulation pattern will be used to test the applicability of neural networks to signal classification.

1.2 Neural Networks

The potential offered by artificial neural networks has sparked a great deal of interest in the research community whose members are eager to apply it within their respective disciplines. Although traditional algorithms have been very successful at tasks that are characterized by formal logical rules, they have had little success at tasks which are difficult to characterize this way. These are often tasks that the human brain performs well, such as pattern recognition and common sense reasoning. Some of the features of neural networks that are attractive on a theoretic level are:

Learning: The ability to change behaviour based on sample inputs (and possibly desired outputs) rather than through program changes.

Generalization: The ability to recognize that an input belongs to a certain class in spite of noise or distortion.

Abstraction: The ability to extract common features of inputs to create separate classifications.

These features lead to a number of practical advantages:

Fault tolerance: Some processing capability can exist even if part of the network is destroyed.

Ease of maintenance: Programming is not required to change the behaviour of the network; adaptation to new variations of input data is accomplished through retraining.

High performance operation: Neural networks are very well suited to parallel computation. Special hardware with massively parallel architectures are being designed to exploit this feature.

Ease of integration: With the introduction of neural networks on chips, existing and developmental systems can readily take advantage of this new technology.

The applications for which neural networks are particularly suited include signal filtering, pattern recognition, noise removal, classification, data compression, image processing and auto-associative recall or synthesis. Neural networks that are currently in use or in the research phase include: interpreting medical images (for cancerous cells), detecting bombs in suitcases, controlling the rollers in steel mills, predicting currency exchange rates, and steering autonomous land vehicles. Perhaps surprisingly, neural networks are very poor at other kinds of computation such as arithmetic and syllogism (if ... then ... else ... reasoning).

This paper is concerned with classification of radar signals. Other authors have investigated similar problems using neural networks. Anderson [1] studied the classification of individual pulses. He used a network to group pulses with similar parameters together to identify emitters. He makes the interesting point that while many clustering algorithms are known, and there is no reason to expect a neural network to be any better than the best traditional algorithm, the fact that neural networks can be both fast and tolerant of noise make them particularly suited to his application. Coat and Hill [2] attempted emitter identification by using a neural network to analyze the inner structure of single pulses. Willson [14] investigated the use of neural networks to perform deinterleaving (to sort individual pulses into "bins" associated with the radar emitter that each pulse came from) and perform radar identification based on already classified pulse parameter information. Brown [3] employed a neural network to classify different probability distributions of pulsed random agile RF signals.

2.0 THEORY

The basic component of an artificial neural network is the artificial neuron. By connecting a number of neurons together, a network is created.

2.1 The Artificial Neuron

An artificial neuron, from here on referred to as a *neuron* or a *node*, is represented in Figure 1.

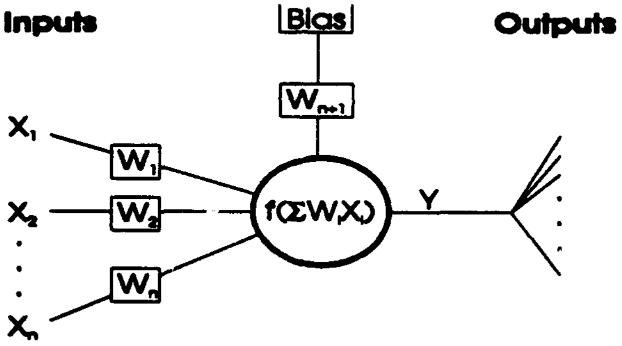


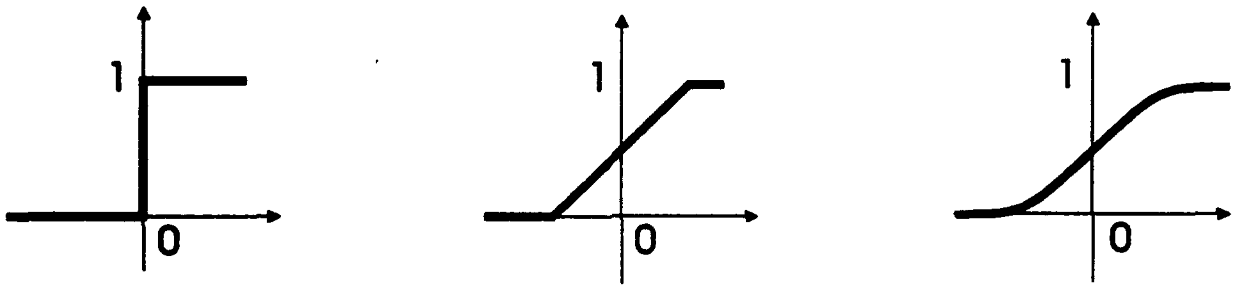
Figure 1: Structure of an Artificial Neuron

A neuron receives information from its inputs and passes on information via its outputs. The ***inputs*** to a neuron are supplied either externally, i.e. from data generated outside of the network, or internally from the output of another neuron in the network. In addition to the N inputs, a ***bias*** B is added, the utility of which will be seen later. Associated with each of the inputs X_n is a weight W_n which diminishes or reinforces this particular input. The bias has an associated weight as well. The ***net input*** to a neuron is the sum of the N inputs and the bias adjusted by the weights according to the following formula:

$$\sum_{1}^{N} W_{nx_n}+W_{n+1}B$$

To produce the final output *y*, the net input is further processed by a (usually) non-linear *transformation*, called a *transfer function* or *activation function*. The most common transfer functions (step, ramp, and sigmoid) are represented in Figure 2.

$$y = \begin{cases} 0 & if\ x<0 \\ 1 & otherwise \end{cases} \qquad y = \begin{cases} 0 & if\ x<-a \\ 1 & if\ x>a \\ \frac{x}{2a}+\frac{1}{2} & otherwise \end{cases} \qquad y = \frac{1}{1+e^{-ax}}$$



a) step

b) ramp

c) sigmoid

Figure 2: Common Transfer Functions

A displacement of these functions along the x axis can be obtained with the bias (generally set to 1) multiplied by its weight — a positive weight resulting in a displacement of the function to the right, a negative weight resulting in a displacement of the function to the left. These three functions can also be defined to be symmetric with respect to the x axis with the function taking on values between -1 and 1 in the case of the step and the ramp, and between -½ and ½ in the case of the sigmoid.

Note that any function defined over the input domain, that is monotonically increasing, has a lower and upper limit, can be used as a transfer function. With its binary output, the step is the simplest of these functions. Finally, some networks modify the behaviour of a node by generating different outputs based on whether the value generated by the transfer function exceeds a threshold.

While these transfer functions are the most common, they are not necessarily adequate for all applications. Some applications use a training algorithm that relies on the differentiability of the transfer function. The step and the ramp functions, being non-differentiable at the points of transition, are used in less sophisticated training algorithms.

2.2 Neural Network Types

In the previous section, structure of the neuron was described. This section deals with the physical organization of the neurons, and the pattern of connectioרּs that exists between them.

The neurons are usually organized in layers. The number of layers in a network varies from a single layer to multiple layers. In networks containing three or more layers, the middle layers are called ***hidden layers*** as they have no direct connection to the outside of the network. The first layer of the network, called the *input layer*, receives data simultaneously at each of its nodes, hence the notion of an *input vector*. At the *output layer* of the network, the outputs are generated in parallel to yield an *output vector*.

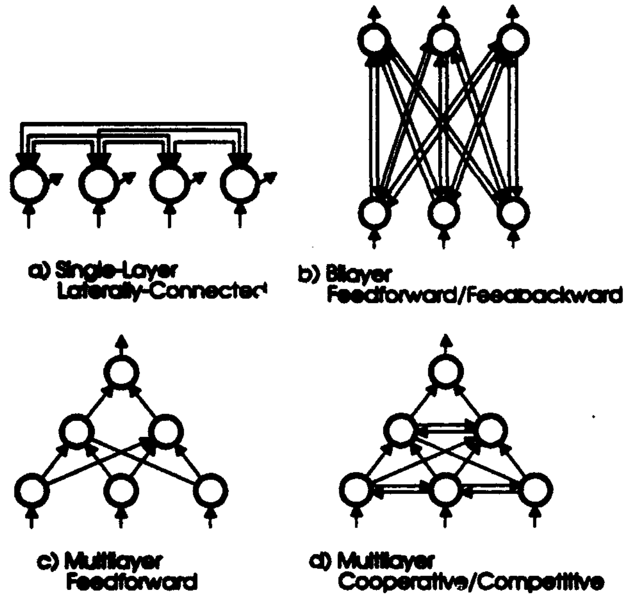
The nodes of a network can be connected in different patterns (see Figure 3):

Feed-forward: The flow of information is transmitted from layer to layer from the input layer to the output layer, with no feedback between layers or nodes. There are no connections between nodes occurring in the same layer, so no information is exchanged between nodes in any given layer.

Lateral: Every node is connected to every other node in the network. Recursive connections are permitted as well (a node can be connected to itself). Since there are no layers *per se*, networks of this type are often referred to as mono-layered networks.

Feedforward/Feedbackward: During the learning or training phase, information is transmitted from one layer to the next in both directions (from the input layer to the output layer and vice-versa). The number of weights is therefore double that of feed-forward networks.

Cooperative/Competitive: This is a combination of the previous two types, i.e., the connections between nodes in adjacent layers can be in both directions as well as between the nodes in any given layer.



a) Single-Layer Laterally-Connected

b) Bilayer Feedforward/Feedbackward

c) Multilayer Feedforward

d) Multilayer Cooperative/Competitive

Figure 3: Network Types (from Handbook of Neural Computing [10])

For certain problems, a single network may either be inadequate or ineffective in producing a satisfactory solution. In current work, a common approach is to break a problem into smaller tasks which are processed by connecting several networks together to work serially (in a cascade) or in parallel.

2.3 Training

The training process of a neural network is carried out by presenting a sequence of training examples or input vectors in a random order to the network. The weights in the network are updated after each presentation according to the network's learning algorithm. This process is repeated until a pre-determined criteria is satisfied. For example, the criteria may be the minimization of an error function, where the error represents the difference of the actual outputs and the expected outputs. The goal upon finishing the training period is to have the network produce the anticipated result for any input not contained in the set of training vectors. In the case of ***supervised training***, a pair of vectors is presented to the network: an input vector and a corresponding output vector. When the network is presented with an input vector, the generated output should match the corresponding output vector. If the input and output trai vectors are the same, the training process is said to be auto-associative, otherwise it is said t hetero-associative. There is also an intermediate form of training called reinforcement in which the network is simply informed if the output is correct or not. In the case of ***unsupervised training***, only the input vector is supplied.

In contrast to other systems, a characteristic attribute of neural networks is their ability to store information in the form of weights on the connections. Before training, the values of the weights are random, but as training proceeds, the network stores the information by changing the values of its weights. This information is distributed throughout the set of nodes. The weights therefore represent the network's current state of knowledge. Each individual training vector influences the entire set of weights and each individual weight depends on the entire set of training vectors.

2.3.1 Training Algorithms

A training algorithm specifies the way in which weights in the network are to be updated in order to improve the network's performance. In the case of supervised training, the algorithm is usually a variation of one of the following three types [7]:

Hebbian learning: The weight of an input connection to a node is increased if the value associated with the input connection and the value of the node's output are both large. This algorithm is designed to simulate the phenomena of learning through repetition and the fonnation of habits.

Delta rule learning: The input connection weights of a neuron are adjusted so as to reduce the error between the actual output and the desired output of the neuron.

Competitive learning: The nodes in a layer compete amongst themselves. The node that generates the greatest output for a given input modifies its weights to generate an even greater output. The other nodes in the layer modify their weights to decrease their output.

Unsupervised learning is less commonly used. The basic idea is that the network organises itself so that each output node responds (generates a large output) to a set of input vectors possessing a particular characteristic.

2.4 Summary

In this chapter, the micro-structure, the macro-structure and the training algorithms of artificial neural networks have been described. In a continuous attempt to improve the performance of neural networks, researchers have developed a wide variety of networks: the Perceptron, the Madaline, the Adaline, Brain-State-in-a-Box, Hopfield nets, back-propagation networks, self organizing networks, Boltzman machines, and networks that apply adaptive resonance theory, to name but a few. Among this vast choice of networks, back-propagation networks are without doubt the most popular and the most commonly used. This is due to the simplicity of their training algorithms and their effectiveness in handling an ever growing number of practical applications. Training with feedback (back-propagation) will be the subject of discussion in the following chapter.

3.0 BACK-PROPAGATION

3.1 Background

Back-propagation (or feedback) training is used in networks containing multiple layers: an input layer, an output layer, and one or two hidden layers. More than two hidden layers could be used if desired, but it has been proven that no advantage is to be gained in doing so [10]. The nodes in the input layer have a linear transfer function which distributes the input values to all of the nodes in the next layer. For all subsequent layers, including the output layer, the transfer functions of the nodes are non-linear and generally sigmoidal in nature. (As will be seen later, back-propagation learning relies on the uniform differentiability of the transfer function. The sigmoid function satisfies this constraint.) The connections are strictly feed-forward and the training is supervised.

The aim of the back-propagation learning rule (often referred to as the Generalized Delta Rule), is to minimize the difference between the actual output and the desired output, of a network for the entire set of input training vectors. To accomplish this, training vectors are presented one by one to the network, and for each one, the error is minimized by adjusting all of the weights in the network. In effect, all of the nodes, including their input weights, are jointly responsible for the global error. The adjustment is based on the *gradient* method (a method commonly used in optimization problems). This method consists of stepping along the *error function* to be minimized in the opposite direction of the gradient, i.e., in the direction where the value of the function decreases. For non-linear problems, as is the case here, the method must be used iteratively until convergence is attained. The number of iterations, or presentations necessary before convergence to a minimum point is obtained is extremely variable, being typically anywhere between 100 to 10000. The training period required using the back-propagation technique is in general a lengthy one, which is the principle weakness of the technique. Another problem inherent in this algorithm, is that there is no guaranty that the minimum value found is indeed the global minimum, and in fact is often just a local minimum. Furthermore, there is no guaranty that the global minimum of the error function is actually zero. It is incumbent upon the user, therefore, to verify that the minimum found is satisfactory for his particular application.

Once the training process is terminated, the performance of the network is evaluated with a set of test cases. This set of test vectors must be completely disjoint from the set of training vectors, as the network may provide excellent results for the training vectors but yield mediocre results for other vectors. This would indicate that the network has only learned specific cases, rather than acquiring the ability to generalize (interpolation for new input vectors becomes impossible). This phenomena can arise if the network is over-trained.

3.2 Output Layer Learning Rule

The goal of back-propagation learning is to adjust the weights in the network so as to minimize the error function. This is done by comparing the output vector with the desired output vector and adjusting the weights, starting with the output nodes and working backwards to the input nodes. In a network with N inputs and L hidden nodes arranged in a single layer and M outputs, the error to be minimized for each training vector is the sum of the squares of the outputs. Therefore, the global error for a training vector p is given by:

$$E_p = \frac{1}{2}\sum_{k=1}^{M} \delta_{pk}^2 \tag{1}$$

where δ_{pk} is the error for the vector p at the output node k between the expected output value y_{pk} and the actual output value o_{pk}.

To determine the relative responsibility and the direction in which each weight will be adjusted, the gradient of the error ∇E_p with respect to the weights ω_{kj} (weights associated with the connection between node j and node k) is calculated. Each weight in the output layer is adjusted proportionally in the negative direction of the partial derivative of the error E_p with respect to this weight:

$$-\frac{\partial E_p}{\partial \omega_{kj}^o} = (y_{pk} - o_{pk}) f_k^{o\prime} (\sum_{j=1}^{L} \omega_{kj}^o i_{pj} + \theta_k^o) i_{pj} \tag{2}$$

$$= (y_{pk} - o_{pk}) f_k^{o\prime} (net_{pk}^o) i_{pj} \tag{3}$$

where net_{pk}^o is the net input, $f_k^{o\prime}$ the derivative of the transfer function, θ_k^o the bias of the output node k and i_{pj} the output of the node j in the hidden layer. Note that because the partial derivatives make use of the derivative of the transfer function, this function must be differentiable for all values. The new weight is obtained by adding an amount proportional to the partial derivative to the former weight. With a constant of proportionality η, the adjusted weight is obtained by using the following formula (see [4] for the complete derivation):

$$\omega_{kj}^o(t + 1) = \omega_{kj}^o(t) + \eta (y_{pk} - o_{pk}) f_k^{o\prime} (net_{pk}^o) i_{pj} \tag{4}$$

$$= \omega^o_{kj}(t) + \eta\,\delta_{pk} f_k^{o'}(net^o_{pk}) i_{pj} \qquad (5)$$

Finally, by defining a value $\delta^o_{pk} = \delta_{pk} f_k^{o'}(net^o_{pk}) i_{pj}$ one obtains

$$\omega^o_{kj}(t+1) = \omega^o_{kj}(t) + \eta\,\delta^o_{pk} i_{pj} \qquad (6)$$

3.3 Hidden Layer Learning Rule

As the error also depends on the weights associated with the hidden layers of the network, similar calculations to those done for the weights associated with the output layer must be done for these weights as well. The error E_p^h for the hidden layer can be expressed as a function of the outputs of the hidden layer:

$$E_p^h = \frac{1}{2}\sum_{k=1}^{M} (y_{pk} - f_k^o(\sum_{j=1}^{L} \omega^o_{kj} i_{pj} + \theta^o_k))^2 \qquad (7)$$

Using this equation to calculate the gradient with respect to the weights ω^h_{ji} in the hidden layer, the following equation is obtained:

$$\frac{\partial E_p^h}{\partial \omega^h_{ji}} = -\sum_{k=1}^{M} (y_{pk} - o_{pk}) f_k^{o'}(net^o_{pk}) \omega^o_{kj} f_j^{h'}(net^h_{pj}) x_{pi} \qquad (8)$$

If an adjustment, proportional to the partial derivatives with a constant of proportionality equal to η is made, then the value of the adjusted weight is given by (see [4] for the complete derivation):

$$\omega^h_{ji}(t+1) = \omega^h_{ji}(t) + \eta f_j^{h'}(net^h_{pj}) x_{pi} \sum_{k=1}^{M} (y_{pk} - o_{pk}) f_k^{o'}(net^o_{pk}) \omega^o_{kj} \qquad (9)$$

$$= \omega^h_{ji}(t) + \eta f_j^{h'}(net^h_{pj}) x_{pi} \sum_{k=1}^{M} \delta^o_{pk} \omega^o_{kj} \qquad (10)$$

Notice that the new δ depends on all of the δ^o_{pk} of the output layer. In the case of a

$$= \omega^h_{ji}(t) + \eta\,\delta^h_{pj}x_{pi} \qquad (11)$$

network containing more than one hidden layer, the equation would be similar, except that the new δ would depend on the δ^h_{pj} of the second hidden layer instead of the output layer. This is why the adjustment of the weights begins at the output layer, then continues from layer to layer in the opposite direction from the usual flow of information (hence the term *back-propagation*).

3.4 Internal Behaviour

In the process of iterative training, the nodes in the hidden layers detect different characteristics found in the input vector in the sense that their outputs are activated (output value is large) if they detect a particular characteristic and are deactivated (output value is small) otherwise. Since there are usually fewer hidden nodes than input nodes, this phase of training is comparable to a reduction in the number of dimensions of the input space. If the network is used for the purposes of classification, the nodes in the output layer arrange themselves to partition the output space into separate regions based on the characteristics supplied by the hidden nodes. In general, each separate region represents a distinct class of the inputs.

When training is finished, the test vectors are presented to the network. The network interprets these new vectors by detecting the absence or presence of characteristics already seen in the training vectors. If a good generalization has been obtained, and a test vector shares common characteristics with a class of training vectors, then the network provides excellent results. The network can respond to new test vectors by interpolating a response from the set of training vectors. For example, if the network generates an output *A′* from a training vector *A* and an output C′ from a training vector *C*, then, if *B* is "between" *A* and *B*, the network will generate B′ between A′ and C′. The network does not in general provide good results in the context of extrapolation. The network does not learn features when it has never been given examples containing these features. Note that an addition of noise is considered to be an interpolation and not an extrapolation as the prime characteristics are still present and the noise, only adds superfluous information that is removed by the network.

