2011

Monte Carlo comparison of back-propagation, conjugate-gradient, and finite-difference training algorithms for multilayer perceptrons

Stephen Wehry

Monte Carlo Comparison of Back-Propagation, Conjugate-Gradient, and Finite-Difference Training Algorithms for Multilayer Perceptrons

By Stephen J. Wehry

A Dissertation fulfillment of the requirements for the Master's Degree in Mathematics

Thesis Adviser: Dr. Alejandro Engel

Rochester Institute of Technology
Department of Mathematics and Statistics
Spring 2011 Quarter

MASTER'S DISSERTATION

OF

STEPHEN JOSEPH WEHRY

April, 7, 2011

APPROVED:

Dissertation Committee:

Major Professor _____

Dr. Alejandro Engel

_____

Dr. James Marengo

_____

Dr. Michael Radin

ROCHESTER INSTITUTE OF TECHNOLOGY

Table of Contents

ABSTRACT....4

INTRODUCTION....5

TEST PROBLEMS AND ENCODING SCHEMES....9

- Precision Test - Pattern Recognition....9
- Generalization Test - Bit Counting....10
- Heuristics Used....10

EXAMPLE MULTILAYER PERCEPTRON – "XOR" FUNCTION....12

TRAINING ALGORITHMS TESTED....16

- The Back-Propagation Algorithm....16
- The Conjugate-Gradient Algorithm....21
- The Finite-Difference Algorithm....26

MONTE CARLO SIMULATION....28

- Test Methodology....28
- Simulation Results....29
  - *A Note About ρ Values....29*
  - *Pattern Recognition....29*
  - *Bit Counting....31*

CONCLUSIONS AND FUTURE WORK....37

REFERENCES....39

APPENDICES....40

- Appendix A: Pattern Recognition Calculator Digits....40
- Appendix B: Bit Counting Training Example Set....41
- Appendix C: Derivation Resulting from Higher-Order ρ Values....42
- Appendix D: Raw Monte Carlo Simulation Data....43
  - *D1: Back-Prop Pattern Recognition....43*
  - *D2: C-G Pattern Recognition....47*
  - *D3: Finite-Difference Pattern Recognition....54*
  - *D4: Back-Prop Bit Counting....58*
  - *D5: C-G Bit Counting....71*
  - *D6: Finite-Difference Bit Counting....84*

Abstract

Monte Carlo Simulation is used to compare the performance of the Back-Propagation, Conjugate-Gradient, and Finite-Difference algorithms when training simple Multilayer Perceptron networks to solve pattern recognition and bit counting problems. Twelve individual simulations will be run for each training algorithm-test problem combination, resulting in an overall total of 72 simulations. The random elements in each Monte Carlo simulation are to be the individual synaptic weights between layers, which will be uniformly distributed. Two other factors, the size of the hidden layer and the exponent of the error function, will also be tested within the simulation plan outlined above.

Introduction

Multilayer perceptrons (hereafter denoted as **MLPs**) are a well-studied type of neural network, dating back to the pioneering work of Rosenblatt in 1958[1]. This paper describes a comparison of three training methods, **back-propagation**, **conjugate-gradient**, and **finite-difference** algorithms, used to train simple MLPs to solve each of two different problems: pattern recognition and bit counting. Each of these training methods follow the supervised learning paradigm[2] in which training examples are presented from which an **error signal** is generated. This error signal is then used to train the network to learn to solve the problem at hand in different (but related) ways.

All of the MLPs used in this paper have the same general structure depicted in Figure 1 below.

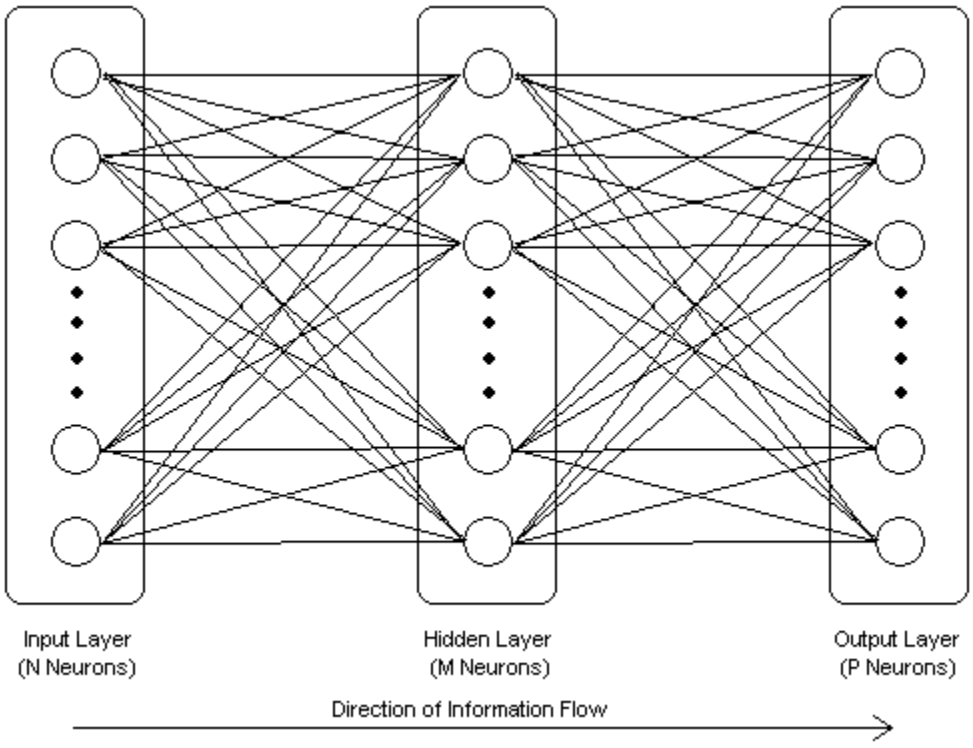

Figure 1: 3-Layer Fully-Connected MLP Topology

Each MLP in this study consists of three layers, one **input layer**, one **hidden layer**, and one **output layer**. The number of **neurons** in the input and output layers are determined by the input and output encoding schemes used to define the problem to be

solved; the number of hidden neurons was varied as part of this study. All of the networks used in this paper have the following properties in addition to those already mentioned:

- They are **fully-connected**, meaning that each neuron in both the input and hidden layers is connected to every neuron in the subsequent layer[3].
- They are **feed-forward**, meaning that during computation information flows in one direction from the input, through the hidden, and ending in the output layer. This means that there is no use of online feedback[4]. An additional constraint on the topology of the networks in this study is that there is no skipping of layers.
- All non-input neurons utilize a nonlinear **activation function**[5].
- Input neurons are for input only, hidden neurons are for computation only, and output neurons are for computation and output. In this context computation means the process of summation of the input signals to an individual neuron then the application of the activation function.

In the following discussion, as well as the rest of this paper, the notation and terminology used will closely follow that in Haykin (1999). The computational structure of the ith input node is shown schematically in Figure 2.

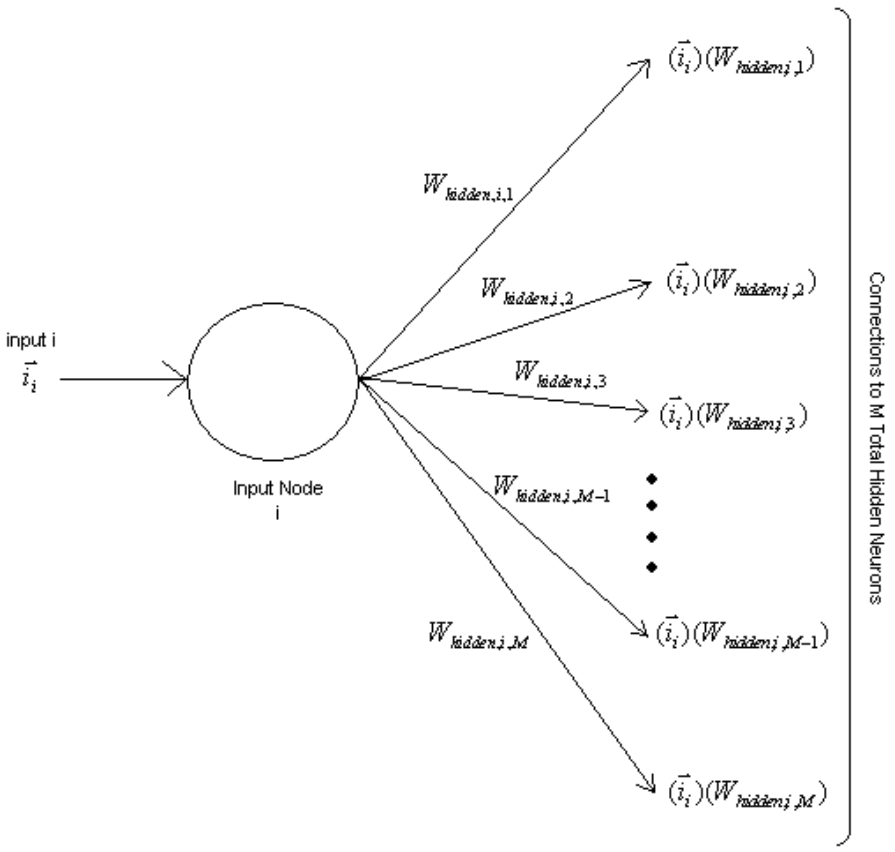

Figure 2: Computational Structure of Input Node i

The function of an input node is to receive its single input signal and pass it on to each neuron in the hidden layer after applying the appropriate synaptic weight[6.] for that connection.

The computational structure of the jth hidden neuron is shown on the next page in Figure 3.

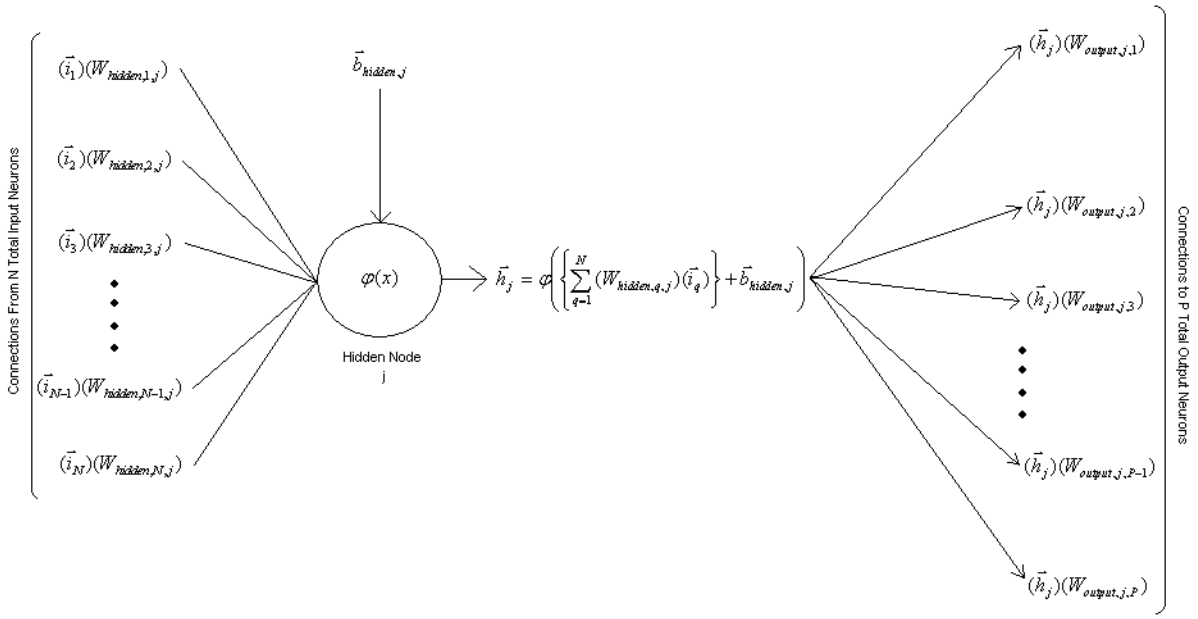

Figure 3: Computational Structure of Hidden Node j

Figure 3 shows that a hidden node first sums up its inputs from the previous layer, adds its bias, then applies the activation function to this value. The resulting value is then passed on to become part of the input of each neuron in the output layer, again, after the appropriate synaptic weight has been applied[7].

The computational structure of the kth output neuron is shown below in Figure 4.

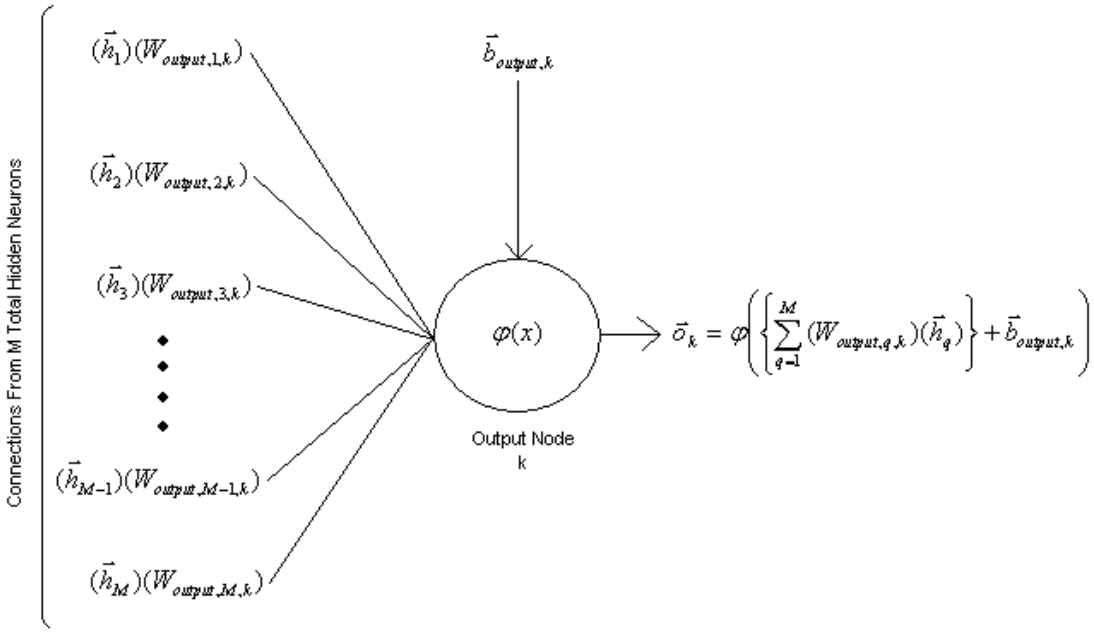

Figure 4: Computational Structure of Output Neuron k

An output neuron acts just like a hidden neuron, except that no weight is applied to the output of the activation function in these nodes[8].

Finally, consider a few miscellaneous notes on the overall properties of the network. First, the functional form and parameters of φ(x) were heuristically chosen for two main reasons. A sigmoid function, such as the hyperbolic tangent function used in this study, has the necessary properties of nonlinearity, monotonicity, antisymmetricity, and range-boundedness. The parameters were chosen to maximize the curvature of this function at $x = y = \pm 1$, which helps to prevent neurons from being driven to saturation, a serious obstacle to the convergence of the training algorithm. This leads to the second general property of the networks used, which is that both the input and output spaces of each problem are encoded into vectors $\mathbf{x}$ with $-1 \leq \mathbf{x}_i \leq 1$[9].

Test Problems and Encoding Schemes

Precision Test - Pattern Recognition

The pattern recognition problem used in this study is for the MLP to correctly identify ten simple five-by-three pixel calculator digits detailed in Appendix A. To illustrate the input space encoding scheme, consider the fifteen pixel calculator digit for zero, shown below in Figure 5.

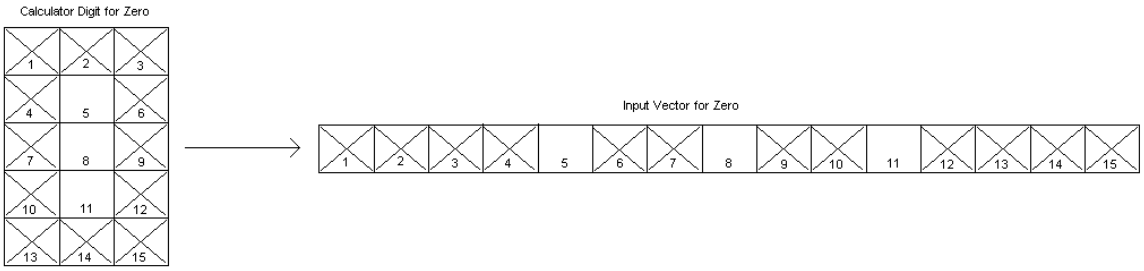

Figure 5: Pattern Recognition Input Space Encoding Scheme

As seen in Figure 5, the fifteen pixel digit is represented by the corresponding fifteen component input vector. The digit zero, therefore, is represented by the input vector [1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1]. The input vectors for the remaining digits are obtained in the same manner. The output space encoding scheme for this problem is to simply represent the numeric value of each digit by its 4-bit binary representation. Thus the output corresponding to zero would be [0, 0, 0, 0], that corresponding to nine would be

[1, 0, 0, 1], and so on. Finally, the network will be presented with one example of each of the ten digits during each epoch of training.

Pattern Recognition is a so-called precision test because there are a small number of possible inputs, meaning that all of them can be presented to the network during each training epoch. This means that since no generalization outside of the training space will be required, the network's only task is to learn these examples to a high degree of accuracy.

Generalization Test - Bit Counting

The bit counting problem used in this study is to have the network count the number of "1"s in an arbitrary binary string seven bits long. Naturally the input space for this problem will consist of vectors with seven components. The output space of the network will be a binary representation of the numeric value of the answer. Since there are a maximum of seven "1"s in a seven bit string, the output space will be a vector with three components.

In this study there are $2^7 = 128$ possible input strings, but only a subset of 68 them will be used as training examples. Since the network will be tested against each of the 128 possible inputs, this test presents the network an additional challenge, that of generalization from the known to the unknown.

Heuristics Used

Several heuristics for improving the performance of MLPs have grown out of the cumulative work that has been done over the years since their introduction. Many have been incorporated into and used in this study; they are listed below.

- Pre-processing of input and output spaces – Remove the mean of each to ensure that the components of the input and output vectors are not all positive or all negative[10].
- "Euclidean Distance Output" calculation – both pattern recognition and bit counting are problems in which valid outputs are restricted to a discrete set of possible values. In this study the raw output of the MLP, the vector whose components are the values of the output neurons, is compared to each of the possible output vectors to determine which valid output is closest in a Euclidean-

distance sense. That closest valid output vector is what is used as the reported output of the network.

- Synaptic weight initialization - Calculate initial values for synaptic weights according to a uniform distribution function; use layer size to determine variance of same[11].
- In the Back-Prop Algorithm, each synaptic weight has its own individually adjustable learning rate parameter[12].

Example Multilayer Perceptron – "XOR" Function

As an illustrative example of using an MLP is to consider the MLP diagrammed below in Figure 6, designed to solve the "Exclusive Or", or "XOR" problem.

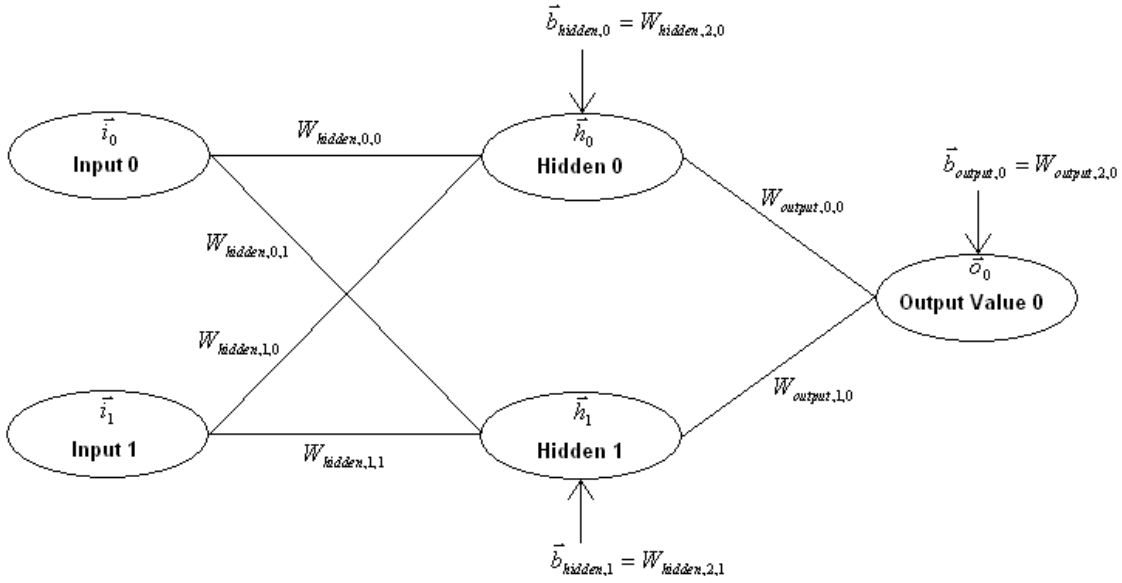

Figure 6: "XOR" MLP Schematic

The XOR function, of course, gives values according to the following truth table.

Input 0	Input 1	Output Value
0	0	0
1	0	1
0	1	1
1	1	0

Table 1: "XOR" Truth Table

It requires two input values, each either 0 or 1, and calculates a single output value, also either 0 or 1. This determined the number of input and output neurons in the design of the MLP in Figure 7. For this example a hidden layer of two neurons was also chosen. One possible example of an MLP trained to solve the "XOR" problem, is given in the following table of parameter values.

Parameter	Value After Successful Training
$W_{hidden,0,0}$	0.362985
$W_{hidden,0,1}$	0.418378
$W_{hidden,1,0}$	-0.464486
$W_{hidden,1,1}$	-0.554121
$W_{hidden,2,0}$	-0.720958
$W_{hidden,2,1}$	0.504430
$W_{output,0,0}$	0.620124
$W_{output,1,0}$	-0.446396
$W_{output,2,0}$	0.692502

Table 2: Example MLP "Trained" Parameter Values

Following the heuristics described in the previous section, the first step in using this example MLP is to pre-process the input values by subtracting out the mean of the input space. The four possible input vectors in the "XOR" problem are (0,0), (1,0), (0,1), and (1,1). The mean value of the first component of the possible input space is (0+1+0+1) / 4 = 0.5, and that of the second component is also (0+0+1+1) / 4 = 0.5. Therefore before we present the network with an input, we first subtract the vector (0.5, 0.5), resulting in the following table of possible pre-processed input values.

Original Input Vector	Pre-Processed Input Vector (Zero-Mean)
(0, 0)	(-0.5, -0.5)
(1, 0)	(0.5, -0.5)
(0, 1)	(-0.5, 0.5)
(1, 1)	(0.5, 0.5)

Table 3: Example MLP Zero-Mean Input Values

For this example, the network will be tested with the input (1, 1); see Appendix C for example calculations using the other three possible "XOR" inputs. First, we use Table 3 to determine the zero-mean input vector that corresponds to (1, 1), which is (0.5, 0.5). This input vector is assigned to the input nodes in the example MLP, as seen in Figure 7 on the next page.

