USING ARTIFICIAL NEURAL NETWORKS FOR FORENSIC FILE TYPE IDENTIFICATION

by Ryan M. Harris

Center for Education and Research in
Information Assurance and Security,
Purdue University, West Lafayette, IN 47907-2086

USING ARTIFICIAL NEURAL NETWORKS FOR FORENSIC FILE TYPE IDENTIFICATION

A Thesis

Submitted to the Faculty

of

Purdue University

by

Ryan M. Harris

In Partial Fulfillment of the

Requirements for the Degree

of

Master of Science

May 2007

Purdue University

West Lafayette, Indiana

ACKNOWLEDGMENTS

The author would like to express appreciation to Dr. Marcus Rogers for his assistance with this thesis. Without his help, this project never would have been completed. He would also like to thank Dr. Victor Raskin for his guidance throughout his stay at Purdue University and Professor Rick Mislan for his neural network advice.

TABLE OF CONTENTS

Page

LIST OF TABLES

LIST OF FIGURES

ABSTRACT

Harris, Ryan M. M.S., Purdue University, May 2007. Using Artificial Neural Networks for Forensic File Type Identification. Major Professor: Dr. Marcus K. Rogers.

Current forensic software relies upon accurate identification of file types in order to determine which files contain potential evidence. However, current type recognition mechanisms are susceptible to simple attacks that enable a criminal to confuse the detection algorithm. This study investigated whether artificial neural networks were superior to existing mechanisms at responding to modern evidence tampering techniques and concluded that the tested neural networks were not better than the existing methods. However, the study yielded avenues for future investigation.

1 THE PROBLEM

1.1 Introduction

With the rapid development of computer technology, crime involving digital evidence is becoming more commonplace (Kruse & Heiser, 2002). Digital forensics involves an investigation of digital evidence to enable investigators to determine the truth about what happened (Kruse & Heiser, 2002). However, to achieve this goal, law enforcement must have tools and technologies that enable them to examine the evidence accurately. Unfortunately, as computer technology has advanced, criminals have found a myriad of ways to avoid law enforcement detection.

The forensics community seems to be ill prepared for these anti-forensic techniques. In fact, most of the discussion about these methods is taking place outside the law enforcement community (Harris, 2006). Since anti-forensics can be used to ". . . compromise the availability or usefulness of evidence to the forensics process" (Harris, 2006, p. S45), it is imperative that investigators begin to develop techniques for responding to these threats.

1.2 Statement of the Problem

A child predator might attempt to hide an image by changing its extension to ".doc" so it appears to be a Microsoft Word document. Some forensics software looks at a file's contents to determine what information it contains (Kruse & Heiser, 2002). This allows investigators to detect the type of the contents even if the file extension has been changed. However, the software currently detects images based on hard coded file signatures. The first few bytes of the file are examined to determine what type of information the file contains (Foster & Liu, 2005). This method of detecting file types works well in cases where the file has been otherwise unaltered; but it may fail when the file's contents do not match the predetermined signatures.

The signature-based method of detecting file types leaves the software susceptible to "evidence counterfeiting." Evidence counterfeiting manipulates existing evidence to hide its purpose or creates new evidence that is deceptive (Harris, 2006). If a suspect alters any of the first few bytes of a file, the forensics packages are no longer able to detect what type of information is in the file (Foster & Liu, 2005).

1.3 Significance of the Problem

Even though files can be rendered undetectable to present forensics software, suspects are still able to open them normally. This means that a suspect can alter files so they are not detected during a scan of the computer for incriminating images. A suspect may go free simply because the investigative software used was unable to see the files.

The implications for child exploitation investigations are scary. The Internet has made it increasingly easy for sexual predators to traffic illicit images (Heimbach, 2002; Federal Bureau of Investigation, n.d.) and engage in other illicit acts (Hernandez, 2006). During the last ten years, child pornography cases have increased over 1750 percent (Federal Bureau of Investigation, n.d.). One Internet trading group that the FBI investigated had over 7,200 members (Heimbach, 2002). If suspects can reliably hide files from forensic tools, many of these individuals may not be caught.

1.4 Purpose of the Study

The purpose of this research was to determine whether artificial neural networks could provide a more robust method of detecting file types than that provided by current signature based methods. The expectation was that artificial neural networks' ability to respond to changes in data would allow reliable detection of file types even if the contents had been subtly altered. This research attempted to determine whether artificial neural networks provided an acceptable substitute for existing detection algorithms. During the process of pursuing the research goal, the research also attempted to identify which file types could be identified most accurately using neural networks.

1.5 Delimations

Because of the wide variety of files in use on modern computers, the research was unable to cover all file types comprehensively. This thesis concentrated on files that could not yield useful information without being identified first. This included files that store information only in a binary format such as images, executables and archives. Other file types such as spreadsheets and documents may store binary information, but the plain text is still available to any search algorithm used by the forensic software. Therefore, this study did not seek to identify file types that included plain text but instead concentrated on files whose contents could not be searched. Specifically, this study only included JPEG, GIF, TIFF, BMP and PNG image files since those files were widely available on the Internet and were commonly encountered during forensic investigation (Table 1).

Signature analysis could occur in two forms in forensic investigation. If a file's extension has been changed, signatures might be used to determine the type of data the file contains. Another use of signature identification is in file carving. During file carving, a forensics program attempts to identify the start and end of files based on pre-defined signatures (Richard & Roussev, 2005). This technique is typically used to find deleted files. While it was expected that this study's results might be useful for file carving, the investigation only centered on identifying files whose contents were known.

1.6 Limitations

The layout of files generated within different software packages may have varied slightly due to differences in interpretation of the format specifications. Since it was impossible to obtain a complete sample of all the different variations of the file formats,

Table 1

Number of images of each type contributing to the testing and training sets

File Type	Training Set	Testing Set
JPEG	5000	500
PNG	5000	500
BMP	1000	100
GIF	1000	100
TIFF	1000	100
TOTAL	13000	1300

this research employed opportunity based sampling to select images based on their availability rather than attempting to meet a specific statistical constraint. Images were selected from a variety of image databases on the Internet. The images in these databases may have been primarily furnished by photographers who used professional software and equipment. As such, the training data may not have accurately represented image files that came from lower-end software or equipment. This limitation may have affected the applicability of the generated neural networks to the community of image files at large. However, the file formats of the images are constrained by specifications, so the impact should have been minimal.

Another limitation of this research involved the complexity of creating classification mechanisms. General-purpose classification schemes are difficult to create (Duda, Hart, & Stork, 2001). While the study yielded a general classification mechanism, the mechanism was not optimal for all of the file types. Future studies would be necessary to determine whether individualized classification mechanisms are necessary for specific types of file. For example, GIF files might need a separate input feature choice method than JPEG files to be detected in an optimal manner.

1.7 Definitions

The following definitions explain the usage of key terms used throughout this document.

- Activation function – An activation function is the function that determines the output value of a neuron in an artificial neural network based upon the weight assigned to the connections to that neuron (Duda, Hart, & Stork, 2001).

- ANOVA – An ANOVA (analysis of variance) is a statistical test that attempts to determine whether there is a statistically significant difference between two or more data groups.
- Anti-forensics – “Any attempts to compromise the availability or usefulness of evidence to the forensics process” (Harris, 2006, p. S45).
- Artificial neural network – An artificial neural network uses a series of neurons to transform a set of inputs into a desired set of outputs (Bishop, 1995).
- Backpropagation – Backpropagation is the process by which an artificial neural network adjusts the weights assigned to the connections between neurons to bring the current output value of the neuron closer to the expected value (Duda, Hart, & Stork, 2001). Backpropagation only occurs during neural network training.
- Clustering – Clustering is a technique for grouping similar data points together to aid in classifying the data (Duda, Hart, & Stork, 2001; Mena, 2003).
- Epoch – An epoch is one presentation of the complete training set to the neural network (Duda, Hart, & Stork, 2001).
- Error – Error is the difference between an expected neuron output value, and the actual output value.
- Error backpropagation – See *backpropagation*.
- Filesystem – A filesystem contains data which helps the computer locate and access files stored on the drive.
- File carving – File carving is a technique for extracting file contents from a disk by searching for signatures which identify the start and end of the file contents (Richard & Roussev, 2005).

- Forensics - “The application of science to those criminal and civil laws which are enforced by police agencies in a criminal justice system” (Saferstein, 1998, p. 4).
- Hidden layer – A hidden layer is a layer whose inputs and outputs are connected to other neurons (Duda, Hart, & Stork, 2001).
- Input layer – An input layer is made up of neurons whose inputs are the data that the network is expected to process (Duda, Hart, & Stork, 2001).
- Incremental training algorithm – An incremental training algorithm adjusts for differences between the actual output value and the expected output value after each sample in the training set is presented to the network rather than adjusting for the error at the end of each epoch (Fast Artificial Neural Network Library, n.d.).
- Layer – A layer is a group of nodes that all accept inputs from the same data set and output to the same data set. There are three types of layers: input layers, hidden layers, and output layers (Duda, Hart, & Stork, 2001). There are connections between the layers that have weights assigned to them to allow the network to adjust the actual output values so that they match with the desired output values (Duda, Hart, & Stork, 2001).
- Metadata – Metadata is data about other data. In a filesystem, the filename, file size, date last accessed, etc. are all metadata; they are data about the file itself.
- Mean square error (MSE) – The mean of the squared differences between the expected and actual output values from neurons in a neural network.
- Neuron - A neuron takes one or more input values and transforms them using an activation function to produce an output value (Duda, Hart, & Stork, 2001).