3.5 Variations of Back-propagation Training

One of the principal drawbacks of the back-propagation technique is its slow training time. One of the ways to accelerate the learning time is to use a large learning coefficient (the value η in equation 6 in section 2.2 and in equation 11 in section 2.3). A large learning coefficient in theory, permits faster training times, since the steps taken in the opposite direction of the gradient are greater. This has the potential drawback, however, of causing oscillation of the total error when the curvature of the error surface is great. As the algorithm assumes that the surface is locally linear, the steps taken will continuously overstep the global minimum. A small learning coefficient stabilizes this process, but results in very long training periods. In addition, it increases the likelihood that the algorithm will become trapped in a local minimum. To alleviate these problems, variations of the original algorithm have been developed.

3.5.1 Varying the Coefficients

One can accelerate the training process by varying the value of the learning coefficient. By starting with a large value for η, large steps can be taken to quickly move towards the minimum of the error function and to avoid being trapped in a local minima. As training progresses, the value of η can be lowered to prevent oscillations about the global minimum.

3.5.2 The Momentum Method

A common method to speed up learning is the *momentum method*. As its name indicates, the momentum method consists of adding momentum in the direction of the weight change. In calculating a new adjustment to the weights, it takes into account the previous adjustment. Therefore a fraction α of the previous step is added to the formula to produce the following equation:

$$\omega^{o}_{kj}(t+1) = \omega^{o}_{kj}(t) + \eta\,\delta^{o}_{pk} i_{pj} + \alpha \nabla_p \omega^{o}_{kj}(t-1) \qquad (12)$$

where α is between 0 and 1. The new term acts as a low-pass filter on the weight error and reduces the oscillation of the network's global error, because it favours a continuation of movements in the same direction. With this method, faster rates of learning are obtained, while keeping the learning coefficient η small at the same time.

3.5.3 Cumulative Update of Weights

Cumulative back-propagation is another method that can improve the rate of convergence of the algorithm. It consists of accumulating the error for several pairs of input/output training vectors before updating the weights. When a single update is made, the error function is reduced for that particular pair only. The overall error function may actually increase. A global update, on the other hand, guarantees a reduction in the overall error function. Caution must be exercised though; the number of calculations greatly increases with the number of accumulated training pairs. The benefits of this technique are lost if the number of pairs is too large.

3.6 Summary

This chapter has described the back-propagation learning rule, its internal behaviour, and three methods for accelerating the training process. The next chapter covers a number of practical considerations, that all neural networks builders must deal with when constructing networks to effectively solve concrete problems.

4.0 PRACTICAL CONSIDERATIONS

Two important practical concerns in the design of a network are:

Convergence of the error function: A problem that can occur during the training period that will prevent convergence is node saturation. For transfer functions such as the sigmoid that attain their minimum and maximum values asymptotically, a large input at a node will be mapped into a region of the sigmoid whose derivative is virtually zero. If the derivative is zero, the learning adjustment will be zero, which implies that the node ceases to learn. When this happens, the node is said to be *saturated*. If too many nodes become saturated during training, learning could simply stop before a minimum in the error function can be found.

The speed of convergence is also an important consideration. Some techniques for accelerating the training period for back-propagation were described in the previous chapter.

Network performance: It is important that the network perform efficiently and effectively on real inputs when the training period is finished.

There are a variety of techniques that address these network design issues. These are described in the following sections.

4.1 Pre-treatment of Input Data

Raw data is rarely presented to a network without some form of pre-treatment. In certain cases, to ensure convergence of the error function, a simple normalization of the input and output vectors is performed to map the data values to within a narrow zone (outside of the asymptotic regions of the transfer function). For example, input values should be normalized to lie within the range [-1, 1] and the output training values to lie within [0, 1]. Freeman and Skapura [4] suggest an additional precaution when the transfer function of the output nodes is sigmoidal: normalize the output training values to [0.1, 0.9] rather than [0.0, 1.0] to avoid saturation of the nodes, since the values of 0.0 and 1.0 can only be obtained asymptotically at the output of this transfer function.

In other cases, the pre-treatment of the input data requires more attention. A transformation may be necessary to eliminate insignificant variations and superfluous details (translations, rotations, deformities, ...) while at the same time accentuating the pertinent information. Without doubt, the most commonly used transformation is the Fourier transform, but other transformations such as the Cepstrum, the Gabor transform and the spectrogram are used as well. Recently, Principal Component Analysis (PCA) has received attention [10]. This technique reduces the number of dimensions of a problem, but keeps the parameters for which the eigen vectors of the autocorrelation matrix have the greatest energy. When the dynamic range of a variable goes beyond several octaves, a logarithmic transformation may be necessary to retain the small variations that might otherwise be lost in normalization. There are other techniques such as Feature Extraction in which only the important characteristics of the raw data are presented to the network as inputs. This reduces the work and complexity of the network but one must ensure the relevance of the chosen characteristics. In general, it is difficult to know in advance which transformations will produce the best results. Experimentation is often required to determine the best approach.

If the network is to operate in a noisy environment, then once the inputs have been transformed and normalized to fit the application, appropriate noise must be added to any synthetically generated training vectors. Freeman and Skapura [4] have found that noise added to the inputs can facilitate the convergence of the network even if the network is destined to operate in a clean environment.

4.2 Weight Initialization and Adjustment

Before training begins, the weights in the network are randomly initialized. It is recommended that values of the weights lie between ±0.5 (some authors recommend even lower limits). Doing so prevents the neurons from saturating at the outset of training. Once training is in progress, it is important that the training vectors be presented in a random order so that the network learns the characteristics of the inputs rather than a specific order of inputs. Of even greater importance is to avoid presenting all of the vectors of one class, followed by the vectors of the second, and so on. There is a risk that the network will *forget* what it has learned from the preceding class during the training of the next class.

Another method of preventing saturation in the nodes is to add a small quantity ε to the derivative of the transfer function. As the adjustment of the weights depends on the derivative, this ε permits a slight adjustment to nodes that would otherwise cease to learn.

4.3 Choosing the Number of Training Vectors

Choosing the correct number of training vectors is an important consideration. When there are not enough vectors in the training set, there is a risk that the network will fail to generalize. The network will produce correct results for vectors in the training set but will fail on test vectors. According to the Baum and Haussler result [6], if the allowable error for any test vector is to be less than ε then the network will "almost certainly" generalize if the allowable error for the training vectors is less than $\varepsilon/2$ and the number of training vectors is greater than W/ε where W equals the number of weights in the network. For example, if the test vectors are to have errors less than 0.2 and there are 200 connections in the network, then there should be at least 1000 training vectors and the training should proceed until the value of the error function is less than 0.1.

4.4 Network Design

The decision concerning the number of layers and nodes to use for a particular application is of great importance as it influences the likelihood that the error function converges and that the network generalizes. While there are heuristics to guide these choices, experimentation is often required to achieve acceptable results. This experimental aspect is one of the prime sources of criticism directed towards the neural network approach.

4.4.1 Number of Layers

Because the computational performance of the network is directly proportional to the number of layers and nodes, it is desirable to limit these values as much as possible. It has been proposed that in most cases one or two hidden layers is sufficient. The neural network interpretation of Kolmogorov's theorem by Hecht-Nielsen proves that a network with a single hidden layer, whose nodes have transfer functions that are not constrained to be the same, can represent an arbitrary function of the inputs [10]. Unfortunately, the theorem does not say how to choose the transfer functions and has therefore little practical application. Cybenko shows that two hidden layers, whose nodes all use sigmoidal transfer functions, are sufficient to compute an arbitrary function of the inputs, and that a single hidden layer is sufficient for classification problems [10]. According to Maren et al. [10] experimentation has confirmed these results.

4.4.2 Number of Modes Per Layer

The number of nodes in the input layer is fixed by the number of points in the input vector. For the output layer, the number of nodes is set to the number of desired outputs (e.g. classes, categories, probabilities). Any attempt to reduce the number of nodes in the output layer by an encoding of the outputs should be applied with caution, as the additional burden imposed on the network may only prove workable with the inclusion of a second hidden layer.

Choosing the number of nodes in the hidden layers is more difficult. The objective is to keep the number of nodes to a minimum as the addition of extra nodes can result in a network that learns particular cases, yet fails to learn general characteristics, as well as increasing the training and operational complexity. Here again, the decision can be guided by theoretical and empirical limits. According to Hecht-Nielson's interpretation of the Kolmogorov theorem, $2N + 1$ nodes (where N is the number of inputs) are necessary to calculate an arbitrary function. According to Kudrycki, the optimum ratio between the first and the second hidden layer is three to one [10]. For small networks where the number of inputs is greater than the number of outputs, it has been proposed that the geometric average of the inputs and outputs provides a good estimate of the optimum number of nodes to use. There is general agreement however, that the number of nodes in each hidden layer should be fewer than the number of inputs.

4.5 Summary

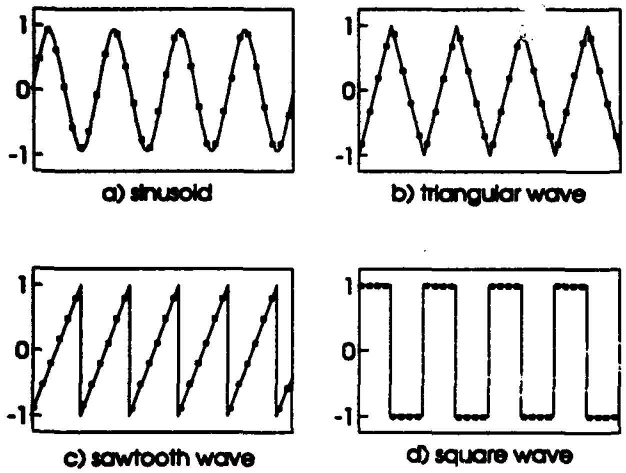
This chapter has dealt with practical considerations in the design of a network. The two main issues are convergence of the error function and network performance. A number of techniques that address these issues were presented, including pre-treatment of the input data, weight initialization and adjustment, the number of vectors in the training set, and the choice of the number of layers and nodes in the network.

5.0 CLASSIFICATION OF RADAR SIGNALS

The advent of complex radar systems has lead to the appearance of frequency and pulse repetition interval (PRI) agile emitters. Emitters can vary these parameters, either randomly or periodically following a well defined modulation pattern. The application of neural networks described in this paper is limited to the common modulation patterns shown in Figure 4. (The vertical and horizontal axes represent the frequency of the pulse and the time of arrival respectively). These include the sinusoid, the triangular wave, the sawtooth wave (including both negative and positive sloping teeth) and the rectangular wave. A signal exhibiting a constant pulse frequency or PRI could also be viewed as having a distinct modulation pattern, but there are already simple algorithms to detect such signals.

Both pulses and modulation patterns are measured in frequency. The frequency of a pulse is typically measured in megahertz or gigahertz, while the frequency of the modulation pattern will be in the order of millihertz or even hertz. The use of the term signal refers to the stream of pulses received by an ESM system.

This chapter describes the results of a number of experiments that were devised to determine the applicability of neural networks to the recognition and classification of such modulation patterns, as well as to assess the performance of different network architectures and the effect of treating the data by transforming it from the time domain to the frequency domain. In each experiment, a network is presented with a number of training vectors, followed by a number of test vectors. Each vector represents a stream of pulses received in one illumination. All network inputs, both training vectors and test vectors, are computer generated synthetic data.



a) sinusoid

b) triangular wave

c) sawtooth wave

d) square wave

Figure 4: Signal Modulation Patterns

5.1 Time Domain Experiments

The goal of a time domain network is to classify an input test vector as one of the following modulation pattern types: sine, triangular, negative sloping saw-tooth, positive sloping saw-tooth or square. Two assumptions are made in generating the data. First, it is assumed that all radar pulses seen in an illumination are equidistant in time, i.e. constant PRI, and second, it is assumed that the signal average is zero, i.e. no DC component is added to the signal generation equations. (For real data, a pre-treatment step to subtract the average value, assuming it is non-zero, is necessary.) It is also assumed that the parameters vary as follows:

Time: The length in time of an illumination is variable. In other words, an illumination may contain a variable number of pulses.

Frequency: The frequency of the modulation pattern can also be viewed as a variation in the number of periods contained in one illumination. The number is assumed to be relatively small, varying between one period and three periods.

Phase: An illumination can start at any point in the cycle of the modulation pattern; the phase shift varying anywhere between 0 and 2π.

The classifier must find shared characteristics in the inputs, subject to the above parameter variations, that permit it to differentiate between different signal patterns.

All of the networks used in the experiments are of the back-propagation type, all have a single hidden layer, and all use the cumulative update training method. The elements of an input vector represent the pulse frequencies in a single illumination. The output vector contains one element for each possible modulation pattern. The element with the highest value is taken to be the classification of the input signal.

Two sets of experiments were carried out. In the first, the number of pulses per illumination is assumed to be no more than 16, and in the second, no more than 32. Where the illumination contains fewer pulses than the maximum, the corresponding vector is padded with zeroes.

Before generating training and test vectors, the number of nodes in the various layers must be decided upon. According to the Baum and Haussler result, in order to obtain an error of less than ε in the test vectors, the number of training vectors must be at least $1/\varepsilon$ times the number of connections in the network. For these experiments ε was chosen to be 0.2. Therefore the number of training vectors must be at least five times the number of connections in the network. To compare the relative merits both in terms of performance and training times, five different networks were built for each set of experiments. In the first network, the number of nodes in the hidden layer was equal to the number of nodes in the output layer, in the second network, the number of nodes in the hidden layer was equal to two times the number of nodes in the output layer, and so on. This was done to test the Kudrycki hypothesis that the ratio of the number of nodes in the hidden layer to the output layer should be three to one.

The notation used in this paper to define the architecture of the various networks is *i-h-o* where *i* indicates the number of nodes in the input layer, *h* the number of nodes in the hidden layer, and *o* the number of nodes in the output layer. The networks used in the first set of experiments are therefore denoted as 16-5-5, 16-10-5, 16-15-5, 16-20-5, and 16-25-5.

The number of weights or connections in a three layer back-propagation network is given by $(i * h) + (h * o)$. For the above networks this yields 105, 210, 315, 420, and 525 connections respectively. Using the Baum-Haussler rule, the number of training vectors to be generated for each network should be at least 525, 1050, 1575, 2100 and 2626 respectively.

In generating the training vectors, there are four parameters of concern: the number of different classifications (or output nodes), the variation in the number of pulses in the illumination (in this case, some number between 1 and 16), the variation in the number of periods in the illumination and the variation in the phase. The basic algorithm for generating the training vectors is as follows:

```
FOR EACH classification type c DO
   FOR x = pulse min TO pulse max BY pulse step DO
      FOR y = period min TO period max BY period step DO
         FOR z = phase min TO phase max BY phase step DO
            Generate a new vector v (c, x, y, z);
            Generate the corresponding output vector w;
            Output v and w;
         END
      END
   END
END
```

To avoid saturation, the entries in the input training and test vectors are computed so as to lie within the range of -1 to +1, and as Freeman [4] suggests, the entries in the output training vector all lie within the range of 0.1 to 0.9.

To give equal weight to each of the variations, the number of possible values taken on by each of *x*, *y* and *z* in the above algorithm should be the same, say *n*. The number of training vectors generated is then equal to Cn^3, where C is the number of classifications, i.e. the number of output nodes. To satisfy the Baum-Haussler result n must be such that Cn^3 is greater than or equal to five times the number of connections in the network. The value of n can therefore be computed by taking the ceiling of the cube root of $5/C$ times the number of connections in the network. Since C is equal to five in this case, n is simply equal to the ceiling of the cube root of the number of connections in the network. Table I shows the minimum number of training vectors to be generated to satisfy Baum-Haussler, and the actual number of training vectors that were generated for all of the time domain network experiments.

The parameters x, y, and z are evenly spread across the desired range of values. For example, if the period of the modulation pattern is to vary between one and three, and five values are to be computed, then the values will be 1.0, 1.5, 2.0, 2.5 and 3.0. If seven values are needed, the values could be 1.002, 1.335, 1.668, 2.001, 2.334, 2.667, and 3.0. Table II shows the number of values desired for each variation (n), and the minimum, maximum and step size required to compute values for x, y, and z. The first five rows show the range of values used to generate the training vectors for the 16 input networks. The sixth row shows the ranges used to generate the test vectors used in testing all of the 16 input networks. The ranges used to generate the training and test vectors for the 32 input networks are shown in the subsequent six rows.

The test vectors are generated using the same algorithm, although care is taken that no test vector is identical to any of the training vectors. This is to ensure that the network is tested against previously unseen vectors.

The second set of experiments was carried out using networks with 32 entries, i.e, 32-5-5, 32-10-5, 32-15-5, 32-20-5, and 32-25-5. Apart from the change in dimension of the input vector, the same algorithm was used to generate the training and test vectors.

The actual experimental process consisted of generating the training and test vectors, and building the corresponding network using the commercial product NeuralWorks Professional II, produced by NeuralWare Inc. Each network was initially presented with 20,000 training vectors randomly chosen from the generated training vector set. (The number 20,000 was determined experimentally to be a convenient interval for assessing the improvement in performance over time.) The performance of the network was then determined by computing the percentage of correctly classified signals represented by the test vectors. Every network used in these experiments has five outputs, each output corresponding to a particular classification. A signal is considered to be correctly classified if the value in its associated output node is greater than any value in the other output nodes. Although this is a weak measure of success, it is sufficient for comparison purposes. (A stronger measure of correctness would insist that not only should the associated output node's value be the greatest, but that it exceed some threshold as well.) In addition to the performance of the network, the length of time required to train the network with the 20,000 vectors is recorded. The training and testing process is repeated for a total of 20 times (400,000 presentations of training vectors). Table III shows the results for the networks with 16 inputs, and Table III, Table IV shows the results for the networks with 32 inputs. Figure 5 to Figure 14 in Appendix A show the results in a graphical format.