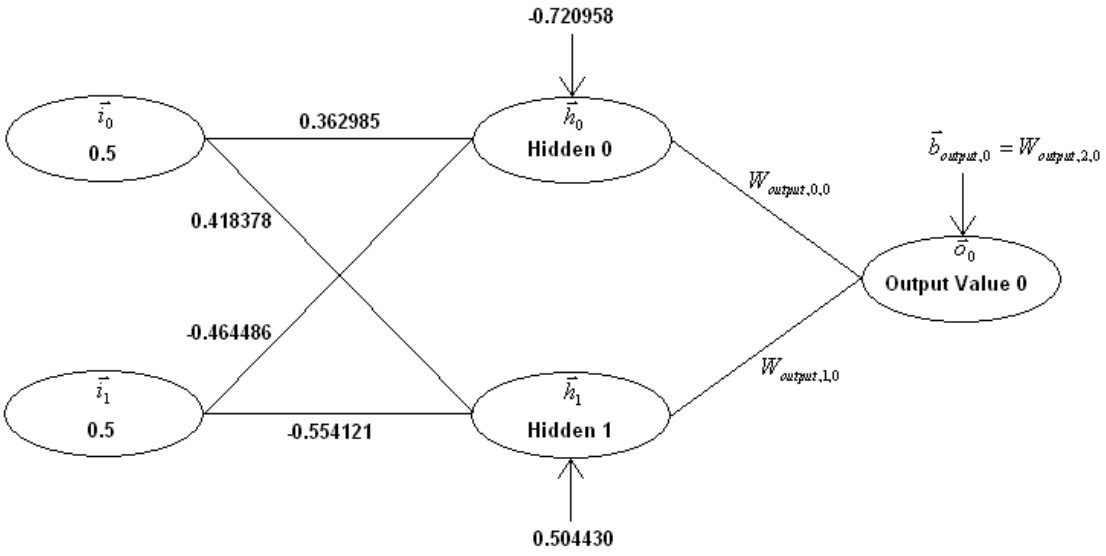

Figure 7: "XOR" MLP Calculation: Input Layer

Now the values of the hidden layer neurons must be calculated, using the values of the input layer along with the synaptic weights and biases that are shown in Figure 7.

$$\vec{h}_0 = \varphi\left\{(\vec{i}_0)(W_{hidden,0,0}) + (\vec{i}_1)(W_{hidden,1,0}) + \vec{b}_{hidden,0}\right\}$$

$$\vec{h}_0 = \varphi\left\{(0.5)(0.362985) + (0.5)(-0.464486) + (-0.720958)\right\}$$

$$\vec{h}_0 = \varphi(-0.771709) = 1.7159\tanh\left(\tfrac{2}{3}(-0.771709)\right) = -0.812346$$

Similarly,

$$\vec{h}_1 = \varphi\left\{(\vec{i}_0)(W_{hidden,0,1}) + (\vec{i}_1)(W_{hidden,1,1}) + \vec{b}_{hidden,1}\right\}$$

$$\vec{h}_1 = \varphi\left\{(0.5)(0.418378) + (0.5)(-0.554121) + 0.504430\right\}$$

$$\vec{h}_1 = \varphi(0.436559) = 1.7159\tanh\left(\tfrac{2}{3}(0.436559)\right) = 0.485756,$$

which results in the network state shown in Figure 8 on the next page.

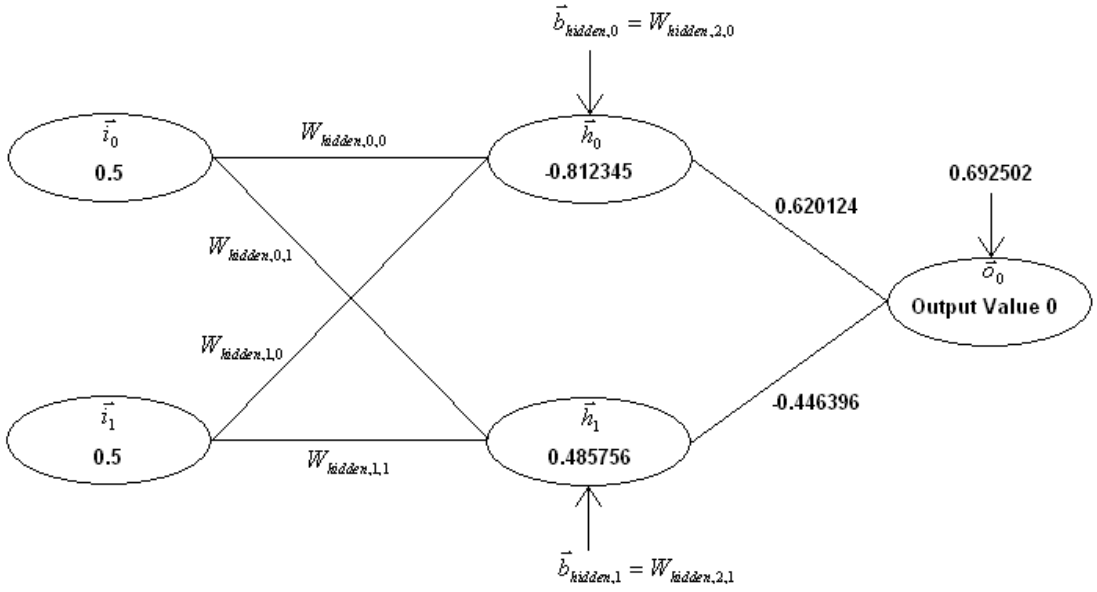

Figure 8: "XOR" MLP Calculation: Hidden Layer

The next step is to calculate the value of the output neuron:

$$\vec{o}_0 = \varphi\left\{(\vec{h}_0)(W_{output,0,0}) + (\vec{h}_1)(W_{output,1,0}) + \vec{b}_{output,0}\right\}$$

$$\vec{o}_0 = \varphi\left\{(-0.812345)(0.620124) + (0.485756)(-0.446396) + 0.692502\right\}$$

$$\vec{o}_0 = \varphi(-0.028092) = 1.7159\tanh\left(\tfrac{2}{3}(-0.028092)\right) = -0.032132,$$

which results in the final network state shown in Figure 9.

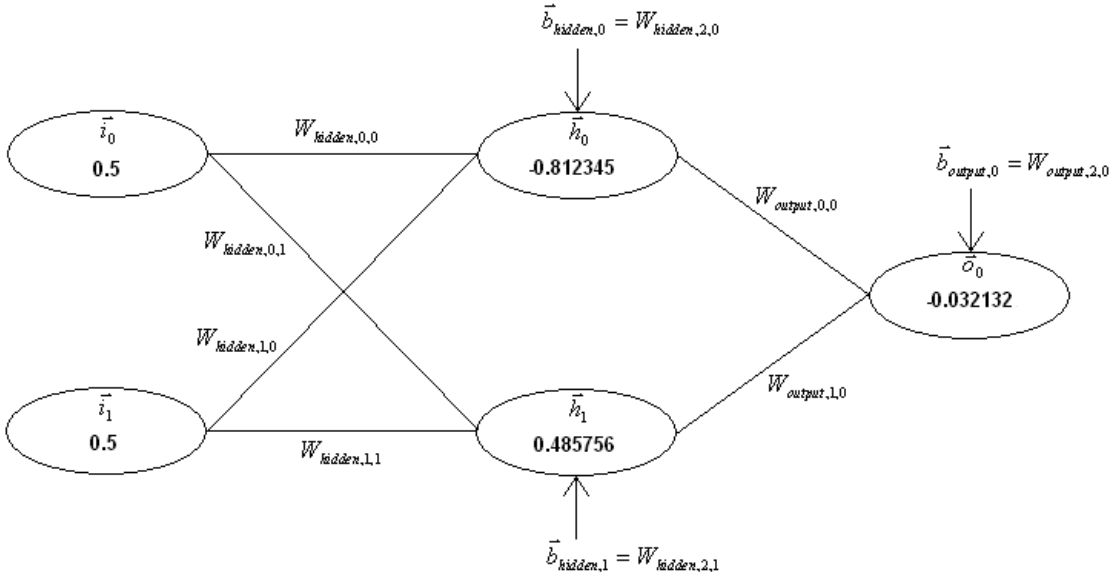

Figure 9: "XOR" MLP Calculation: Output Layer

The calculated output of the example MLP to the input (1, 1) is -0.032132, but the possible outputs of the actual "XOR" problem are restricted to either 1 or 0. This is because this MLP was trained using zero-mean output values (another of the heuristics listed in the previous section), which are tabulated below in Table 4.

Original Output Vector	Zero-Mean Output Vector
(0)	(-0.5)
(1)	(0.5)

Table 4: Example MLP Zero-Mean Output Values

So, the possible zero-mean output values for the "XOR" problem are -0.5 or 0.5 – to which does the output value -0.032132 of our example MLP correspond? This final step in the MLP calculation involves computing the Euclidean distance of our output vector to each of the possible zero-mean output vectors, and choosing that which is closest. We calculate

$$\|(-0.032132) - (0.5)\| = \sqrt{[(-0.032132) - (0.5)]^2} = 0.532132$$

$$\|(-0.032132) - (-0.5)\| = \sqrt{[(-0.032132) - (-0.5)]^2} = 0.467868,$$

so the final output of the example MLP to the "XOR" of (1, 1) is 0.

Training Algorithms Tested

As mentioned in the Introduction, both training methods used in this study employ the **supervised learning paradigm**, in which adjustments are made to the network's parameters due to errors in the output response to training examples[13]. However, even small MLPs like those in this study have a large number of such parameters in the form of synaptic weights between and biases of their neurons. Which parameters should be adjusted when, and by how much, during training? This is what is known as the **"credit assignment problem"** for MLPs that training algorithms are designed to solve[14].

The Back-Propagation Algorithm

One of the first computationally efficient solutions to the credit assignment problem, and still one of the most powerful and popular neural network training algorithms, is the back-propagation (back-prop) algorithm[15]. Back-propagation is so-called because it adjusts the free parameters of the network by using the familiar Chain

Rule of Calculus to propagate the error signal of each training example back, layer-by-layer, through the network. **Hebbian Learning** adjustments are made to the parameters of each layer in proportion to the contribution of each individual parameter to the overall error signal[16].

The back-prop formulas are derived as follows. First, assume an MLP with N input, M hidden, and P output neurons. Let **d** be the vector (dimension P) of desired response for the training example presented to the network as input vector **i** (dimension N). Given the current state of the network (i.e., the values of the weight and bias parameters for each neuron), the network will calculate an output response vector **o** (dimension P), the vector of the output values of the output neurons. The output values will differ from the desired response by an amount

$$\vec{e}_k = \vec{d}_k - \vec{o}_k, \quad k = 1, \ldots, P \tag{1.}$$

for each output neuron k. This value is the error signal for the kth neuron in response to the current training example. An energy-like total error in response to the current training example can be computed as

$$E = \tfrac{1}{2}\sum_{k=1}^{P} (\vec{d}_k - \vec{o}_k)^2 = \tfrac{1}{2}\sum_{k=1}^{P} (\vec{e}_k)^2 . \tag{2.}$$

This is the error signal that will be propagated back through the network to effect training. Using the Hebbian Learning analogy, we wish to adjust each parameter by an amount proportional to its individual contribution to this total error, as in

$$\Delta W_{ij} = -\lambda \left(\frac{\partial E}{\partial W_{ij}} \right), \tag{3.}$$

where the value λ is the called the learning rate parameter. The following derivation[17] provides a value for $\frac{\partial E}{\partial W_{ij}}$.

Consider a non-input neuron j. Let N be the number of neurons in the previous layer (indexed by the variable i), and M be the number of neurons in the layer (indexed by the variable j) containing neuron j. Neuron j will receive N+1 total input signals (see Figures 3 and 4 above) for its calculation, the outputs of the N neurons in the previous layer plus the bias "signal" of neuron j. Call these input signals to neuron j the values $\mathbf{y}_i$, i

= 0 to N, where $\mathbf{y_0} = 1$ is the "signal" which, when multiplied by the bias value $\mathbf{b}_j = W_{0j}$, gives the bias term of the calculation. Let W_{ij} be the matrix of synaptic weights from the previous layer to the layer containing neuron j.

The value

$$\vec{v}_j = \sum_{i=0}^{N} (W_{ij})(\vec{y}_i) \tag{4.}$$

is called the induced local field of neuron j. The output signal of neuron j in response to the current training example is

$$\vec{y}_j = \varphi(\vec{v}_j). \tag{5.}$$

Now, using the Chain Rule of Calculus,

$$\frac{\partial E}{\partial W_{ij}} = \frac{\partial E}{\partial \vec{e}_j}\frac{\partial \vec{e}_j}{\partial \vec{y}_j}\frac{\partial \vec{y}_j}{\partial \vec{v}_j}\frac{\partial \vec{v}_j}{\partial W_{ij}}, \tag{6.}$$

which gives an expression for the value we want in terms of values we can calculate. First, differentiate equation (2.) to get

$$\frac{\partial E}{\partial \vec{e}_j} = \vec{e}_j. \tag{7.}$$

Recall that for non-input neuron j,

$$\vec{e}_j = \vec{d}_j - \vec{y}_j, \tag{8.}$$

from which we get, via differentiation,

$$\frac{\partial \vec{e}_j}{\partial \vec{y}_j} = -1. \tag{9.}$$

To get the fourth factor of the right side of equation (6.), we differentiate equation (5.) to get

$$\frac{\partial \vec{y}_j}{\partial \vec{v}_j} = \varphi'(\vec{v}_j). \tag{10.}$$

Finally, by differentiating equation (4.) we get

$$\frac{\partial \vec{v}_j}{\partial W_{ij}} = \vec{y}_i. \tag{11.}$$

Plugging these four partial derivatives back into equation (6.) we get

$$\frac{\partial E}{\partial W_{ij}} = [-\vec{e}_j][\varphi'(\vec{v}_j)][\vec{y}_i], \tag{12.}$$

which gives the following expression for equation (3.):

$$\Delta W_{ij} = \lambda[\vec{e}_j][\varphi'(\vec{v}_j)][\vec{y}_i]. \tag{13.}$$

The value $[\vec{e}_j][\varphi'(\vec{v}_j)]$ is denoted by $\vec{\delta}_j$ and is called the local gradient of neuron j.

Note that if neuron j is an output node equation (12.) is readily evaluated, but due to the presence of the desired response in $\vec{e}_j$, further calculation to define this value is necessary in the case where neuron j is a hidden neuron. The local gradient $\vec{\delta}_j$ is defined as

$$\vec{\delta}_j = [\vec{e}_j][\varphi'(\vec{v}_j)] = -\frac{\partial E}{\partial \vec{v}_j} \tag{14.}$$

$$= -\frac{\partial E}{\partial \vec{y}_j}\frac{\partial \vec{y}_j}{\partial \vec{v}_j} \tag{15.}$$

$$= -\frac{\partial E}{\partial \vec{y}_j}\varphi'(\vec{v}_j), \tag{16.}$$

where the Chain Rule is used in equation (15.). Following Haykin (1999), note that in equation (2.) the summation is over the output nodes k. Differentiating equation (2.) yields

$$\frac{\partial E}{\partial \vec{y}_j} = \sum_{k=1}^{P}\left(\vec{e}_k\frac{\partial \vec{e}_k}{\partial \vec{y}_j}\right). \tag{17.}$$

Again the chain rule is used to get

$$\frac{\partial E}{\partial \vec{y}_j} = \sum_{k=1}^{P}\left(\vec{e}_k\frac{\partial \vec{e}_k}{\partial \vec{v}_k}\frac{\partial \vec{v}_k}{\partial \vec{y}_j}\right). \tag{18.}$$

Since neuron k is an output neuron, recall from equation (8.) that

$$\vec{e}_k = \vec{d}_k - \vec{y}_k \tag{8.}$$

$$= \vec{d}_k - \varphi(\vec{v}_k). \tag{19.}$$

Therefore we can differentiate to get

$$\frac{\partial \vec{e}_k}{\partial \vec{v}_k} = -\varphi'(\vec{v}_k) \tag{20.}$$

Using equation (4.) we can write the induced local field of neuron k as

$$\vec{v}_k = \sum_{j=0}^{M} \left([W_{jk}][\vec{y}_j]\right), \tag{21.}$$

so we get that

$$\frac{\partial \vec{v}_k}{\partial \vec{y}_j} = W_{jk}\,. \tag{22.}$$

Finally, plugging the two partials (20.) and (22.) back into equation (18.) yields

$$\frac{\partial E}{\partial \vec{y}_j} = -\sum_{k=1}^{P} \left([\vec{e}_k][\varphi'(\vec{v}_k)][W_{jk}]\right) \tag{23.}$$

and

$$\frac{\partial E}{\partial \vec{y}_j} = -\sum_{k=1}^{P} \left([\vec{\delta}_k][W_{jk}]\right). \tag{24.}$$

Plugging equation (24.) into equation (15.) yields what is known as the **back-propagation formula**,

$$\vec{\delta}_j = \varphi'(\vec{v}_j)\sum_{k=1}^{P} (\vec{\delta}_k W_{jk}) \tag{25.}$$

for hidden neuron j. This provides the algorithmic mechanism to calculate the "local contribution" of hidden neuron j to the overall error signal for the current training example[18.].

A pseudocode version of the complete back-prop algorithm used in this study is shown in Table 5 on the next page.

Line #	**3-Layer MLP with N input, M hidden, and P output neurons**	Comments
1	Initialize epoch counter = 0	
2	Initialize (M+1 by P) output layer weights matrix $\mathbf{W}_{\mathbf{output}}$	Uniform Distribution with $\mu = 0$, $\sigma^2 = 1 / M$
3	Initialize (M+1 by P) output layer learning rate matrix $\boldsymbol{\lambda}_{\mathbf{output}}$	Use an initial value of 0.01
4	Initialize (N+1 by M) hidden layer weights matrix $\mathbf{W}_{\mathbf{hidden}}$	Uniform Distribution with $\mu = 0$, $\sigma^2 = 1 / N$
5	Initialize (N+1 by M) hidden layer learning rate matrix $\boldsymbol{\lambda}_{\mathbf{hidden}}$	Use an initial value of 0.01
6	While not done training do	
7	For x = 0 to NUM_EXAMPLES	
8	Set input vector **i** = training example x	
9	Set desired response vector **d**	
10	Compute hidden layer vector **h**	See Figure 3 above.
11	Compute output layer vector **o**	See Figure 4 above.
12	Compute error signal component E_x	$E_x = \frac{1}{2}\sum_{i=1}^{P} (\vec{d}_i - \vec{o}_i)^2$
13	Calculate output layer local gradient vector $\boldsymbol{\delta}_{\mathbf{output}}$	$\vec{\delta}_{output,k} = \varphi'(\vec{v}_{output,k})(\vec{d}_k - \vec{o}_k); k = 1...P$
14	Calculate hidden layer local gradients vector $\boldsymbol{\delta}_{\mathbf{hidden}}$	$\vec{\delta}_{hidden,j} = \varphi'(\vec{v}_{hidden,j})\sum_{k=1}^{P} (\vec{\delta}_{output,k} W_{output,j,k}); j = 1...M$
15	Adjust output layer weights matrix $\mathbf{W}_{\mathbf{output}}$	$W_{output,j.k} = (W_{output,j.k})_{previous} + (\lambda_{j,k})(\vec{\delta}_{output,k})(\vec{h}_j); j = 1...M; k = 1...P$
16	Adjust hidden layer weights matrix $\mathbf{W}_{\mathbf{hidden}}$	$W_{hidden,i.j} = (W_{hidden,i.j})_{previous} + (\lambda_{i,j})(\vec{\delta}_{hidden,j})(\vec{i}_i); i = 1...N; j = 1...M$
17	Adjust individual learning rates matrices $\boldsymbol{\lambda}_{\mathbf{hidden}}$, $\boldsymbol{\lambda}_{\mathbf{output}}$	
18	Next x	
19	Calculate Average Error Value E_{avg}	$E_{avg} = \left[\left(\frac{1}{NE}\right)\sum_{x=1}^{NE} E_x\right], NE = NUM\_EXAMPLES$
20	If stopping criteria is not met Then increment epoch count	
	Else done training is true	
21	Loop back to line 4	
22	Record # epochs, total training time, initial average error, and final average error	
23	STOP	

Table 5: The Back-Propagation Algorithm

The Conjugate-Gradient Algorithm

The **method of conjugate gradients** is an algorithm for exactly minimizing the quadratic vector equation $\frac{1}{2}(\mathbf{x}^T\mathbf{A}\mathbf{x}) - \mathbf{b}\mathbf{x} + c = \mathbf{0}$, when the n-by-n matrix **A** is positive-definite[19]. This algorithm adjusts the n-by-1 vector **x** to minimize the equation along each of a constructed sequence of n **A**-conjugate vectors, the resulting value of **x** being the desired minimizing value. If the matrix **A** is known and positive-definite the sequence of **A**-conjugate vectors can be directly calculated and the algorithm finishes in at most n steps.

To begin our discussion of this algorithm, we must define the concept of **A**-conjugacy. A non-zero set of N vectors $\{\mathbf{s}_0, \mathbf{s}_1, \mathbf{s}_2, \ldots, \mathbf{s}_{N-1}\}$ is called **A**-conjugate with respect to the N-by-N positive definite matrix **A** if

$$\vec{s}_i^{\,T} A \vec{s}_j = 0, i \neq j \qquad (26.)$$

The concept of **A-conjugacy** is a generalization of the more commonly used idea of orthogonality of vectors. Like a set of mutually orthogonal vectors, a set of **A**-conjugate vectors is linearly independent. This can be proven by contradiction as follows. Without loss of generality, assume we have the **A**-conjugate set $\{\mathbf{s}_i\}$, i=0,…,N-1, and assume that

$$\vec{s}_0 = \sum_{j=1}^{N-1} a_j \vec{s}_j \tag{27.}$$

in other words, that the set of vectors is linearly dependent. Left-multiply Equation (27.) by the matrix **A** to get

$$A\vec{s}_0 = \sum_{j=1}^{N-1} a_j A\vec{s}_j \ . \tag{28.}$$

Again left-multiply by $\vec{s}_0^{\,T}$ to get

$$\vec{s}_0^{\,T} A\vec{s}_0 = \sum_{j=1}^{N-1} a_j \vec{s}_0^{\,T} A\vec{s}_j \ , \tag{29.}$$

but because we assumed that the set $\{\mathbf{s}_i\}$ was **A**-conjugate, the right side of Equation (29.) is zero, so we see that

$$\vec{s}_0^{\,T} A\vec{s}_0 = 0 \ . \tag{30.}$$

This is our contradiction because by definition $\mathbf{s}_0$ is nonzero and the matrix **A** is positive-definite, therefore the set of **A**-conjugate vectors must be linearly independent[20].

The conjugate-gradients algorithm can be adapted for use in a generic nonlinear minimization problem due to the theoretical fact that any function can be Taylor expanded to "look" like a second-order system in the neighborhood of a local minimum. The training of an MLP can be considered as such a problem, where the average squared error over a training epoch is used as the nonlinear function to be minimized. In the neighborhood of a local minimum, the error surface is described by

$$E_{Avg}(\vec{w}) \approx \tfrac{1}{2}\vec{w}^T A\vec{w} - \vec{b}^T\vec{w} + c \ . \tag{31.}$$

Clearly in the general neural network application the matrix **A**, the vector **b**, and the scalar c are unknown, so we develop a method to calculate the sequence of **A**-conjugate vectors without the need to explicitly calculate any of these values[21].

Given the situation in Equation (31.), the idea behind conjugate-gradients is to generate a sequence of **A**-conjugate search vectors $\mathbf{s}_0, \mathbf{s}_1, \mathbf{s}_2, \ldots, \mathbf{s}_{N-1}$ to adjust the parameter vector **w** using the update equation

$$\vec{w}_{n+1} = \vec{w}_n + \eta \vec{s}_n \text{, n=0,…,N-1,} \tag{32.}$$

where the scalar η is defined as the value that minimizes the value of $E_{Avg}(\vec{w}_n + \eta \vec{s}_n)$ when both $\mathbf{w}_n$ and $\mathbf{s}_n$ are held constant. First, define the residual vector

$$\vec{r}_n = \vec{b} - A\vec{w}_n = \left. \frac{\partial E_{Avg}(\vec{w})}{\partial \vec{w}} \right|_{\vec{w}=\vec{w}_n}, \tag{33.}$$

the gradient vector of the error surface evaluated at the current point **w** on the error surface.