- Node – See *neuron*.
- Output layer – An output layer is a layer made up of neurons which output the end result of the network's processing (Duda, Hart, & Stork, 2001).
- Overfitting – Overfitting occurs when the neural network begins to approximate individual data samples in the training set rather than generalizing the pattern that these data values represented.
- Present – To present a sample is to supply it to the input layer of a neural network in order to determine what the output values of the neural network will be.
- Symmetric sigmoid – A symmetric sigmoid function is a type of activation function that is smooth and non-linear (Duda, Hart, & Stork, 2001), ranging from -1 to 1 (Fast Artificial Neural Network Library, n.d.).
- Tanh – Tanh is an abbreviation for the function hyperbolic tangent. This function is a member of the sigmoid class of functions (Duda, Hart, & Stork, 2001).
- Testing set – A testing set is a set of samples which are used to test how well the network has learned the training set. Each sample in the testing set also includes the expected output value of the neural network so that it can be compared with the actual output value.
- Training – Training is the process by which a neural network "learns" patterns. Generally, data is presented to the neural network, and the actual output values are compared with the expected output values. Any error is corrected through backpropagation (Duda, Hart, & Stork, 2001).
- Training set – A training set is a set of samples that are used to train the network. Each sample in the training set also includes the expected output value of the

neural network so that the network can be trained to output that value when presented with a similar input.

2 LITERATURE REVIEW

2.1 Attempts to Evade Detection

It is important to understand how anti-forensic methods may affect computer file systems since they store a variety of evidence that can be useful to an investigation. In addition to file contents, file systems also store a wide range of metadata (Carrier, 2005) which is used to enhance usability. The wealth of information provided by the file system may be extremely valuable to an analysis. Without valid metadata, the forensic process can become extremely difficult to complete.

Therefore, criminals have started using the anti-forensic techniques of counterfeiting and destruction (Harris, 2006) to manipulate the forensic process. A criminal might choose to create counterfeit file system metadata to hide the true purpose of a file (Harris, 2006; Foster & Liu, 2005). For example, a criminal could choose to rename an image file so that it would appear to be a Microsoft Word document. However, the actual contents of the file would still be an image. Another technique that a criminal could use would be to destroy the file system metadata that points to the file. This would make it difficult for the investigator to find and identify the file. However, removing the file system information does not destroy the actual file contents (Carrier, 2005; Geiger, 2005; Mallery, 2001). Therefore, if the file contents can be identified, they may become valuable evidence.

2.2 Counteracting Attempts to Evade Detection

Forensic software vendors have recognized that criminals attempt to hide files by changing the file's extension. As a result, it was essential to identify files based on the actual data they contain rather than the names that they had been assigned. As early as 1973, the UNIX operating system provided the "file" command to enable people to identify files based on their contents (Darwin, 1999). The command used a few bytes from the beginning of the file (referred to as the "magic") to identify the file type (Darwin, 1999). Forensics software borrowed this technique and current software attempts to identify files based on a brief series of bytes at the beginning of the files (Foster & Liu, 2005).

File carving is another technique quite similar to file type identification that is used to recover deleted files. When a file's metadata has been destroyed, file carving attempts to find the original files by identifying their contents (Richard & Roussev, 2005). File carving can be viewed as an extended type of signature based file identification. This is because the first few bytes of each sector are checked for a signature that identifies the start of a known file type (Richard & Roussev, 2005).

2.3 Difficulties with File Type Identification

One of the primary difficulties with identifying data based on a known signature is that the signature must be static. If a signature is not picked correctly, it may fail to identify files consistently. In some cases, a signature may result in false positive values when it is not sufficiently restrictive. For example, Foster and Liu (2005) detailed how some forensic software can misidentify text files as executables if the first two bytes in the file are “MZ.” However, this is not the only difficulty. If a signature is too restrictive, it may result in files not being identified as they should be. Forensic software may fail to identify JPEG images if the second two bytes are changed, even though these two bytes are not significant in identifying an image as a JPEG (Harris, 2006).

Since file carving is a special type of file identification, it has similar problems to those encountered by standard file identification. The first activity of a carving tool is identifying the start of files using a signature detection algorithm (Richard & Roussev, 2005). However, even if we assume that the signature used for determining where the file starts is completely correct, there are still several difficulties with this approach. After a carving tool has identified the start of a file, it will scan in a similar manner for a signature that identifies the end of the file (Richard & Roussev, 2005). The signatures that identify the end of the file may not fall on an even 512-byte boundary. Therefore, a file carving tool must be even more carefully created to avoid falsely detecting the end of a file.

After an end signature is found, the tool then assumes that the bytes between the start and end signatures belong to a single file. However, file systems can become fragmented so the data stored in-between the start marker and end marker could be from different files. File fragmentation occurs frequently during normal computer use (Kinsella, 2006), so fragmentation can be expected to be a norm on most file systems. Therefore, there is a high likelihood that files will be incorrectly identified if the software is unable to detect whether the in-between sectors have come from a similar file type.

2.4 A Possible Approach

Usable files must have a standardized enough format that a program is able to parse the data they contain and use it. Logically then, all usable files must have some sort of format or pattern to the data that they contain. Since each usable file type must have a pattern, if this pattern can be discerned and expressed uniquely from other file types, then a file can be identified by the pattern of the data it contains. Therefore, file type identification may not need to be a search for a specific hard-coded byte-sequence. An algorithm could simply search for an identifiable pattern to determine what type of data a file contains.

A file may contain mostly random information with little organized structure. Therefore, one difficulty with this method is that a pattern might be hard to recognize through the noise of the file's data. Neural netwo rks may provide a solution since they can learn patterns that are difficult to discern (Mena, 2003).

Neural networks offer a couple of advantages for pattern recognition. First, they are extremely effective at recognizing patterns (Mena, 2003). This is a distinct advantage when processing large volumes of training data where the actual underlying pattern is unknown (Mena, 2003). Having a human search for discernible patterns may take quite a while, yet a neural network may be able to pick out the pattern easily.

Another advantage of neural networks is the speed with which they operate once they have been trained (Mena, 2003). Speed is an essential factor in a forensic investigation where almost every file on a drive will need to be identified and classified. As hard drive size increases, the number of files that can be stored on the drive increases as well. An algorithm that takes too much time to classify each file would not be useful to an investigation.

2.5 Neural Networks for Pattern Recognition

Neural networks are a well-established field. Initial research into neural networks started over 50 years ago (Mena, 2003). Extensive research has investigated using neural networks in pattern recognition tasks (Bishop, 1995). This research was based on the premise that the best framework for assessing patterns is that which is provided by statistics (Bishop, 1995).

Neural networks have been designed to solve a variety of pattern related problems including forecasting, clustering, classifying and generalizing (Mena, 2003). Prediction uses neural networks to determine future events based on history (Mena, 2003). While this might be useful in some branches of forensics, it appears to have little usefulness for analyzing file types.

Clustering attempts to group data according to similarities (Mena, 2003) which allows the researcher to find related data points. For file type identification, clustering could be useful to determine which features of a specific file type are most useful in creating a classification scheme. For example, a clustering algorithm could identify bytes within JPEG files that normally are identical.

Classification attempts to identify data as either inside or outside a set (Mena, 2003). Classification seems to be useful approach for a system that identifies file types. An algorithm might classify a file as belonging to a specific type of file as either a GIF or not a GIF image. However, this is not the goal of robust file type identification. A signature-based system is quite capable at identifying and classifying files based on specific attributes.

The most useful aspect of neural networks appears to be the ability to generalize data. Generalization attempts to recognize a pattern between cases (Bishop, 1995; Mena, 2003). When identifying file types, the algorithm must be powerful enough to enable it to identify files that have not been seen before. Neural networks might generalize file patterns enough to identify file types even when the file has been altered.

2.6 Capabilities and Limitations

Neural networks appear to have distinct advantages when used for file type identification. However, there are several significant limitations. Neural networks are vastly misunderstood in common literature (Bishop, 1995). As a result, neural networks may receive undue attention as a complete solution to every pattern recognition problem. Any investigation using neural networks cannot assume that they are the ideal solution.

One of the primary difficulties with neural networks is determining how much information must be provided at the inputs (Bishop, 1995). Intuitively, it would seem that the more information that is provided to the network, the better the matching capability. Nevertheless, this is not the case. Large numbers of input values actually may adversely affect the capability of the recognition system (Bishop, 1995). This occurs because the number of training data points must increase with the number of input nodes into the neural network (Bishop, 1995).

Another difficulty with neural networks is determining how many hidden layer nodes are necessary to represent the data accurately. Adding more hidden neurons may more accurately represent the data when the input data values are tightly related (Duda, Hart, & Stork, 2001). However, adding too many hidden layer inputs provides little value (Liang, Moskowitz, & Yih, 1992). Duda, Hart and Stork (2001) acknowledge that there "is no foolproof method for setting the number of hidden units before training" (p. 310). So, the only way to correctly determine the best number of hidden layer nodes is through experimentation.

3 PROCEDURES AND METHODS

3.1 Sample Selection

To gather an appropriately large sample, it was necessary to rely on opportunity-based sampling. Stratification by image type was used to increase the likelihood that each type of image was adequately represented in the training and testing sets. Table 1 indicates the number of images of each type that were assigned to the testing and training sets.

From the beginning, it was understood that file formats may have differed slightly from package to package (for example, a JPEG created in Adobe Photoshop will differ from one created by a Canon camera). So, the file sources were chosen with the hope of reducing this bias. The files were selected from several different Internet file repositories. Repositories that contained images from many different sources were favored over ones that contained images from a single source. This was done to reduce the likelihood that the image file formats in the repository would be identical. For example, Flickr was assumed to have more variety in its images than a stock photography site would have. This assumption was based on the fact that a stock photography site might use images from a small group of photographers.

In addition to controlling the repository selection, each image type was selected from more than one repository. Again, this was done to reduce the likelihood that all the images were passed through the same image processing algorithms. Table 2 provides a summary of the repositories from which images were selected for possible inclusion in the testing or training sets and how many images of each type were selected from that source. The images selected from each source had an equal likelihood of being included in either the testing or training set. However, once an image had been included in one of the sets, it was no longer eligible for inclusion in the second set.