Network	Number of Connections (Weights)	Required Number of Training Vectors	Number of Variations (*n*)	Actual Number of Training Vectors
16-5-5	105	525	5	625
16-10-5	210	1050	6	1080
16-15-5	315	1575	7	1715
16-20-5	420	2100	8	2560
16-25-5	525	2625	9	3645
32-5-5	185	925	6	1080
32-10-5	370	1850	8	2560
32-15-5	555	2775	9	3645
32-20-5	740	3700	10	5000
32-25-5	925	4625	10	5000

Table I: Time Domain: Number of Training Vectors

Net.	*n*	Pulse min	Pulse max	Pulse step	Period min	Period max	Period step	Phase min	Phase max	Phase step
16-5-5	5	12.0	16.0	1.0	1.0	3.0	0.5	0.0	0.996	0.249
16-10-5	6	11.0	16.0	1.0	1.0	3.0	0.4	0.0	0.995	0.199
16-15-5	7	10.0	16.0	1.0	1.002	3.0	0.333	0.0	0.996	0.166
16-20-5	8	9.0	16.0	1.0	1.005	3.0	0.285	0.0	0.994	0.142
16-25-5	9	8.0	16.0	1.0	1.0	3.0	0.25	0.0	0.992	0.124
Test 16	4	9.0	15.0	2.0	1.001	2.999	0.666	0.0	0.999	0.333
32-5-5	6	27.0	32.0	1.0	1.0	3.0	0.4	0.0	0.995	0.199
32-10-5	8	25.0	32.0	1.0	1.005	3.0	0.285	0.0	0.994	0.142
32-15-5	9	24.0	32.0	1.0	1.0	3.0	0.25	0.0	0.992	0.124
32-20-5	10	23.0	32.0	1.0	1.002	3.0	0.222	0.0	0.999	0.111
32-25-5	10	23.0	32.0	1.0	1.002	3.0	0.222	0.0	0.999	0.111
Test 32	4	25.0	31.0	2.0	1.001	2.999	0.666	0.0	0.999	0.333

Table II: Time Domain: Pulse Count, Period and Phase Settings

T	16-5-5		16-10-5		16-15-5		16-20-5		16-25-5	
Iteration	% Correct	Time	% Correct	Time	% Correct	Time	% Correct	Time	% Correct	Time
0	17.812	0:00:00	20.000	0:00:00	21.250	0:00:00	20.000	0:00:00	20.000	0:00:00
20000	37.812	0:01:23	37.188	0:01:45	24.688	0:04:31	20.000	0:05:33	20.000	0:06:36
40000	44.375	0:02:46	61.875	0:03:30	42.500	0:09:02	32.500	0:11:06	20.000	0:13:12
60000	54.688	0:04:09	73.125	0:05:15	64.062	0:13:33	47.500	0:16:39	26.875	0:19:48
80000	55.938	0:05:32	73.750	0:07:00	72.812	0:18:04	52.188	0:22:12	47.812	0:26:24
100000	55.938	0:06:55	74.688	0:08:45	85.312	0:22:35	69.062	0:27:45	44.062	0:33:00
120000	57.500	0:08:18	76.250	0:10:30	84.375	0:27:06	80.000	0:33:18	55.625	0:39:36
140000	61.875	0:09:41	76.250	0:12:15	89.375	0:31:37	90.000	0:38:51	64.688	0:46:12
160000	57.812	0:11:04	73.750	0:14:00	90.000	0:36:08	92.188	0:44:24	82.500	0:52:48
180000	59.375	0:12:27	74.688	0:15:45	91.562	0:40:39	94.375	0:49:57	86.562	0:59:24
200000	60.312	0:13:50	75.625	0:17:30	91.562	0:45:10	95.000	0:55:30	92.500	1:06:00
220000	64.375	0:15:13	74.375	0:19:15	89.375	0:49:41	95.312	1:01:03	89.062	1:12:36
240000	62.188	0:16:36	74.062	0:21:00	90.938	0:54:12	96.250	1:06:36	93.125	1:19:12
260000	56.562	0:17:59	75.312	0:22:45	91.250	0:58:43	95.938	1:12:09	94.688	1:25:48
280000	49.375	0:19:22	74.688	0:24:30	92.812	1:03:14	95.625	1:17:42	95 .000	1:32:24
300000	58.750	0:20:45	75.000	0:26:15	92.188	1:07:45	96.250	1:23:15	95.312	1:39:00
320000	63.750	0:22:08	74.688	0:28:00	92.812	1:12:16	96.562	1:28:48	95.625	1:45:36
340000	55.625	0:23:31	74.688	0:29:45	94.062	1:16:47	96.250	1:34:21	96.875	1:52:12
360000	63.438	0:24:54	75.312	0:31:30	94.062	1:21:18	96.250	1:39:54	96.875	1:58:48
380000	62.188	0:26:17	75.312	0:33:15	93.750	1:25:49	96.250	1:45:27	96.562	2:05:24
400000	62.812	0:27:40	74.375	0:35:00	94.688	1:30:20	96.250	1:51:00	96.875	2:12:00

Table III: Time Domain: Network Results 16 Input Nodes

T	32-5-5		32-10-5		32-15-5		32-20-5		32-25-5	
Iteration	% Correct	Time	% Correct	Time	% Correct	Time	% Correct	Time	% Correct	Time
0	20.000	0:00:00	19.375	0:00:00	20.000	0:00:00	18.75	0:00:00	20.000	0:00:00
20000	20.000	0:04:40	30.938	0:06:06	30.938	0:06:43	20.000	0:07:14	20.000	0:07:45
40000	20.000	0:09:20	45.938	0:12:12	40.000	0:13:26	20.000	0:14:28	20.000	0:15:30
60000	20.000	0:14:00	72.812	0:18:18	54.375	0:20:09	27.500	0:21:42	20.000	0:23:15
80000	27.812	0:18:40	76.562	0:24:24	58.125	0:26:52	43.750	0:28:56	39.062	0:31:00
100000	37.188	0:23:20	82.500	0:30:30	75.312	0:33:35	50.938	0:36:10	65.938	0:38:45
120000	48.750	0:28:00	85.312	0:36:36	83.75	0:40:18	76.875	0:43:24	69.688	0:46:30
140000	51.562	0:32:40	87.812	0:42:42	91.875	0:47:01	87.812	0:50:38	87.500	0:54:15
160000	53.125	0:37:20	87.188	0:48:48	94.375	0:53:44	89.688	0:57:52	95.000	1:02:00
180000	51.875	0:42:00	86.875	0:54:54	95.000	1:00:27	90.625	1:05:06	97.500	1:09:45
200000	52.500	0:46:40	85.000	1:01:00	95.312	1:07:10	93.125	1:12:20	97.812	1:17:30
220000	52.188	0:51:20	87.812	1:07:06	95.938	1:13:53	95.938	1:19:34	99.062	1:25:15
240000	53.438	0:56:00	87.188	1:13:12	95.625	1:20:36	97.812	1:26:48	100.000	1:33:00
260000	57.188	1:00:40	83.750	1:19:18	98.125	1:27:19	99.375	1:34:12	100.000	1:40:45
280000	58.125	1:05:20	86.875	1:25:24	98.750	1:34:02	97.812	1:41:16	99.688	1:48:30
300000	57.188	1:10:00	88.438	1:31:30	98.750	1:40:45	99.375	1:48:30	100.000	1:56:15
320000	60.625	1:14:40	87.812	1:37:36	99.062	1:47:28	98.750	1:55:44	100.000	2:04:00
340000	58.750	1:19:20	87.188	1:43:42	99.062	1:54:11	99.062	2:02:58	100.000	2:11:45
360000	62.812	1:24:00	85.000	1:49:48	98.750	2:00:54	99.688	2:10:12	99.688	2:19:30
380000	66.875	1:28:40	88.125	1:55:54	98.750	2:07:37	99.375	2:17:26	100.00	2:27:15
400000	65.938	1:33:20	86.562	2:02:00	99.375	2:14:20	99.688	2:24:40	100.000	2:35:00

Table IV: Time Domain: Network Results 32 Input Nodes

5.2 The Frequency Domain

One of the goals of this project is to analyze the performance of neural networks using data that has been transformed from the time domain to the frequency domain. Transforming data in this way, most commonly using a Fourier Transform, is a common technique for analyzing signals in digital signal processing. The Fourier transform is typically applied to an amplitude versus time representation of a signal. This project, however, deals with the modulation of pulse frequencies over time. A transformation into the *frequency* domain in this case yields a pulse frequency versus modulation frequency representation of the signal.

Note that in a fielded application the additional cost of pre-treating data must be weighed against any possible benefits.

5.2.1 The Fourier Transform

The Fast Fourier Transform is an algorithm that quickly computes the Fourier transform of a signal. To use this algorithm, it is necessary that the number of points in the signal be a power of two. However, the number of points (pulses) is variable and padding the signal, (i.e. the input vector) with zero entries causes a problem: the discontinuity introduced with the addition of the zero entries can significantly alter the behaviour of the amplitude derived by the Fourier transform (the amplitude of the signal being the pulse frequency in our application). To remove this discontinuity, the signal values (pulse frequencies) are multiplied by a Blackman [11] window prior to the addition of the zero values. This has the property of creating a function whose values begin at zero, gradually rises to a maximum of one at the centre of the window and then gradually descend towards zero:

$$w[n] = \begin{cases} 0.42 - 0.5\cos(2\pi n/M) + 0.08\cos(4\pi n/M) & 0 \le n \le M \\ 0.0 & \textit{otherwise} \end{cases}$$

The new signal therefore has a value close to zero before the zero data points are added.

It is also important to note that with the Fourier transformation, the two variations of the sawtooth wave pattern will result in the identical function in the frequency domain (the transformation of the negative and positive wave patterns differing by a phase shift of 180 degrees). The total number of distinct signal classifications is therefore reduced to four. The distinction between the two sawtooth patterns can be easily made after the neural network has produced its output.

5.2.2 Frequency Domain Experiments

As for the time domain experiments, separate sets of training and test vectors were generated to serve as input to a new set of neural networks. For these experiments, five networks with 16 input nodes were created ranging from four hidden nodes to twenty: 16-4-4, 16-8-4, 16-12-4, 16-16-4, and 16-20-4, and five networks with 32 inputs: 32-4-4, 32-8-4, 32-12-4, 32-16-4, and 32-20-4. The basic algorithm to generate the input vectors is the same. Again, the parameters that can vary are the classification of the signal, the number of pulses in the signal, the number of periods in the illumination and the phase of the signal:

```
FOR EACH classification type c DO
   FOR x = pulse min TO pulse max BY pulse step DO
      FOR y = period min TO period max BY period step DO
         FOR phase min TO phase max BY phase step DO
            Generate a new vector v (c, x, y, z);
            Apply Blackman window to v;
            Apply Fourier Transform to v;
            Generate the corresponding output vector w;
            Output v and w;
         END
      END
   END
END
```

Table V lists the number of training vectors to be generated for each network. Table VI shows the ranges and step sizes used to specify the *x*, *y* and *z* values of the algorithm. Table VII shows the results for the networks with 16 inputs, and Table VIII shows the results for the networks with 32 inputs. Figure 15 to Figure 24 in Appendix A show the results in a graphical format.

Network	Number of Connections (Weights)	Required Number of Training Vectors	Number of Variations (*n*)	Actual Number of Training Vectors
16-4-4	80	400	5	500
16-8-4	160	800	6	864
16-12-4	240	1200	7	1372
16-16-4	320	1600	8	2048
16-20-4	400	2000	8	2048
32-4-4	144	720	6	864
32-8-4	288	1440	8	2048
32-12-4	432	2160	9	2916
32-16-4	576	2880	9	2916
32-20-4	720	3600	10	4000

Table V: Frequency Domain: Number of Training Vectors

Net	*n*	Pulse min	Pulse max	Pulse step	Period min	Period max	Period step	Phase min	Phase max	Phase step
16-4-4	5	12.0	16.0	1.0	1.0	3.0	0.5	0.0	0.996	0.249
16-8-4	6	11.0	16.0	1.0	1.0	3.0	0.4	0.0	0.995	0.199
16-12-4	7	10.0	16.0	1.0	1.002	3.0	0.333	0.0	0.996	0.166
16-16-4	8	9.0	16.0	1.0	1.005	3.0	0.285	0.0	0.994	0.142
16-20-4	8	9.0	16.0	1.0	1.005	3.0	0.285	0.0	0.994	0.142
test16	4	9.0	15.0	2.0	1.001	3.0	0.666	0.0	0.999	0.333
32-4-4	6	27.0	32.0	1.0	1.0	3.0	0.4	0.0	0.995	0.199
32-8-4	8	25.0	32.0	1.0	1.005	3.0	0.285	0.0	0.994	0.142
32-12-4	9	24.0	32.0	1.0	1.0	3.0	0.25	0.0	0.992	0.124
32-16-4	9	24.0	32.0	1.0	1.0	3.0	0.25	0.0	0.992	0.124
32-20-4	10	23.0	32.0	1.0	1.002	3.0	0.222	0.0	0.999	0.111
test32	4	25.0	31.0	2.0	1.001	2.999	0.666	0.0	0.999	0.333

Table VI: Frequency Domain: Pulse Count, Period and Phase Settings

F	16-4-4		16-8-4		16-12-4		16-16-4		16-20-4	
Iteration	% Correct	Time	% Correct	Time	% Correct	Time	% Correct	Time	% Correct	Time
0	25.000	0:00:00	25.000	0:00:00	25.000	0:00:00	25.000	0:00:00	25.000	0:00:00
20000	55.859	0:01:13	53.125	0:01:34	53.906	0:02:17	43.359	0:05:07	51.562	0:06:09
40000	61.719	0:02:26	55.469	0:03:08	60.547	0:04:34	54.297	0:10:14	54.688	0:12:18
60000	57.031	0:03:39	62.500	0:04:42	63.672	0:06:51	60.547	0:15:21	54.688	0:18:27
80000	57.031	0:04:52	63.672	0:06:16	62.500	0:09:08	60.938	0:20:28	61.328	0:24:36
100000	60.547	0:06:05	62.891	0:07:50	55.859	0:11:25	64.062	0:25:35	65.625	0:30:45
120000	59.375	0:07:18	64.062	0:09:24	62.891	0:13:42	61.328	0:30:42	66.797	0:36:54
140000	60.156	0:08:31	69.531	0:10:58	62.109	0:15:59	58.594	0:35:49	62.891	0:43:03
160000	64.062	0:09:44	66.797	0:12:32	62.109	0:18:16	64.844	0:40:56	66.016	0:49:12
180000	63.281	0:10:57	69.922	0:14:06	65.625	0:20:33	60.938	0:46:03	56.641	0:55:21
200000	63.281	0:12:10	63.672	0:15:40	66.016	0:22:50	61.719	0:51:10	63.281	1:01:30
220000	61.719	0:13:23	66.016	0:17:14	68.359	0:25:07	69.922	0:56:17	65.234	1:07:39
240000	67.188	0:14:36	71.484	0:18:48	71.094	0:27:24	57.422	1:01:24	62.500	1:13:48
260000	64.844	0:15:49	68.750	0:20:22	63.281	0:29:41	66.406	1:06:31	67.188	1:19:57
280000	64.453	0:17:02	67.969	0:21:56	68.359	0:31:58	61.719	1:11:38	66.016	1:26:06
300000	60.156	0:18:15	71.094	0:23:30	70.703	0:34:15	64.453	1:16:45	67.188	1:32:15
320000	66.406	0:19:28	71.875	0:25:04	73.438	0:36:32	69.531	1:21:52	58.594	1:38:24
340000	64.453	0:20:41	72.656	0:26:38	71.094	0:38:49	69.141	1:26:59	67.188	1:44:33
360000	65.625	0:21:54	76.172	0:28:12	67.188	0:41:06	62.109	1:32:06	65.625	1:50:42
380000	66.016	0:23:07	66.406	0:29:46	74.219	0:43:23	71.484	1:37:13	67.188	1:56:51
400000	66.016	0:24:20	76.953	0:31:20	71.094	0:45:40	60.938	1:42:20	73.047	2:03:00

Table VII: Frequency Domain: Network Results 16 Input Nodes

F	32-4-4		32-8-4		32-12-4		32-16-4		32-20-4	
Iteration	% Correct	Time	% Correct	Time	% Correct	Time	% Correct	Time	% Correct	Time
0	25.000	0:00:00	25.000	0:00:00	25.000	0:00:00	25.000	0:00:00	25.000	0:00:00
20000	50.781	0:01:42	39.844	0:05:31	44.531	0:06:20	66.016	0:06:25	66.016	0:07:10
40000	67.969	0:03:24	57.422	0:11:02	67.969	0:12:40	60.156	0:12:50	66.797	0:14:20
60000	64.453	0:05:06	57.422	0:16:33	63.281	0:19:00	66.797	0:19:15	71.875	0:21:30
80000	79.688	0:06:48	64.453	0:22:04	59.766	0:25:20	73.047	0:25:40	59.766	0:28:40
100000	72.656	0:08:30	70.703	0:27:35	71.484	0:31:40	80.469	0:32:05	71.484	0:35:50
120000	66.406	0:10:12	75.000	0:33:06	80.469	0:38:00	81.641	0:38:30	72.656	0:43:00
140000	73.047	0:11:54	69.141	0:38:37	77.344	0:44:20	80.469	0:44:55	67.578	0:50:10
160000	78.906	0:13:36	75.781	0:44:08	87.891	0:50:40	92.578	0:51:20	80.078	0:57:20
180000	75.391	0:15:18	92.969	0:49:39	92.578	0:57:00	92.969	0:57:45	91.797	1:04:30
200000	75.000	0:17:00	86.328	0:55:10	89.844	1:03:20	83.984	1:04:10	94.141	1:11:40
220000	67.969	0:18:42	92.188	1:00:41	96.484	1:09:40	92.188	1:10:35	92.188	1:18:50
240000	86.328	0:20:24	89.453	1:06:12	93.359	1:16:00	92.969	1:17:00	86.328	1:26:00
260000	77.344	0:22:06	90.234	1:11:43	93.359	1:22:20	89.844	1:23:25	97.266	1:33:10
280000	68.359	0:23:48	95.703	1:17:14	93.359	1:28:40	94.922	1:29:50	93.750	1:40:20
300000	83.203	0:25:30	94.922	1:22:45	91.016	1:35:00	96.875	1:36:15	93.359	1:47:30
320000	85.547	0:27:12	93.359	1:28:16	97.266	1:41:20	94.922	1:42:40	93.359	1:54:40
340000	91.406	0:28:54	98.047	1:33:47	91.406	1:47:40	94.922	1:49:05	95.312	2:01:50
360000	89.844	0:30:36	98.047	1:39:18	96.094	1:54:00	94.531	1:55:30	98.047	2:09:00
380000	86.719	0:32:18	88.281	1:44:49	96.875	2:00:20	97.656	2:01:55	98.438	2:16:10
400000	90.234	0:34:00	96.484	1:50:20	96.875	2:06:40	89.453	2:08:20	98.828	2:23:20

Table VIII: Frequency Domain: Network Results 32 Input Nodes

5.3 Observations

The experiments show that either type of network can be used successfully to classify radar signals. However, the acceptability of a network's performance depends on the application and is usually defined as the ability of the neural network to achieve some threshold of success over a set of test vectors. It is important to note that in general, neural networks cannot guarantee a one hundred percent success rate for all possible inputs. (One of the networks in the experiments (32-25-5) did achieve a success rate of one hundred, but this was on a limited set of test vectors).

For example, if a 95 percent success rate is considered to acceptable, then the time domain networks: 16-20-5, 16-25-5, 32-15-5, 32-20-5, and 32-25-5 all pass this criterion based on the set of test vectors presented to them. In the frequency domain, the networks: 32-8-4, 32-12-4, 32-16-4, and 32-20-4 meet this criterion. However, if the application calls for a 99 percent success rate than only the time domain networks 32-15-5, 32-20-5, and 32-25-5 can satisfy this requirement. None of the frequency domain networks meet this standard.

The experimental results concur with Kudrycki's result that the number of nodes in the hidden layer must be at least three times the number of nodes in the output layer to obtain adequate performance. Apart from the frequency domain network 32-8-4, the highest rate of success, for any of the networks having equal or double the number of nodes in the hidden layer as in the output layer, was 91.406 percent. For the networks with only 16 input nodes, the highest rating was 76.953 percent. Equal or better performance is obtained from networks whose hidden layer had four or five times the number of nodes in the output layer. This greater degree of success, however, was achieved at the cost of additional training time. It should also be noted that these larger networks cannot execute as fast as the networks with fewer connections, (unless parallel processing techniques are employed) .

The training time was found to be proportional to the number of connections in the network and in all cases fairly lengthy. The times shown in Tables III, IV, VII, and VIII indicate only the training time, and yet this alone adds up to more than 30 hours. This total does not include the time required to generate the training and test vectors, the time to build the networks, the time to run the tests, nor the time to record the results. The entire process actually required more than two person-weeks. However, to achieve the same results using traditional programming methods would require many times this effort. Of course, if one decides in advance of the training process that a certain percent success rate is acceptable, then the training time can be reduced by stopping the training process when this level of success has been met. One still needs an upper limit, as the success rate may never reach this level. In the experiments performed in this project, the number of iterations of training (400,000) was chosen arbitrarily on the assumption that if the training error function had failed to converge after this amount of training, then it was unlikely to ever do so. The reader will note that even after 400,000 iterations of training, 8 of the 20 networks had failed to reach the 95 percent threshold.

In comparing the general performance of the two types of networks it is clear that the time domain networks were the better performers. Taking a 95 percent success rate as the acceptable performance criterion, two of the 16 input time domain networks 16-20-5 and 16-25-5 were able to achieve this rate, whereas the highest success rate of all of the 16 input frequency domain networks was only 76.953. To achieve this level of success in the frequency domain requires a 32 input network. This is significant because it is generally easier to generate data for networks with smaller input vectors, and networks with fewer connections result in faster execution times. The actual time required to obtain the 95 percent rate for the time domain network 16-20-5 was 55 minutes and 30 seconds, whereas one hour, 17 minutes and 14 seconds was required to achieve this rating for the frequency domain network 32-8-4. Note also that in the frequency domain case, additional time is required to perform the required transformations in generating the training and test vectors. The poorer performance of the frequency domain networks could be a result of the Blackman window transformation. The distortion of the input data caused by this transformation could increase the similarity between two modulation pattern types thereby decreasing the network's ability to distinguish between the two different patterns.

6.0 CONCLUSIONS

This paper has described the fundamentals of neural networks: their construction, training, and behaviour. Particular attention has been payed to the class of neural networks called *back-propagation* networks due to the simplicity of their training algorithm and their effectiveness in solving classification type problems. Theoretical issues involving the architecture, training algorithm, the execution algorithm, and several methods for accelerating the training process of this class of neural networks were covered. Practical consideration were dealt with as well. These include the pre-treatment of input data, initialization of weights, the choice in the number of training vectors, the number of layers and the number of nodes in each layer.

This background material was then followed by a description of a series of experiments which investigated the applicability of the neural network technique in classifying the modulation patterns of pulse-to-pulse frequency agile emitters. The experiments showed that it is possible to construct a neural network to perform this classification task. They also showed that there are no clear-cut rules to follow in the construction of a network. It is somewhat of a black art — one must rely on rules of thumb and "tweaking" to produce a successful network.