We begin the algorithm by choosing an arbitrary initial value of $\mathbf{w}_0$, and setting our initial search vector

$$\vec{s}_0 = \vec{r}_0 = \vec{b} - A\vec{w}_o = \left. \frac{\partial E_{Avg}(\vec{w})}{\partial \vec{w}} \right|_{\vec{w}=\vec{w}_0}. \tag{34.}$$

We'll use the **conjugate Gram-Schmidt**[22.] procedure to construct the remaining search vectors such that

$$\vec{s}_i = \vec{r}_i + \sum_{k=0}^{i-1} \beta_{ik} \vec{s}_k \,. \tag{35.}$$

Take the inner product of Equation (35.) with the vector $\mathbf{As}_j$, $0 < j < i$, to get

$$(\vec{s}_i) \cdot (A\vec{s}_j) = \left[\vec{r}_i + \sum_{k=0}^{i-1} \beta_{ik} \vec{s}_k \right] \cdot (A\vec{s}_j) \tag{36.}$$

$$(\vec{s}_i) \cdot (A\vec{s}_j) = (\vec{r}_i) \cdot (A\vec{s}_j) + \sum_{k=0}^{i-1} \beta_{ik} \left[(\vec{s}_k) \cdot (A\vec{s}_j) \right] \tag{37.}$$

$$\vec{s}_i^{\,T} A\vec{s}_j = \vec{r}_i^{\,T} A\vec{s}_j + \sum_{k=0}^{i-1} \beta_{ik} \left[\vec{s}_k^{\,T} A\vec{s}_j \right]. \tag{38.}$$

Recall that the set $\{\mathbf{s}_i\}$ is **A**-conjugate, so

$$\vec{s}_i^{\,T} A\vec{s}_j = 0, i \neq j \,, \tag{39.}$$

so

$$0 = \vec{r}_i^{\,T} A\vec{s}_j + \beta_{ij} \vec{s}_j A\vec{s}_j \,, \tag{40.}$$

and solving Equation (40.) for β_{ij} gives

$$\beta_{ij} = -\frac{\vec{r}_i^{\,T} A\vec{s}_j}{\vec{s}_j^{\,T} A\vec{s}_j}. \tag{41.}$$

However, we want to be able to compute β_{ij} without explicitly using or even knowing the matrix **A**, so we must continue our calculation. From the method of steepest descent we know that the residual vectors we have defined are related by the recurrence relation

$$\vec{r}_{i+1} = \vec{r}_i - \alpha_i A\vec{s}_i. \tag{42.}$$

Taking the inner product

$$(\vec{r}_i)\cdot(\vec{r}_{j+1}) = (\vec{r}_i)\cdot(\vec{r}_j) - \alpha_j(\vec{r}_i)\cdot(A\vec{s}_j) \tag{43.}$$

$$\vec{r}_i^{\,T}\vec{r}_{j+1} = \vec{r}_i^{\,T}\vec{r}_j - \alpha_j \vec{r}_i^{\,T} A\vec{s}_j \tag{44.}$$

and rearranging Equation (44.) yields

$$\alpha_j \vec{r}_i^{\,T} A\vec{s}_j = \vec{r}_i^{\,T}\vec{r}_j - \vec{r}_i^{\,T}\vec{r}_{j+1}, \tag{45.}$$

and therefore

$$\vec{r}_i^{\,T} A\vec{s}_j = \begin{cases} \frac{1}{\alpha_i}\vec{r}_i^{\,T}\vec{r}_i, i = j \\ -\frac{1}{\alpha_{i-1}}\vec{r}_i^{\,T}\vec{r}_i, i = j+1 \\ 0, otherwise \end{cases}. \tag{46.}$$

Plugging Equation (46.) into Equation (41.) gives

$$\beta_{ij} = \begin{cases} \left(\frac{1}{\alpha_{i-1}}\right)\frac{\vec{r}_i^{\,T}\vec{r}_i}{\vec{s}_{i-1}^{\,T} A\vec{s}_{i-1}}, i = j+1 \\ 0, i > j+1 \end{cases} \tag{47.}$$

The parameter α_{i-1} is the steepest descent parameter

$$\alpha_{i-1} = \frac{\vec{s}_{i-1}^{\,T}\vec{r}_{i-1}}{\vec{s}_{i-1}^{\,T} A\vec{s}_{i-1}}, \tag{48.}$$

and

$$\frac{1}{\alpha_{i-1}} = \frac{\vec{s}_{i-1}^{\,T} A\vec{s}_{i-1}}{\vec{s}_{i-1}^{\,T}\vec{r}_{i-1}}. \tag{49.}$$

Plugging Equation (49.) into Equation (47.) gives

$$\beta_i = \left(\frac{\vec{s}_{i-1}^T A \vec{s}_{i-1}}{\vec{s}_{i-1}^T \vec{r}_{i-1}} \right) \left(\frac{\vec{r}_i^T \vec{r}_i}{\vec{s}_{i-1}^T A \vec{s}_{i-1}} \right) \tag{50.}$$

$$\beta_i = \frac{\vec{r}_i^T \vec{r}_i}{\vec{s}_{i-1}^T \vec{r}_{i-1}}, \tag{51.}$$

where we drop the j subscript from β as it is no longer necessary. We then take the inner product of the conjugate Gram-Schmidt Equation (35.) with $\mathbf{r}_j$ to get

$$(\vec{s}_i) \cdot (\vec{r}_j) = \left[\vec{r}_i + \sum_{k=0}^{i-1} \beta_{ik} \vec{s}_k \right] \cdot (\vec{r}_j) \tag{52.}$$

$$\vec{s}_i^T \vec{r}_j = \vec{r}_i^T \vec{r}_j + \sum_{k=0}^{i-1} \beta_{ik} \vec{s}_k^T \vec{r}_j , \tag{53.}$$

and when i = j,

$$\vec{s}_i^T \vec{r}_i = \vec{r}_i^T \vec{r}_i + \sum_{k=0}^{i-1} \beta_{ik} \vec{s}_k^T \vec{r}_i . \tag{54.}$$

However, we know that

$$\vec{s}_k^T \vec{r}_i = 0, k < i , \tag{55.}$$

which is the case with each term in the summation in Equation (54.), so

$$\vec{s}_i^T \vec{r}_i = \vec{r}_i^T \vec{r}_i . \tag{56.}$$

At long last, substitution of Equation (56.) into Equation (51.) gives what is known as the **Fletcher-Reeves equation**,

$$\beta_i = \frac{\vec{r}_i^T \vec{r}_i}{\vec{r}_{i-1}^T \vec{r}_{i-1}} . \tag{57.}$$

This is the equation that, when used in Equation (35.), allows the calculation of the **A**-conjugate search vectors $\{\mathbf{s}_i\}$ using only the values of the residual vectors[23].

Practical results have shown that a modification of the Fletcher-Reeves formula is more effective in the general case. This is called the **Polak-Ribiere formula**[24], and is given by

$$\beta_i = \max\left\{ 0, \frac{\vec{r}_i^T \left(\vec{r}_i - \vec{r}_{i-1} \right)}{\vec{r}_{i-1}^T \vec{r}_{i-1}} \right\} . \tag{58.}$$

The reason is that the generated search vectors $\{\mathbf{s}_i\}$ are only perfectly **A**-conjugate when the function $E_{Avg}(\mathbf{w})$ is a quadratic function. For a highly nonlinear error function or for

values of **w** far from a local minimum, this tends not to be the case. For this reason the generated sequence of search vectors to "lose conjugacy", which means they are no longer linearly independent, and thus the search can get "stuck" and fail to converge to a minimum. The Polak-Ribiere formula avoids this pitfall by periodically "resetting" the conjugate-gradients search at the current value of **w** by setting the adjustment term β_i to zero, which means the gradient is once again used as the search vector (recall that this is how the conjugate-gradients algorithm starts in the first place)[25].

A summary of the conjugate-gradients algorithm that was used in this paper is shown below in Table 6.

Line #	**3-Layer MLP with N input, M hidden, and P output neurons**	Comments
1	Initialize epoch counter = 0	
2	Initialize the weights vector **w**	Uniform Distribution with $\mu = 0$, $\sigma^2 = 1 / M$
3	Set Initial Learning Rate η=0.10; Annealing Rate η_{rate}=0.9	
4	Set stopping criteria α=0.001	
5	Calculate initial vectors **g**, **s**, and $\mathbf{r}_0$	All three have dimension [M*(N + P) + M + P]
6	While not done training do	
7	Use Brent's Method to calculate value of η which	
8	Minimizes $E_{Avg}(\mathbf{w}+\eta\mathbf{s})$ where **w** and **s** are held constant	
9	Update weight vector $\mathbf{w} + \mathbf{w} + \eta\mathbf{s}$	
10	Use Back-Prop to calculate new gradient vector **g**	
	Set $\mathbf{r}_{prev} = \mathbf{r}$	
11	Set $\mathbf{r} = -\mathbf{g}$	
12	Use Polack-Ribiere formula to calculate β	$\beta = \max\left\{0, \frac{\vec{r}^{\,T}\left(\vec{r} - \vec{r}_{prev}\right)}{\vec{r}_{prev}^{\,T}\vec{r}_{prev}}\right\}$
13	Calculate new $\mathbf{s} = \mathbf{r} + \beta\mathbf{s}$	
14	If stopping criteria is not met Then increment epoch count	Stopping criterion met when all training examples have been learned with 100% accuracy.
15	Else done training is true	
16	Loop back to Line 6	
17	Record # epochs, total training time, initial average error, final average error, and percent correct	Percent Correct is calculated after training time is computed. It is the number of correct answers in a test of all 128 possible input values.
18	STOP	

Table 6: The Conjugate-Gradient Algorithm

The Finite-Difference Algorithm

The third training algorithm that will be tested in this study is the so-called **finite-difference** algorithm. This algorithm is identical to back-prop except for the computation of gradients during the back-propagation phase. Instead of using the local gradient calculations in equations (12.), (14.), and (25.), this algorithm will directly calculate the central-difference approximation to each gradient with respect to each weight in the network. Thus, for the finite-difference approximation algorithm,

$$\frac{\partial E}{\partial W_{ij}} = \frac{E(W_{ij} + \varepsilon) - E(W_{ij} - \varepsilon)}{2\varepsilon}, \qquad (59.)$$

where $E(W_{ij})$ is the calculated error value, averaged over a training epoch, at that particular value of the individual weight W_{ij}. The strategic motivation for this algorithm is to utilize a multiprocessor computer to calculate each central-difference in parallel. This study, however, uses an implementation coded for a single process on a single processing BUS, so the potential for productivity gains through parallel computation are immediately forfeit. However, since we are interested in testing only accuracy, generalization ability, and number of training iterations, this study will still be a valid test of the performance of the finite-difference algorithm relative to the other two. A summary of the finite-difference algorithm used in this study is shown below in Table 7.

Line #	**3-Layer MLP with N input, M hidden, and P output neurons**	Comments
1	Initialize epoch counter = 0	
2	Initialize (M+1 by P) output layer weights matrix $\mathbf{W_{output}}$	Uniform Distribution with $\mu = 0$, $\sigma^2 = 1 / M$
3	Initialize (M+1 by P) output layer learning rate matrix $\boldsymbol{\lambda}_{\mathbf{output}}$	Use an initial value of 0.01
4	Initialize (N+1 by M) hidden layer weights matrix $\mathbf{W_{hidden}}$	Uniform Distribution with $\mu = 0$, $\sigma^2 = 1 / N$
5	Initialize (N+1 by M) hidden layer learning rate matrix $\boldsymbol{\lambda}_{\mathbf{hidden}}$	Use an initial value of 0.01
6	While not done training do	
7	For x = 0 to NUM_EXAMPLES	
8	Set input vector **i** = training example x	
9	Set desired response vector **d**	
10	Compute hidden layer vector **h**	See Figure 3 above.
11	Compute output layer vector **o**	See Figure 4 above.
12	Compute error signal component E_x	$E_x = \frac{1}{2}\sum_{i=1}^{P}(\vec{d}_i - \vec{o}_i)^2$
13	Calculate output layer central-difference gradients	$\frac{\partial E}{\partial W_{ij}} = \frac{E(W_{ij}+\varepsilon) - E(W_{ij}-\varepsilon)}{2\varepsilon}, \forall\, i, j; \varepsilon = 0.0001$
15	Adjust output layer weights matrix $\mathbf{W_{output}}$	$W_{output,j.k} = (W_{output,j.k})_{previous} + (\lambda_{j,k})(\vec{\delta}_{output,k})(\vec{h}_j); j = 1...M; k = 1...P$
16	Adjust hidden layer weights matrix $\mathbf{W_{hidden}}$	$W_{hidden,i.j} = (W_{hidden,i.j})_{previous} + (\lambda_{i,j})(\vec{\delta}_{hidden,j})(\vec{i}_i); i = 1...N; j = 1...M$
17	Adjust individual learning rates matrices $\boldsymbol{\lambda}_{\mathbf{hidden}}$, $\boldsymbol{\lambda}_{\mathbf{output}}$	
18	Next x	
19	Calculate Average Error Value E_{avg}	$E_{avg} = \left[\left(\frac{1}{NE}\right)\sum_{x=1}^{NE} E_x\right], NE = NUM\_EXAMPLES$
20	If stopping criteria is not met Then increment epoch count	
	Else done training is true	
21	Loop back to line 4	
22	Record # epochs, total training time, initial average error, and final average error	
23	STOP	

Table 7: The Finite-Difference Algorithm

Monte Carlo Simulation

Test Methodology

The Monte Carlo Simulation designed to test the effectiveness of the three training algorithms is outlined in Table 8, below.

Simulation # (*)	ρ Values (**)	Hidden Layer Size (***)	Test Problem	Training Algorithm
1-12	2, 4, 6	4, 10, 15, 20	Pattern Recog.	Back-Prop
13-24	2, 4, 6	4, 10, 15, 20	Pattern Recog.	C-G
25-36	2, 4, 6	4, 10, 15, 20	Pattern Recog.	Finite-Diff.
37-48	2, 4, 6	4, 6, 8, 10	Bit Counting	Back-Prop
49-60	2, 4, 6	4, 6, 8, 10	Bit Counting	C-G
61-72	2, 4, 6	4, 6, 8, 10	Bit Counting	Finite-Diff.

(*) Each individual simulation in this study is composed of 500 trials. A single trial in this study is defined as one complete run through the training algorithms outlined in Table 1, Table 2, or Table 3.
(**) "ρ" is the exponent of the energy-like error function E to be minimized during training. See Appendix C.
(***) Both the hidden layer size and ρ value were varied in this study, but held constant during an individual simulation. *For example, a Hidden Layer Size of 10 means that there were 10 neurons in the hidden layer during that simulation.*

Table 8: Monte Carlo Simulation Outline

As shown in Table 8, both the size of the hidden layer and the exponent ρ of the energy-like error function E were varied to determine the best value of each to use when comparing the three training algorithms for each problem. See Appendix C for the changes to the preceding training algorithm equations that result from a ρ value greater than 2.

The randomly-generated, uniformly-distributed initial values of the synaptic weights were the random elements in each trial of a given simulation. The effectiveness of the training algorithms in each case was determined by using the data to answer two basic questions:

1. **Did the network learn the problem sufficiently well?**
2. **How much work was required to complete the training?**

For each trial in this study, the following was used as the **stopping criteria** for all of the three training algorithms tested. *The supervised-training trial was said to converge if the network was able to learn 100% of the training examples (10 for pattern*

recognition, 128 for bit counting) in a given epoch. If the network's performance in the training example set was less than 100%, training continues epoch-by-epoch until either convergence or the hard limit of maximum epochs was reached. This hard limit, determined by empirical testing, was chosen to be 5,000 epochs for pattern recognition and 15,000 epochs for bit counting.

Finally, each of the simulations described above were performed on a Compaq Presario F700 laptop computer with a 1.90GHz AMD Athlon Dual-Core Processor and 2.00GB of RAM.

Simulation Results

A Note About ρ Values

The purpose of this study was to test the most competitive network configurations for each problem-training algorithm combination. In every problem-training algorithm combination in this study the best performance was achieved with a ρ value of 2. In most cases the networks using higher values of ρ performed substantially worse, especially with respect to convergence. It was therefore decided to omit discussion of ρ values greater than 2 from the following discussion, although the relevant performance data for these omitted trials can still be found in Appendix D.

Pattern Recognition

1. Did the network learn the problem sufficiently well?

In the pattern recognition problem, the network was presented with all 10 training examples, and the stopping criteria was when all 10 training examples have been learned. Therefore, if a training trial did not converge, the network was not able to learn the pattern recognition problem. The convergence results for the pattern recognition trials (all ρ = 2) are shown on the next page in Table 9.

Algorithm	Hidden Layer Size	Total Trials	Convergent Trials
Back-Propagation	4	500	500
Back-Propagation	10	500	500
Back-Propagation	15	500	500
Back-Propagation	20	500	500
Conjugate-Gradient	4	500	499
Conjugate-Gradient	10	500	500
Conjugate-Gradient	15	500	500
Conjugate-Gradient	20	500	499
Finite-Difference	4	500	500
Finite-Difference	10	500	500
Finite-Difference	15	500	500
Finite-Difference	20	500	500

Table 9: Pattern Recognition Convergence Results

As Table 9 shows, only 2 out of the 6,000 total pattern recognition trials failed to converge, both with the conjugate-gradient method. These two trials do not affect the overall message of Table 9, however, which is that the network was able to learn this problem equally well in all four configurations with each of the three training algorithms.

2. How much work was required to complete the training?

The work required to train the network was measured by the number of epochs necessary to train the network to convergence. The summary statistics for each of the twelve distributions generated in this study are shown below in Table 10.

Algorithm	Hidden Layer Size	Min. Epochs	Mean Epochs	Median Epochs	Max. Epochs	StDev Epochs
Back-Propagation	4	42	174	66	436	70
Back-Propagation	10	30	93	160	224	31
Back-Propagation	15	12	79	89	186	26
Back-Propagation	20	22	69	76	145	21
Conjugate-Gradient	4 (*)	4	14	14	35	4
Conjugate-Gradient	10	4	9	9	24	2
Conjugate-Gradient	15	3	8	8	24	2
Conjugate-Gradient	20 (*)	3	7	7	20	2
Finite-Difference	4	32	105	98	411	44
Finite-Difference	10	19	61	57	177	22
Finite-Difference	15	13	53	51	142	17
Finite-Difference	20	16	48	45	142	15

(*) Only the 499 convergent trials were used to calculate these summary statistics.

Table 10: Pattern Recognition Epochs – Summary Statistics

Table 10 shows that the conjugate-gradient algorithm was by far the most efficient in terms of number of epochs required to learn the pattern recognition problem. Among the conjugate-gradient results Table 10 shows that a hidden layer of 20 neurons was the best performing of the configurations tested.

Bit Counting

1. Did the network learn the problem sufficiently well?

In the bit counting problem, the stopping criteria for each training algorithm was the same as in pattern recognition – stop training when the network achieves 100% accuracy across the training set or the hard limit in epochs is reached. Table 11 below is a summary of the convergence results for the bit counting trials (all $\rho = 2$) performed in this study.

Algorithm	Hidden Layer Size	Total Trials	Convergent Trials
Back-Propagation	4	500	328
Back-Propagation	6	500	471
Back-Propagation	8	500	500
Back-Propagation	10	500	500
Conjugate-Gradient	4	500	347
Conjugate-Gradient	6	500	476
Conjugate-Gradient	8	500	499
Conjugate-Gradient	10	500	500
Finite-Difference	4	500	21
Finite-Difference	6	500	250
Finite-Difference	8	500	413
Finite-Difference	10	500	484

Table 11: Bit Counting Convergence Results

Table 11 clearly shows that all of the three training algorithms showed better convergence results as the size of the hidden layer was increased from 4 to 10 neurons. Figure 10 below shows the same data in a graphical format – here one clearly sees the poor bit counting performance of the finite-difference algorithm relative to the other two.

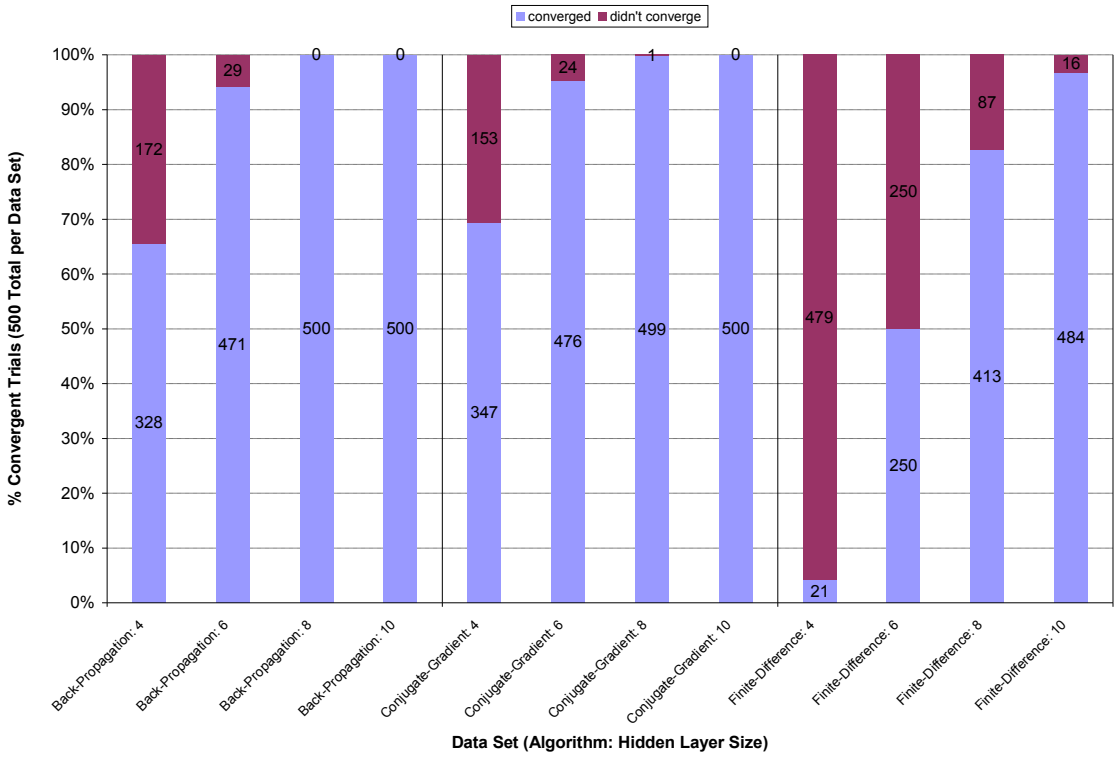

Figure 10: Percent Convergent Trials by Algorithm and Hidden Layer Size

Analysis of convergence behavior is a measure of how effectively the network was able to learn the examples within training set. For bit counting, however, there is another dimension to the notion of how well the network was able to learn the problem, and that is the generalization performance against the possible inputs that were not part of the training set. This performance was evaluated by testing the newly trained network against each of the 128 possible bit counting inputs and recording the results on a scale from 0 to 100% correct. The summary statistics for resulting distributions of test scores is shown below in Table 12.

Algorithm	Hidden Layer Size	Min. % Correct	Mean % Correct	Median % Correct	Max. % Correct	StDev % Correct
Back-Propagation	4	12.5	98.6	100.0	100.0	6.7
Back-Propagation	6	81.3	97.6	100.0	100.0	3.8
Back-Propagation	8	70.3	90.0	90.6	100.0	7.0
Back-Propagation	10	67.2	84.6	83.6	100.0	7.2
Conjugate-Gradient	4	0.0	93.8	100.0	100.0	20.2
Conjugate-Gradient	6	89.1	97.9	98.4	100.0	2.7
Conjugate-Gradient	8	75.0	92.5	93.8	100.0	5.5
Conjugate-Gradient	10	71.9	87.4	87.5	100.0	5.8
Finite-Difference	4	16.4	73.5	81.3	100.0	23.9
Finite-Difference	6	8.6	89.9	96.1	100.0	14.2
Finite-Difference	8	21.9	90.8	93.0	100.0	9.4
Finite-Difference	10	50.0	85.6	86.3	100.0	8.5

Table 12: Bit Counting % Correct Summary Statistics (All Data)

Another useful way to compare the data that generated Table 12 is to construct and plot empirical cumulative distribution functions from the data, as seen in Figures 11 through 13, below.