The samples for the experiment were stratified by file type. This was done to ensure that all the file types that were to be trained on the system were adequately represented. As shown in Table 2, in some cases the samples were removed from consideration for training and testing. These samples were removed as a double stratification measure to ensure that one size or shape of image was not over-represented. Thumbnails were eliminated since they may have biased the sample toward images of that size and shape.

Table 2

Image sources by file type

	JPEG		PNG		BMP		GIF		TIFF		
Source	D	U	D	U	D	U	D	U	D	U	Total
usenet-replayer.com	620	611	1090	1082	1121	1103	1553	1538	734	734	5068
wpclipart.com	0	0	15189	15189	0	0	0	0	0	0	15189
flickr.com	23191	9038	95	95	0	0	161	161	0	0	9294
cs.sfu.ca*	0	0	0	0	0	0	0	0	223	223	223
nps.gov	0	0	0	0	0	0	0	0	31	31	31
sipi.usc.edu*	0	0	0	0	0	0	0	0	215	215	215
TOTAL	23811	9649	16374	16366	1121	1103	1714	1699	1203	1203	

D	Number of images downloaded
U	Number of images that were considered for the testing or training sets
*	Test image archives

During the initial run of the data for this investigation, it became obvious that it was necessary to impose further controls on the samples. The results of the partial data run showed that the size of the individual files was biasing training results. File types that tended to be larger (such as a TIFF image) were being learned disproportionately to the other file types. This phenomenon occurred because the larger files supplied more training data to the network. To eliminate this bias, the experiment was stopped and a file size control was added before the experiment was restarted. This control was designed to reduce the bias caused by the file size differences and better represent each file type. The file size control specified that only the first five kilobytes (first 10 blocks) of each file would be fed to the networks for training and testing. The 10 block limit was chosen since most of the files were at least 5 kilobytes in size.

3.2 Software and Hardware

The neural networks for the experiment were constructed and tested using the Fast Artificial Neural Network library (or FANN). The FANN library was picked because it provided a flexible design that met the requirements of the project. It was necessary to use a beta version (version 2.1.0 beta) of FANN since the release version at the time of the experiment was known to have issues in some configurations. The FANN graphical interface was used for the neural network design and much of the testing. Command line interfaces were used for the training and the remainder of the testing to enable logging results more easily.

The FANN library was used in floating point mode. The smallest network used in this experiment had over 250 nodes and ten times as many connections between the layers. This meant that over two thousand floating-point calculations were required for each training or testing record. Since training required processing many times more records than testing, the neural networks were trained on four separate desktop computer systems. Since the testing did not require as much processor activity, it was completed on a single laptop. Table 3 summarizes the configuration of each of the training and testing systems.

Table 3

Configuration of training and testing hardware

System	Chassis	Processor	RAM	HD	Usage
blackbeast	Desktop	Athlon64 2800	2.0	750.0	Training Character Frequency (10)
slimjim	Desktop	Sempron 2800	1.0	60.0	Training Character Frequency (20)
mythtv	Desktop	Athlon64 3200	1.0	120.0	Training Character Frequency (30)
newhorizon	Desktop	Pentium D 820	1.0	250.0	Training Raw (10, 20, 30)
graydawn	Laptop	Pentium M 1.6GHz	1.5	80.0	Testing All

3.3 Neural Network Construction

Two feature extraction methods were used: raw filtering and character code frequency. For both methods, the file was divided into blocks of 512 bytes. As noted in prior sections, only the first 10 blocks of each file were processed. Raw filtering essentially took each byte from a block in the input file and supplied it as an input into one neuron of the neural network. It was assumed that this technique would be most useful for files that had regularly spaced data structures (for example 64 bytes for each data structure) since the patterns would occur at regular intervals.

Character code frequency filtering was borrowed from the cryptographic community. The inputs to the character frequency filtered neural networks were comprised of the number of times each character code was used in each of the blocks in the file to be identified. This technique was assumed most useful for files that had non-regularly spaced file structures that were started with uniform identifiers.

There were many possible methods of constructing the networks and it was difficult to determine which layout would be most appropriate for this research. Consequently, the research examined several different node counts at each juncture to determine their effectiveness. The research used neural networks with two different input node counts. Networks that were created for use with raw input filtering had 512 input nodes and the character-code frequency networks had 256 input nodes. There were significant amounts of random data fed to the networks because of the number of nodes present at the input layer. Theoretically, determining optimal input clusters would create better results. However, this trade-off was picked in the interest of creating a general network structure that would be useful on a wide variety of files.

There was only a single hidden layer in the neural networks. This hidden layer had nodes numbering ten, twenty or thirty. The hidden node count was limited to no more than thirty because it was expected that there would be a point at which additional hidden nodes would provide no additional benefit. This expectation was based on a phenomenon mentioned by Liang, Moskowitz, and Yih (1992), where extra hidden layer nodes increase computational complexity without providing substantial value. Output node count was held constant at five to allow for representing the types of input files as a bit sequence.

Duda, Hart and Stork (2001) mentioned that a linear activation function would effectively negate the benefits of a tri-layer network. For this reason, the activation functions were set to hyperbolic tangent (tanh) for all the networks. These functions were selected since they were the most accurate available in the library of supported functions (Fast Artificial Neural Network Library, n.d.).

In artificial neural networks, the error encountered during each iteration is used to adjust the weights of the individual neurons in the network. This adjustment is referred to as backpropagation (Ripley, 1996). In this research, the error was adjusted using tanh which is generally more effective than a standard linear error function (Fast Artificial Neural Network Library, n.d.).

Based on Duda, Hart and Stork's (2001) suggestions, the network learning rate was set to 0.1 anticipating that a small value would reduce the variability during the training and would also reduce the likelihood of consistently overshooting the correct connection weight. Although the software allowed for more advanced training methods, the author chose to use the incremental error backpropagation algorithm. This decision was made to increase the likelihood of usable results since the other available algorithms are only useful under specific circumstances (Fast Artificial Neural Network Library, n.d.).

3.4 Inputs and Outputs

As mentioned earlier, a variable number of input nodes were fed through a hidden layer into a fixed number of output nodes. The input nodes were fed using *n* consecutive bytes from the file being identified or from the character frequency table (where *n* was the number of input nodes in the neural network). Because of software limitations and general neural network limitations, the bytes were processed before being submitted at the artificial neural network input nodes. During preliminary testing, it was found that the software only allowed decimal values to be provided for inputs and outputs, so the input byte-codes were divided by one thousand to accommodate this limitation. Additionally, the input values were mathematically shifted to center around zero with byte-code 128 being the center. This was done to ensure that null values in the files (byte-code 0) would not adversely affect the output node values.

The output node values were treated like bits in a byte with a positive 0.9 indicating "on" and a negative 0.9 indicating "off." This range was chosen to reduce the likelihood of falsely classifying a file that was subtly altered. While +/-0.9 was the intended output value, any output greater than either 0.5 or less than -0.5 was assumed to be on or off respectively. This was in-line with Duda, Hart and Stork's (2001) recommendations that the chosen output values should be less than the saturation point of the output nodes since

Table 4

Expected output node value by image type

Type	Bit 1	Bit 2	Bit 3	Bit 4	Bit 5
JPEG	On	Off	Off	Off	Off
PNG	Off	Off	On	Off	Off
BMP	Off	Off	Off	Off	On
GIF	Off	Off	Off	On	Off
TIFF	Off	On	Off	Off	Off

the neural network outputs nodes might never reach the saturation point. Table 4 summarizes the bit values that were assigned to each file type.

3.5 Training the Networks

Before training, the neural network node weights were initialized to random values between 0.1 and -0.1. These minimum and maximum values were chosen to ensure that the nodes were not initialized to large values. According to Ripley (1996), large initial weights could have caused the output node values to be saturated at the outset (too close to zero or one).

There were two training data sets. Both sets were constructed using the same input samples; the only difference between the sets was in the filtering mechanism. The first data set consisted of the training samples filtered through the raw filtering mechanism. The second data set consisted of training samples that were processed with the character frequency filter. As noted earlier, due to a change in experimental protocol, only the first 10 blocks of 512 bytes each were taken from each file, and then filtered and processed for the training sets.

The neural network software allowed for both a variable number of training iterations and a fixed limitation (Fast Artificial Neural Network Library, n.d.). The variable limit used the mean square error (MSE) between the expected output neuron values and the actual output neuron values for that iteration (Fast Artificial Neural Network Library, n.d.). In this research, the variable limit was set to a MSE of 0.001 and the fixed limit was set to 20,000 iterations. If either limit were encountered, the training would stop. The variable limitation was designed to help prevent the neural network from being overfitted. An overfitted network effectively memorizes the individual data points in the training set rather than recognizing the overall pattern (Ripley, 1996).

4 RESULTS

4.1 Network Training Iterations

As was detailed in Chapter 3, the training was set to stop automatically after 20,000 rounds or an MSE of 0.001 was reached. This double stopping condition was designed to provide protection against overfitting. However, the chosen MSE of 0.001 was not reached by any of the networks before they had completed the 20,000 epochs. Since all of the networks reached 20,000 epochs, it was impossible to use the number of epochs as a predictor of how effectively a given network configuration was able to learn the training data.

4.2 Ending Training MSE and Input Node Count

Table 5 provides a summary of the ending MSE of each of the network configurations. Figure 1 provides a graphical summary of the same data. As can be seen from the graph, on average, the raw input network configurations exhibited a much higher error than that shown by the character frequency code filtering network configurations. A two-tailed two way ANOVA was done to assess whether the number of nodes in each layer had an effect on the MSE of the networks at completion (Table 6). The number of input nodes resulted in an $F(1, 5) = 79.4$, $p = 0.012$. This indicated that the number of input nodes had a significant impact on input node configuration and the end of training MSE.