As well, the experiments showed that several networks with different architectures could be trained to perform the same task successfully. In designing a particular neural network for a specific application, one is generally forced to make a trade-off between network performance, training time, and execution time. To find the optimal network, the designer must experiment with different architectures and different representations of the input data. This can be a time consuming task, involving the construction, training and evaluation of multiple networks. These tasks are relatively simple ones, certainly much simpler than implementing a classification algorithm in software. The problem of classifying the five signal patterns used in this project, a problem easily solved by a human, is actually very difficult to solve using traditional software techniques, particularly when the inputs are allowed to vary as much as they do in this project (variable number of inputs and periods with arbitrary phase).

Finally, neural networks are a promising technology for radar signal classification problems. However, more experimentation is required. For example, to test performance when there is a greater variability in the possible signal data. This project also made restrictive assumptions concerning the nature of the signal data, eg., fixed PRI, a minimum of eight pulses, and at least one full period of the modulation pattern per illumination. While it is true that any technique will have difficulty classifying a signal with less than eight pulses and only a fractional portion of one modulation pattern, an effective classifier must be able to handle signals exhibiting PRI agility.

7.0 REFERENCES

[1] J.A. Anderson, *Radar Signal Categorization using a Neural Network*, Department of Cognitive and Linguistic Sciences and Department of Psychology, Brown University, Providence, RI, (September 1988).

[2] I.L. Coat & P.C.J. Hill, *The Application of Neural Networks to Radar Classification*, IEE Electronic Division Colloquium on Electronic Warfare Systems, (January 1991).

[3] A. Brown, *Neural Networks in Naval Electronic Warfare Classification Problems*, Carleton University, Ottawa (1990).

[4] J.A. Freeman & D.M. Skapura, *Neural Networks, Algorithms, Applications, and Programming Techniques*, Addison-Wesley Publishing Company Inc., Reading, MA (1991).

[5] D. Fukunaga, *Introduction to Statistical Pattern Recognition*, Academic Press, New York, NY (1972).

[6] G. Hinton, *Neural Networks for Industry*, Department of Computer Science, University of Toronto, Toronto (December 1991).

[7] V. Klimasauskas, J. Guiver & F. Pelton, *NeuralWorks Porfessional II: User's Guide*, NeuralWare, Inc. Pittsburgh, PA (1989).

[8] R.P. Lipmann, *An Introduction to Computing with Neural Nets*, IEEE ASSP Magazine, (April 1987).

[9] R.P. Lipmann, *Pattern Classification Using Neural Networks*, IEEE Communications Magazine, (November 1989).

[10] A. Maren, C. Harston & R. Pap, *Handbook of Neural Computing Applications*, Academic Press Inc., San Diego, CA (1990).

[11] A.V. Oppenheim & R.W. Schafer, *Discrete-Time Signal Processing*, Prentice-Hall, Inc., NJ (1989).

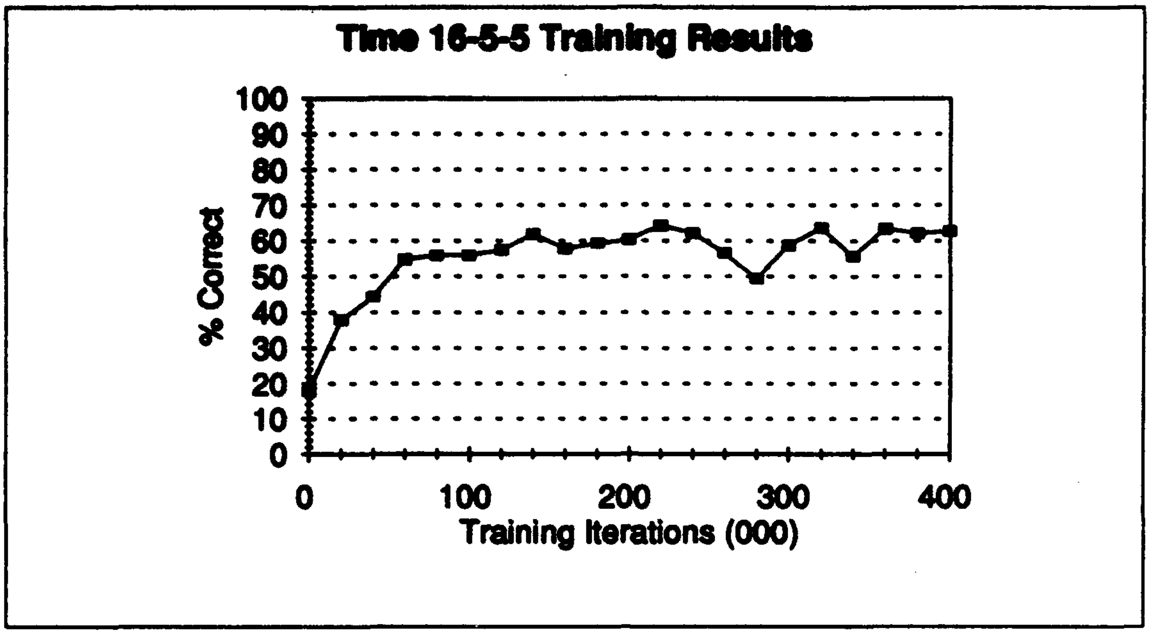
[12] W.H. Press, B.P. Flannery, S.A. Teudolsky & W.T. Vetterling, *Numerical Recipes in C, The Art of Scientific Computing*, Cambridge University Press, Cambridge, MA (1988).

[13] D.E. Rumelhart, J.L. McClelland and the PDP Research Group, *Parallel Distributed Processing, Exploration in the Microstructure of Cognition*, The MIT Press, Cambridge,

MA (1986).

[14] G.B. Willson, *Radar Classification using a Neural Network*, SPIE Vol 1294 Applications of Artificial Neural Networks, (1990).

Appendix A: Figures of Experimental Results



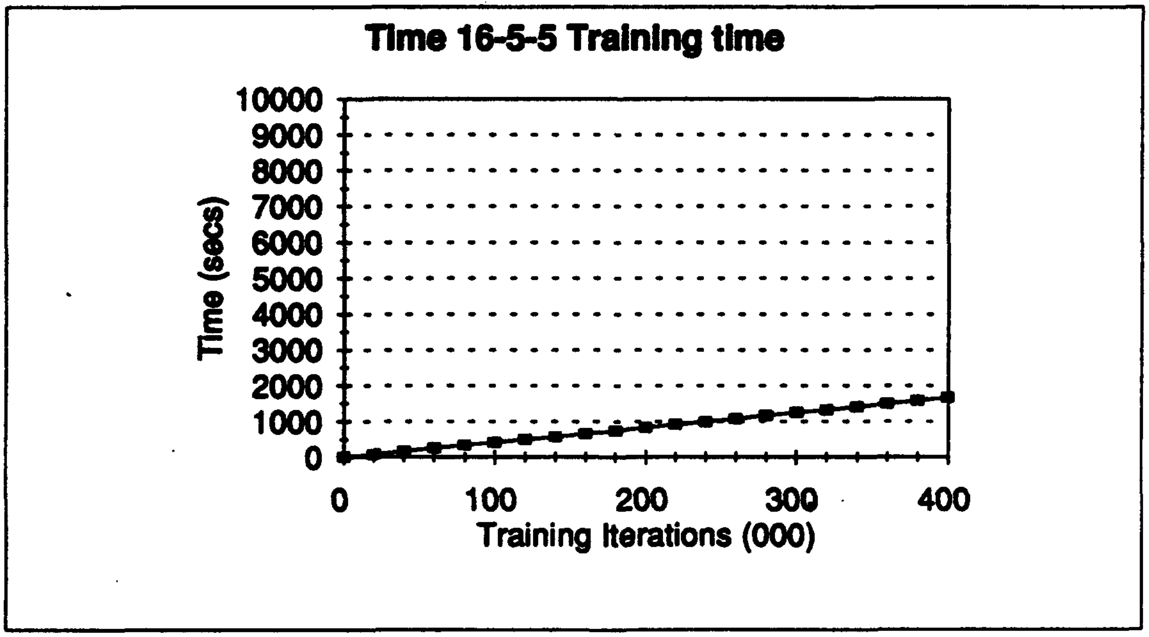
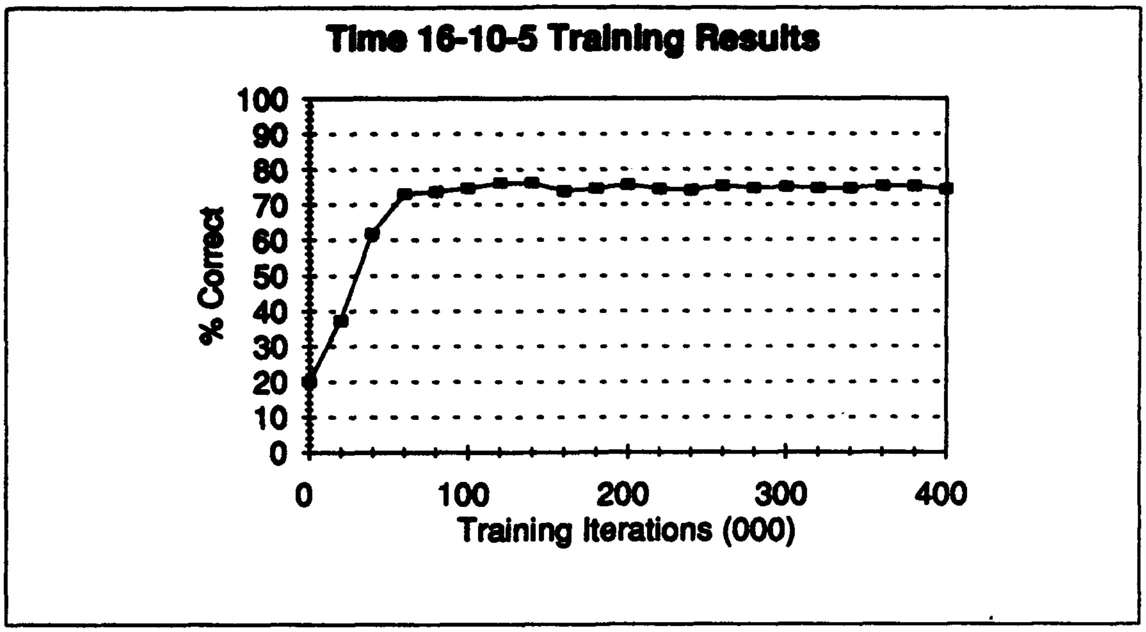


Figure 5: 16-5-5 Results



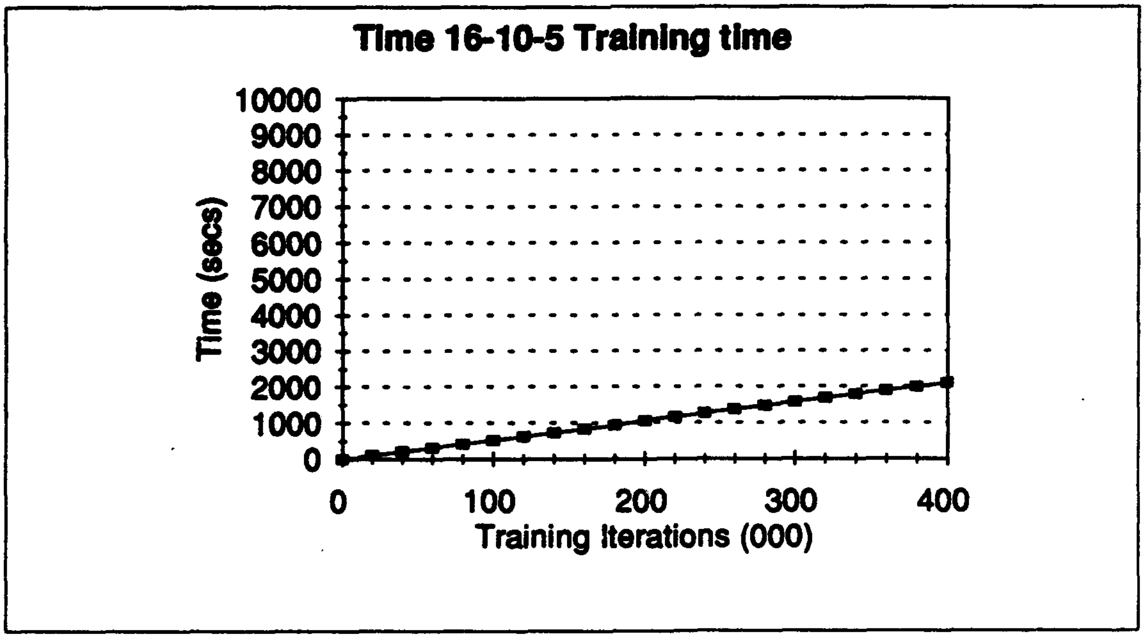
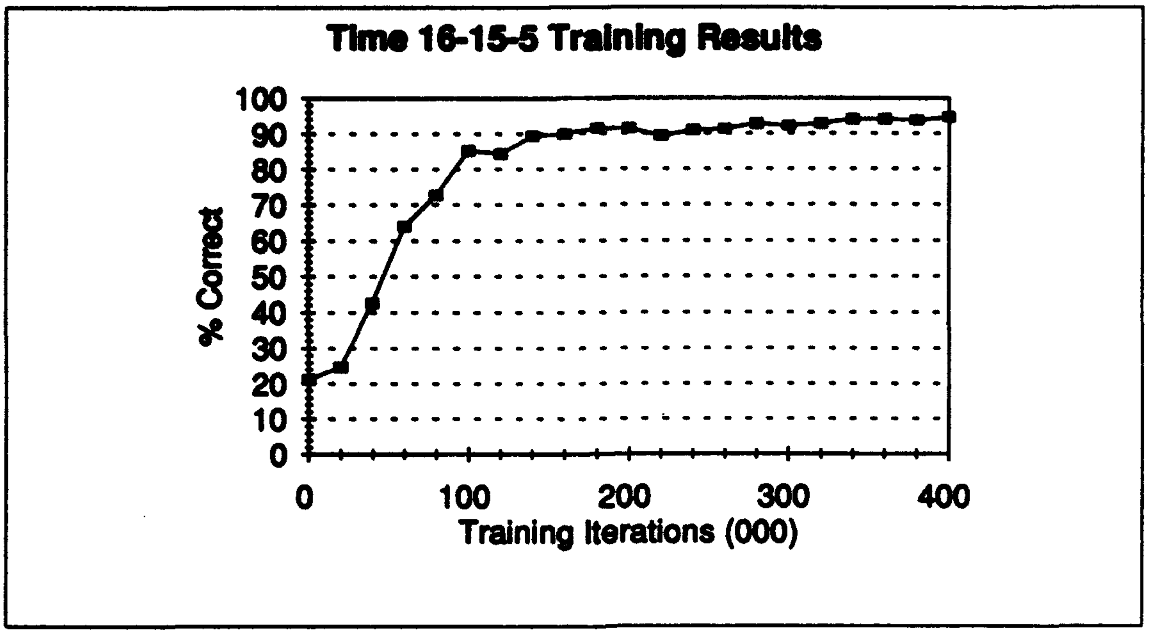


Figure 6: 16-10-5 Results



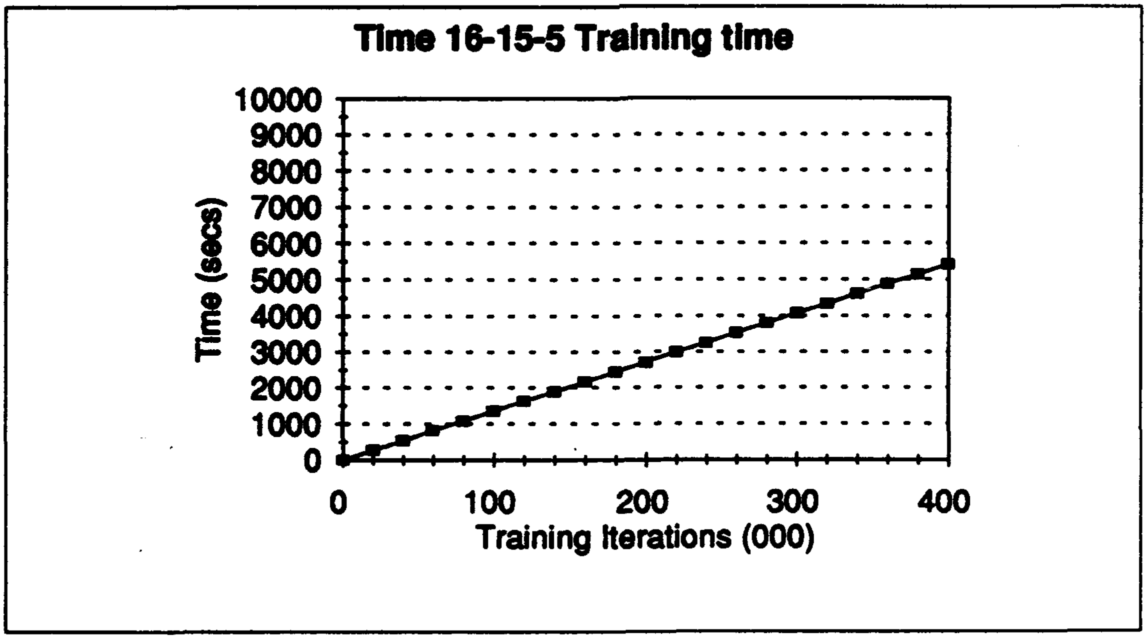
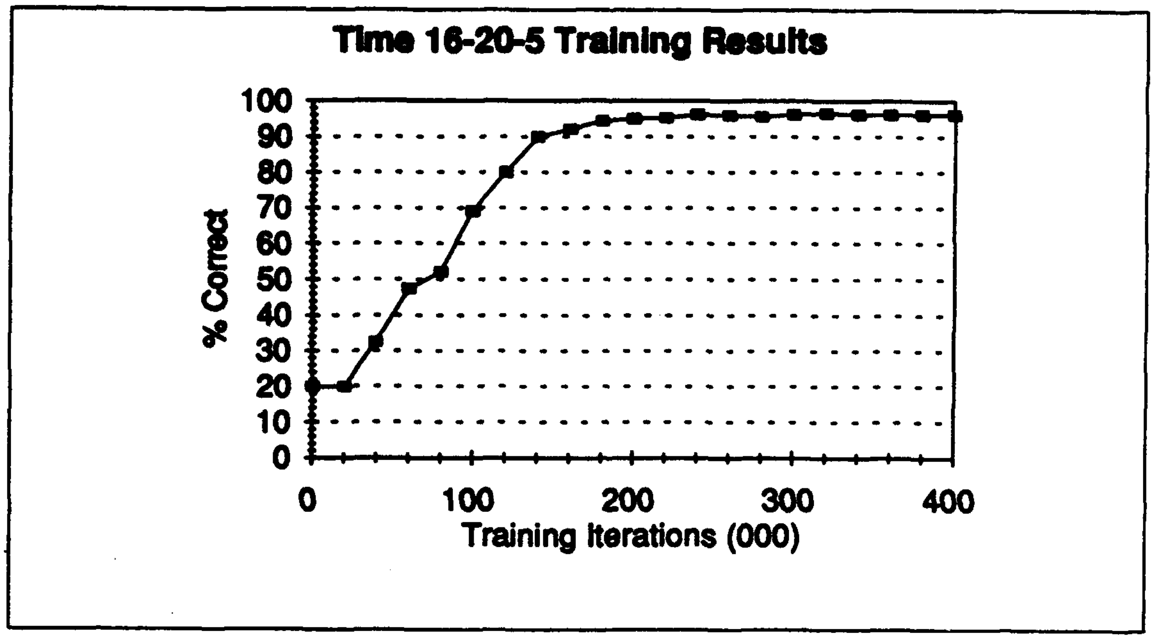