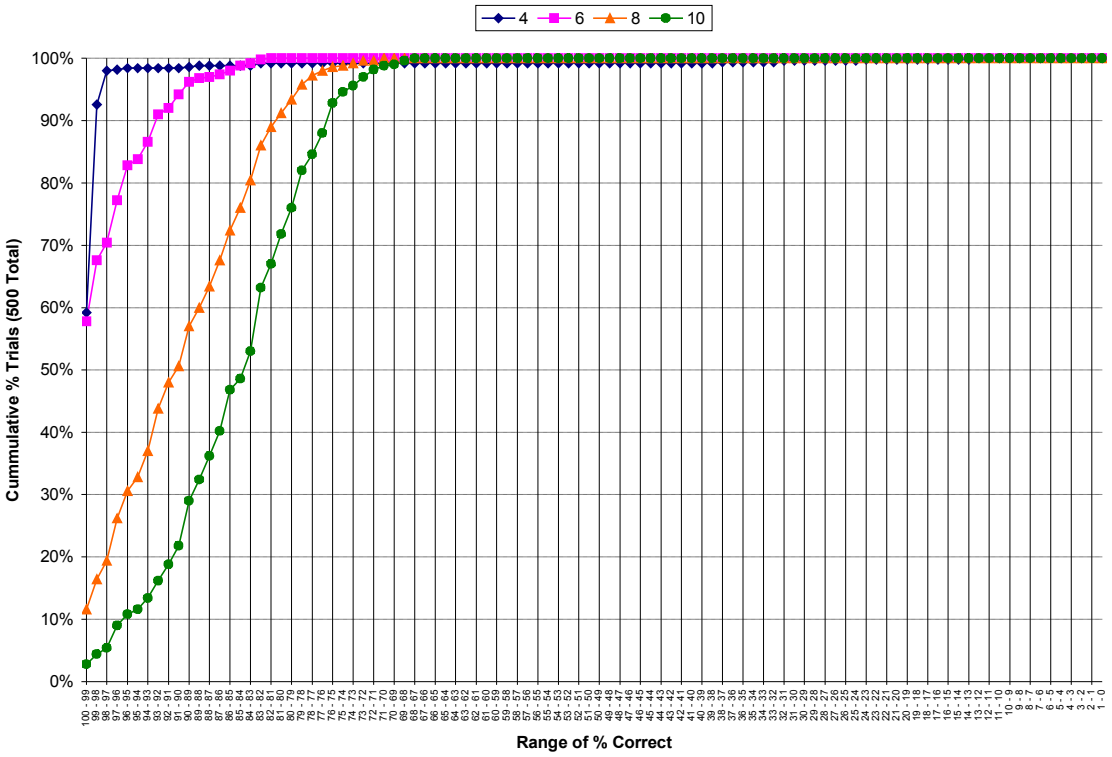

Figure 11: Empirical CDF of Percent Correct for Back-Prop Bit Counting

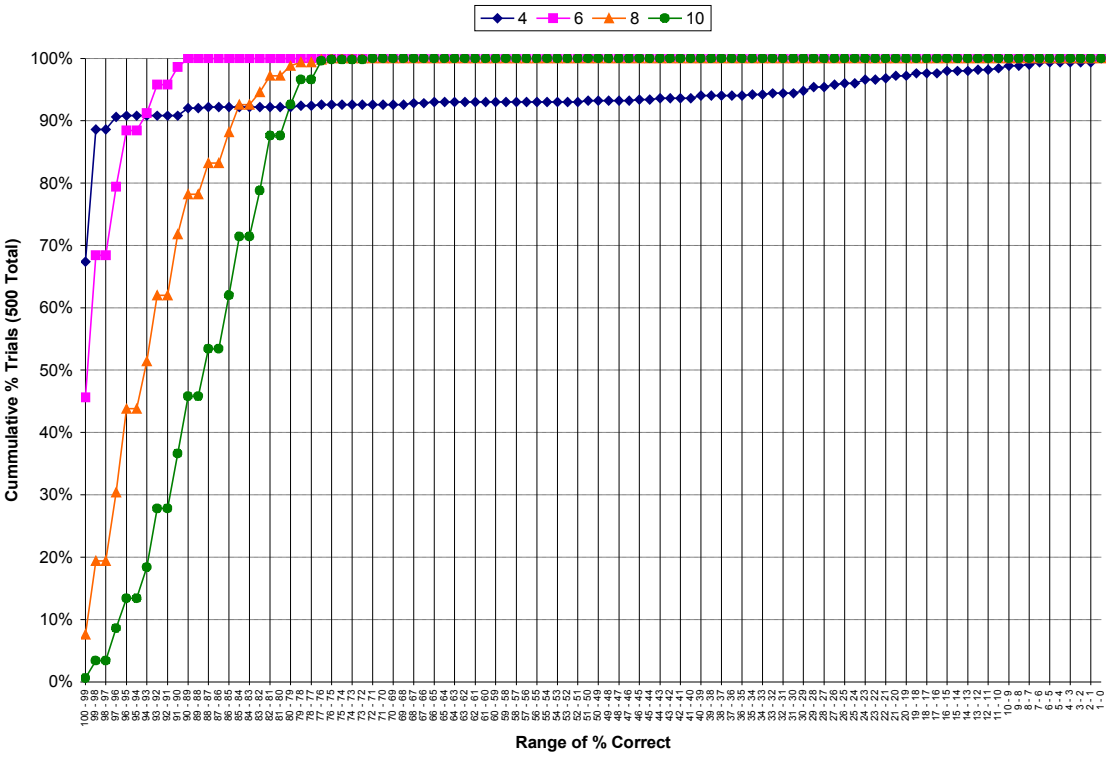

Figure 12: Empirical CDF of Percent Correct for C-G Bit Counting

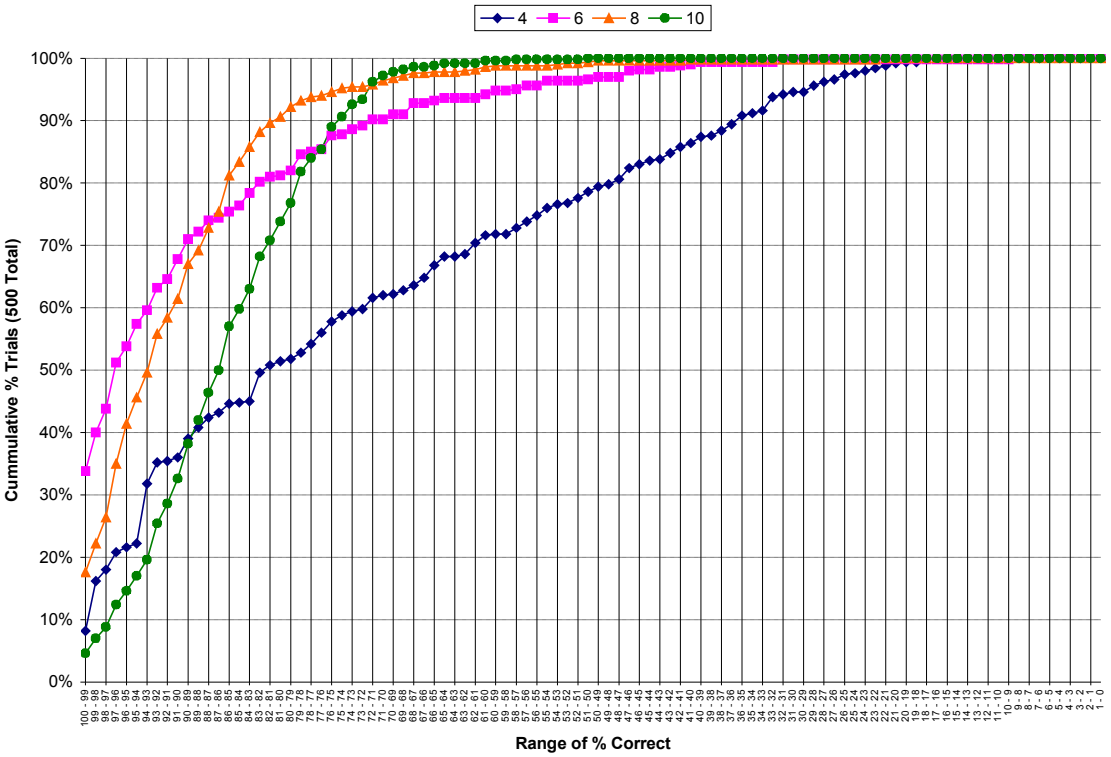

Figure 13: Empirical CDF of Percent Correct for Finite-Diff Bit Counting

From the preceding figures one can conclude that for hidden layer sizes of 6, 8, and 10 neurons the conjugate-gradient algorithm's statistical performance was superior to the other two algorithms. One curious feature of these figures, however, is that the back-prop algorithm appears to have outperformed the other two with a hidden layer size of 4.

2. How much work was required to complete the training?

Again the amount of work required to train the network will be evaluated by comparing the distributions of the number of epochs necessary to train the network. The summary statistics of the distributions generated during the bit counting trials are shown below in Table 13.

Algorithm	Hidden Layer Size	Min. Epochs	Mean Epochs	Median Epochs	Max. Epochs	StDev Epochs
Back-Propagation	4	5,574	11,725	11,370	15,000	2,842
Back-Propagation	6	1,411	5,346	4,630	15,000	3,062
Back-Propagation	8	1,071	2,745	2,768	11,534	958
Back-Propagation	10	967	2,107	1,959	4,682	696
Conjugate-Gradient	4	215	6,837	3,227	15,000	6,241
Conjugate-Gradient	6	105	3,323	1,775	15,000	3,903
Conjugate-Gradient	8	74	1,166	684	15,000	1,520
Conjugate-Gradient	10	70	584	384	4,033	560
Finite-Difference	4	2,148	14,649	15,000	15,000	1,821
Finite-Difference	6	908	10,116	14,949	15,000	5,322
Finite-Difference	8	826	5,822	3,676	15,000	4,792
Finite-Difference	10	801	3,263	2,332	15,000	2,879

Table 13: Bit Counting Epochs Summary Statistics (All Data)

As was done with the percent correct data, empirical cumulative distribution functions were created from the distributions of epochs. These are displayed below in Figures 14 through 16.

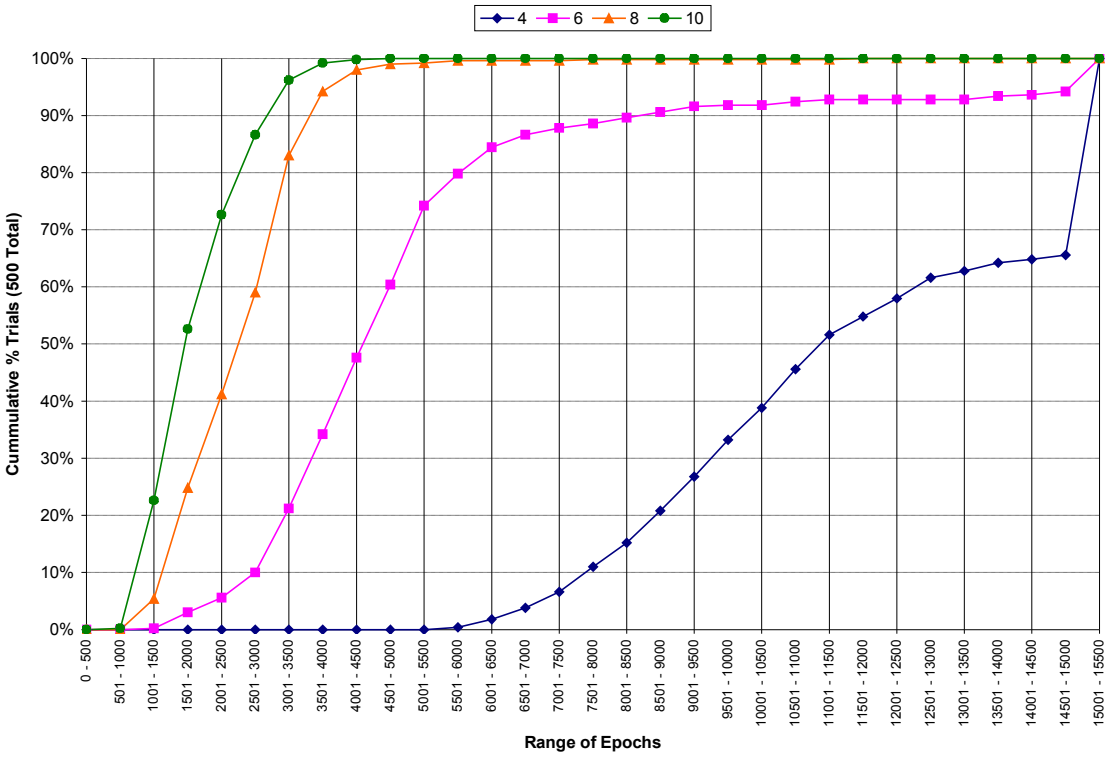

Figure 14: Empirical CDF of Epochs for Back-Prop Bit Counting

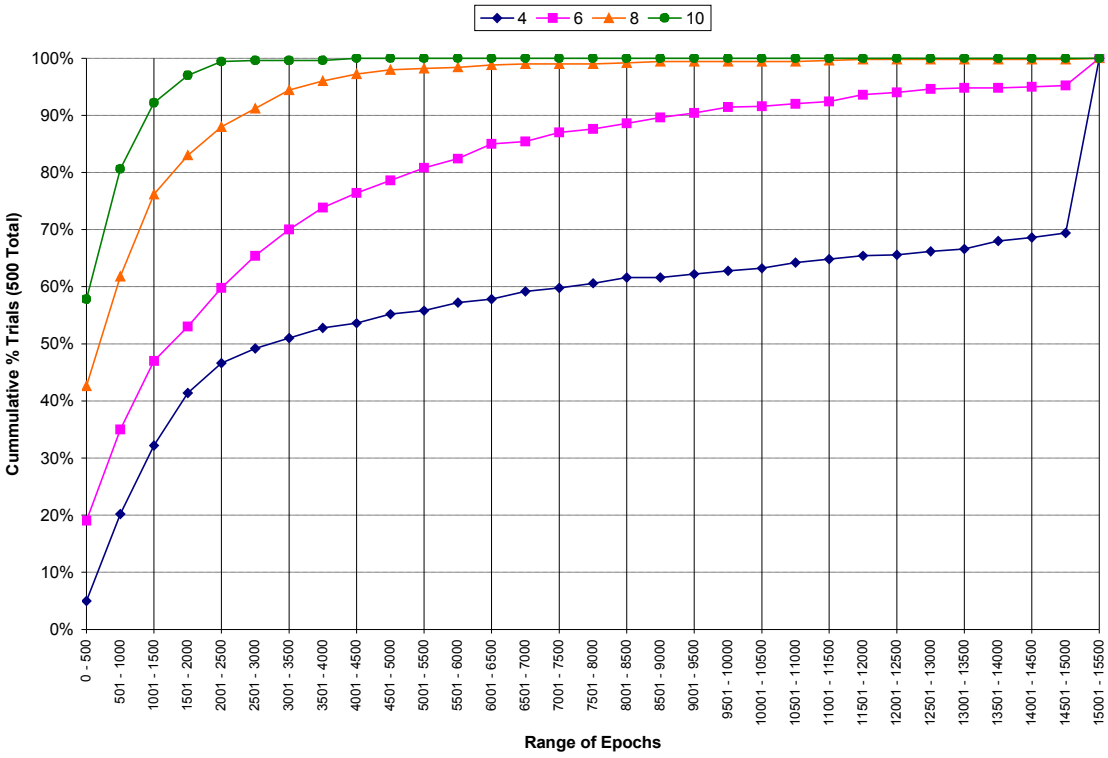

Figure 15: Empirical CDF of Epochs for C-G Bit Counting

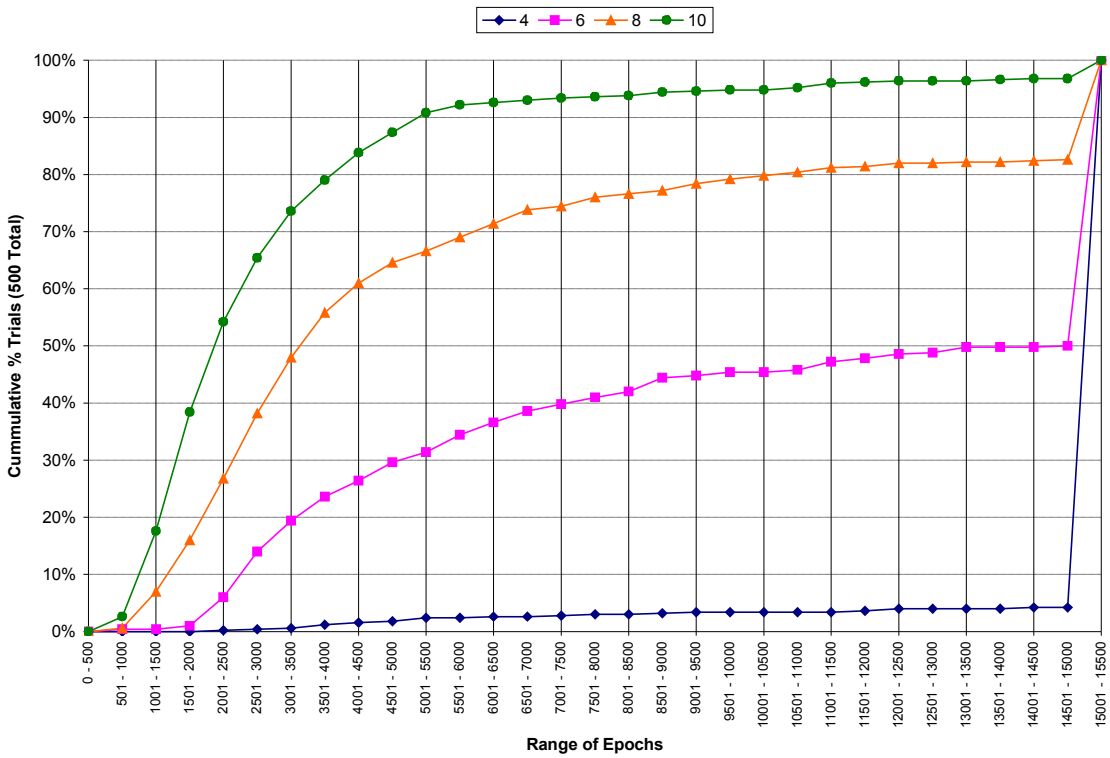

Figure 16: Empirical CDF of Epochs for Finite-Diff Bit Counting

As with pattern recognition, so with bit counting – the conjugate-gradient algorithm required far fewer iterations for each hidden layer size than the other two algorithms by any statistical measure.

Therefore, the results of this study suggest that the conjugate-gradient algorithm was superior to the other two algorithms for bit counting as well. Figures 12 and 15 suggest that within the conjugate-gradient data set, the network with a hidden layer of 6 neurons gave the best overall accuracy performance with an acceptable trade-off in the distribution of epochs.

Conclusions and Future Work

All three algorithms were able to teach the pattern recognition task with near-perfect accuracy regardless of the hidden layer size chosen. The conjugate-gradient algorithm required much fewer training epochs to do so by any statistical measure. Therefore the data suggests that the conjugate-gradient algorithm was the best of the three algorithms for teaching the pattern recognition task. The bit counting problem proved much more difficult for all three training algorithms, but again, the statistical

results of this study demonstrate the overall superiority of the conjugate-gradient relative to the other two.

The fact that a simple problem like bit counting should prove to be much harder for the network to learn than pattern recognition begs some explanation. One reason could have to do with the concept of **Hamming distances**. The Hamming distance between two vectors is defined as the number of components in which the two vectors differ in value[26]. As a representative example, Table 14 below shows the Hamming distances between the vector (0, 0, 0) and each of the other vectors of 0 or 1 of length 3.

Vector 1	Vector 2	Hamming Distance
(0, 0, 0)	(0, 0, 0)	0
(0, 0, **0**)	(0, 0, **1**)	1
(0, **0**, 0)	(0, **1**, 0)	1
(0, **0**, **0**)	(0, **1**, **1**)	2
(**0**, 0, 0)	(**1**, 0, 0)	1
(**0**, 0, **0**)	(**1**, 0, **1**)	2
(0, 0, 0)	(1, 1, 0)	2
(**0**, **0**, **0**)	(**1**, **1**, **1**)	3

Table 14: Example Hamming Distances

The pattern recognition problem in this study has 10 unique input vectors and 10 corresponding unique output vectors, a one-to-one input-output functional relationship. This can be rephrased in terms of Hamming distances as follows: in pattern recognition, when the Hamming distance between two input vectors is greater than zero, the Hamming distance between the corresponding output vectors will also be greater than zero. On the contrary, the 7-digit bit counting problem has 128 unique vectors but only 8 unique output vectors, a many-to-one functional relationship. Again, in terms of Hamming distances: in bit counting, when the Hamming distance between two input vectors is greater than zero, the Hamming distance between the corresponding output vectors will be greater than or equal to zero. The conjecture that the results of this study suggest is that MLPs are more easily trained to perform tasks whose functional model is one-to-one than those whose model is one-to-many or many-to-one.

Some suggestions for future work are listed as follows:

- Try an individually-adjustable ρ value for every synaptic weight that would be an increasing function of the absolute value of the error signal (desired – actual). The

thought behind the higher ρ values is that larger errors would be "trained out" more quickly by the corresponding higher order gradients that would result. However, this is only true when the absolute value of the error signal is greater than or equal to one. When this value is less than one, the larger ρ values would make the gradient much smaller, which could cause small but appreciable error signals to persist.

- Try a variable/adaptive finite-difference epsilon to try and improve the performance of the finite-difference algorithm.

References

1.) Rosenblatt, F., 1958. "The Perceptron: A probabalistic model for information storage and organization in the brain", Psychological Review, vol. 65, pp. 386-408.
2.) Haykin, S. Neural Networks: A Comprehensive Foundation. 2nd Ed. New Jersey: Prentice-Hall; 1999. pp. 63-64.
3.) Ibid., pp.156-159.
4.) Ibid., pp.159-161.
5.) Ibid., pp. 168-169.
6.) Ibid., pp. 135-137.
7.) Ibid., p. 160.
8.) Ibid., p. 162.
9.) Ibid., pp. 179-181.
10.) LeCun, Y., 1993. Efficient Learning and Second-order Methods, A Tutorial at NIPS 93, Denver.
11.) Haykin, 1999. p. 184.
12.) Ibid., p. 184.
13.) Ibid., p. 63.
14.) Minsky, M.L., 1961. "Steps towards artificial intelligence", Proceedings of the Institute of Radio Engineers, vol. 49, pp. 8-30.
15.) Rumelhart, D.E., G.E. Hinton, and R.J. Williams, 1986a. "Learning representations of back-propagation", in D.E. Rumelhart and J.L McCleland, eds., vol 1, Chapter 8, Cambridge, MA: MIT Press.
16.) Hebb, D.O. The Organization of Behavior: A Neuropsychological Theory. New York: Wiley; 1949. p. 62.
17.) Haykin, 1999. pp. 162-163.
18.) Ibid., pp. 164-166.
19.) Shewchuck, J.R. 1994. "An Introduction to the Conjugate Gradient Method Without the Agonizing Pain" [Online]. Available from: http://www-2.cs.cmu.edu/jrs/jrspapers.html#cg. pp.21-24.
20.) Haykin, 1999. pp. 237-238.
21.) Ibid., pp. 234-236.
22.) Shewchuck, 1994. pp. 25-26.
23.) Ibid., pp. 30-32.

24.) Polak, E., and G. Ribiere, 1969. “Note sur la convergence de methods de directions conjuguees”, Revue Francaise Information Recherche Operationnelle, vol. 16, pp. 35-43.
25.) Haykin, 1999. pp. 239-240.
26.) Paul E. Black, “Hamming distance”, in Dictionary of Algorithms and Data Structures [Online], Paul E. Black, ed., U.S. National Institute of Standards and Technology. May 31, 2006. Available from http://www.nist.gov/dads/HTML/HammingDistance.html.