Table 5

Ending mean square error (MSE) from training

Input Filtering	Hidden Node Count 10	20	30	Mean
Character Frequency	0.26	0.22	0.21	0.23
Raw	0.39	0.35	0.30	0.35
Mean	0.33	0.28	0.26	

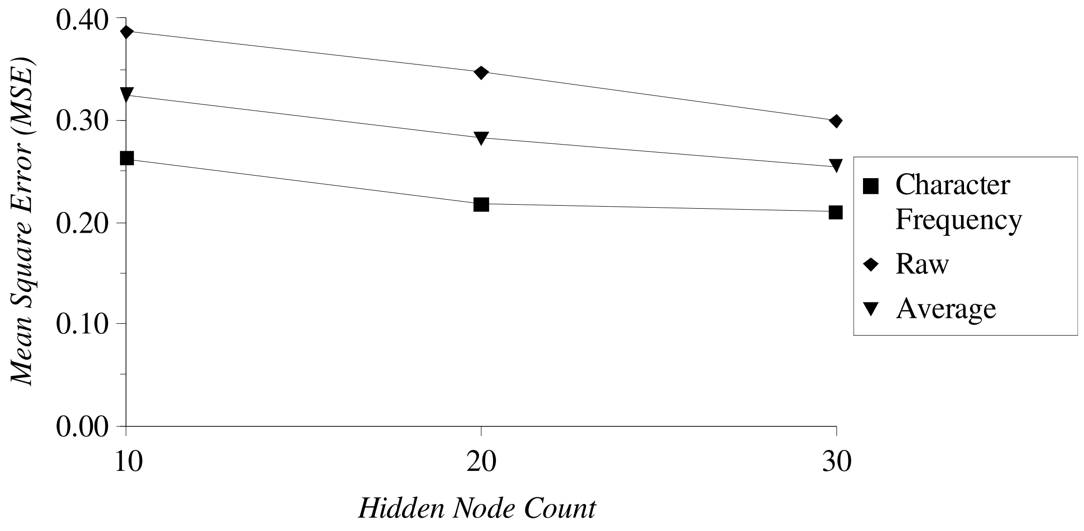

Figure 1

End of training mean square error (MSE)

4.3 Ending Training MSE and Hidden Node Count

Table 5 also provides a summary of the ending MSE based on the number of hidden nodes. As can be seen from Figure 1, as the number of hidden nodes increased, the overall MSE decreased. In a two-tailed two way ANOVA, the number of hidden nodes resulted in an $F(2, 5) = 10.1$, $p = 0.090$ (Table 6). At an alpha of 0.05, there was no significant relationship between the number of hidden nodes and the end of training MSE. Normally, one would expect that the number of hidden layer nodes would have a significant impact on the accuracy of the network (Duda, Hart, & Stork, 2001) and would therefore affect the ending MSE.

Table 6

Multiple analysis of variance for node count and training MSE

Source	*df*	*F*	□	*p*
Input Node Count	1	79.38	18.51	0.01*
Hidden Node Count	2	10.06	19.00	0.09*
Error	2			

* $p < 0.05$

4.4 File Detection Anomaly

A possible anomaly in file type detection was detected when compiling the study results. According to FTK, around one percent of the files were misclassified during the training and testing of the networks. This explained why the percentage of unaltered files detected by FTK was less than 100%. The possibility of such an error was accepted for several reasons. First, this issue was found to affect less than one percent of the training samples overall and affected no one file type more than 2.1%. This amount was considered small enough that it would not decrease the learning capability of the network. Additionally, in many cases the files were downloaded based on specific file type information provided by the source of the files. For example, the bitmaps that were downloaded from usenet-replayer.com were all classified as “image/bmp” by the site's internal recognition algorithm.

4.5 Unaltered File Detection Performance

The neural networks were assessed to determine their effectiveness at detecting unaltered files. A file was detected properly if the average of the output bits for the file's blocks were in the appropriate range for that file type. This detection rate was compared against that of Access Data’s Forensic Toolkit (FTK) Version 1.7 to determine the viability of neural networks in this configuration. Table 7 provides a summary of how well the networks detected unaltered samples of each of the file types and baselines the detection rate against that of FTK.

Table 7

Detection rates for unaltered test files

	Hidden	JPG	PNG	TIF	GIF	BMP
Raw filtering	10	6.2%	7.2%	36.0%	2.0%	17.0%
	20	7.6%	8.6%	37.0%	9.0%	20.0%
	30	10.6%	12.8%	50.0%	1.0%	29.0%
Character code filtering	10	16.0%	35.8%	49.0%	0.0%	31.0%
	20	50.2%	43.8%	57.0%	0.0%	41.0%
	30	42.6%	46.6%	60.0%	0.0%	37.0%
FTK		99.0%	100.0%	99.0%	100.0%	99.0%

As with the end of training MSE, in almost every case, the raw input filtering networks performed more poorly than the character code frequency networks. Overall, the performance of the networks was not very good. None of the networks achieved a mean file detection rate above 50%. The best performance occurred with TIFF files. On average, the recognition rate for TIFF images was almost 15% above the recognition rate of any other file type. The worst performing file type was GIF. The GIF image detection rate was several orders of magnitude lower than any of the other file types. Even the best performing network had no more than a ten percent success rate at detecting GIF images.

The results for character code frequency networks seemed to be affected differently by the hidden node count than the raw filtering networks. For these networks, once the hidden node count hit thirty, the performance actually declined. Therefore, it would seem that an increase in the number of hidden nodes might not have increased the performance of the network. This idea was also supported by the results seen in Figure 1. Once the neural network hidden node count reached thirty, the training MSE leveled off. Therefore, an increase in the number of hidden nodes for character code frequency networks might not have provided a boost in neural network performance.

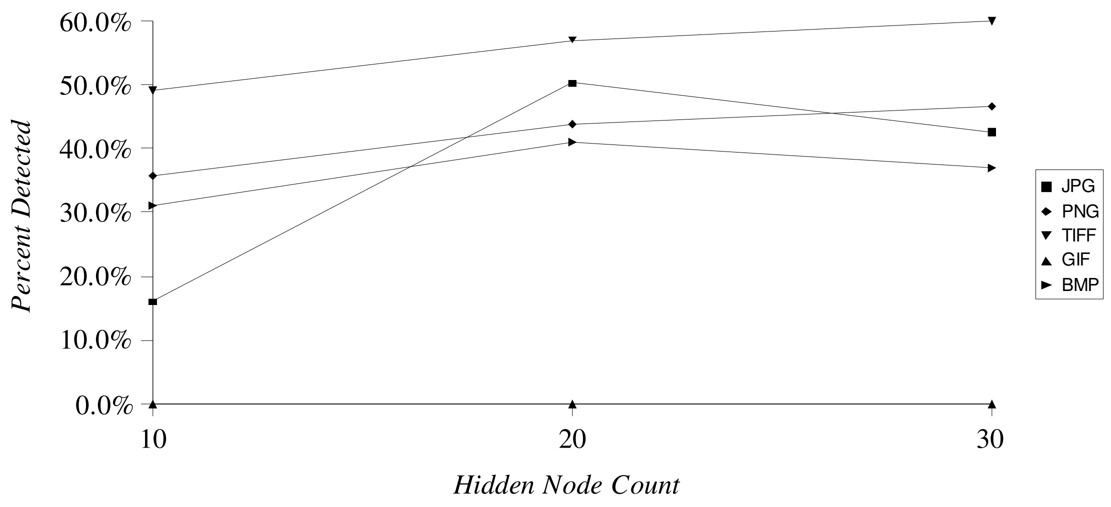

Figure 2

Percent of character code filtered files detected by type (unaltered files)

When the detection rate for each of the network types was compared based on file type (Figures 2 and 3), a completely different picture emerged. As hidden neuron count increased, some of the file types appeared to have increasing detection rates. However, this general trend was not the same for every file type. Figure 2 shows that the JPEG and BMP accuracy decreased for the character code frequency network with 30 hidden nodes. As could be seen in Figure 3, GIF image accuracy declined for the raw filtering network with 30 hidden nodes.

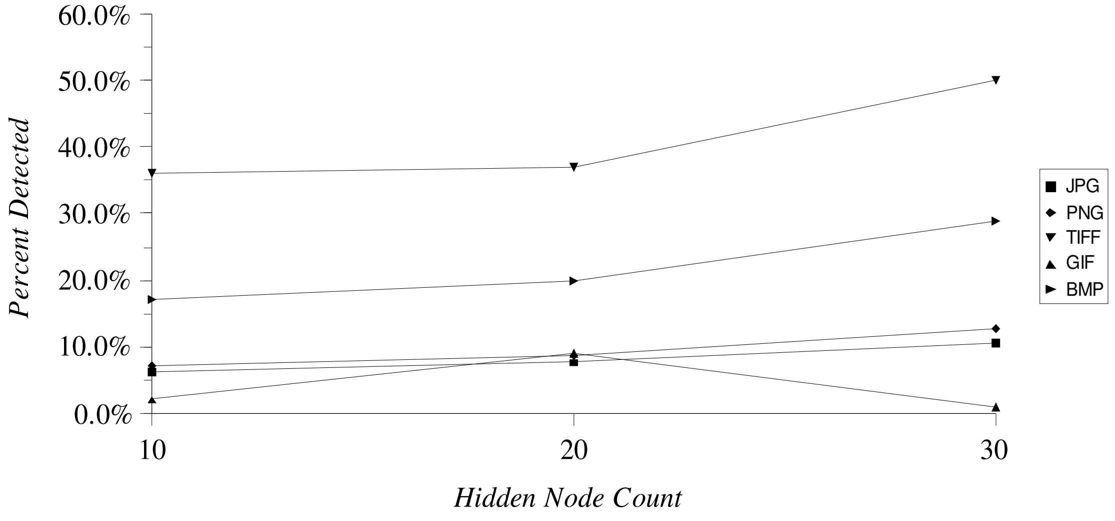

Figure 3

Percent of raw filtered files detected by type (unaltered files)

4.6 Altered File Detection Performance

Table 8 details the performance of the neural networks when files were altered. Figures 4 and 5 provide a graphical summary of the same data. For most file types, the performance decreased. This can be seen in the difference between the detection percentage from Tables 7 and 8. The difference in detection rate was no more than 10% for the raw filtering networks and no more than 8% for the character frequency networks.