Figure 7: 16-15-5 Results



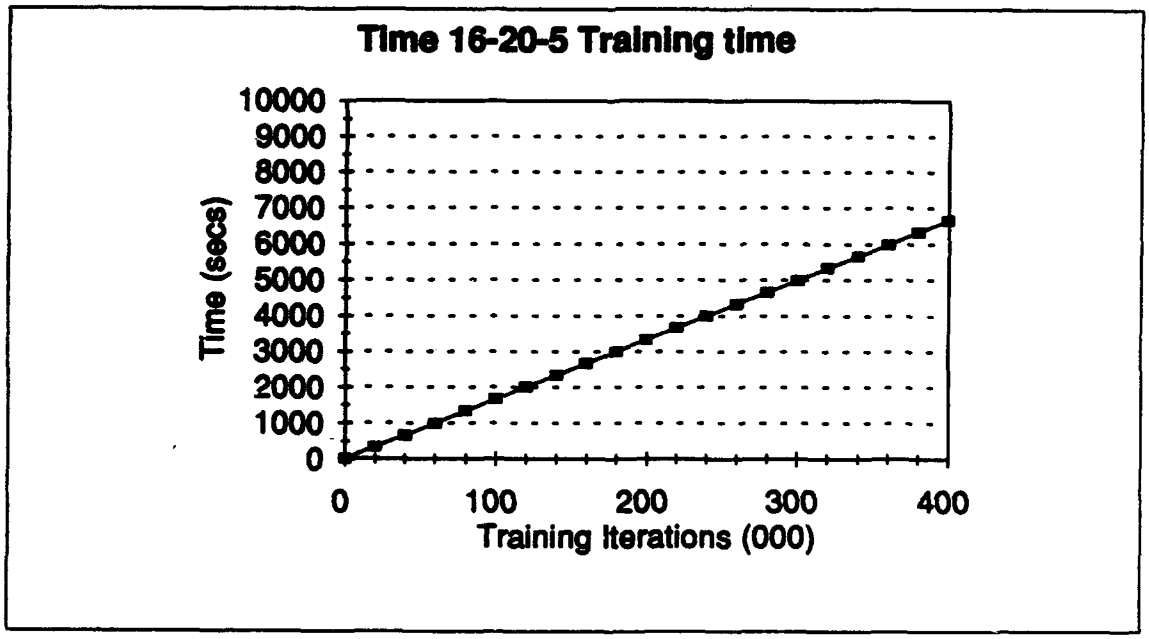


Figure 8: 16-20-5 Results

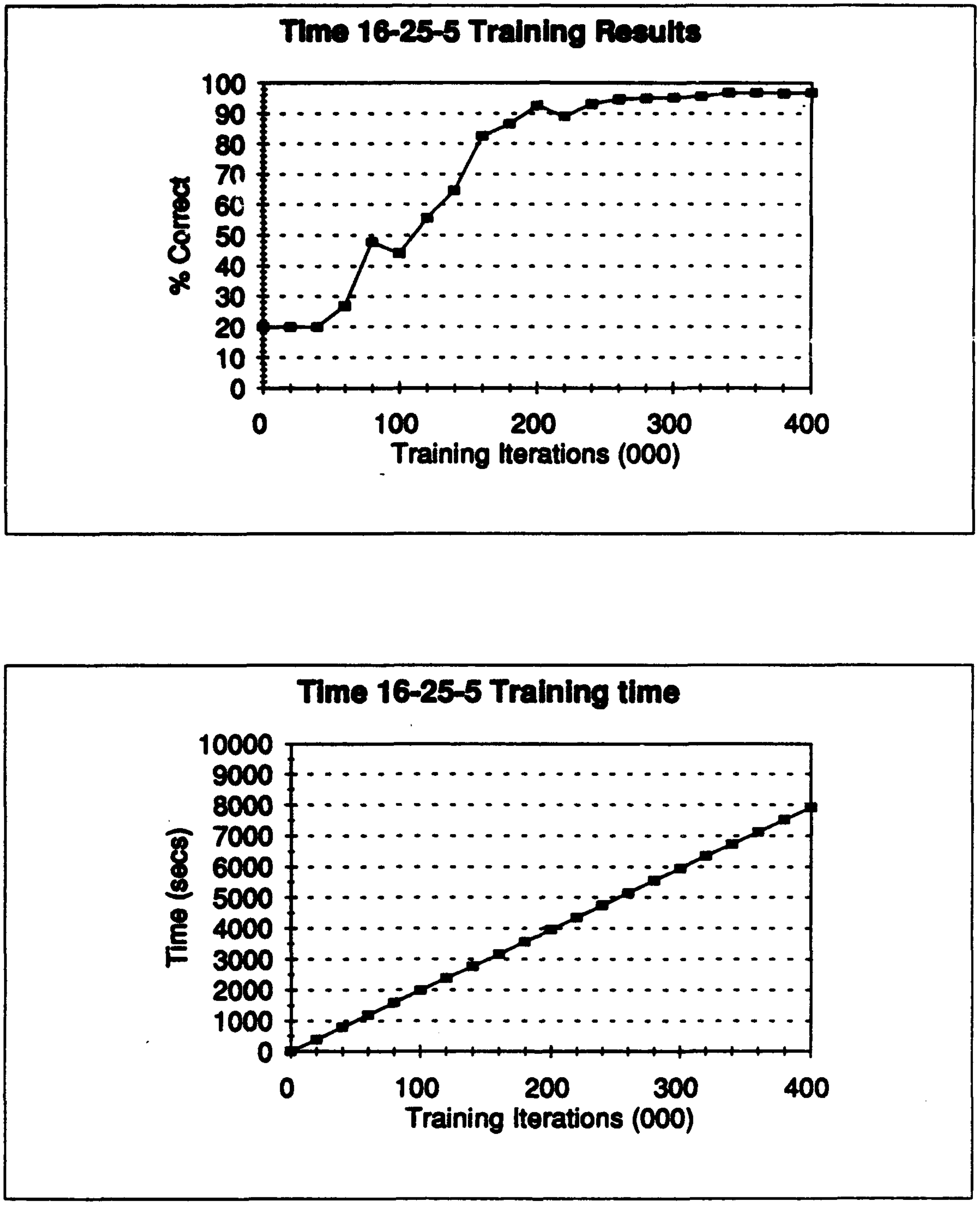
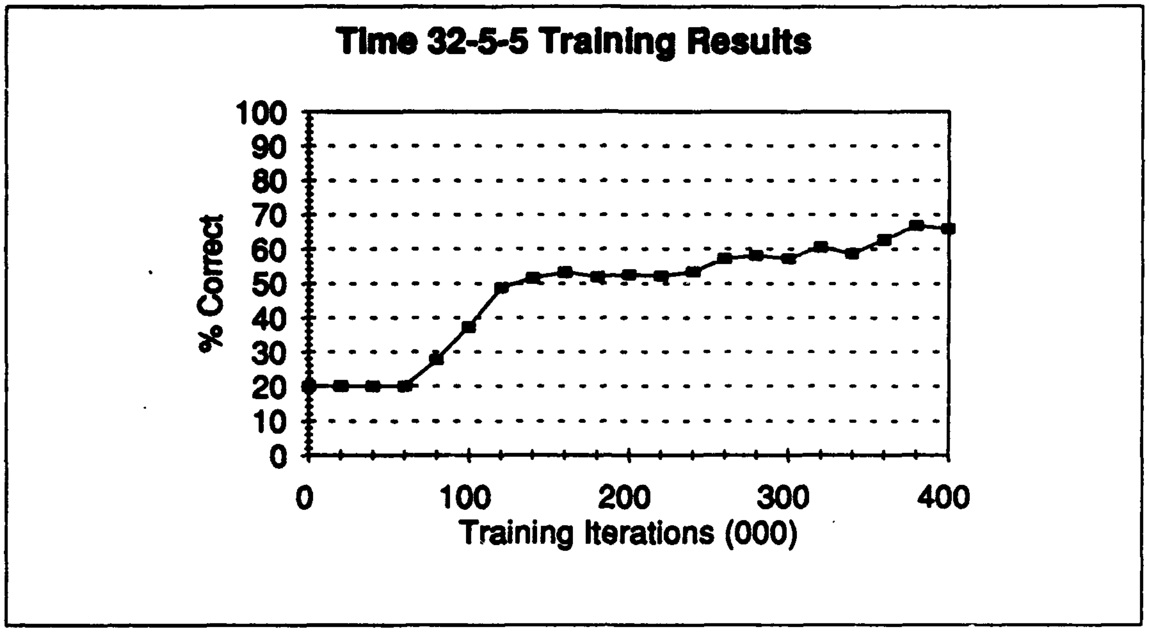


Figure 9: 16-25-5 Results



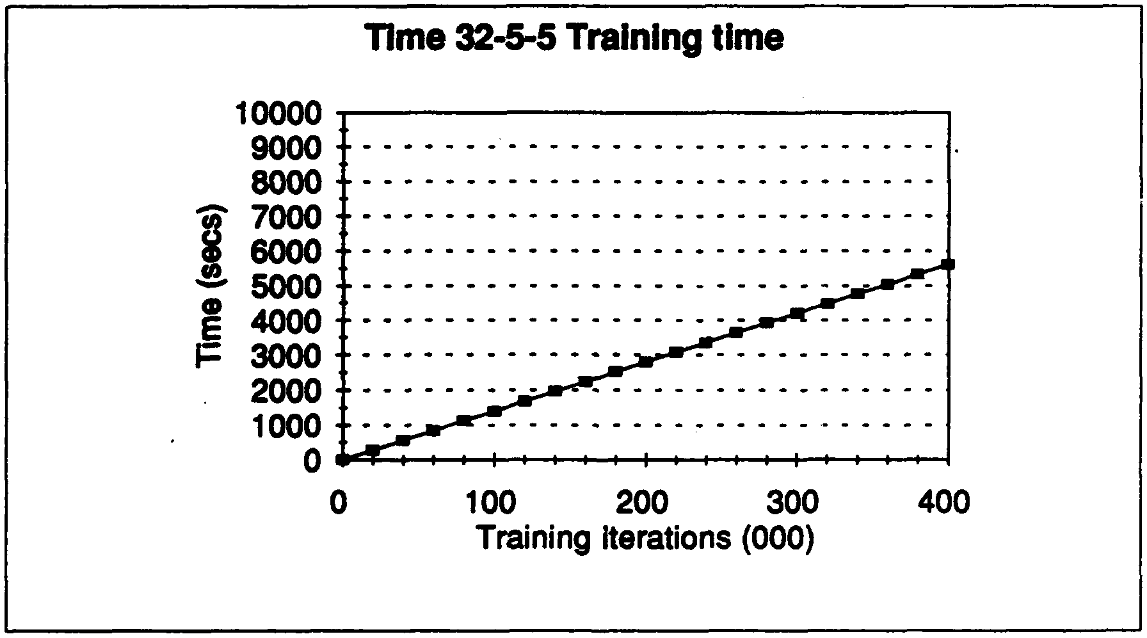
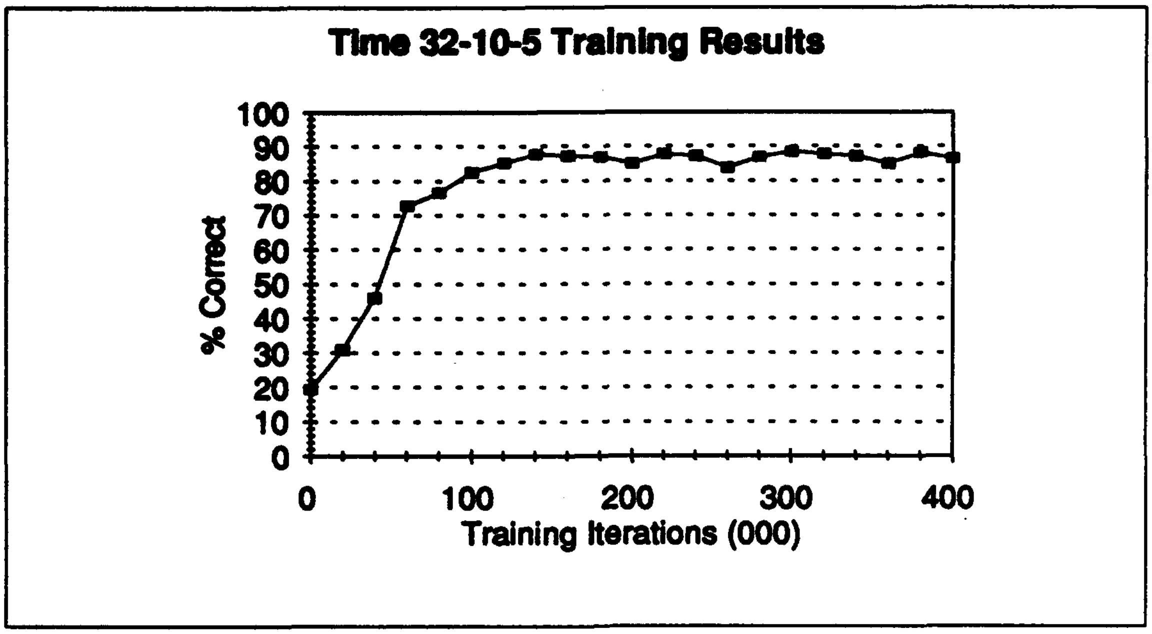


Figure 10: 32-5-5 Results



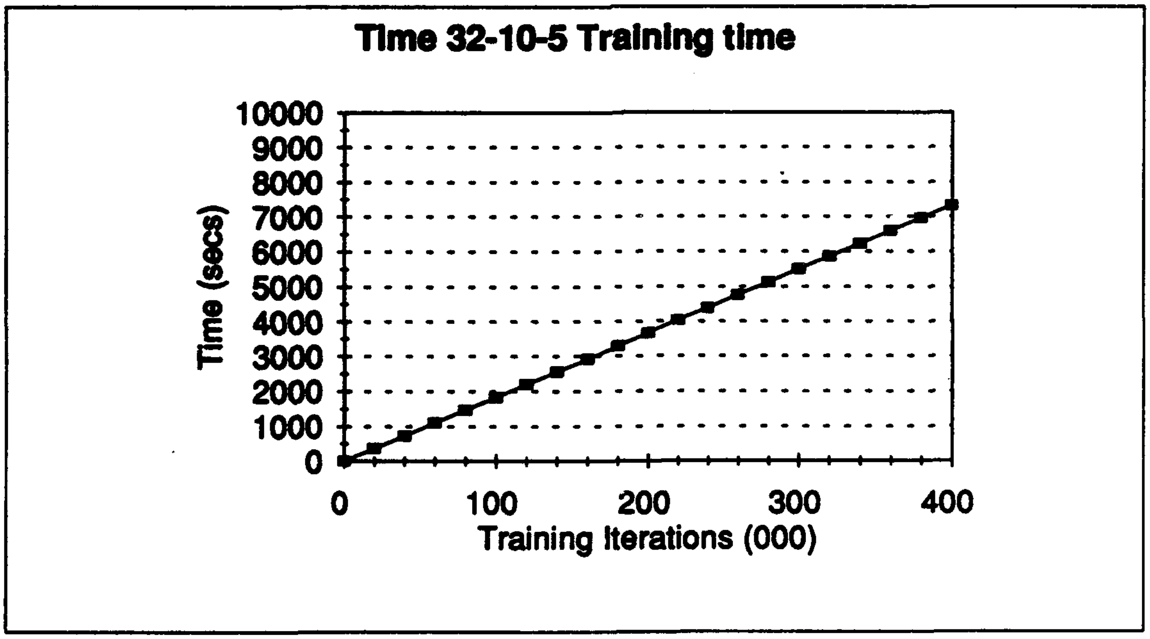
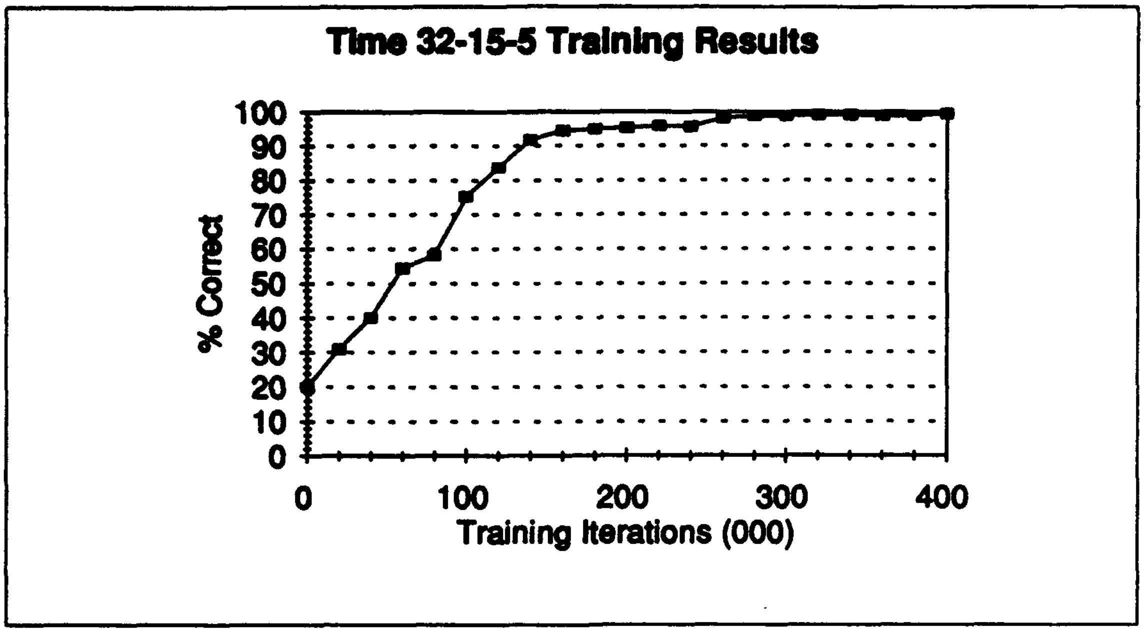


Figure 11: 32-10-5 Results



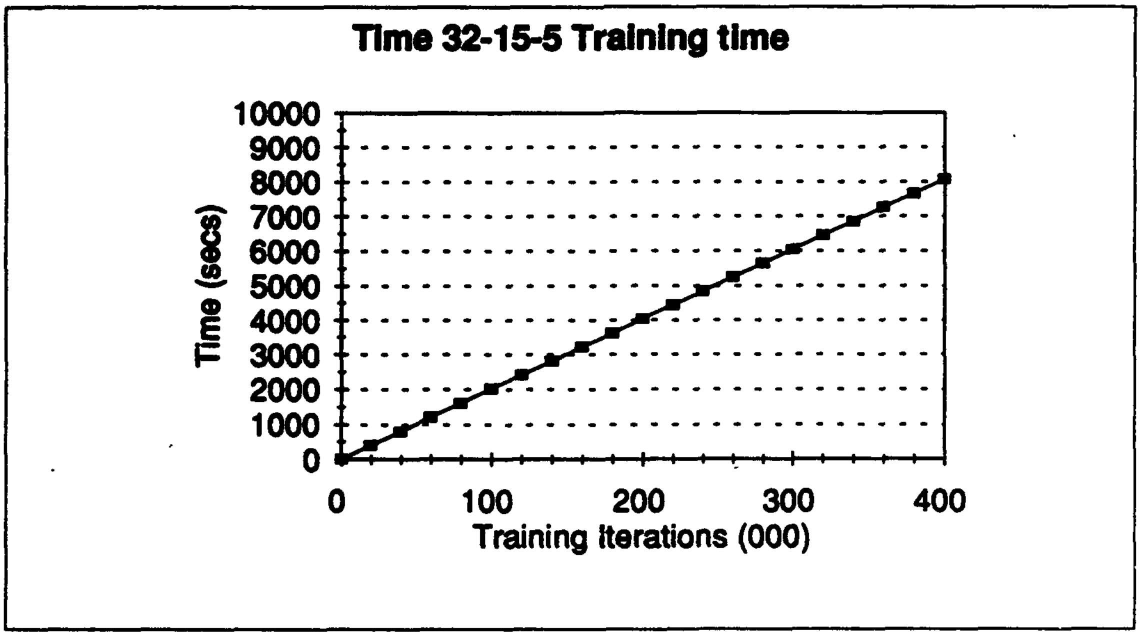
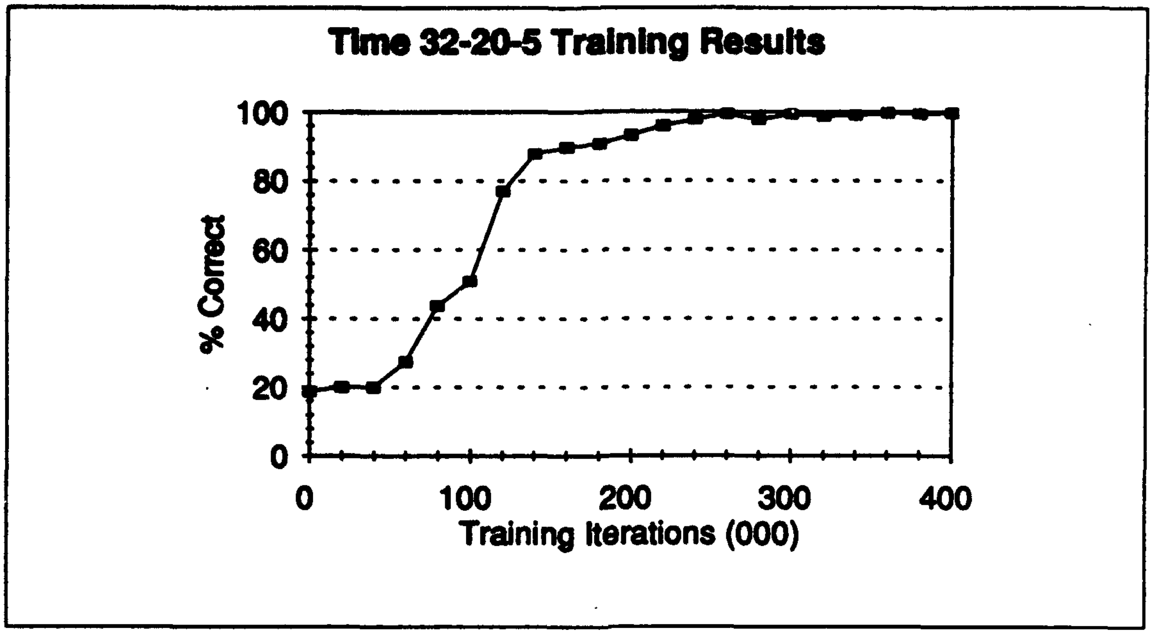