Appendices

Appendix A: Pattern Recognition Calculator Digits

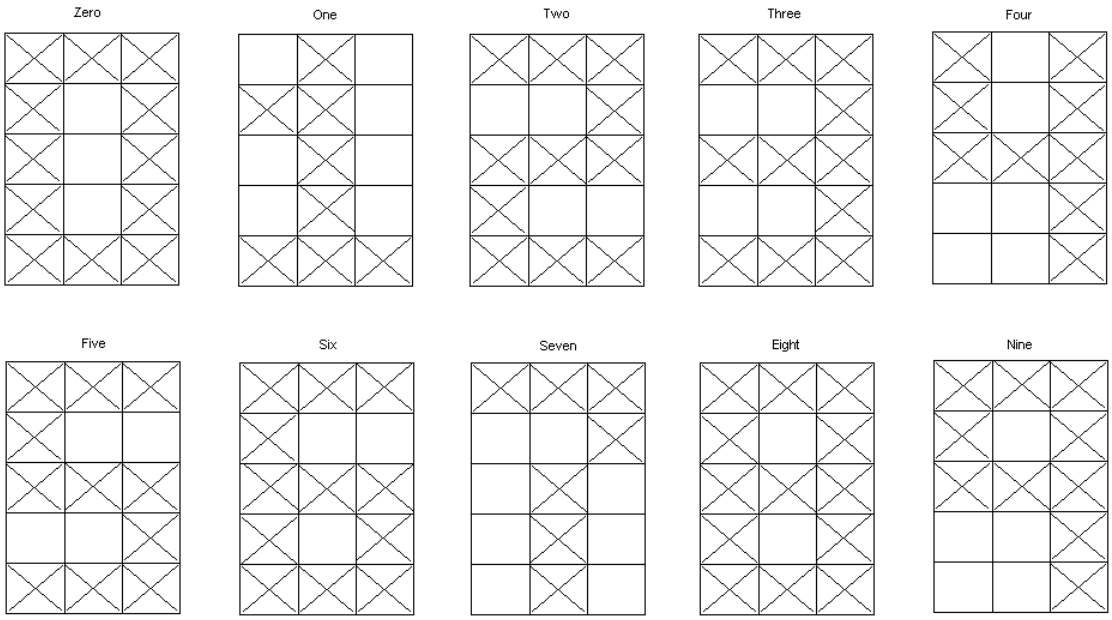

Figure A1: Fifteen Pixel Calculator Digits

Appendix B: Bit Counting Training Example Set

Bit Count	Possible Training Examples (128)	Number of Examples (68)	Training Examples
0	1	1	0000000
1	7	7	1000000 0100000 0010000 0001000 0000100 0000010 0000001
2	21	11	1100000 0110000 0011000 0001100 0000110 0000011 1000001 0010001 0010100 1010000 0000101
3	35	15	1110000 0000111 0011100 0111000 0001110 1010010 1000101 0011010 1100001 1000011 1010001 0100101 0101010 1001001 1010100
4	35	15	1111000 0001111 1100011 0111100 0011110 0101011 1010101 1101010 0110110 0110101 1110100 0010111 0111001 1001110 1100101
5	21	11	1111100 1110011 1001111 0011111 1110110 0111110 1010111 1011101 1101101 0101111

			1111010
6	7	7	1110111 1011111 0111111 1101111 1111011 1111101 1111110
7	1	1	1111111

Table B1: Bit Counting Training Examples

Appendix C: Derivation Resulting from Higher-Order ρ Values

The exponent of the error function E was varied during this study, meaning that instead of Equation 2

$$E = \tfrac{1}{2}\sum_{k=1}^{P}(\vec{d}_k - \vec{o}_k)^2 = \tfrac{1}{2}\sum_{k=1}^{P}(\vec{e}_k)^2 \qquad (2.)$$

we would have

$$E = \tfrac{1}{2}\sum_{k=1}^{P}(\vec{d}_k - \vec{o}_k)^{\rho} = \tfrac{1}{2}\sum_{k=1}^{P}(\vec{e}_k)^{\rho}, \rho = 2,4,6. \qquad (2.C)$$

The following changes to the Back-Prop equations are then necessary. Recall from Equation 6 that

$$\frac{\partial E}{\partial W_{ij}} = \frac{\partial E}{\partial \vec{e}_j}\frac{\partial \vec{e}_j}{\partial \vec{y}_j}\frac{\partial \vec{y}_j}{\partial \vec{v}_j}\frac{\partial \vec{v}_j}{\partial W_{ij}}, \qquad (6.)$$

but now

$$\frac{\partial E}{\partial \vec{e}_j} = \frac{\partial}{\partial \vec{e}_j}\left[\tfrac{1}{2}\sum_{k=1}^{P}(\vec{e}_k)^{\rho}\right] \qquad (7.C.a)$$

$$\frac{\partial E}{\partial \vec{e}_j} = \left(\tfrac{1}{2}\right)(\rho)\left(\vec{e}_j^{\,\rho-1}\right). \qquad (7.C)$$

The rest of Equation 6 is unchanged, so that now Equation 12 becomes

$$\frac{\partial E}{\partial W_{ij}} = -\left[\tfrac{\rho}{2}\right][\vec{e}_j^{\,\rho-1}][\varphi'(\vec{v}_j)][\vec{y}_i], \qquad (12.C)$$

so that for output neurons j, Equation 14 becomes

$$\vec{\delta}_j = \left[\tfrac{\rho}{2}\right][\vec{e}_j^{\,\rho-1}][\varphi'(\vec{v}_j)]. \qquad (14.C)$$

Appendix D: Raw Monte Carlo Simulation Data

D1: Back-Prop Pattern Recognition

Data Set	Total Trials	Convergent Trials	Min Epochs	Mean Epochs	Median Epochs	Max Epochs	StdDev Epochs
4 - 2	500	500	42	174	66	436	70
4 - 4	500	500	45	247	60	1,467	134
4 - 6	500	500	102	609	100	2,594	382
10 - 2	500	500	30	93	160	224	31
10 - 4	500	500	20	101	230	253	38
10 - 6	500	500	45	195	525	646	88
15 - 2	500	500	12	79	89	186	26
15 - 4	500	500	20	79	97	208	30
15 - 6	500	500	36	139	176	569	59
20 - 2	500	500	22	69	76	145	21
20 - 4	500	500	14	65	76	170	25
20 - 6	500	500	22	107	128	289	42

Table D.1.1: Summary Statistics for Number of Epochs by Data Set

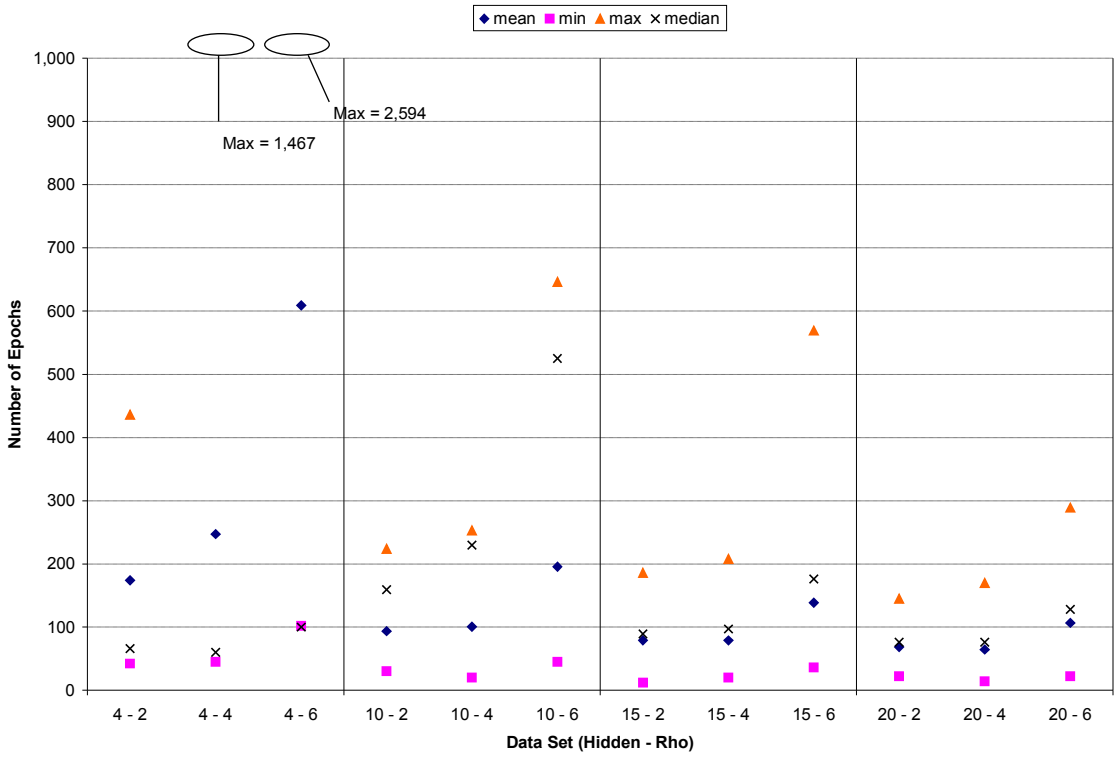

Figure D.1.1: Plot of Summary Statistics for Epochs by Data Set (All Data)

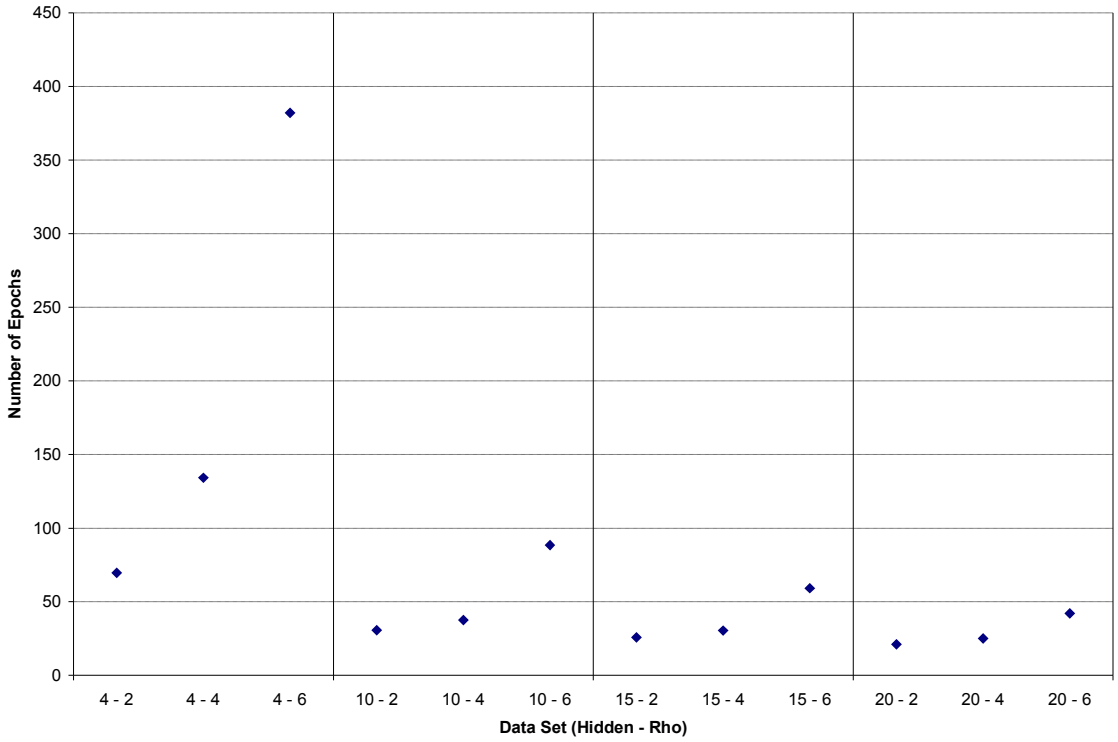

Figure D.1.2: Plot of Standard Deviation of Epochs by Data Set (All Data)

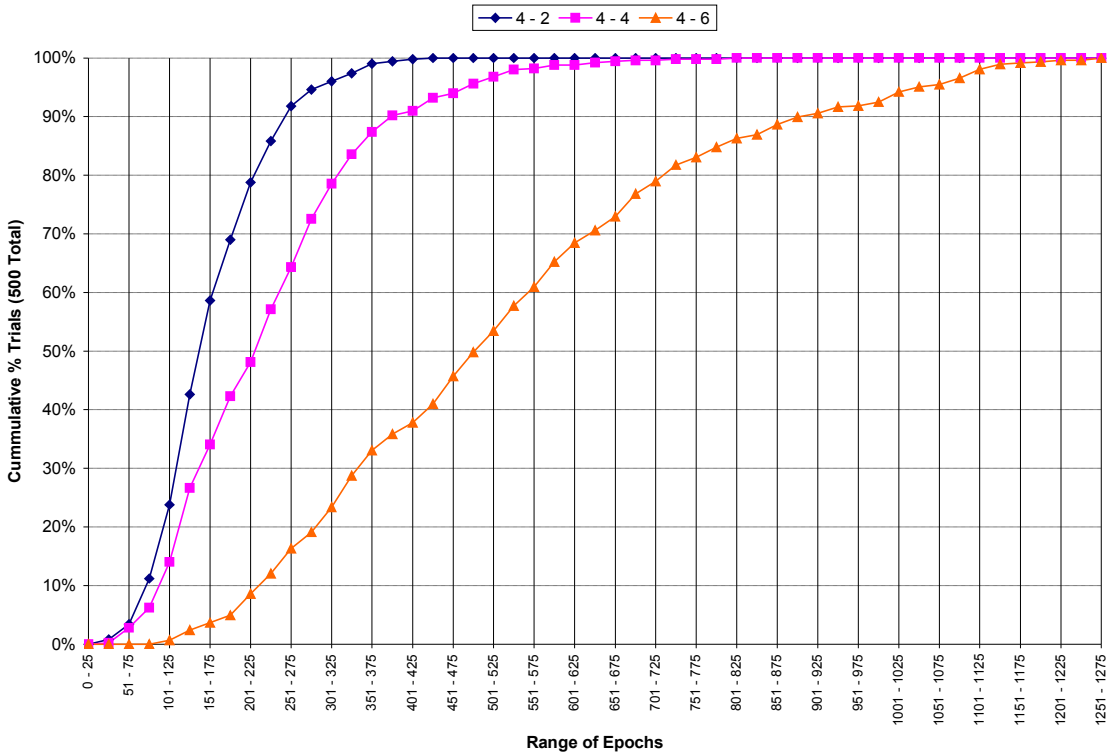

Figure D.1.3: Empirical CDF of Epochs for Hidden Layer Size 4 Networks by ρ Value

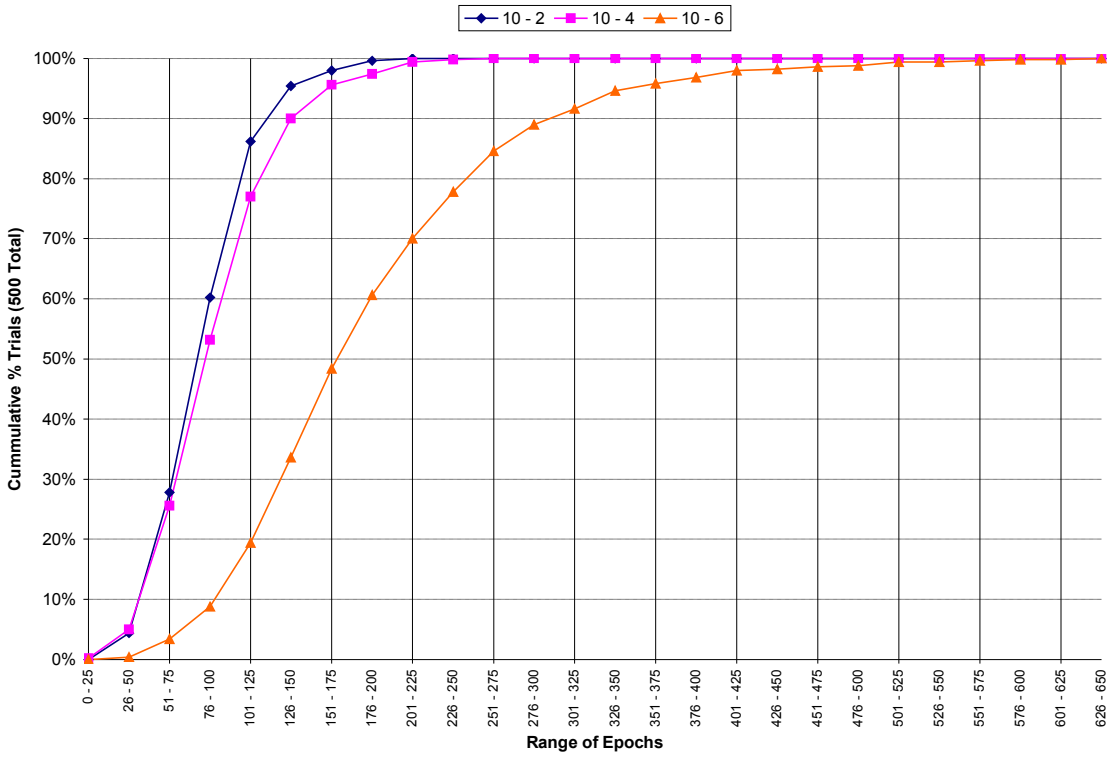

Figure D.1.4: Empirical CDF of Epochs for Hidden Layer Size 10 Networks by ρ Value

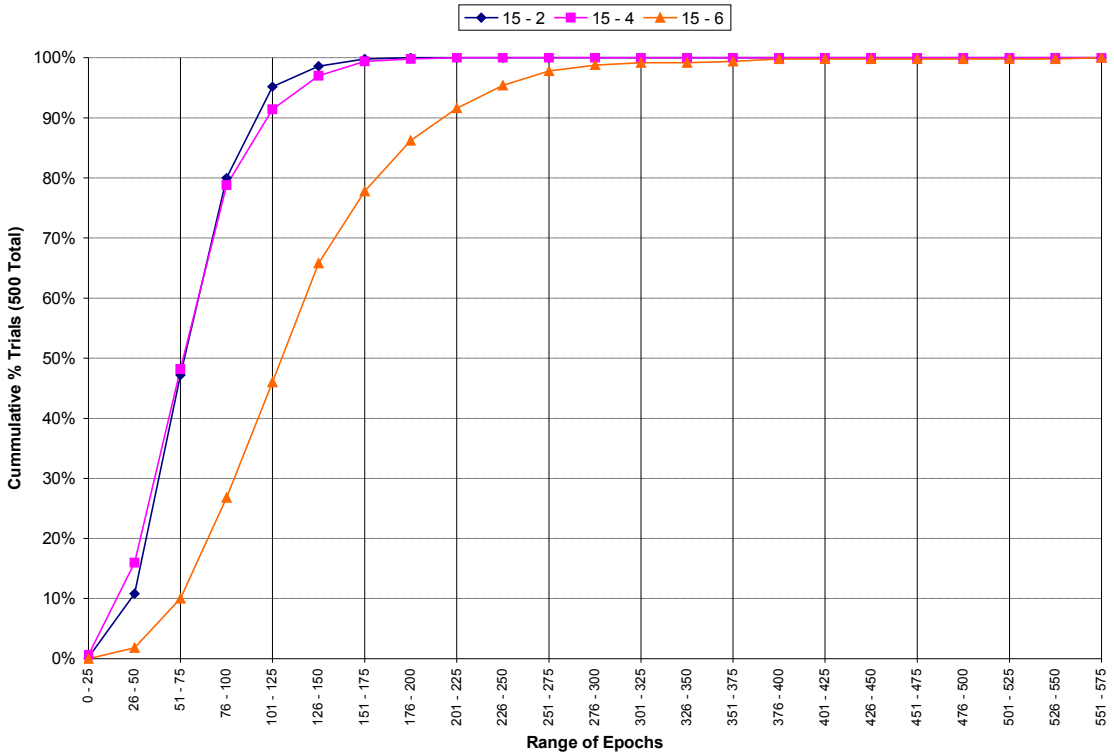

Figure D.1.5: Empirical CDF of Epochs for Hidden Layer Size 15 Networks by ρ Value

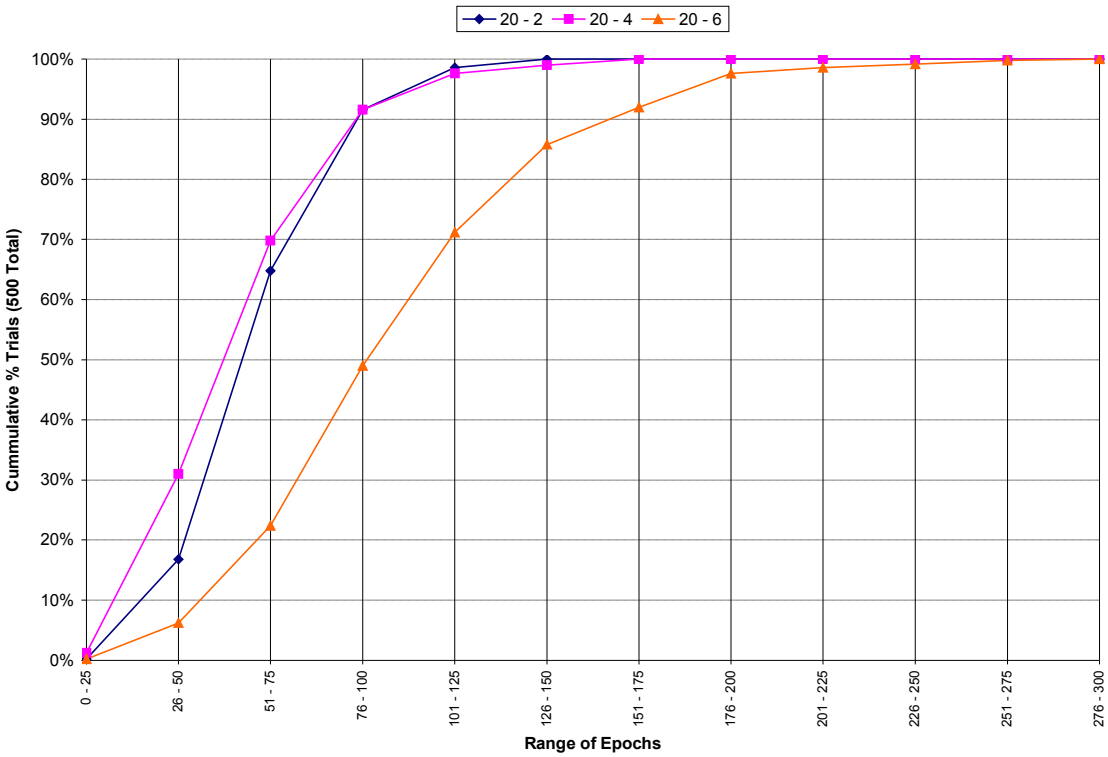

Figure D.1.6: Empirical CDF of Epochs for Hidden Layer Size 20 Networks by ρ Value

D2: C-G Pattern Recognition

Data Set	Total Trials	Convergent Trials	Min Epochs	Mean Epochs	Median Epochs	Max Epochs	StdDev Epochs
4 - 2	500	499	4	24	14	5,000	223
4 - 4	500	473	6	312	15	5,000	1.135
4 - 6	500	314	7	1,888	25	5,000	2.400
10 - 2	500	500	4	9	9	24	2
10 - 4	500	444	4	570	10	5,000	1.575
10 - 6	500	287	6	2,137	14	5,000	2.469
15 - 2	500	500	3	8	8	24	2
15 - 4	500	406	4	947	9	5,000	1,952
15 - 6	500	243	5	2,575	5,000	5,000	2,597
20 - 2	500	499	3	17	7	5,000	223
20 - 4	500	396	3	1,047	8	5,000	2,028
20 - 6	500	204	6	2,964	5,000	5,000	2,455

Table D.2.1: Summary Statistics for Epochs by Data Set (All Data)

Data Set	Convergent Trials	Min Epochs	Mean Epochs	Median Epochs	Max Epochs	StdDev Epochs
4 - 2	499	4	14	14	35	4
4 - 4	473	6	44	15	3,022	178
4 - 6	314	7	45	17	1,761	148
10 - 2	500	4	9	9	24	2
10 - 4	444	4	12	9	103	9
10 - 6	287	6	12	11	38	4
15 - 2	500	3	8	8	24	2
15 - 4	406	4	9	8	42	4
15 - 6	243	5	10	10	22	2
20 - 2	**499**	**3**	**7**	**7**	**20**	**2**
20 - 4	396	3	9	8	37	3
20 - 6	204	6	10	9	22	3