Table 8

Detection rates for altered test files

	Hidden	JPG	PNG	TIFF	GIF	BMP
Raw filtering	10	5.2%	7.4%	28.0%	1.0%	17.0%
	20	2.8%	8.0%	32.0%	9.0%	21.0%
	30	4.4%	11.8%	40.0%	1.0%	27.0%
Character code filtering	10	14.4%	35.6%	47.0%	0.0%	37.0%
	20	50.8%	42.8%	57.0%	0.0%	48.0%
	30	41.8%	45.6%	61.0%	0.0%	45.0%
FTK		98.0%	98.2%	99.0%	100.0%	91.0%

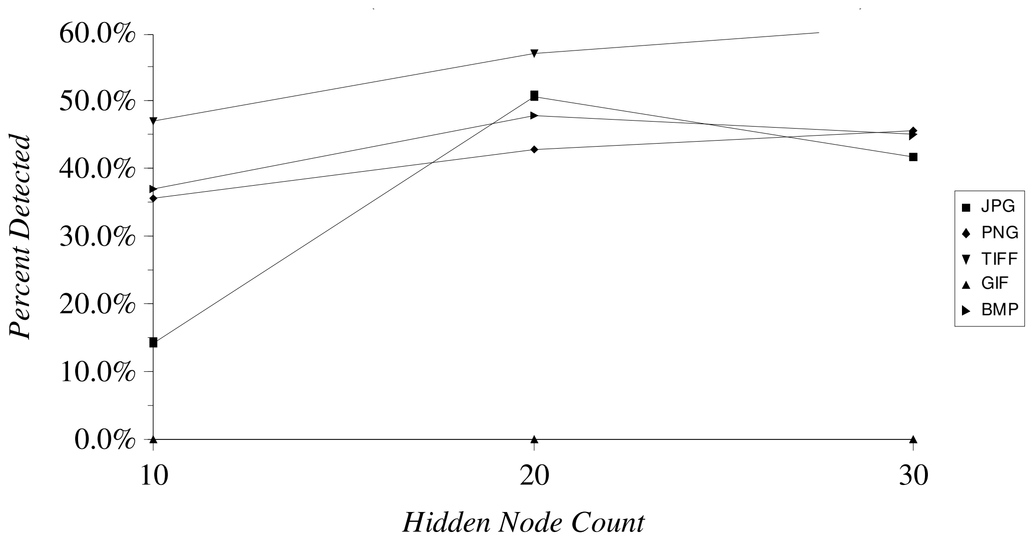

Figure 4

Percent of character code frequency filtered files detected by type (altered files)

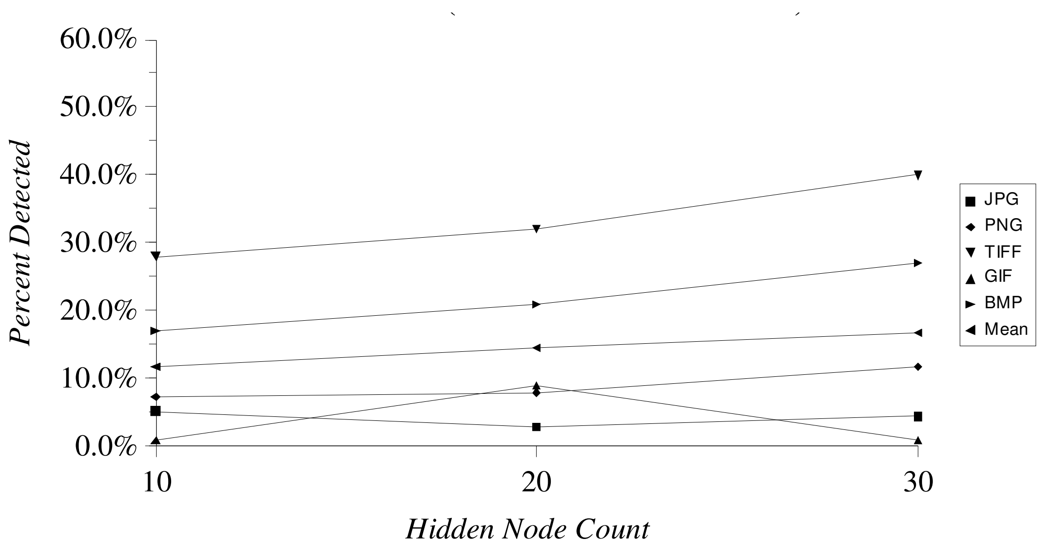

Figure 5

Percent of raw filtered files detected by type (altered files)

4.7 Test File MSE and Node Count

Tables 9 and 10 detail the testing MSE that resulted from each of the network configurations. Table 9 summarizes the MSE for unaltered files; Figure 6 provides a graphical summary of the same data. The MSE for altered files is slightly higher and is detailed in Table 10 and Figure 7.

Table 9

Mean square error (MSE) from unaltered file test

Input Filtering	Hidden Node Count			Mean
	10	20	30	
Character Frequency	0.29	0.24	0.24	0.26
Raw	0.46	0.57	0.66	0.56
Mean	0.37	0.40	0.45	

Table 10

Mean square error (MSE) from altered file test

Input Filtering	Hidden Node Count 10	20	30	Mean
Character Frequency	0.29	0.24	0.25	0.26
Raw	0.46	0.58	0.67	0.57
Mean	0.38	0.41	0.46	

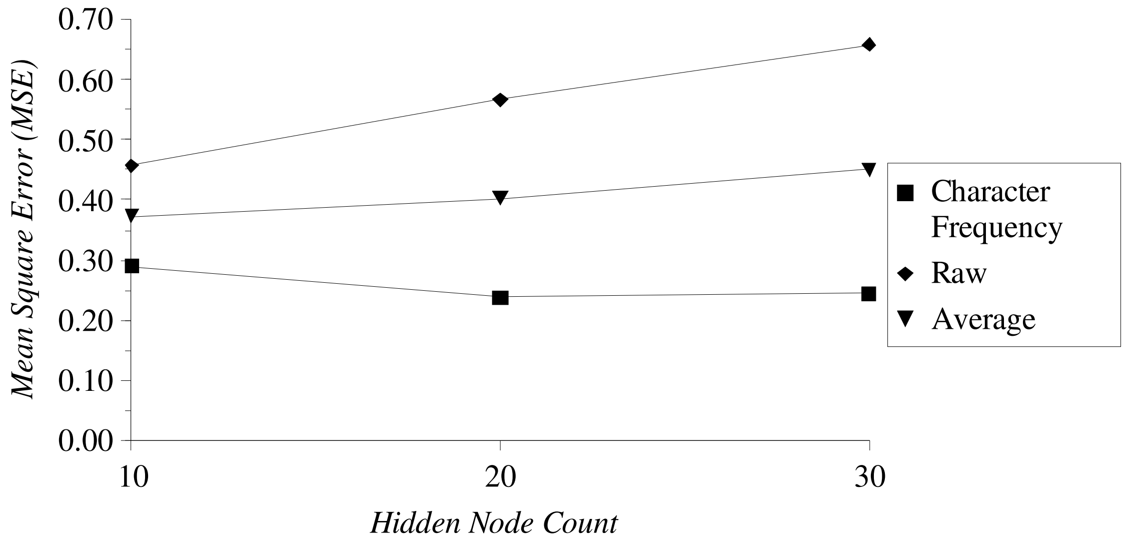

Figure 6

Unaltered file test mean square error (MSE)

A two-tailed two factor ANOVA was performed on the unaltered test file MSE data to assess whether the number of nodes in each layer had an effect on the MSE of the networks at completion. The number of input nodes resulted in an *F*(1, 5) = 17.5, *p* = 0.05. At an alpha level of 0.05, there was a significant relationship between the input node count and the unaltered test file MSE. The number of hidden nodes resulted in an

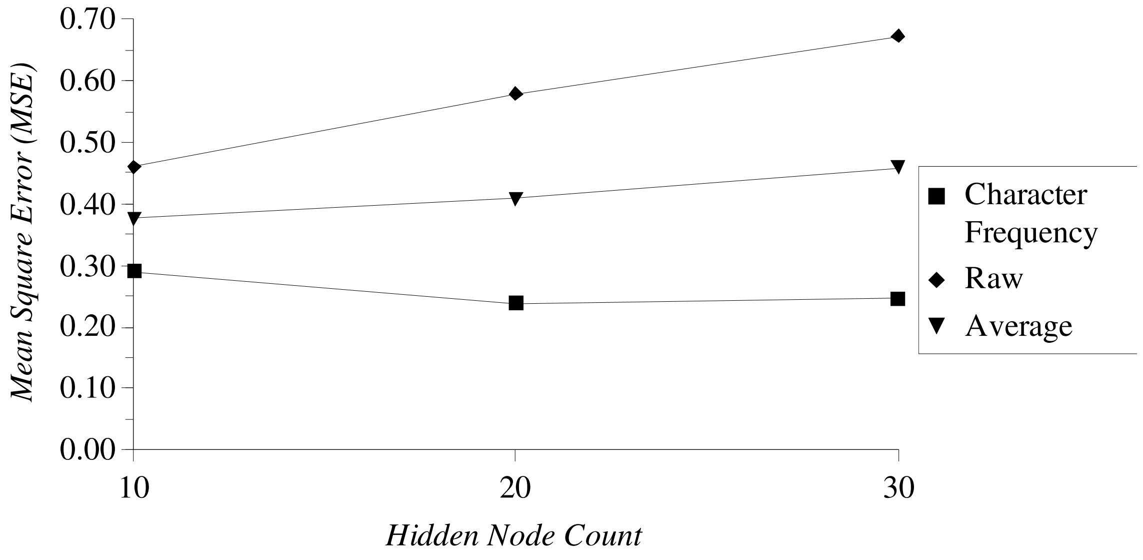

Figure 7

Altered file test mean square error (MSE)

F(2, 5) = 0.40, *p* = 0.72 (Table 11). At an alpha level of 0.05, there was no significant relationship between the hidden node count and the unaltered test file MSE.