Figure 12: 32-15-5 Results



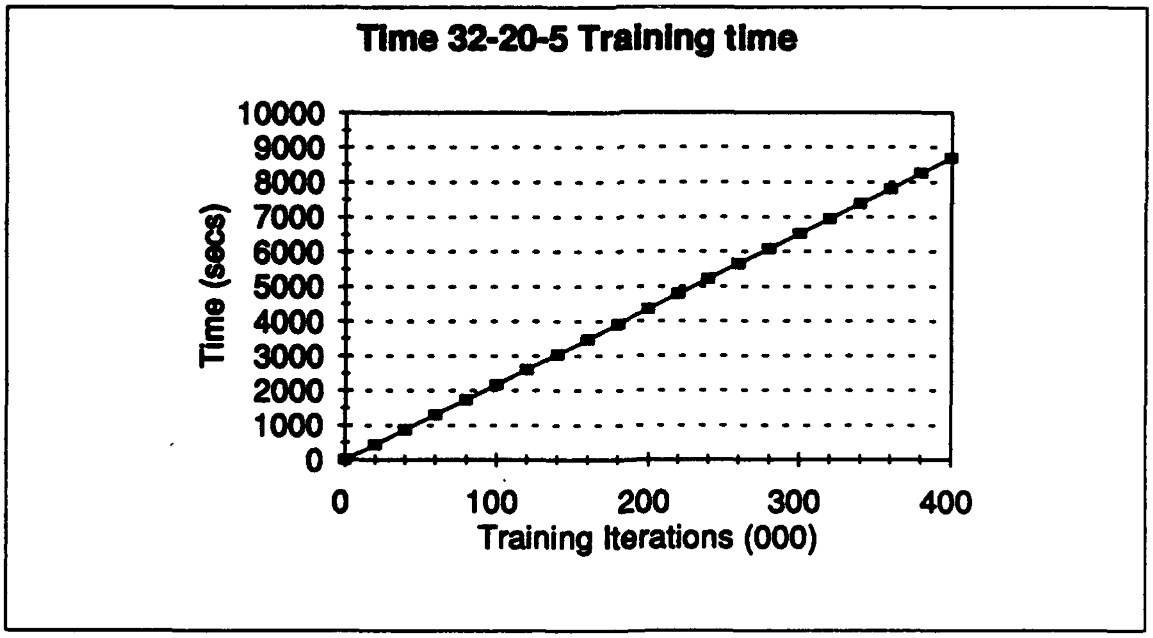
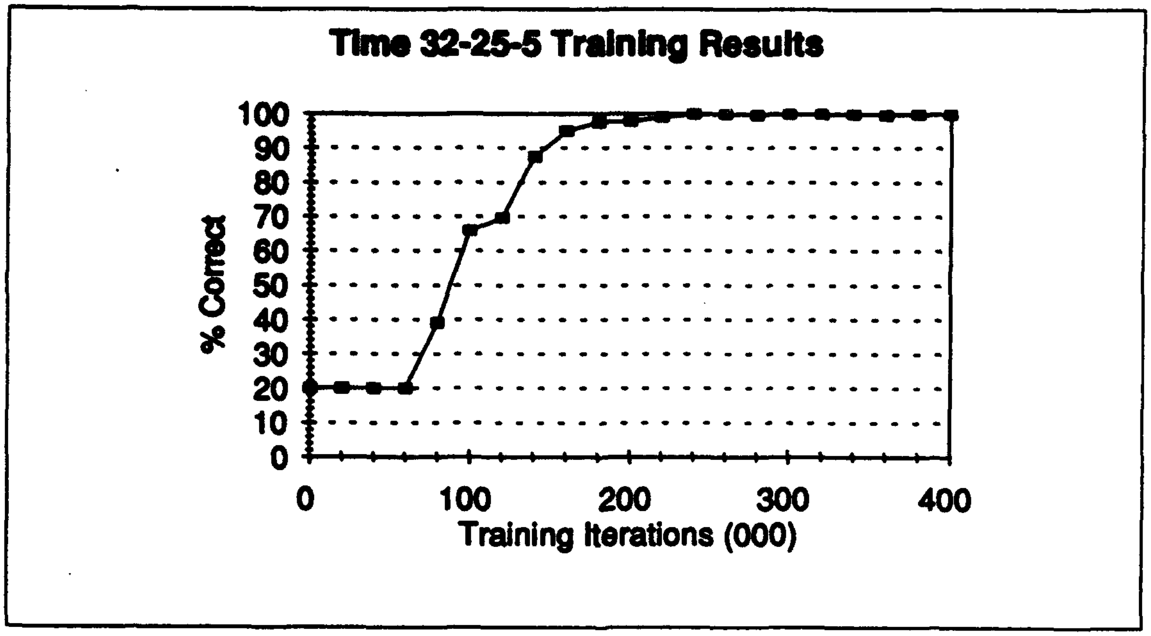


Figure 13: 32-20-5 Results



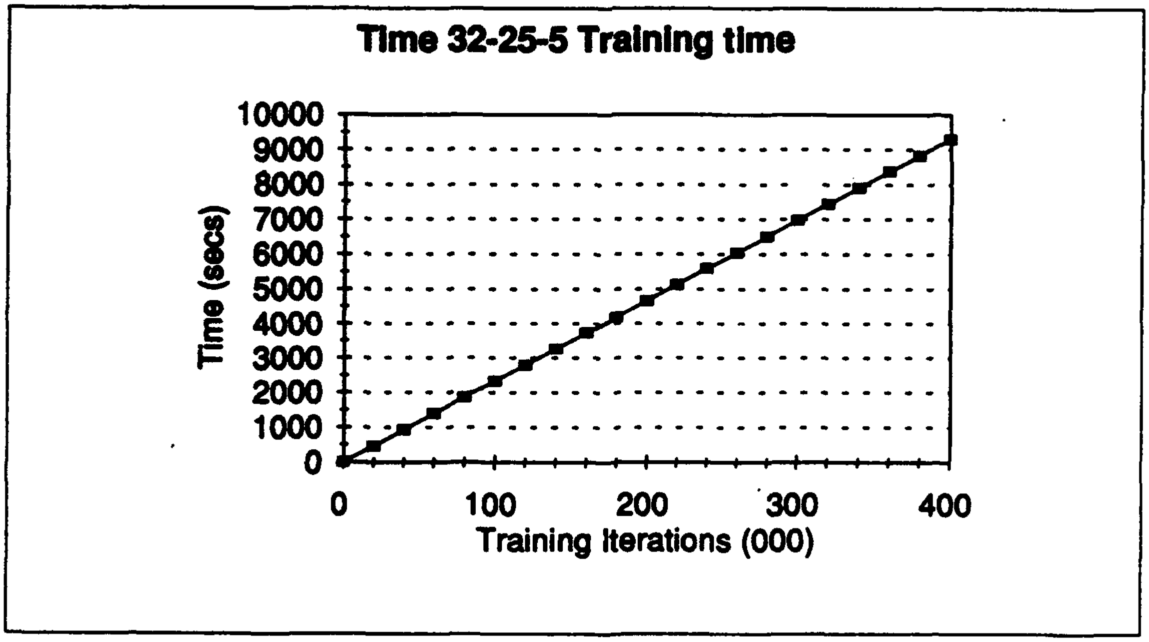
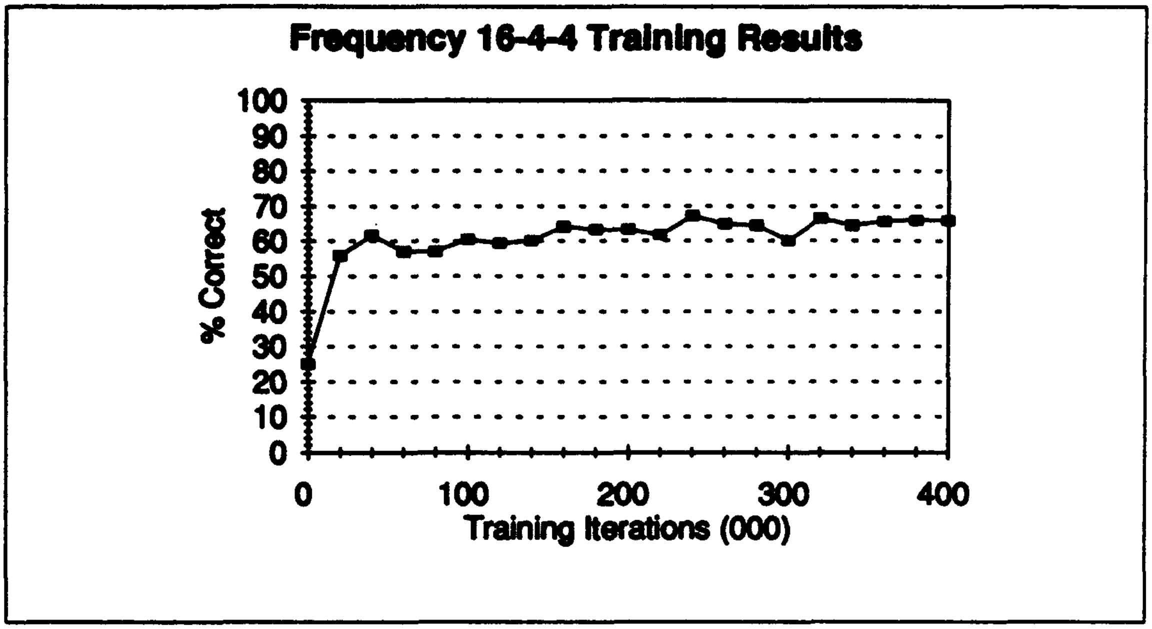


Figure 14: 32-25-5 Results



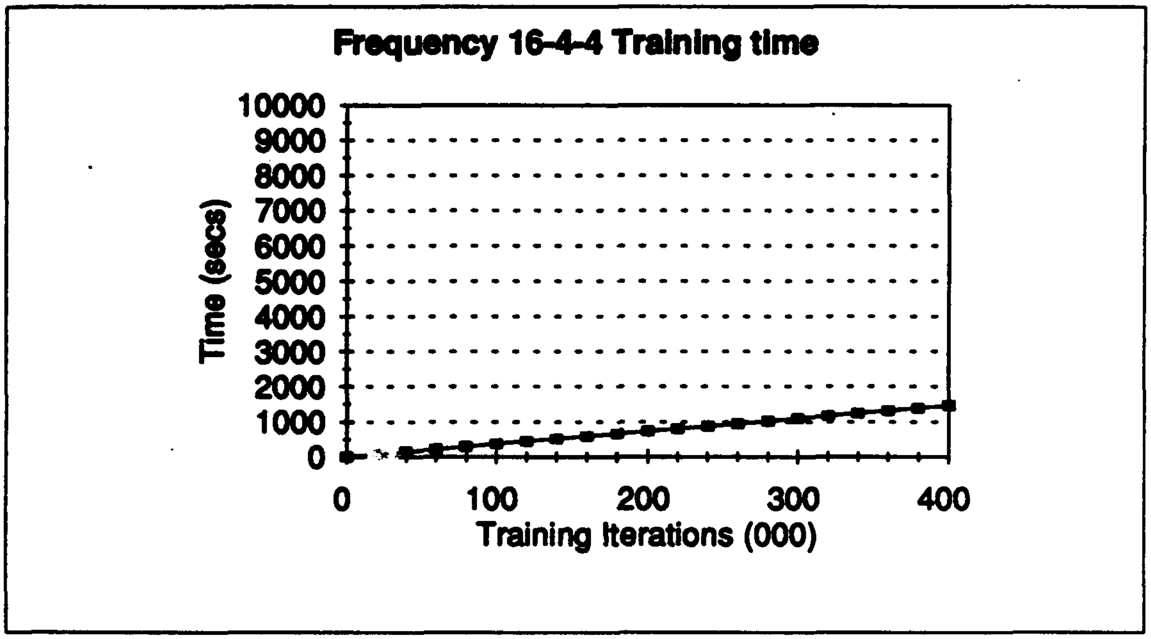
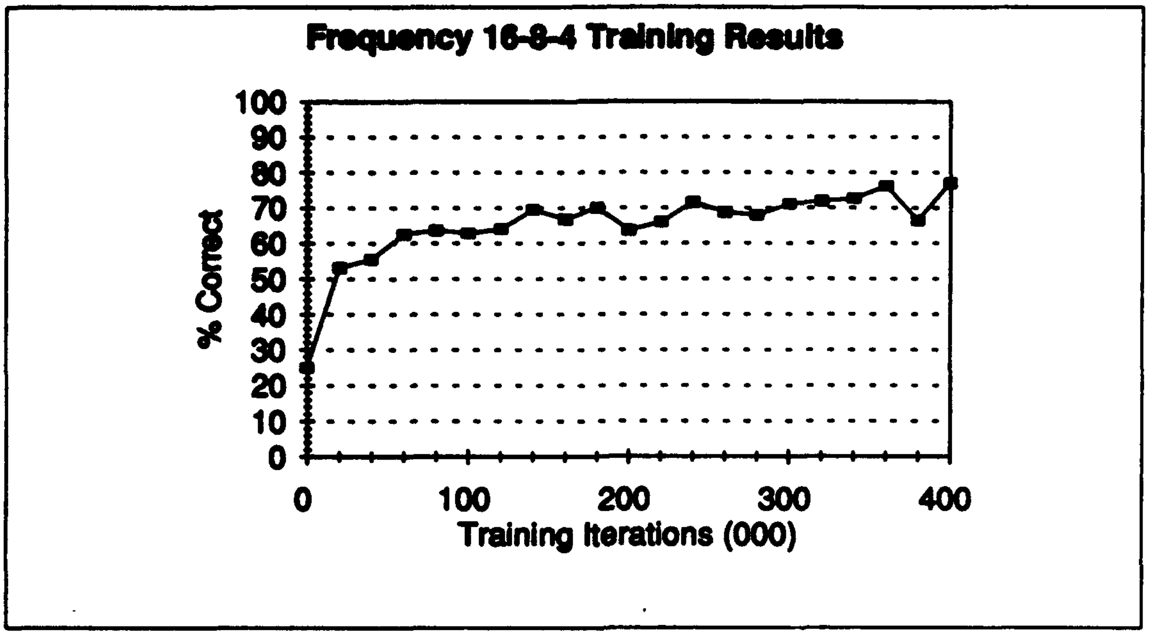


Figure 15: 16-4-4 Results



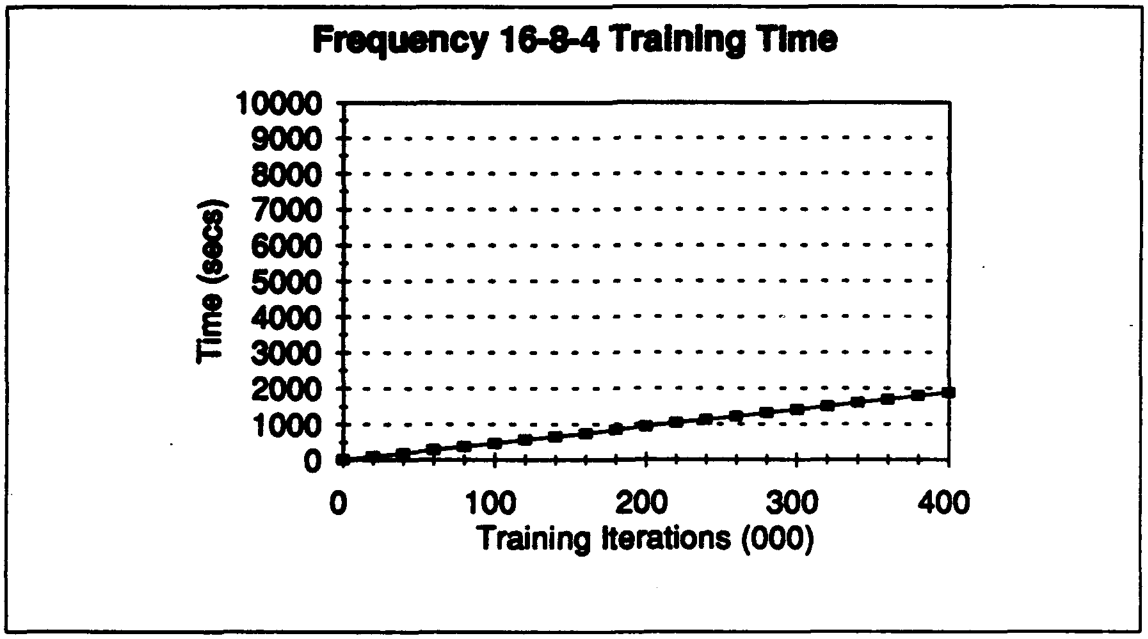
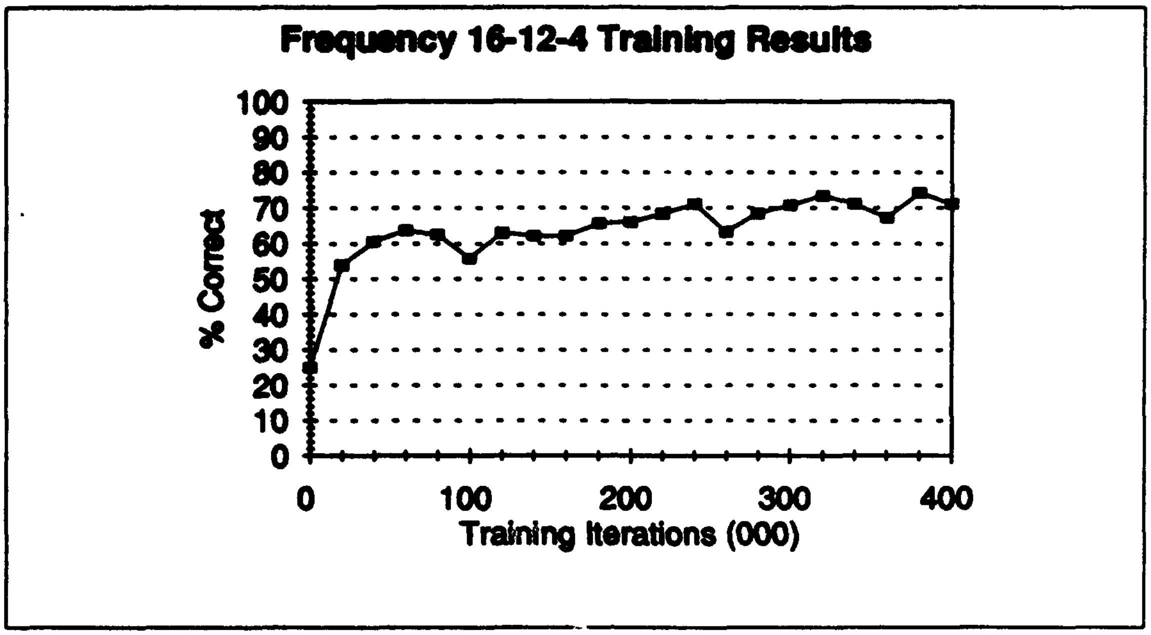