Table D.2.1: Summary Statistics for Epochs by Data Set (Convergent Trials Only)

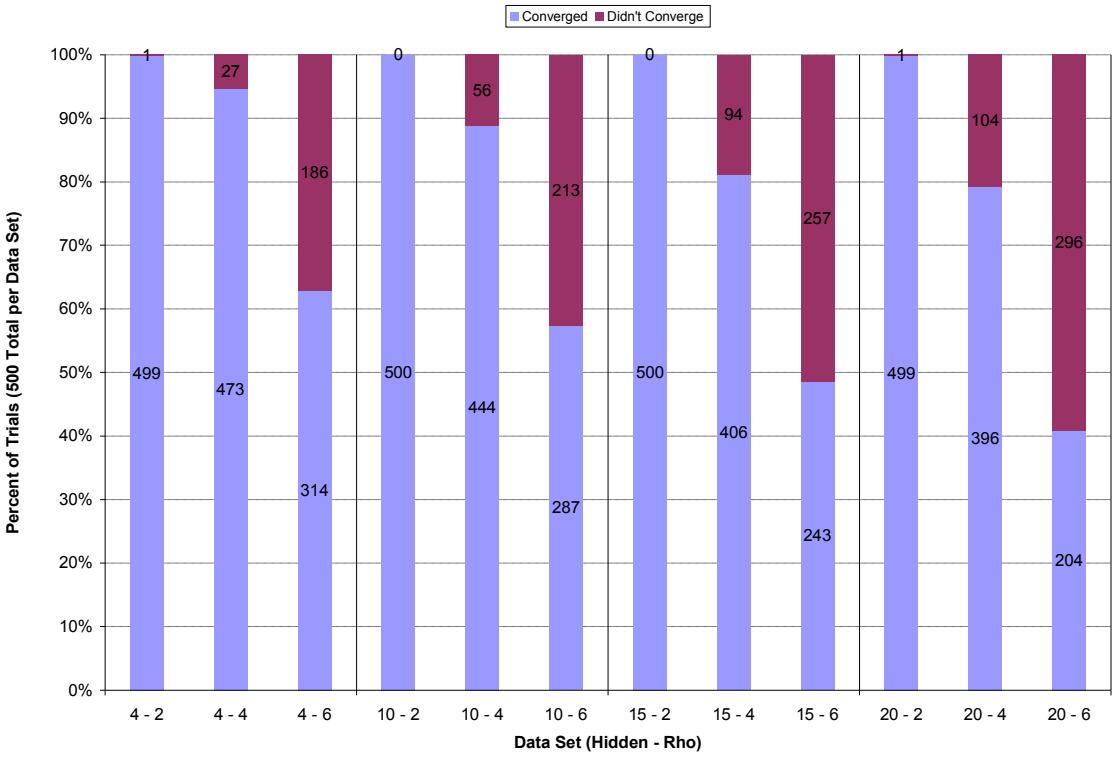

Figure D.2.1: Percent of Convergent Trials by Data Set

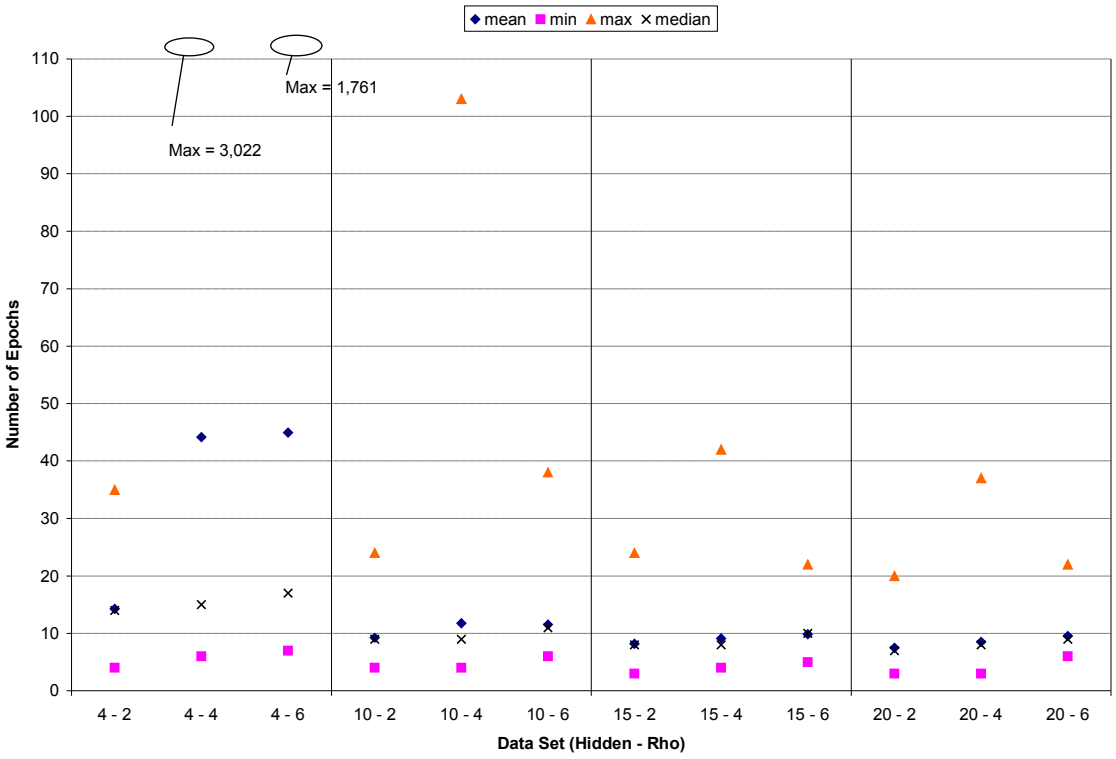

Figure D.2.2: Plot of Summary Statistics for Epochs by Data Set (Convergent Trials Only)

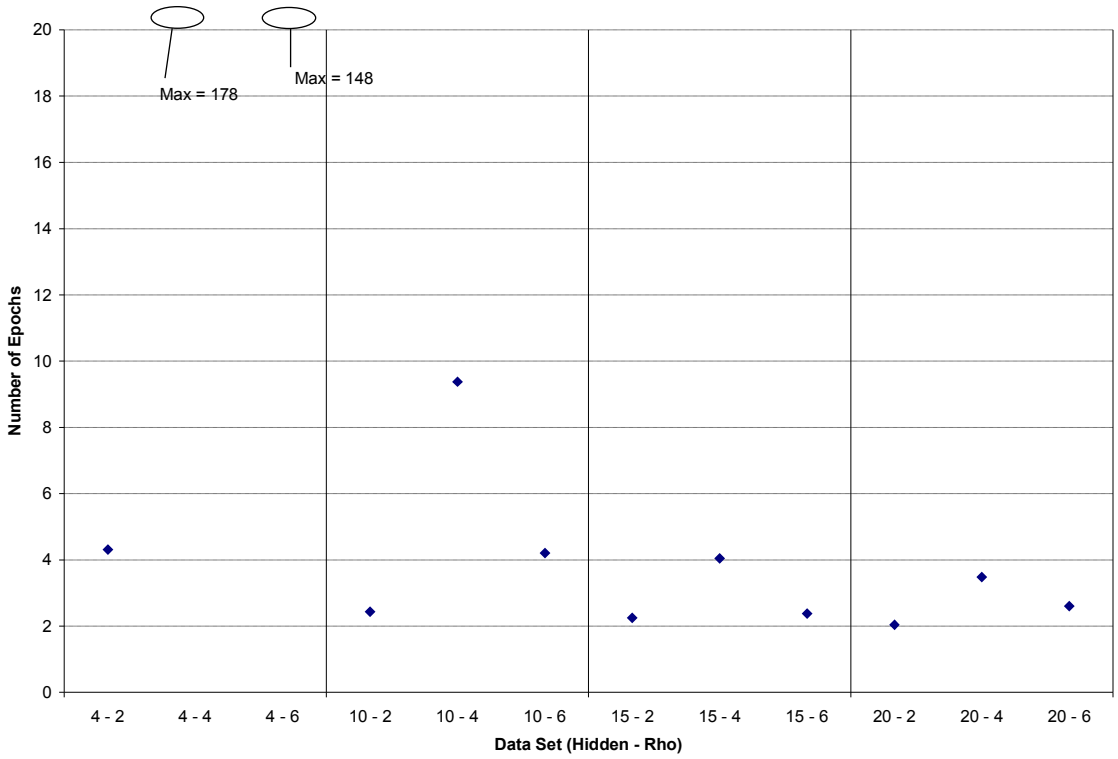

Figure D.2.3: Plot of Standard Deviation of Epochs by Data Set (Convergent Trials Only)

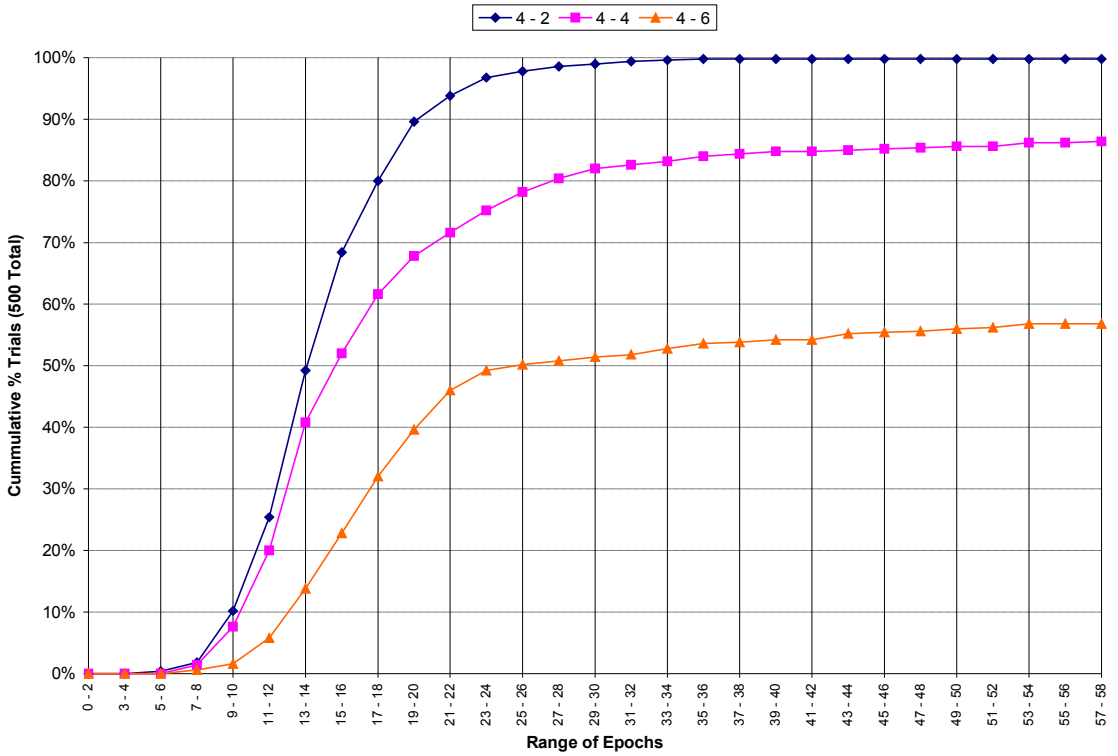

Figure D.2.4: Empirical CDF of Epochs for Hidden Layer Size 4 Networks by ρ Value (All Data)

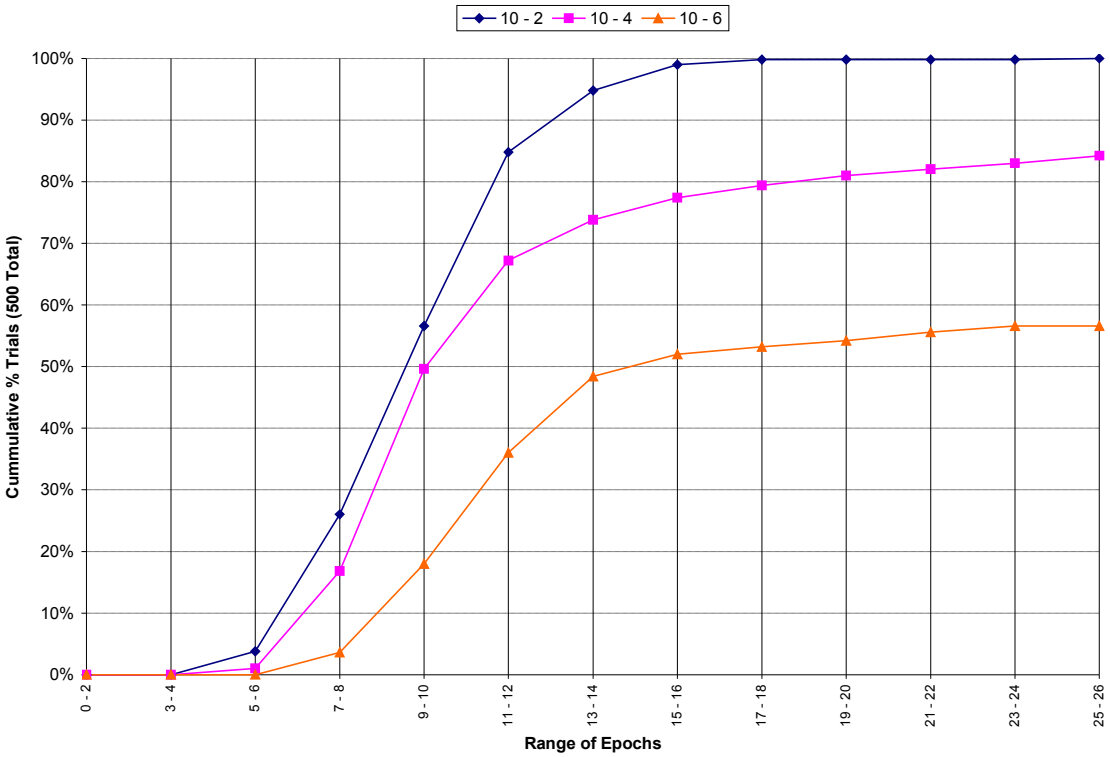

Figure D.2.5: Empirical CDF of Epochs for Hidden Layer Size 10 Networks by ρ Value (All Data)

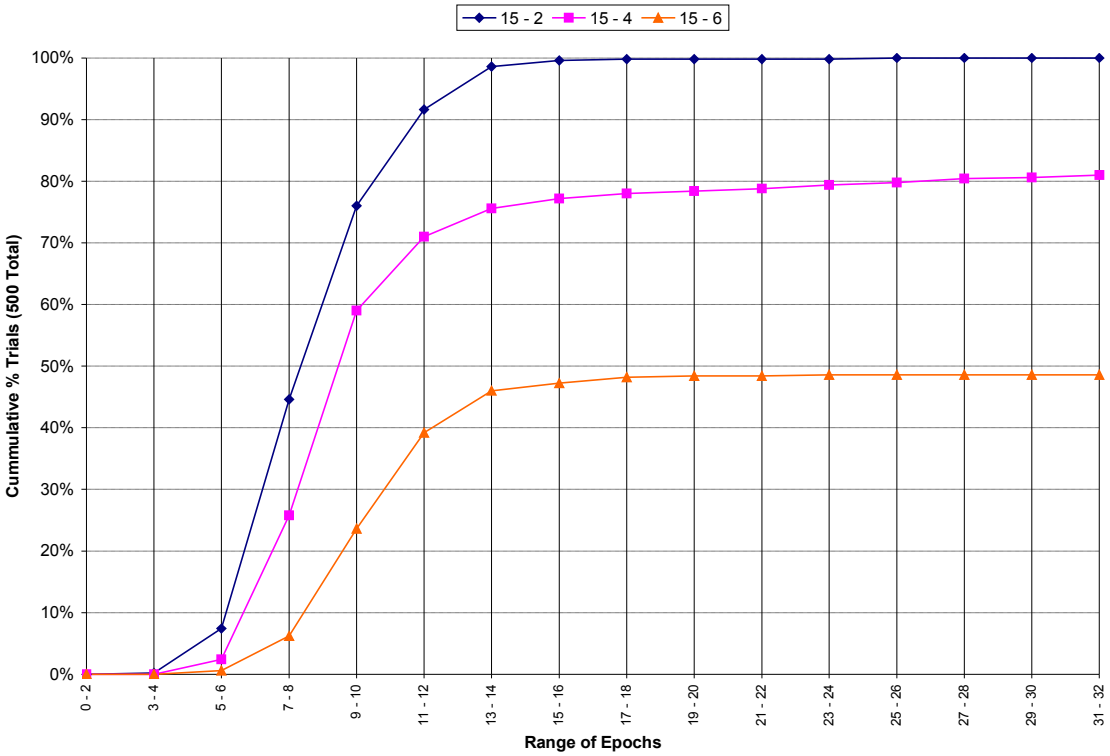

Figure D.2.6: Empirical CDF of Epochs for Hidden Layer Size 15 Networks by ρ Value (All Data)

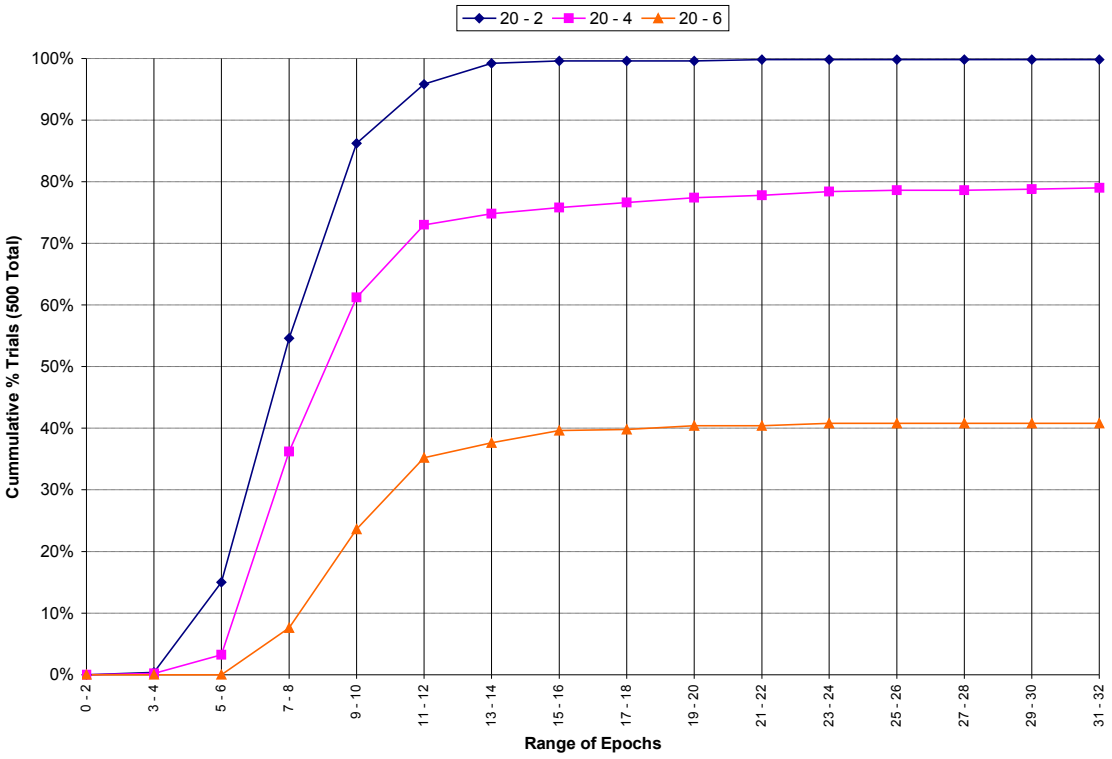

Figure D.2.7: Empirical CDF of Epochs for Hidden Layer Size 20 Networks by ρ Value (All Data)

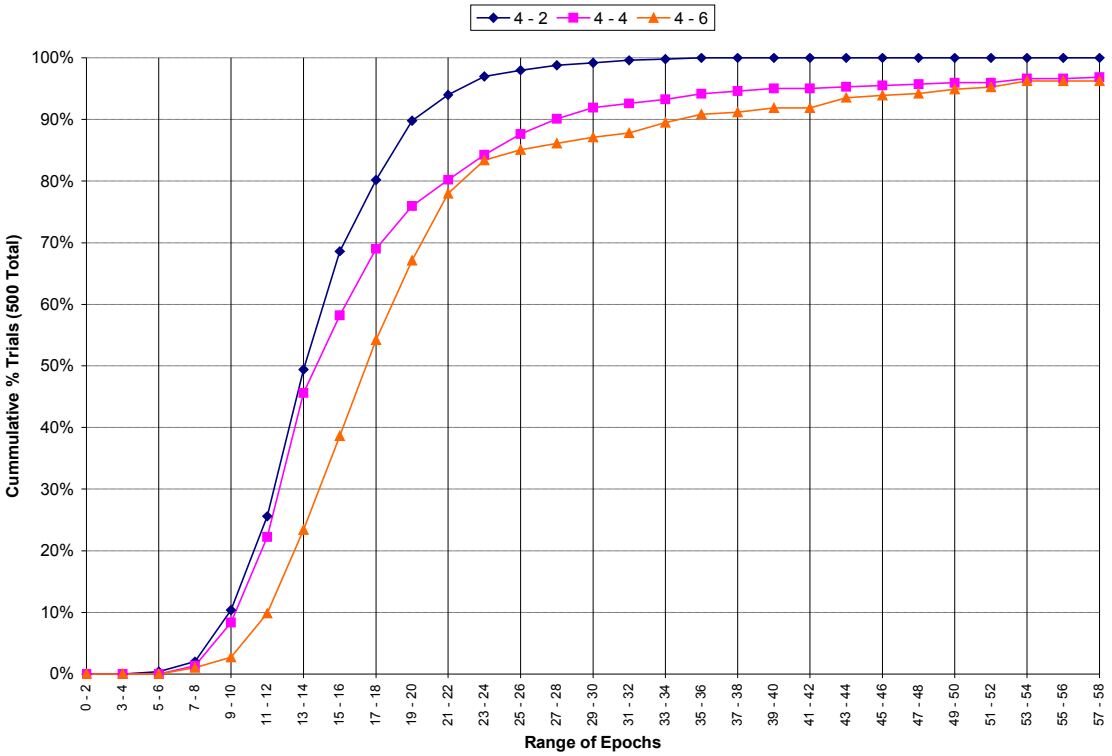

Figure D.2.8: Empirical CDF of Epochs for Hidden Layer Size 4 Networks by ρ Value (Convergent Trials Only)

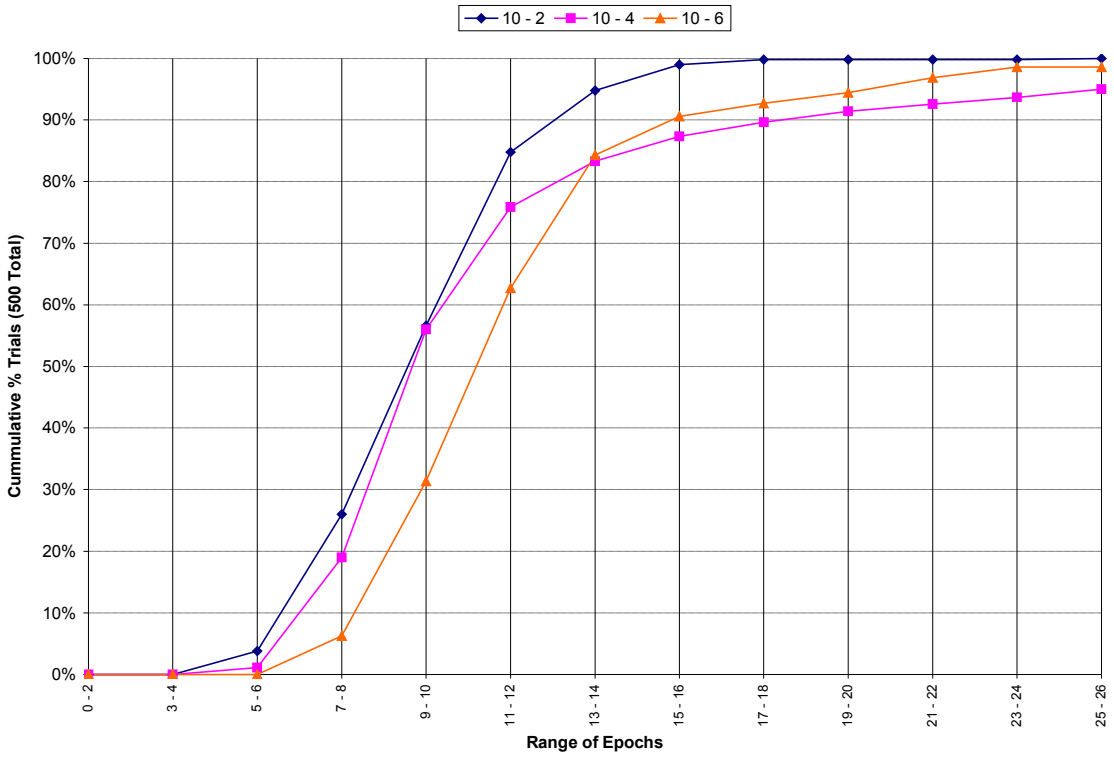

Figure D.2.9: Empirical CDF of Epochs for Hidden Layer Size 10 Networks by ρ Value (Convergent Trials Only)