A two factor ANOVA was also performed on the altered test file MSE data to assess whether the number of nodes in each layer had an effect (Table 12). The number of input nodes resulted in an F-ratio of *F*(1, 5) = 17.03, *p* = 0.05. At an alpha level of 0.05, there was a significant relationship between the input node count and the unaltered test file MSE. The number of hidden nodes resulted in an *F*(2, 5) = 0.42, *p* = 0.71. At an alpha level of 0.05, there was no significant relationship between the hidden node count and the unaltered test file MSE.

Table 11

Two way analysis of variance for node count and unaltered file MSE

Source	*df*	*F*	□	*p*
Input Node Count	1	17.53	18.51	0.05*
Hidden Node Count	2	0.40	19.00	0.72*
Error	2			

* $p < 0.05$

Table 12

Two way analysis of variance for node count and altered file MSE

Source	*df*	*F*	□	*p*
Input Node Count	1	17.03	18.51	0.05*
Hidden Node Count	2	0.42	19.00	0.71*
Error	2			

* $p < 0.05$

5 DISCUSSION

5.1 Number of Training Iterations

Why all the networks ran for the full 20,000 training epochs is not readily apparent. There were four viable explanations. One could argue that the number of iterations might not have been sufficient. Increasing the number of training epochs might have increased the likelihood of reaching a lower training MSE. However, this solution is not as straightforward as it would seem. The appendix shows the relationship between training epochs and mean square error. As can be seen in the graphs, each of the networks reached a point where the training mean square error (MSE) started to decrease at almost infinitesimal rates. Therefore, without further experimentation it would be very difficult to determine whether a change in the number of epochs could have caused the neural networks to reach an MSE of 0.001.

A second explanation could be that an MSE of 0.001 was too low of a stopping condition. In retrospect, this argument would seem to carry some weight. However, in this case it would be difficult to determine what a better stopping MSE would be. As can be seen from the graphs, none of the networks ended training at exactly the same MSE. The networks using raw input filtering did not reach as low of an error as the networks using counted input filtering. So, if one were to pick an MSE such as 0.2, the raw input networks would still have completed the maximum epochs. If a higher MSE such as 0.3

was picked, one could argue that the character frequency input networks were not adequately trained. Therefore, determining a proper stopping MSE retroactively would still be fraught with difficulty.

Another possible explanation for the number of epochs would be to say that the neural networks were incapable of correctly approximating the data. One could argue that there were not enough training entries, the network design was inadequate, or the data was too random for the network to learn. Arguing that there were insufficient training entries appears to be incorrect since there were over 130,000 records in the training set. Simply saying that the network configuration was inadequate would neglect the possibility that the data was too random for any network configuration to reach an MSE of 0.001. Alternatively, blaming the data randomness neglects the possibility of better network designs. So, assessing how these two variables related to the ending MSE would be difficult and would require a new experiment design.

The final possible explanation is that there should have been an additional stopping condition to prevent overfitting. Duda, Hart and Stork (2001) suggest separating out a portion of the training set to validate the network concurrently with the training. According to these authors, overfitting would be detectable by looking at a graph of the validation set's MSE across rounds. For example, the network would be trained for five epochs, and then it would be tested against the validation set to determine whether overfitting was occurring. The training would be stopped if a graph of the MSE across epochs showed an increase (rather than the expected decrease). It would be assumed that the rise in error was caused by the neural network memorizing the training data points rather than generalizing them. At face value, this type of stopping condition would sound like a good solution. However, this type of stopping condition may be illegitimate. According to Ripley (1996), frequently "...after an initial drop the error on the validation set rises slowly...then falls dramatically to a small fraction of the previous minimum" (pp. 154-155). Therefore, this type of stopping condition could have accidentally stopped training when the neural network has not finished learning.

5.2 Input Node Count and Filtering Method

According to the ANOVA, there was a significant relationship between the ending MSE and input node count. This result was not surprising given the significant gap between the ending MSE of the two different network input configurations (Figure 1). From this data, we can conclude that the input configuration of the networks must have had an effect on the ability of the network to be trained. The networks that had 256 input nodes had statistically higher training performance and were more capable of approximating the training data presented to them.

This result did imply that the character frequency networks had better learning performance. However, it did not signify that the difference was completely caused by the actual filtering method used to provide the data. The degree of relationship between ending MSE and input filtering method could not easily be separated from that caused by the actual number of input nodes. The significance in the relationship may have simply come from the number of input nodes in the networks. Based on *purely anecdotal* evidence, some of the significance may have been from the difference in filtering method. The training files were compressed when they were moved between computers. The character frequency filtering files were compressed two times more efficiently.

Compression works by removing repeated patterns. It appears that character frequency filtering had more discernible patterns. However, further testing would be needed to assert definitively that the relationship completely stems from the input filtering method.

It seems that at least one of the file types (GIF) benefited from a different input filtering method than the other types of files. From anecdotal evidence, it appeared that a fair amount of the performance difference was related to the filtering method. But, it is common knowledge in the neural network community that input node count has a performance impact. As a result, it was impossible to quantify how much of the performance of the networks was related to the input node count and how much was related to the actual filtering mechanism chosen.

5.3 Hidden Node Count

The hidden node count in the neural networks seemed to have no statistically significant effect on the overall performance of the networks. This phenomenon could possibly be explained by saying that the neural networks had an incorrect number of hidden nodes or layers. If the number of hidden nodes were several orders of magnitude too low or too high, the networks would all be unable to adequately represent the training data. In this type of situation, one might conclude that the error between the networks was relatively minor since none of the networks were successful at approximating the data. In this case, it appears that the number of hidden nodes or layers may have been too small. This assessment was based on the general direction of the MSE as shown in Figure 1. However, it would be incorrect to assert that this was the case given the fact that the ANOVA found no significance.

However, as noted earlier, there were several possible reasons for this result. First, one could argue that the networks were not trained sufficiently. Another argument would state that the number of nodes in the neural network was dramatically above or below the number that were actually needed. Another likely explanation is that the different file types responded differently to the hidden layer nodes. Some file types benefited from additional nodes, while others were not benefited by these nodes. Figures 2 through 5 seem to indicate that this problem may have occurred.

5.4 File Detection Rates

TIFF images were detected most effectively out of all the file types. This effectiveness could be attributed to the ability of the data to be classified by the neural network. TIFF file types would appear to have data that could be recognized consistently by the neural networks. The patterns in these types of files would appear to be more consistent. One caveat with this assessment should be noted. As can be seen from Table 2, some of the TIFF images were drawn from test image archives. If the images in these archives had a more consistent layout than the images from other sources, then this could have introduced a bias toward TIFF images. However, as noted in Table 1, TIFF images were less represented than either JPEG or PNG images. Therefore, one would expect that any problems introduced by using these image sources would have been minimized by the reduced number of total samples used during training.

GIF images were detected correctly the least out of all the file types. This could be the result of GIF images containing more apparently random information than the other file types. The GIF images caused a negative skew in the mean detection rate for the character frequency networks. In the raw filtered networks, the negative effect of the GIF files was balanced by the TIFF detection rate. This effect could be seen in the difference between the mean (biased estimator) and median (unbiased estimator) as shown in Table 8. It should also be noted that GIF images were the only type of image whose actual detection rate was higher with the raw input filtering networks. This lends some credence to the initial supposition that different filtering methods would be better for different types of files.

None of the file types were correctly detected more than 60% of the time. However, this is not to imply that the neural networks were incapable of detecting file types. Since there were five different file types to be detected, the random chance detection would be no higher than 20%. Therefore, we could assume that neural networks were more successful than random chance at detecting the file types.

There were several possible explanations for the poor performance of the networks. First, as noted in prior sections, one could argue that the neural network hidden node configuration was incorrect. If this assessment were correct, the networks would have been able to generalize the data with which they were presented. At face value, this argument initially seems plausible if one only considers the earlier end of training MSE findings.

However, the difference in detection trends between file types that could provide a possible explanation for much of the learning and detection difficulty for the neural networks. From Figures 2 through 5, it would appear that some file types benefited from additional hidden nodes, while for others, additional nodes were detrimental. This could mean that these file types required a different neural network layout than the other file types. Therefore, the inclusion of these files in larger neural networks actually reduced the performance of the network. So, rather than the network not containing enough nodes to adequately represent the data, it appears that some of the low learning rate could be blamed on the difference in optimal parameters for each of the file types.

The neural network performance for altered files was generally lower than that for unaltered files. This decrease in detection rate was expected because the files no longer matched exactly with a standard pattern for that particular file type. This result lends credence to the notion that neural networks are resilient against changes in the files themselves. If a well performing neural network configuration could be designed, it appears that it would be fairly well suited to changing data.

However, when comparing the altered file detection rate with the unaltered detection rate, there was a slight anomaly in the data. While almost all the file detection rates decreased with altered files, the BMP file detection rate actually increased. At first, it was thought that this was the result of a data recording error. However, this was not the case. Therefore, there must have been another reason for the increase in effectiveness at detecting altered bitmaps.