Figure 16: 16-8-4 Results



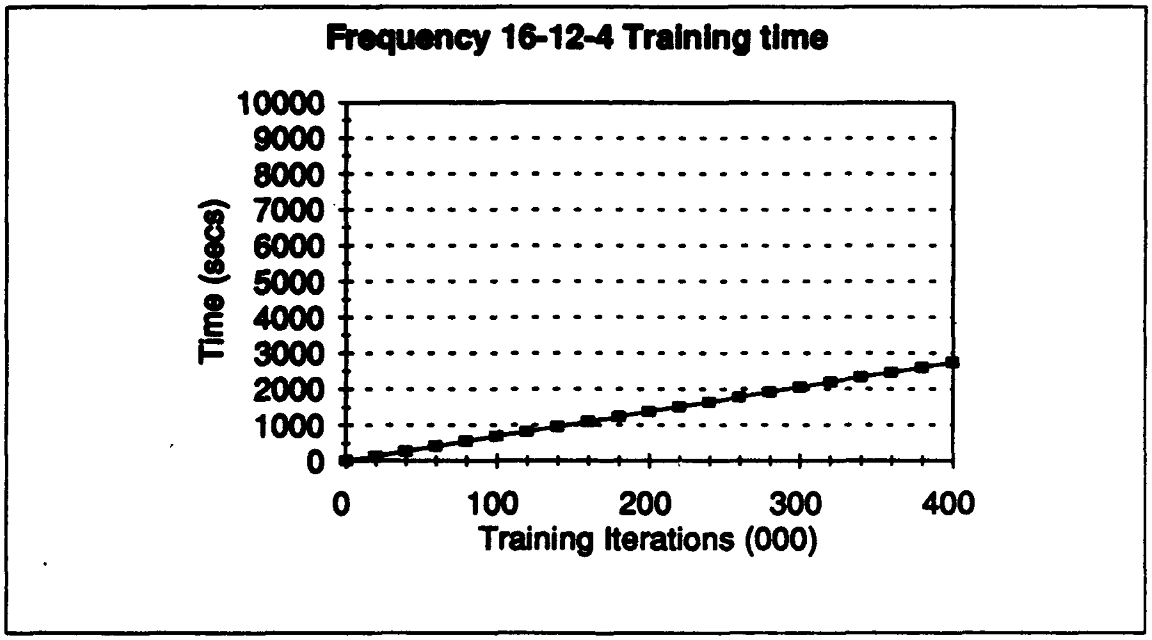
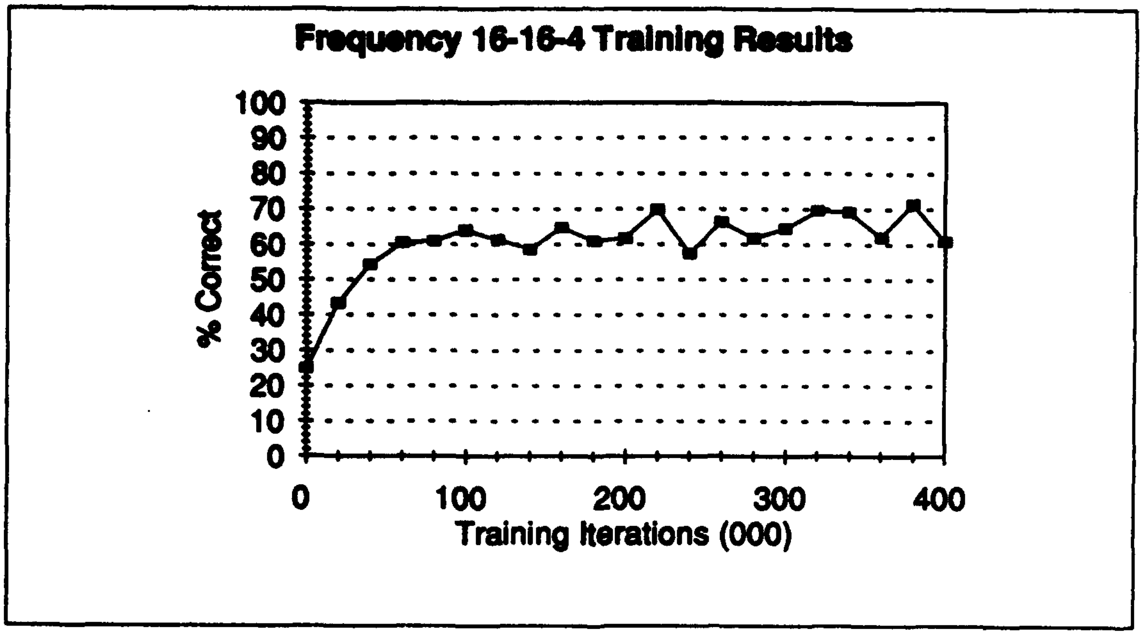


Figure 17: 16-12-4 Results



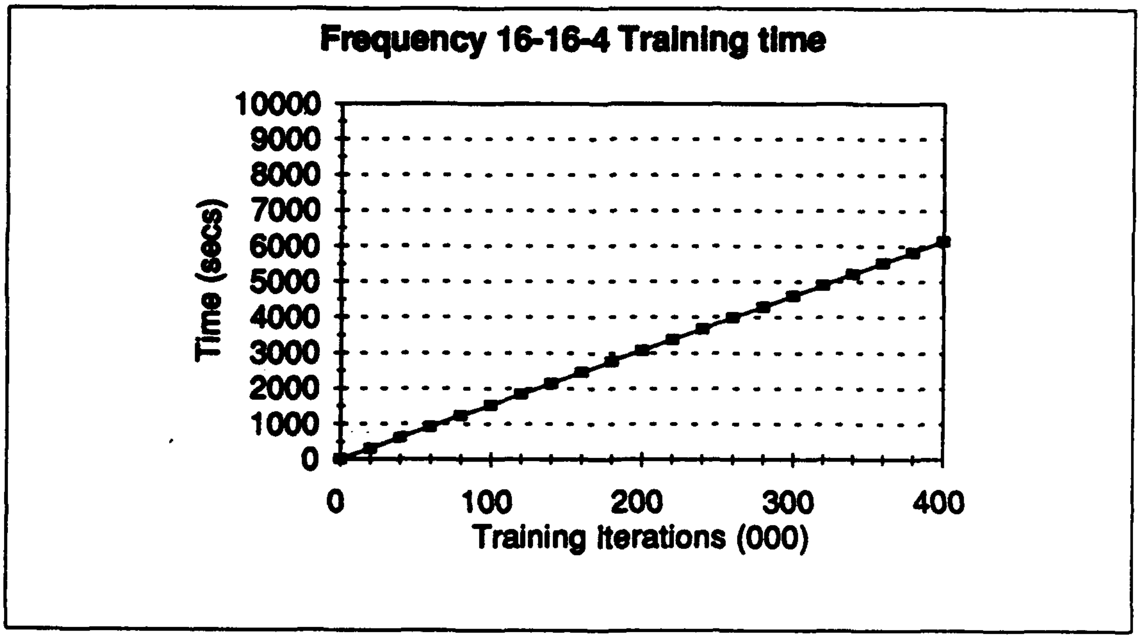
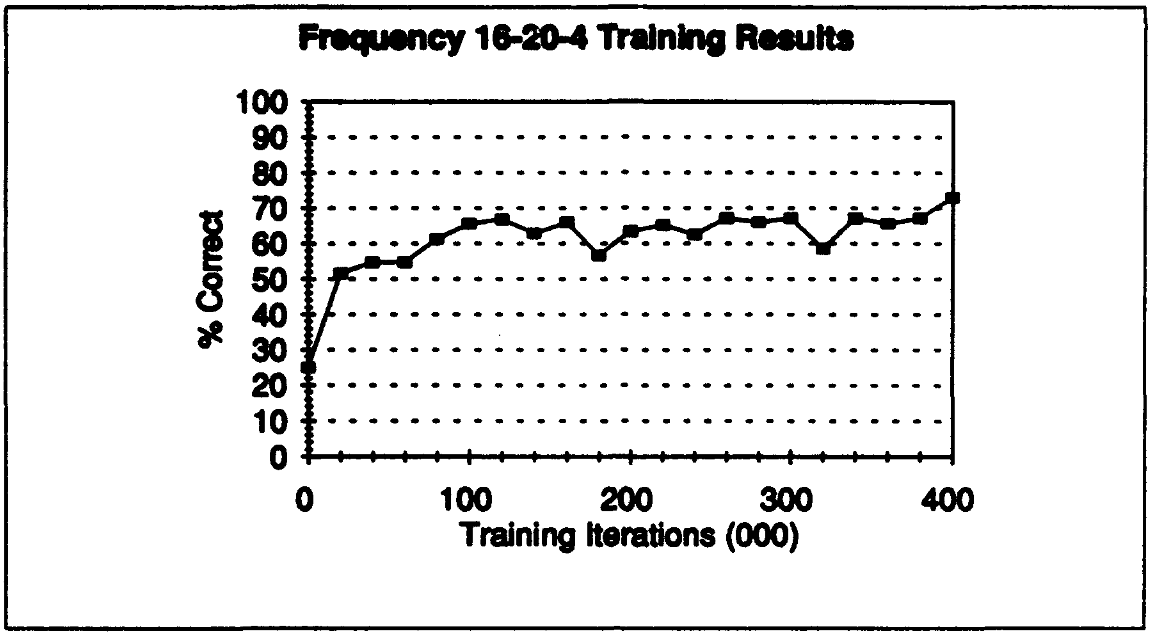


Figure 18: 16-16-4 Results



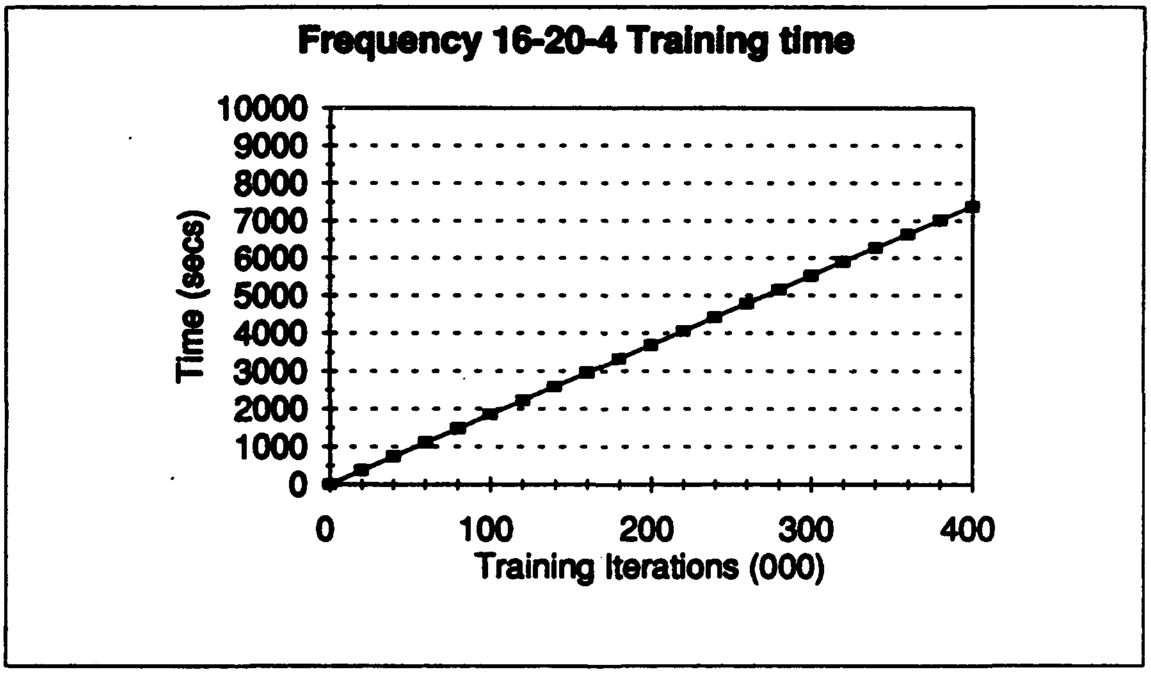
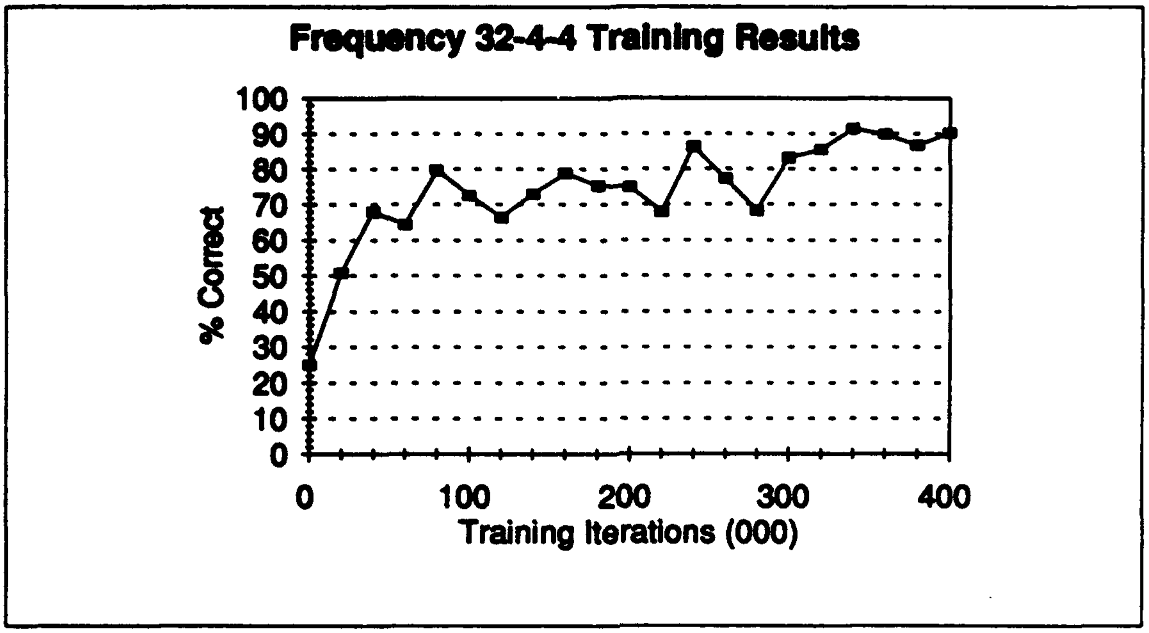


Figure 19: 16-20-4 Results



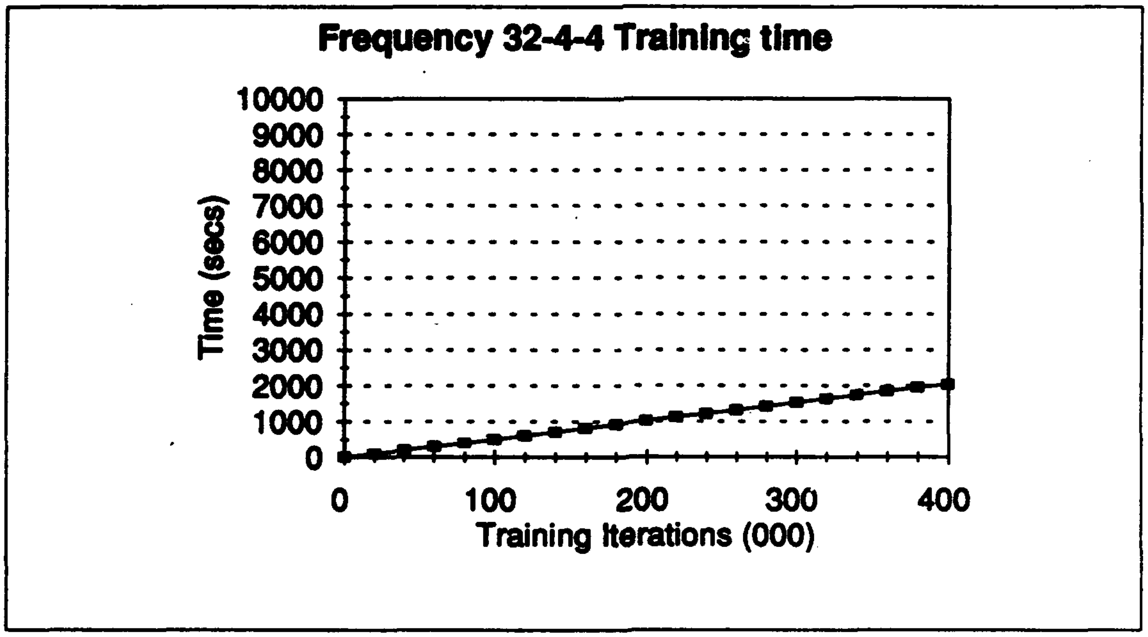
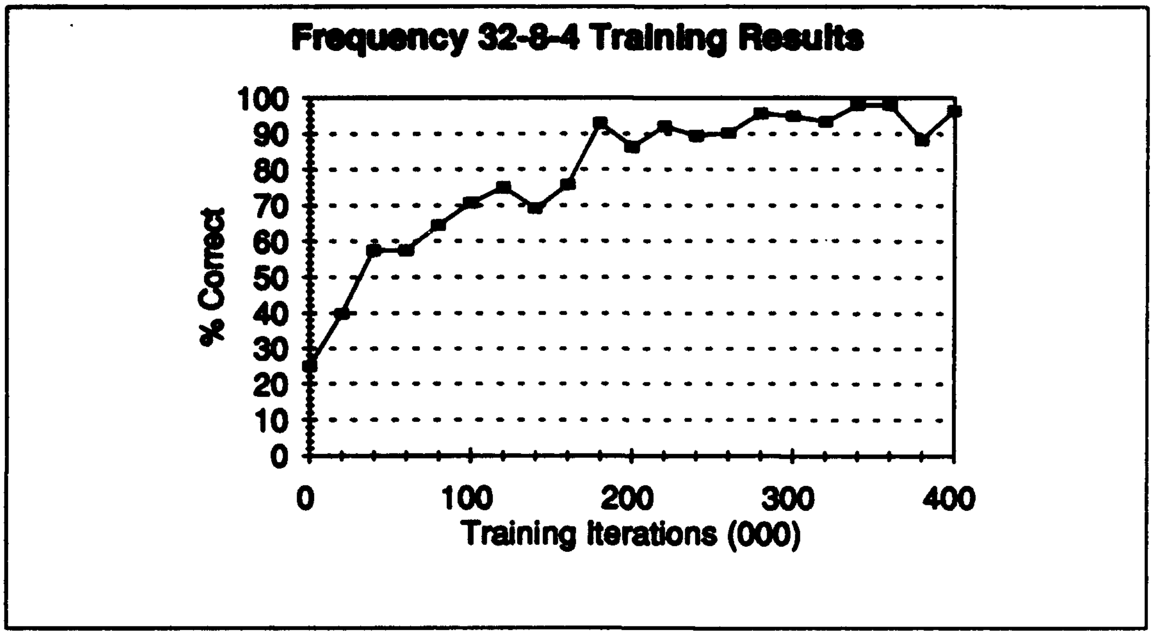