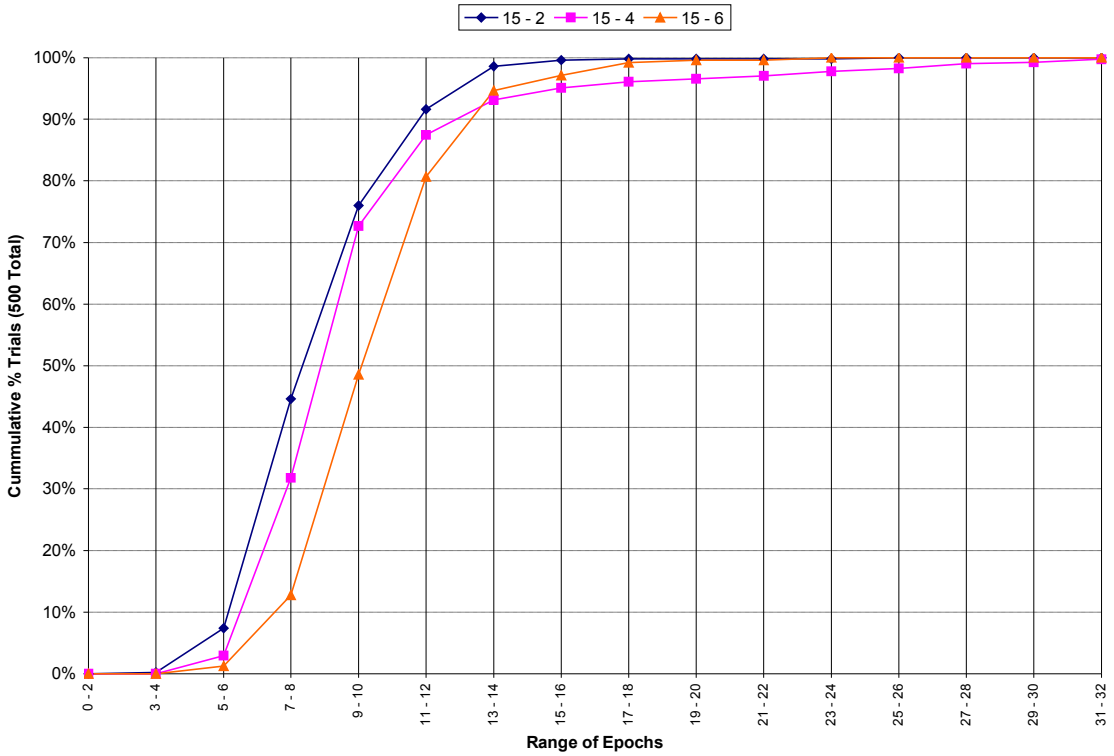

Figure D.2.10: Empirical CDF of Epochs for Hidden Layer Size 15 Networks by ρ Value (Convergent Trials Only)

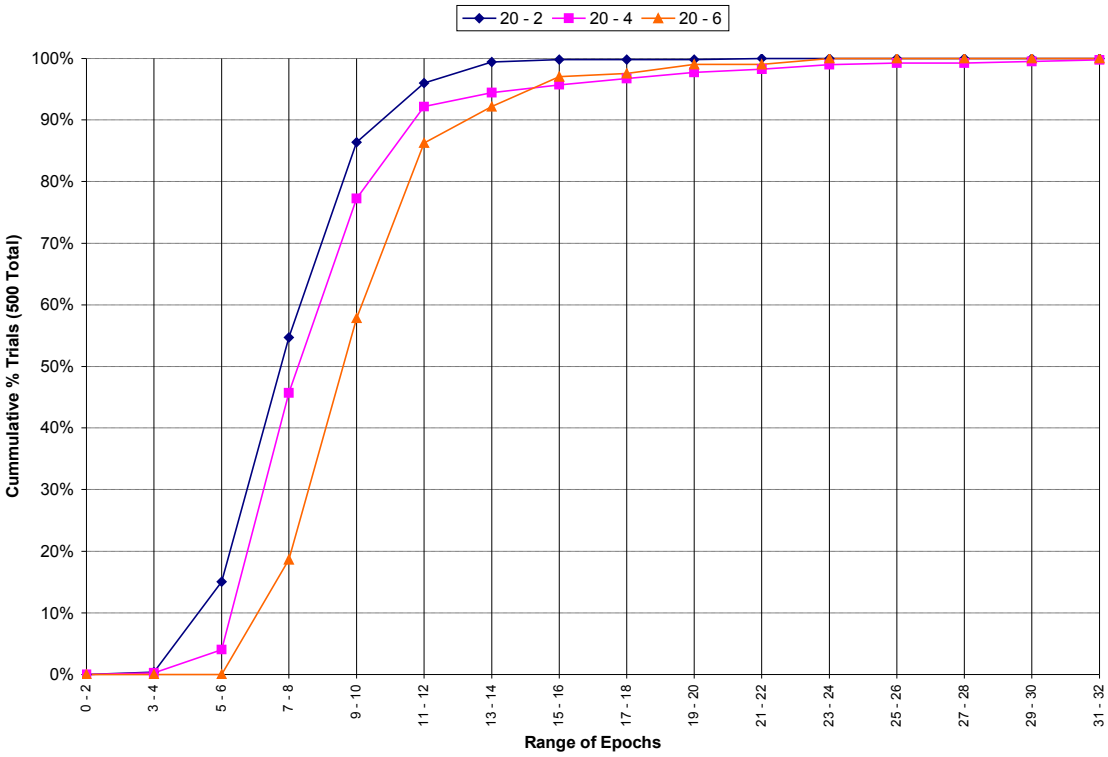

Figure D.2.11: Empirical CDF of Epochs for Hidden Layer Size 20 Networks by ρ Value (Convergent Trials Only)

D3: Finite-Difference Pattern Recognition

Data Set	Total Trials	Convergent Trials	Min Epochs	Mean Epochs	Median Epochs	Max Epochs	StdDev Epochs
4 - 2	500	500	32	105	98	411	44
4 - 4	500	500	18	154	135	853	87
4 - 6	500	500	42	388	342	1,449	226
10 - 2	500	500	19	61	57	177	22
10 - 4	500	500	16	67	64	159	25
10 - 6	500	500	34	125	114	427	58
15 - 2	500	500	13	53	51	142	17
15 - 4	500	500	11	54	51	128	20
15 - 6	500	500	24	87	80	247	37
20 - 2	500	500	16	48	45	142	15
20 - 4	500	500	14	47	44	117	17
20 - 6	500	500	17	75	70	212	30

Table D.3.1: Summary Statistics for Epochs by Data Set (All Data)

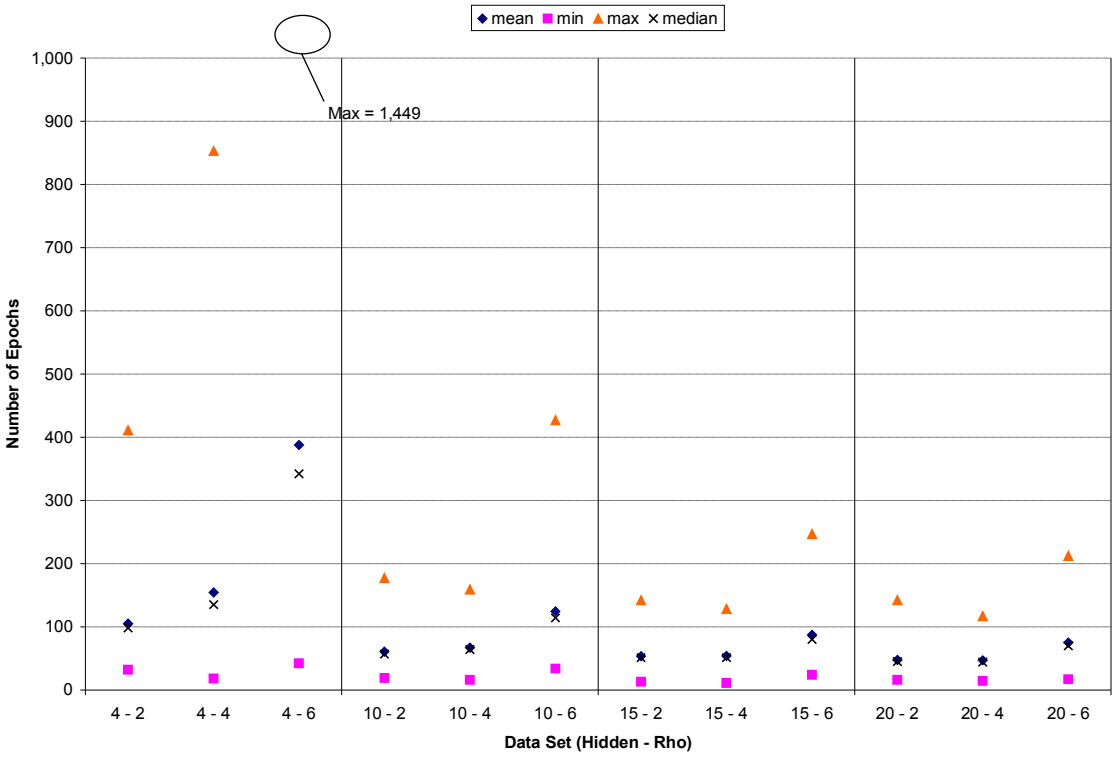

Figure D.3.1: Plot of Summary Statistics for Epochs by Data Set (All Data)

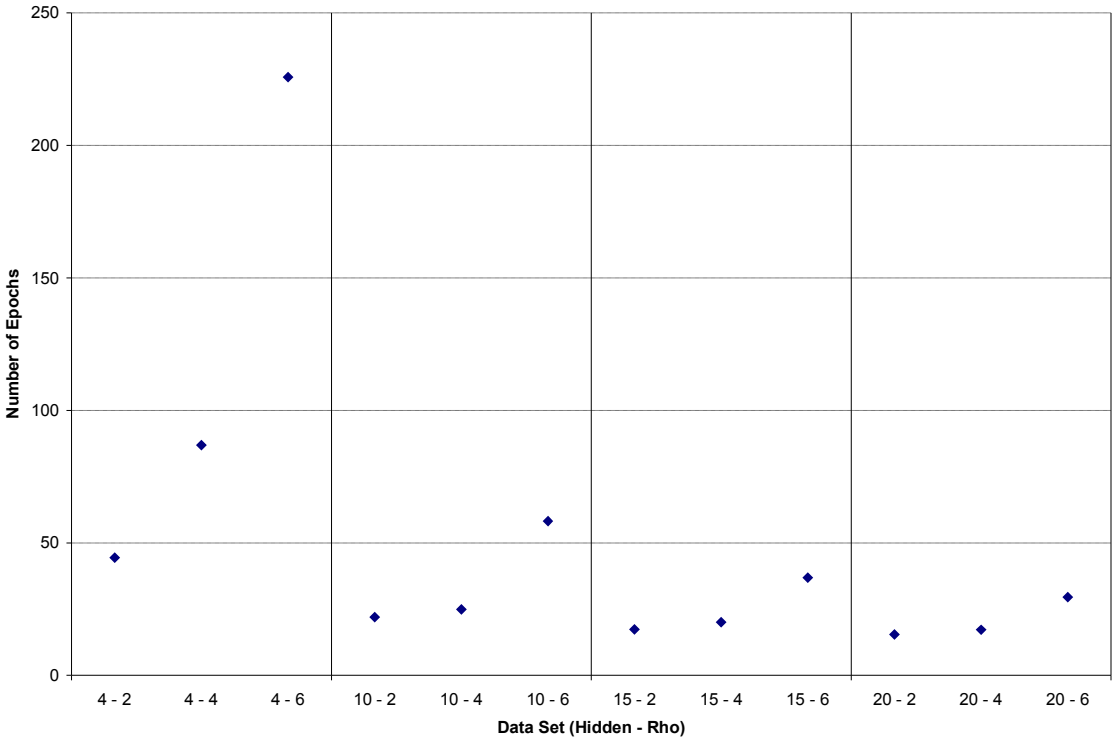

Figure D.3.2: Plot of Standard Deviations of Epochs by Data Set (All Data)

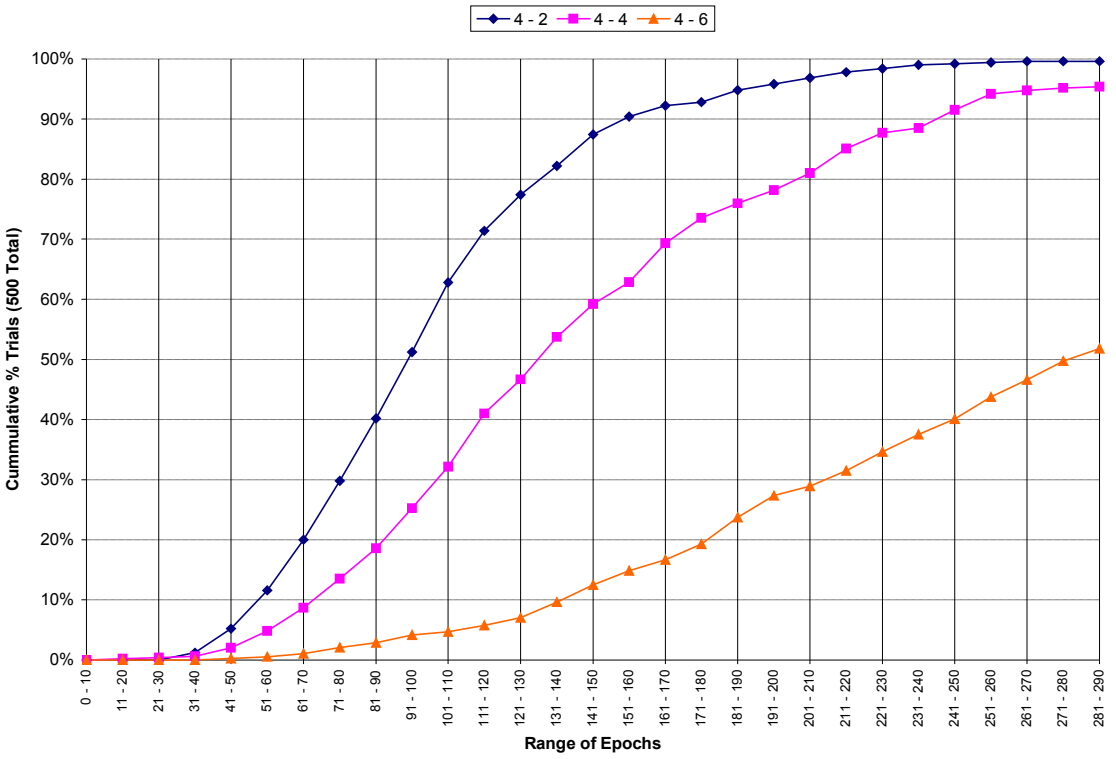

Figure D.3.3: Empirical CDF of Epochs for Hidden Layer Size 4 Networks by ρ Value (All Data)

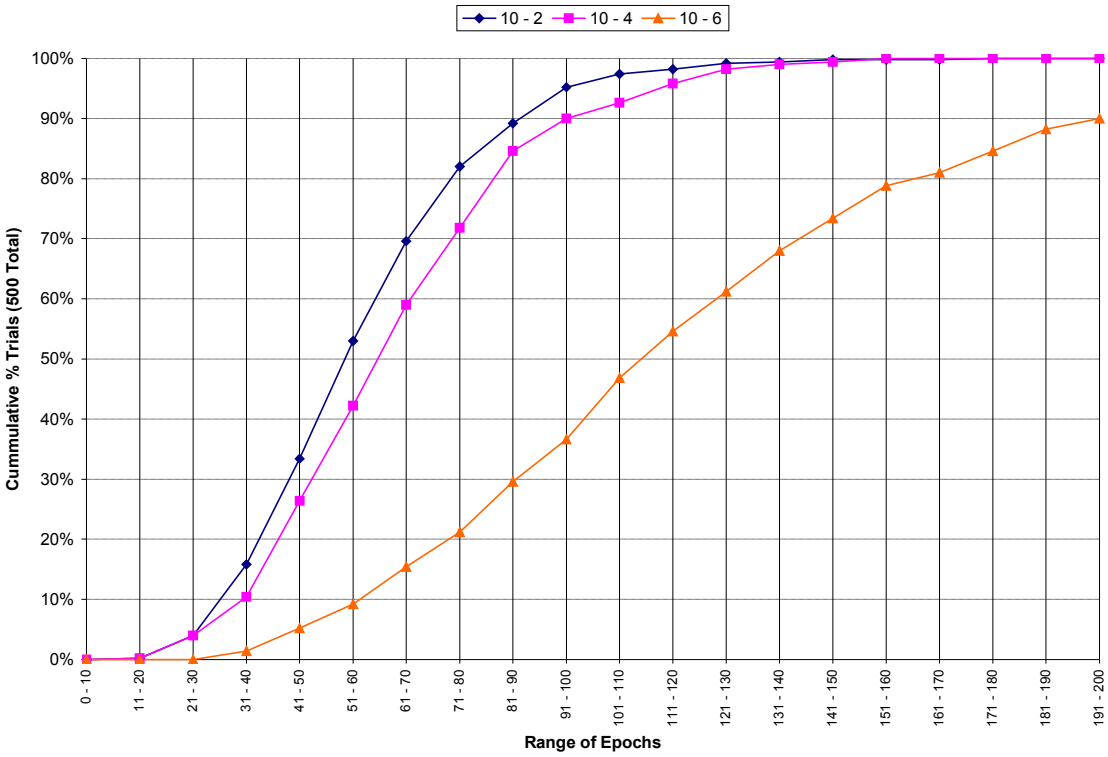

Figure D.3.4: Empirical CDF of Epochs for Hidden Layer Size 10 Networks by ρ Value (All Data)

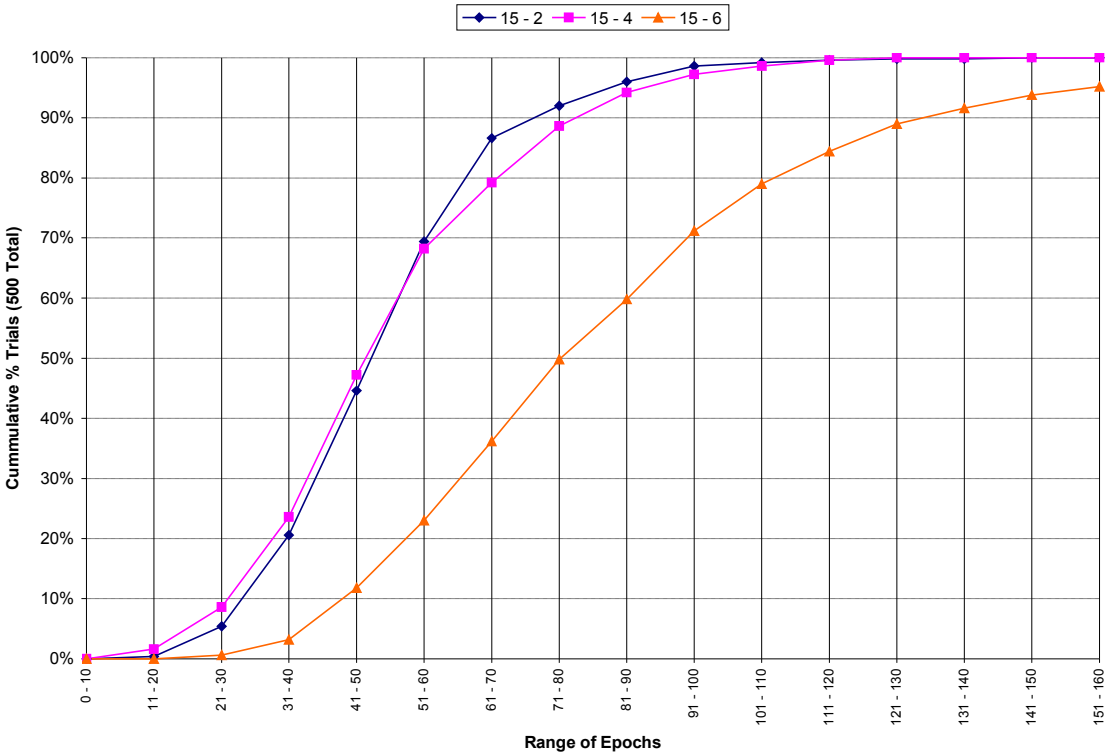

Figure D.3.5: Empirical CDF of Epochs for Hidden Layer Size 15 Networks by ρ Value (All Data)

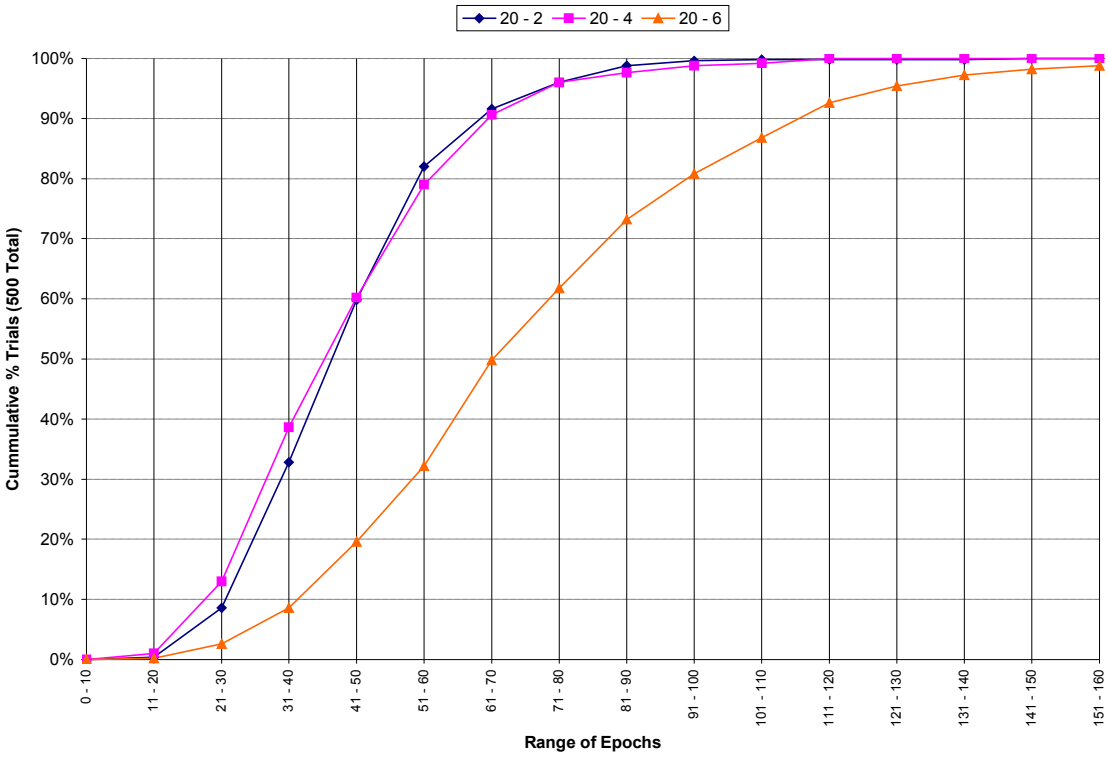

Figure D.3.6: Empirical CDF of Epochs for Hidden Layer Size 20 Networks by ρ Value (All Data)