One possible explanation assumes that the neural networks were not successful at fully generalizing the training data (as noted in several other places in this research). Depending on how well the data was generalized, there could have been an abundance of "borderline cases." These borderline cases would fall just barely outside the detection criteria mentioned earlier. For example, the average of a single output bit might be just below the cutoff of 0.5. In these borderline cases, any alteration of the file might be enough to shift the average bit toward either direction of the cutoff point. To assess whether this might have been true, the list of altered files that were detected correctly was spot-checked against the list of unaltered files. In every checked instance, the unaltered files were a borderline case. Therefore, it appears that the anomaly may have been caused by the learning rate of the networks, rather than any data error.

6 CONCLUSION, RECOMMENDATIONS, AND FUTURE RESEARCH

6.1 Conclusion

In the configurations tested in this paper, neural networks were not a practical method for detecting file types in the real world. However, this research provided a valuable foundation for future studies. Since the detection rate of the neural networks was much better than random chance alone, we may assume that the networks were able to recognize some type of pattern in most of the files' data. This lends credence to the idea that data patterns may be a future way of identifying suspect files.

Additionally, the neural networks' performance was stable even when the files were altered. Therefore, while the neural networks as configured in this research were not effective, there is still some possibility that neural networks will provide a viable method for detecting file types in the future. As such, this research has also pinpointed several possible avenues for future experimentation.

6.2 Recommendations and Future Research

Several possible research projects could center on increasing the effectiveness of the neural network configurations presented here. As detailed throughout the project, it was known from the outset that this project might not result in optimal neural network configurations. Based on the results of the study, there appear to be several ways in which the file type detection rate of neural networks could be improved.

One possible study could look at whether adjusting the number of training epochs would have an impact on the effectiveness of the networks. As noted in the first section of Chapter 4, the number of training iterations may not have been sufficient. Based on results of the study as a whole, the number of epochs appeared to have been adequate. However, an additional study could determine whether this assumption was correct. This type of investigation would probably be most effective if it concentrated specifically on one of the neural network filtering methods.

Another possible future study could attempt to isolate how much of the significance between neural network input configuration and performance was caused by the actual input filtering method. This study was unable to quantify the difference between the performance change due to input node count and the performance change due to filtering method. While there was some evidence that the filtering method played a role in the network effectiveness, a future study could better isolate the variables. One possible method of isolation would be to reduce the raw input filtering to blocks of 256 bytes. However, this might have an unforeseen impact on the training because of the difference in the number of blocks presented to the networks.

The most promising area for future investigation would to isolate the training by file type. Some evidence indicated that the filtering method affected the file types differently. There was also evidence that the number of hidden nodes influenced the file types in separate ways. Therefore, it would be useful to split out the file types and train each file type on individual networks to see if the performance of the networks increases.

If the performance were to increase, investigation could pursue identifying the optimal configuration for each individual file type. Once optimal configurations were found, it might be possible to arrive at standard configurations that have acceptable detection rates for a variety of files.

REFERENCES

REFERENCES

Bishop, C. M. (1995). *Artificial neural networks for pattern recognition.* New York: Oxford University Press.

Carrier, B. (2005). *File system forensic analysis.* Upper Saddle River, NJ: Addison Wesley Professional.

Darwin, I. F. (1999). File - determine file type. *UNIX man pages.* Retrieved November 17, 2006 from http://unixhelp.ed.ac.uk/CGI/man-cgi?file

Duda, R. O., Hart, P. E., & Stork, D. G. (2001). Multilayer neural networks. *Pattern classification (2nd Ed.).* New York: John Wiley & Sons.

Fast Artificial Neural Network Library. (n.d.). *Fast Artificial Neural Network Library (FANN): Version 2.0 reference manual.* Retrieved April 21, 2006 from http://leenissen.dk/fann/html/files/fann-h.html.

Federal Bureau of Investigation. (n.d.) *Innocent images national initiative.* Retrieved November 27, 2006 from http://www.fbi.gov/publications/innocent.htm

Foster, J. C., & Liu, V. (2005). *Catch me, if you can. . .* Retrieved September 30, 2005 from http://www.blackhat.com/presentations/bh-usa-05/bh-us-05-foster-liu-update.pdf.

Geiger, M. (2005). Evaluating commercial counter-forensic tools. *Digital Forensic Research Workshop.* Retrieved November 27, 2006 from http://www.dfrws.org/2005/proceedings/geiger_couterforensics.pdf.

Harris, R. (2006, August). Arriving at an anti-forensics consensus: Examining how to define and control the anti-forensic problem. *Digital Forensic Research Workshop.* S44-S49. Retrieved November 16, 2006 from http://www.dfrws.org/2006/proceedings/6-Harris.pdf

Heimbach, M. J. (2002). *Testimony of Michael J. Heimbach, Crimes against children unit, FBI.* Retrieved November 27, 2006 from http://www.fbi.gov/congress/congress02/heimbach050102.htm.

Hernandez, A. E. (2006). *Statement of Andres E. Hernandez director of the sex offender treatment program, Federal Correctional Institution - Butner, NC.* Retrieved November 27, 2006 from http://www.projectsafechildhood.gov/HernandezTestimonyCongress.pdf

Kinsella, J. (2006). The impact of disk fragmentation. *Windows IT pro.* Retrieved November 17, 2006 from http://files.diskeeper.com/pdf/ImpactofDiskFragmentation.pdf.

Kruse, W. G., II & Heiser, J. G., (2004). *Computer forensics: Incident response essentials.* Upper Saddle River, NJ: Addison Wesley Professional.

Liang, T., Moskowitz, H., & Yih, Y. (1992). Integrating neural networks and semi-Markov processes for automated knowledge acquisition: An application to real-time scheduling. *Decision Sciences, 23*(6). 1297-1314. Retrieved February 21, 2006 from ProQuest database.

Mallery, J. R. (2001). *Secure file deletion, fact or fiction?.* Retrieved February 19, 2006 from http://www.sans.org/rr/whitepapers/incident/631.php.

Mena, J. (2003). *Investigative data mining for security and criminal detection.* Burlington, MA: Butterworth-Heinemann.

Richard, III, G. G., Roussev, V. (2005). Scalpel: a frugal, High performance file carver. *Digital Forensic Research Workshop.* Retrieved November 17, 2006 from http://www.dfrws.org/2005/proceedings/richard_scalpel.pdf.

Ripley, B. D. (1996). Feed-forward neural networks. *Pattern recognition and neural networks.* Cambridge, U.K.: Cambridge University Press.

Shannon, C. E. (1948). A mathematical theory of communication. *The Bell system technical journal, 27.* 379-423, 623-656. Retrieved November 17, 2006 from http://cm.bell-labs.com/cm/ms/what/shannonday/shannon1948.pdf.

APPENDIX

APPENDIX

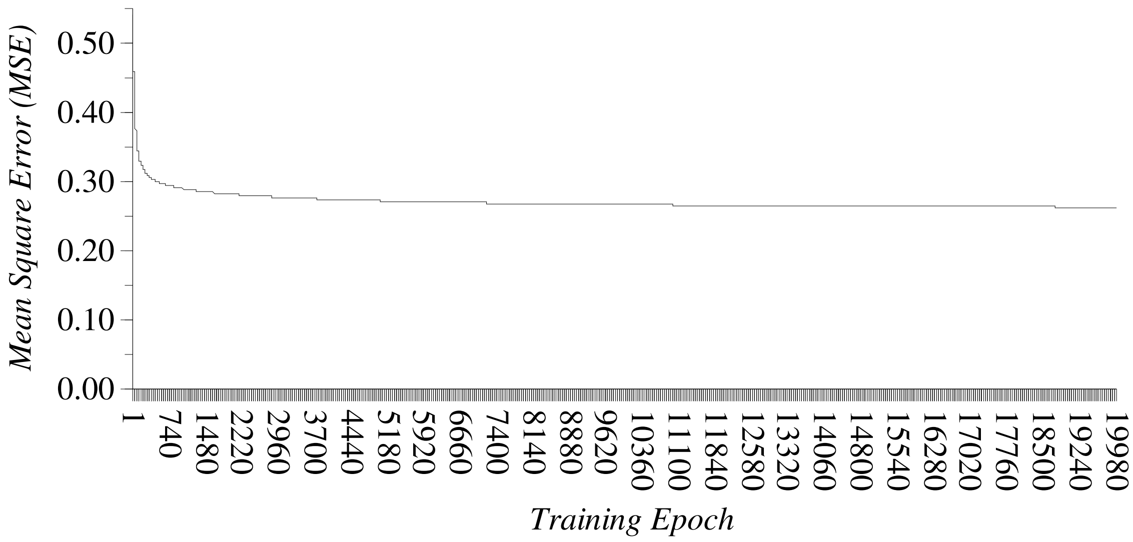

Figure A.1

Mean square error across training epochs for the character code frequency neural network with 10 hidden nodes