Figure 20: 32-4-4 Results



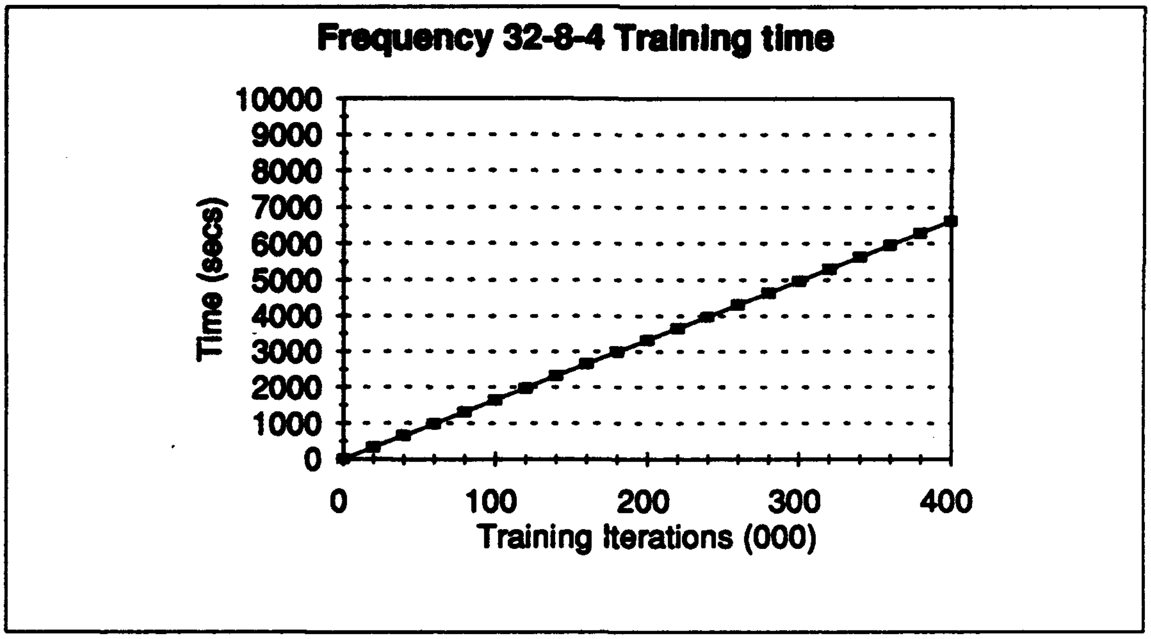
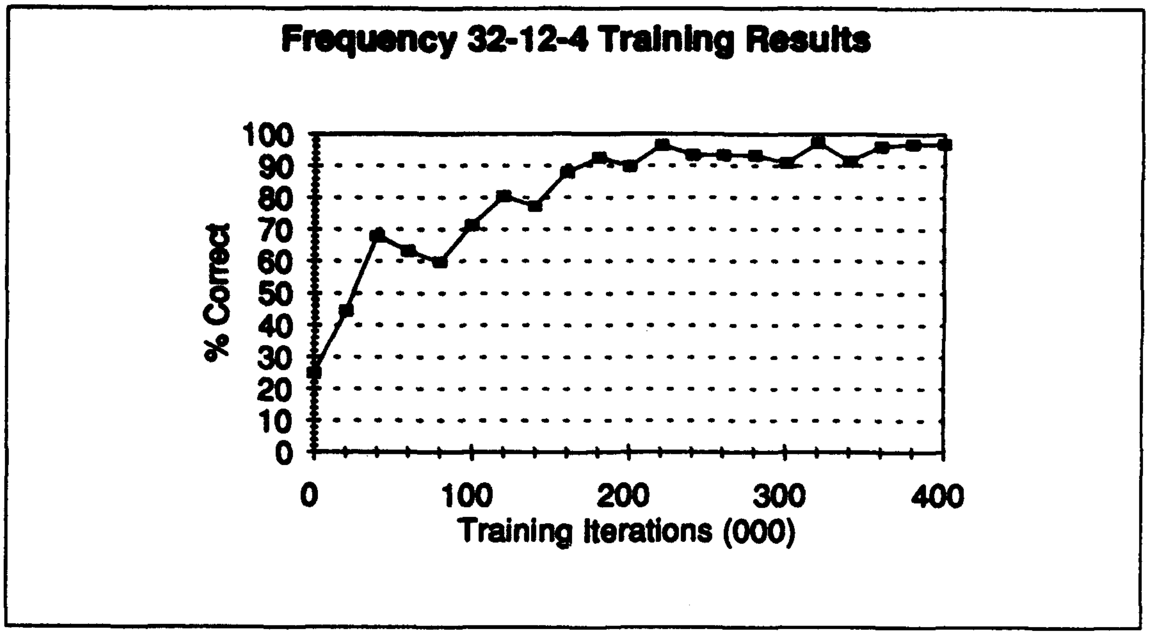


Figure 21: 32-8-4 Results



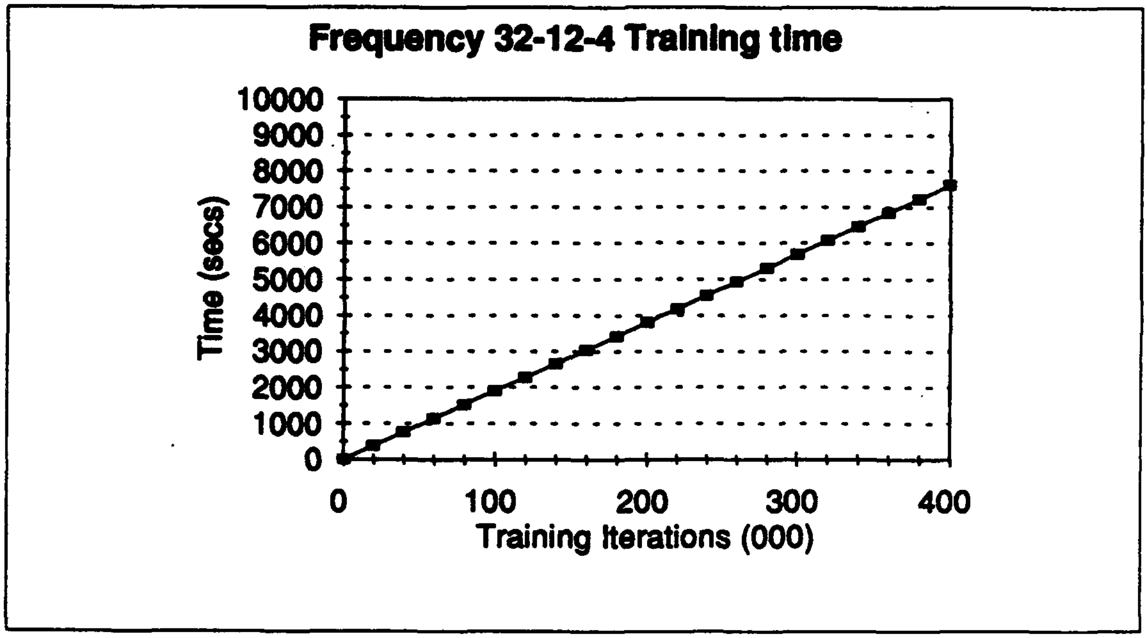
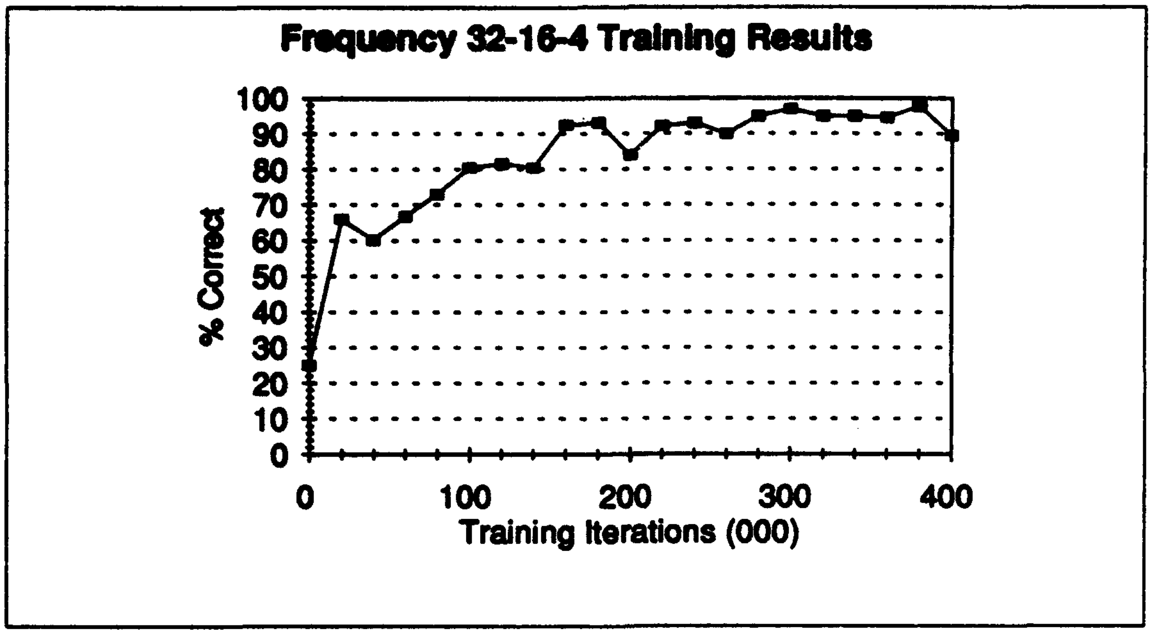


Figure 22: 32-12-4 Results



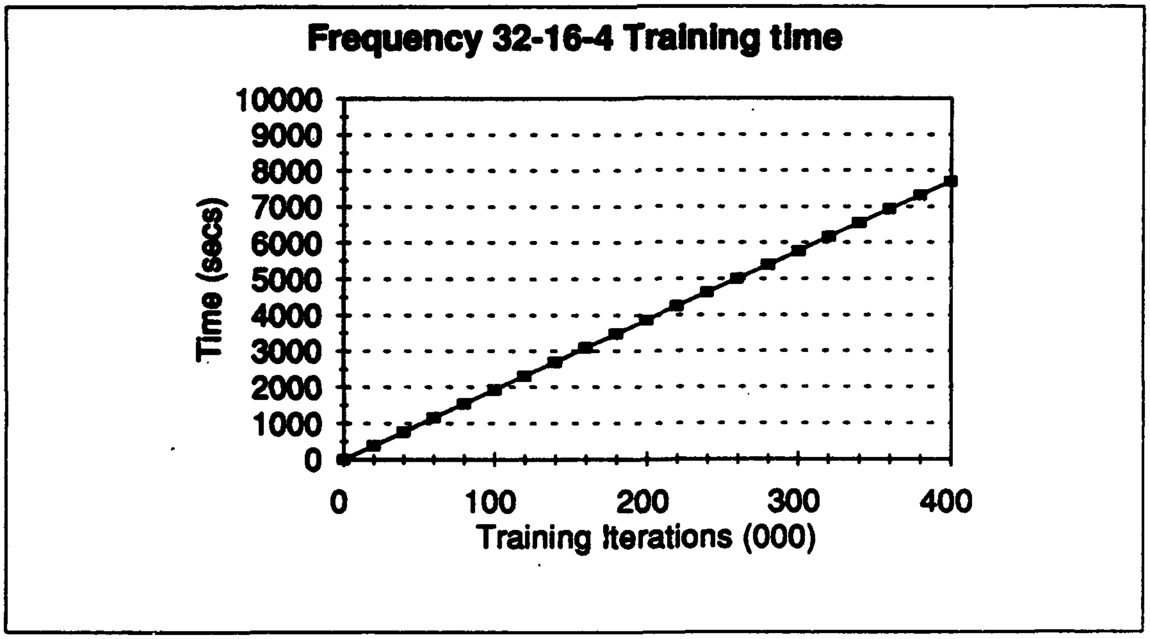
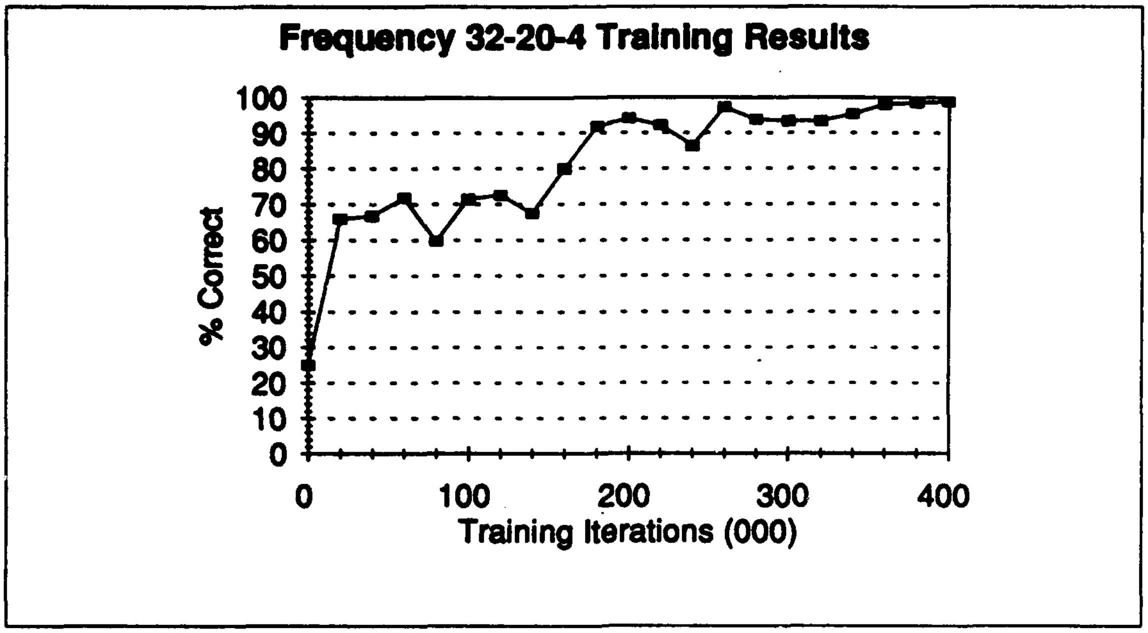


Figure 23: 32-16-4 Results



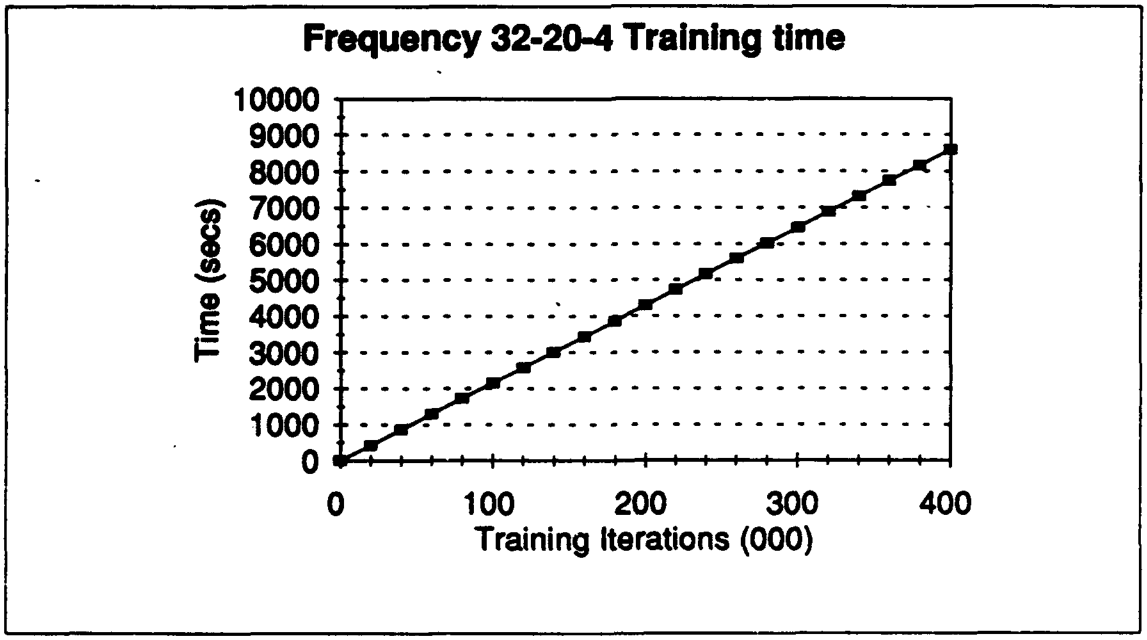


Figure 24: 32-20-4 Results

UNCLASSIFIED

SECURITY CLASSIFICATION OF FORM
(highest classification of Title, Abstract, Keywords)

DOCUMENT CONTROL DATA

(Security classification of title, body of abstract and indexing annotation must be entered when the overall document is classified)

1. ORIGINATOR (the name and address of the organization preparing the document. Organizations for whom the document was prepared, e.g. Establishment sponsoring a contractor's report, or tasking agency, are entered in section 8.)

DEFENCE RESEARCH ESTABLISHMENT OTTAWA
NATIONAL DEFENCE
SHIRLEY BAY, OTTAWA, ONTARIO K1A OK2 CANADA

2. SECURITY CLASSIFICATION (overall security classification of the document including special warning terms if applicable)

UNCLASSIFIED

3. TITLE (the complete document title as indicated on the title page. Its classification should be indicated by the appropriate abbreviation (S,C or U) in parentheses after the title.)

NEURAL NETWORKS FOR CLASSIFICATION OF RADAR SIGNALS (U)

4. AUTHORS (Last name, first name, middle initial)

CARTER, CHRISTOPHER, A. AND MASSE, NATHALIE (APPLIED SILICON INC.)

5. DATE OF PUBLICATION (month and year of publication of document)

NOVEMBER 1993

6a. NO. OF PAGES (total containing information. Include Annexes, Appendices, etc.)

74

6b. NO. OF REFS (total cited in document)

14

7. DESCRIPTIVE NOTES (the category of the document, e.g. technical report, technical note or memorandum. If appropriate, enter the type of report, e.g. interim, progress, summary, annual or final. Give the inclusive dates when a specific reporting period is covered.)

DREO TECHNICAL NOTE

8. SPONSORING ACTIVITY (the name of the department project office or laboratory sponsoring the research and development. Include the address.)

DEFENCE RESEARCH ESTABLISHMENT OTTAWA
NATIONAL DEFENCE
SHIRLEY BAY, OTTAWA, ONTARIO K1A OK2 CANADA

9a. PROJECT OR GRANT NO. (if appropriate, the applicable research and development project or grant number under which the document was written. Please specify whether project or grant)

011LB

9b. CONTRACT NO. (if appropriate, the applicable number under which the document was written)

10a. ORIGINATOR'S DOCUMENT NUMBER (the official document number by which the document is identified by the originating activity. This number must be unique to this document.)

DREO TECHNICAL NOTE 93-33

10b. OTHER DOCUMENT NOS. (Any other numbers which may be assigned this document either by the originator or by the sponsor)

11. DOCUMENT AVAILABILITY (any limitations on further dissemination of the document, other than those imposed by security classification)

(X) Unlimited distribution
( ) Distribution limited to defence departments and defence contractors; further distribution only as approved
( ) Distribution limited to defence departments and Canadian defence contractors; further distribution only as approved
( ) Distribution limited to government departments and agencies; further distribution only as approved
( ) Distribution limited to defence departments; further distribution only as approved
( ) Other (please specify):

12. DOCUMENT ANNOUNCEMENT (any limitation to the bibliographic announcement of this document. This will normally correspond to the Document Availabilty (11). However, where further distribution (beyond the audience specified in 11) is possible, a wider announcement audience may be selected.)

UNLIMITED

13. **ABSTRACT** (a brief and factual summary of the document. It may also appear elsewhere in the body of the document itself. It is highly desirable that the abstract of classified documents be unclassified. Each paragraph of the abstract shall begin with an indication of the security classification of the information in the paragraph (unless the document itself is unclassified) represented as (S), (C), or (U). It is not necessary to include here abstracts in both offical languages unless the text is bilingual).

(U) Radar Electronic Support Measures (ESM) systems are faced with increasingly dense and complex electromagnetic environments. Traditional algorithms for signal recognition and analysis are highly complex, computationally intensive, often rely on heuristics, and require humans to verify and validate the analysis. In this paper, the use of an alternative technique - artificial neural networks - to classify pulse-to-pulse signal modulation patterns is investigated. Neural networks are an attractive alternative because of their potential to solve difficult classification problems more effectively and more quickly than conventional techniques. Neural networks adapt to a problem by learning, even in the presence of noise or distortion in the input data, without the requirement for human programming. In the paper, the fundamentals of network construction, training, behaviour and methods to improve the training process and enhance a network's performance are discussed. As well, a description and the results of the classification experiments are provided.

14. **KEYWORDS, DESCRIPTORS or IDENTIFIERS** (technically meaningful terms or short phrases that characterize a document and could be helpful in cataloguing the document. They should be selected so that no security classification is required. Identifiers, such as equipment model designation, trade name, military project code name, geographic location may also be included. If possible keywords should be selected from a published thesaurus. e.g. Thesaurus of Engineering and Scientific Terms (TEST) and that thesaurus-identified. If it is not possible to select indexing terms which are Unclassified, the classification of each should be indicated as with the title.)

NEURAL NETWORK
RADAR
SIGNAL
PULSE
MODULATION
CLASSIFICATION