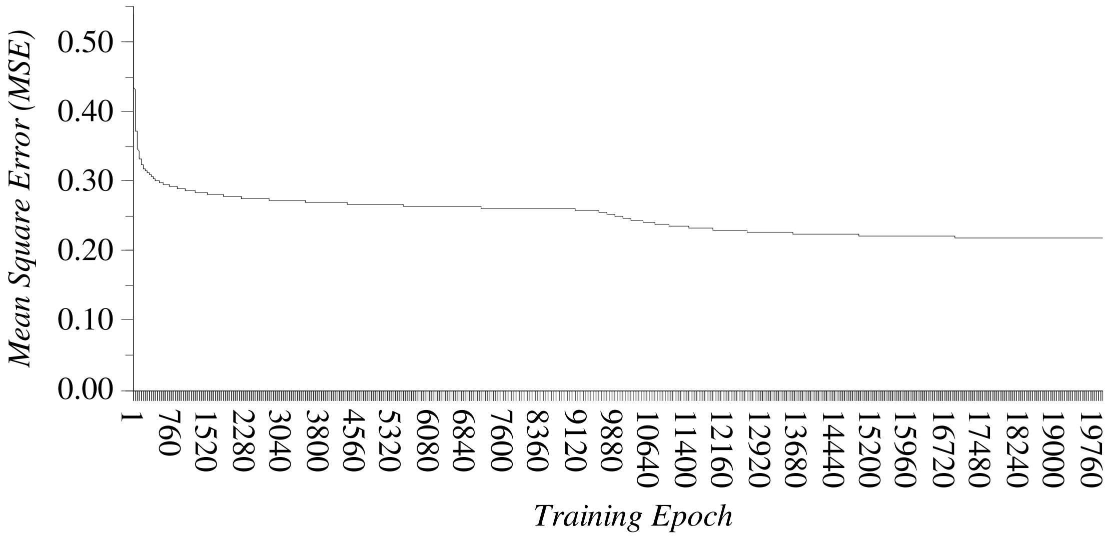

Figure A.2

Mean square error across training epochs for the character code frequency neural network with 20 hidden nodes

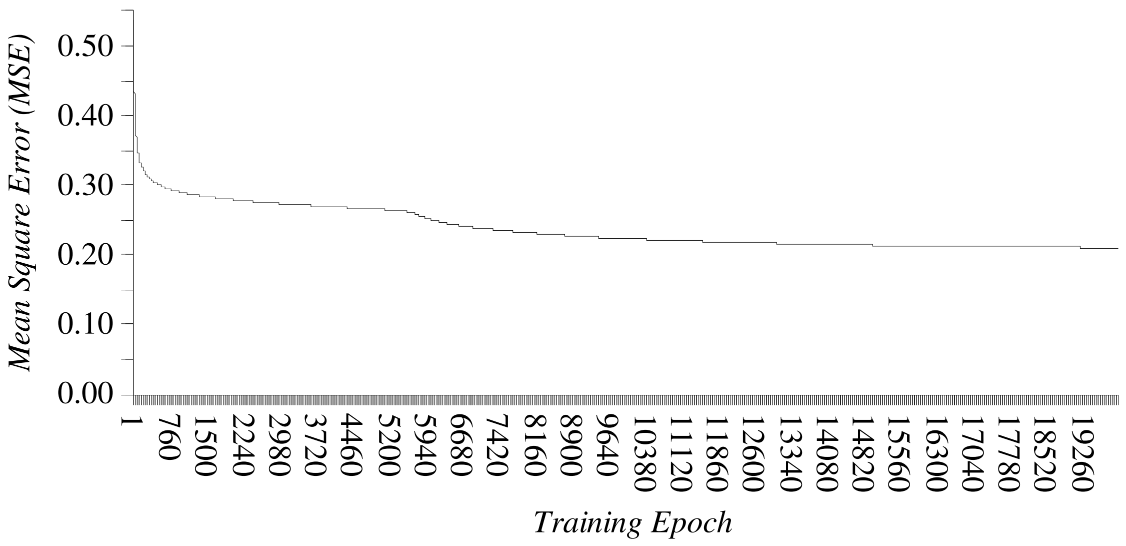

Figure A.3

Mean square error across training epochs for the character code frequency neural network with 30 hidden nodes

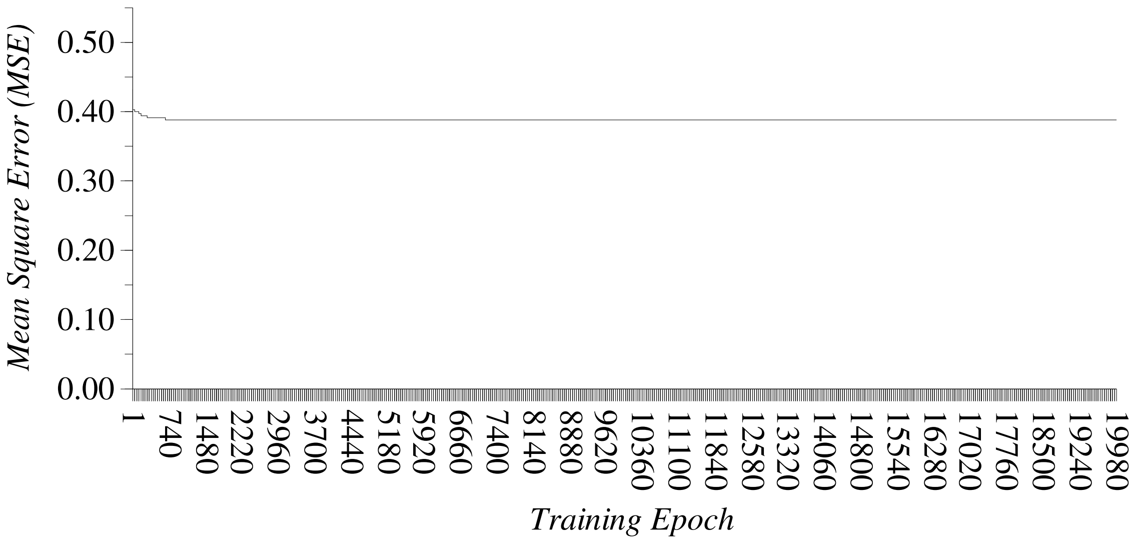

Figure A.4

Mean square error across training epochs for the raw neural network with 10 hidden nodes

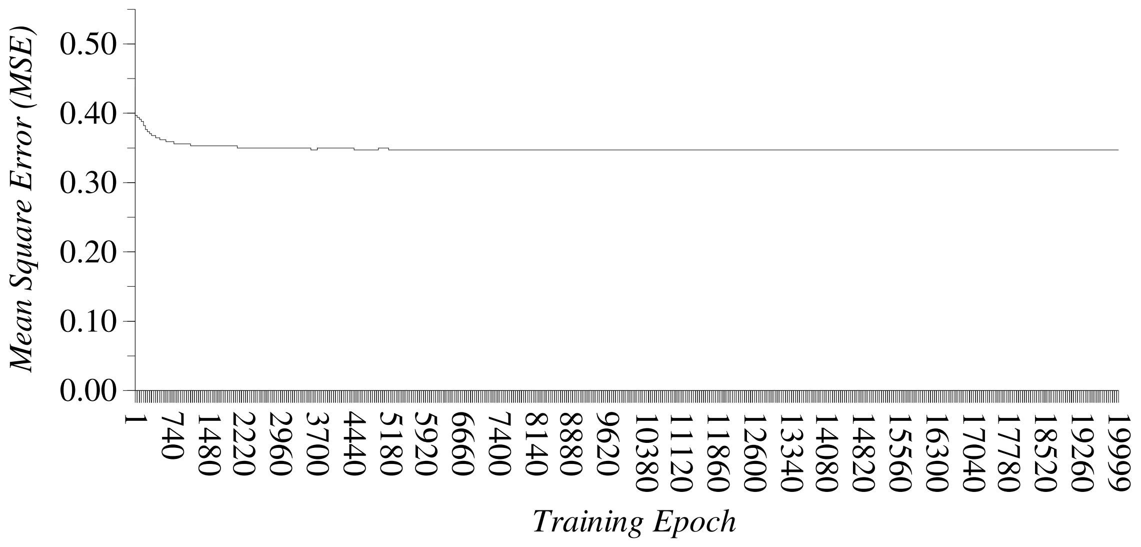

Figure A.5

Mean square error across training epochs for the raw neural network with 20 hidden nodes

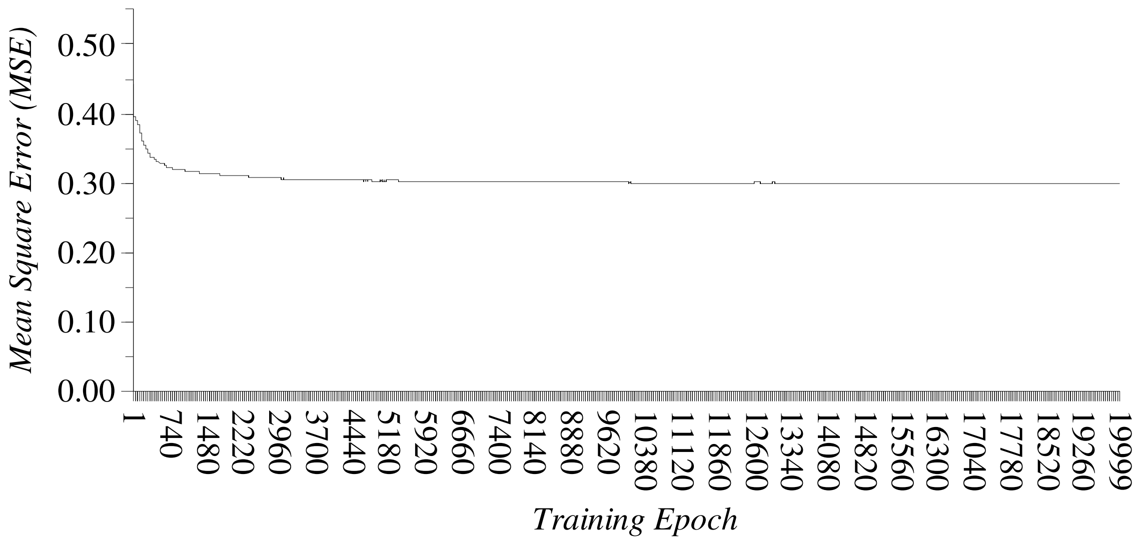

Figure A.6

Mean square error across training epochs for the raw neural network with 30 hidden nodes