D4: Back-Prop Bit Counting

Data Set	Total Trials	Convergent Trials	Min Epochs	Mean Epochs	Median Epochs	Max Epochs	StdDev Epochs
4 - 2	500	328	5,574	11,725	11,370	15,000	2,842
4 - 4	500	361	9,387	12,536	11,943	15,000	1,698
4 - 6	500	0	15,000	15,000	15,000	15,000	0
6 - 2	500	471	1,411	5,346	4,630	15,000	3,062
6 - 4	500	492	2,717	6,381	6,058	15,000	2,160
6 - 6	500	142	7,055	14,257	15,000	15,000	1,542
8 - 2	500	500	1,071	2,745	2,768	11,534	958
8 - 4	500	500	1,970	3,939	3,824	7,813	1,025
8 - 6	500	463	4,995	10,575	10,382	15,000	2,570
10 - 2	500	500	967	2,107	1,959	4,682	696
10 - 4	500	500	1,558	3,150	3,024	6,193	793
10 - 6	500	499	4,327	8,220	8,013	15,000	1,743

Table D.4.1: Summary Statistics for Number of Epochs by Data Set (All Data)

Data Set	Total Trials	Convergent Trials	Min % Correct	Mean % Correct	Median % Correct	Max % Correct	StdDev % Correct
4 - 2	500	328	12.5	98.6	100.0	100.0	6.7
4 - 4	500	361	37.5	96.5	100.0	100.0	7.9
4 - 6	500	0	29.7	52.2	51.6	82.8	3.9
6 - 2	500	471	81.3	97.6	100.0	100.0	3.8
6 - 4	500	492	68.0	86.8	87.1	100.0	7.5
6 - 6	500	142	43.0	64.6	66.4	90.6	9.6
8 - 2	500	500	70.3	90.0	90.6	100.0	7.0
8 - 4	500	500	65.6	74.9	74.2	96.1	5.4
8 - 6	500	463	52.3	69.2	69.5	81.3	3.8
10 - 2	500	500	67.2	84.6	83.6	100.0	7.2
10 - 4	500	500	61.7	71.0	71.1	85.2	3.8
10 - 6	500	499	60.9	68.4	68.8	76.6	2.5

Table D.4.2: Summary Statistics for Percent Correct by Data Set (All Data)

Data Set	Convergent Trials	Min Epochs	Mean Epochs	Median Epochs	Max Epochs	StdDev Epochs
4 - 2	328	5,574	10,007	9,972	14,811	1,928
4 - 4	361	9,387	11,587	11,550	14,756	865
4 - 6	0	0	0	0	0	0
6 - 2	471	1,411	4,752	4,477	14,932	1,962
6 - 4	492	2,717	6,241	6,032	13,304	1,874
6 - 6	142	7,055	12,381	12,750	14,955	1,863
8 - 2	500	1,071	2,745	2,768	11,534	958
8 - 4	500	1,970	3,939	3,824	7,813	1,025
8 - 6	463	4,995	10,221	10,152	14,937	2,332
10 - 2	**500**	**967**	**2,107**	**1,959**	**4,682**	**696**
10 - 4	500	1,558	3,150	3,024	6,193	793
10 - 6	499	4,327	8,206	8,003	13,776	1,718

Table D.4.3: Summary Statistics for by Data Set (Convergent Trials Only)

Data Set	Convergent Trials	Min % Correct	Mean % Correct	Median % Correct	Max % Correct	StdDev % Correct
4 - 2	328	95.3	99.7	100.0	100.0	0.64
4 - 4	361	91.4	99.9	100.0	100.0	0.65
4 - 6	0	0	0	0	0	0
6 - 2	471	81.3	97.7	100.0	100.0	3.8
6 - 4	492	68.0	86.9	87.5	100.0	7.5
6 - 6	142	64.8	73.0	71.9	90.6	4.8
8 - 2	500	70.3	89.9	90.6	100.0	7.0
8 - 4	500	65.6	74.9	74.2	96.1	5.4
8 - 6	463	61.7	69.7	69.5	81.3	3.1
10 - 2	**500**	**67.2**	**84.6**	**83.6**	**100.0**	**7.2**
10 - 4	500	61.7	71.0	71.1	85.2	3.8
10 - 6	499	60.9	68.4	68.8	76.6	2.5

Table D.4.3: Summary Statistics for Percent Correct by Data Set (Convergent Trials Only)

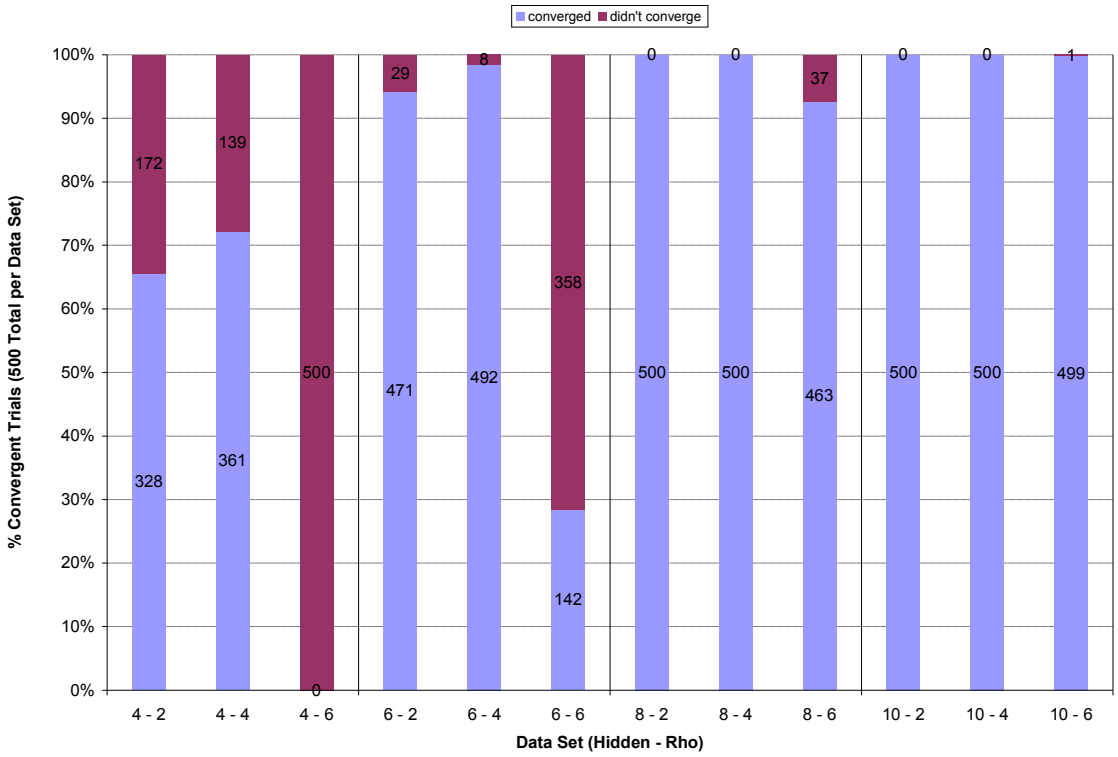

Figure D.4.1: Percent of Convergent Trials by Data Set

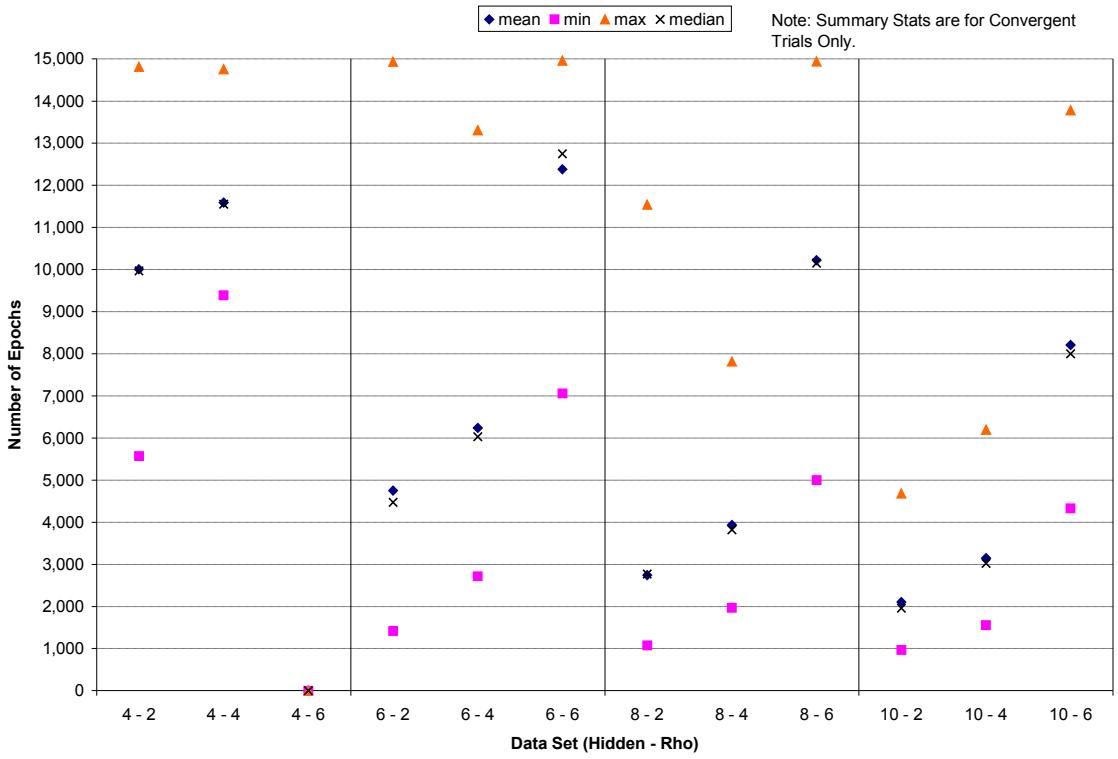

Figure D.4.2: Plot of Summary Statistics for Epochs by Data Set (Convergent Trials Only)

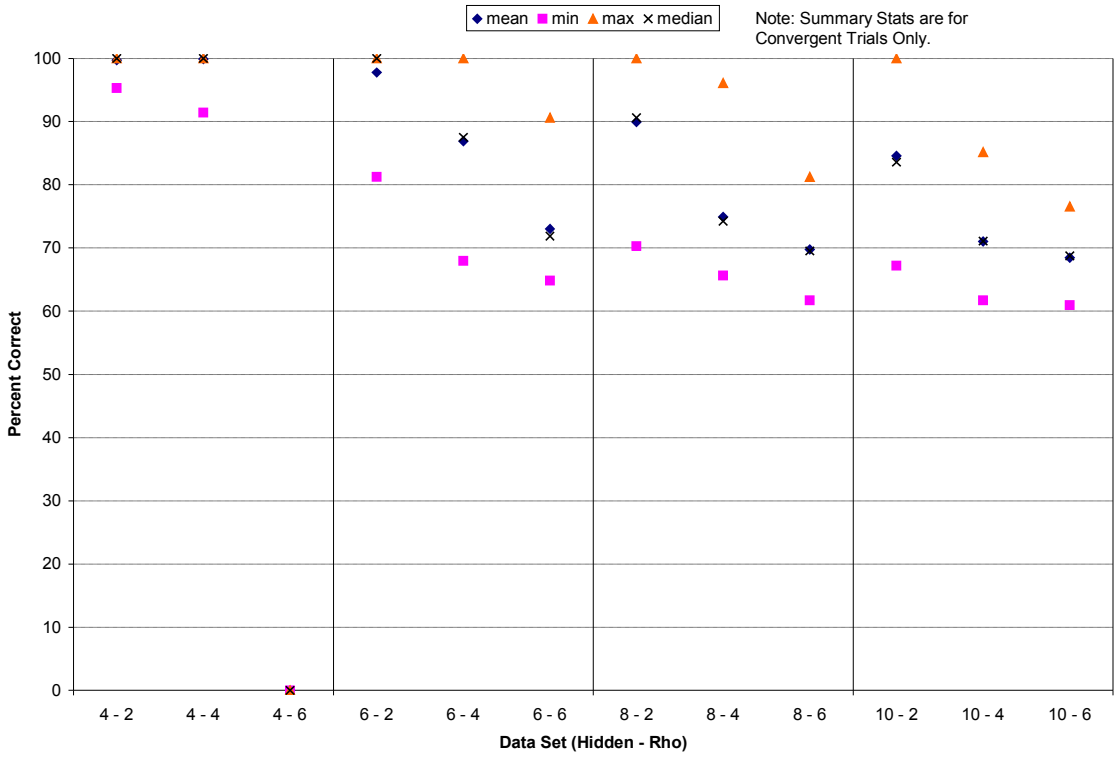

Figure D.4.3: Plot of Summary Statistics for Percent Correct by Data Set (Convergent Trials Only)

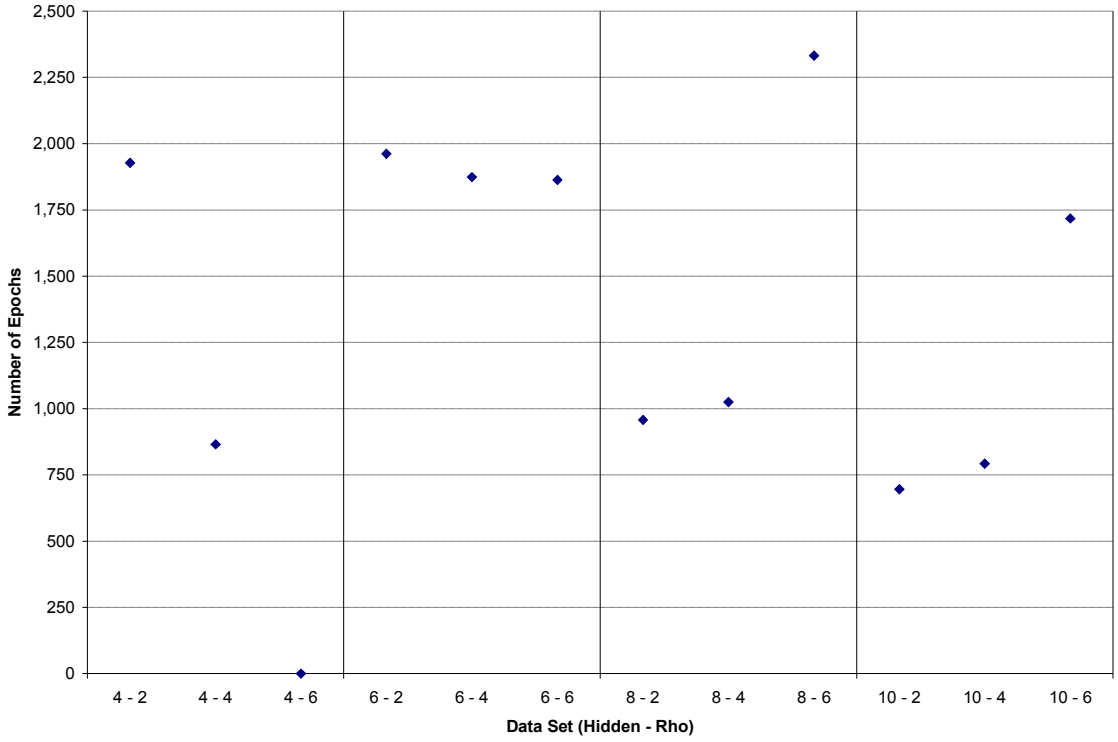

Figure D.4.4: Plot of Standard Deviations of Epochs by Data Set (Convergent Trials Only)

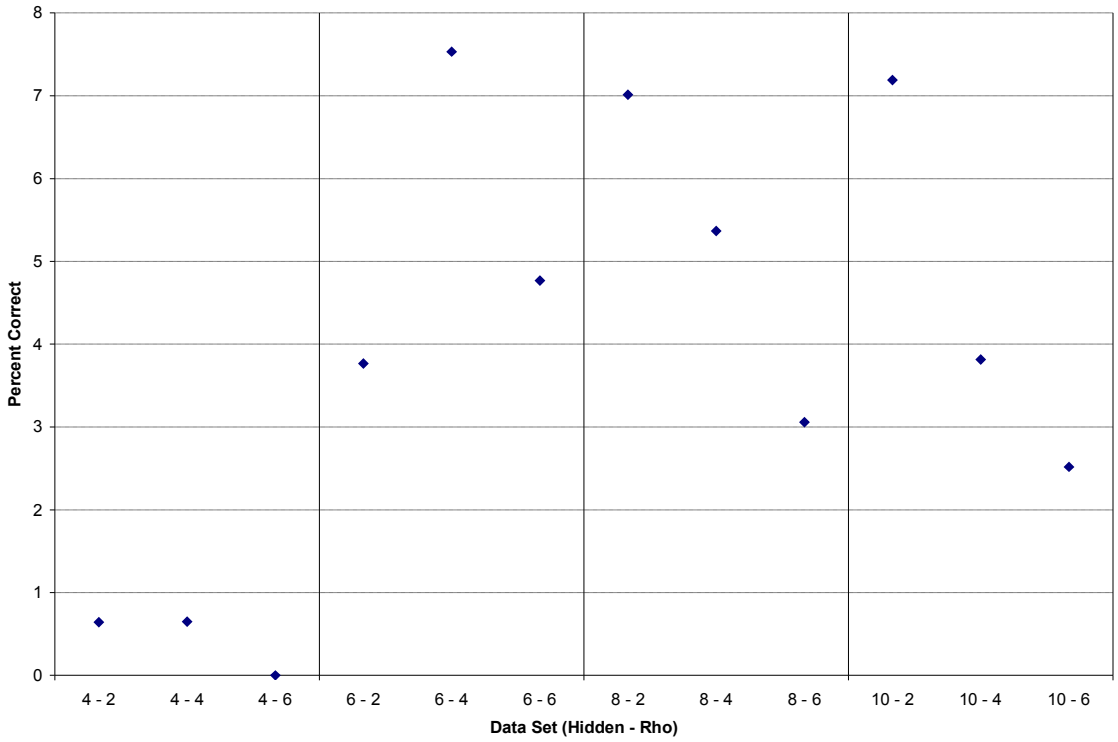

Figure D.4.5: Plot of Standard Deviations of Percent Correct by Data Set (Convergent Trials Only)

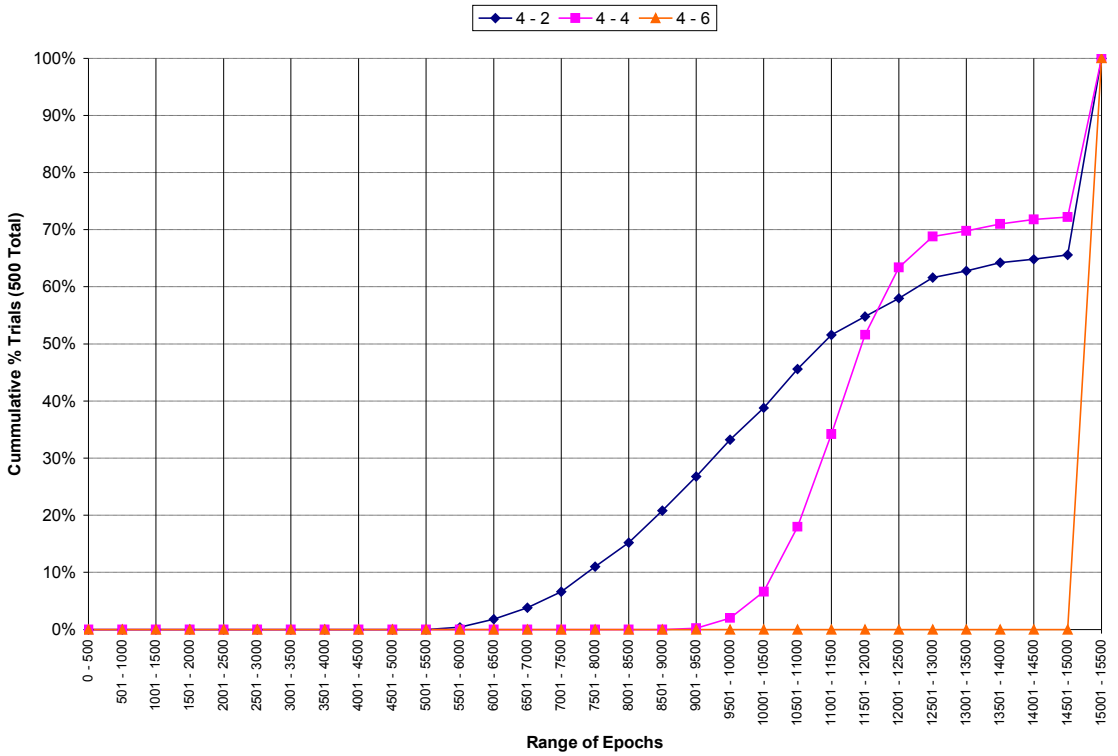

Figure D.4.6: Empirical CDF of Epochs for Hidden Layer Size 4 Networks by ρ Value (All Data)

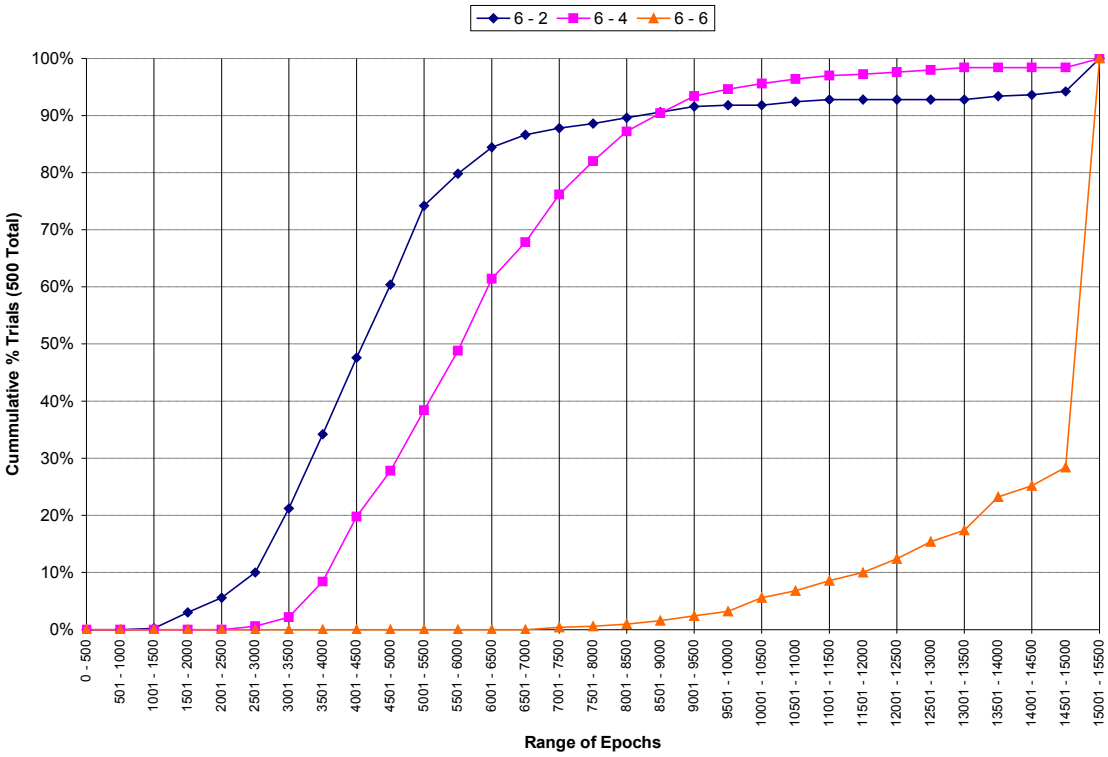

Figure D.4.7: Empirical CDF of Epochs for Hidden Layer Size 6 Networks by ρ Value (All Data)

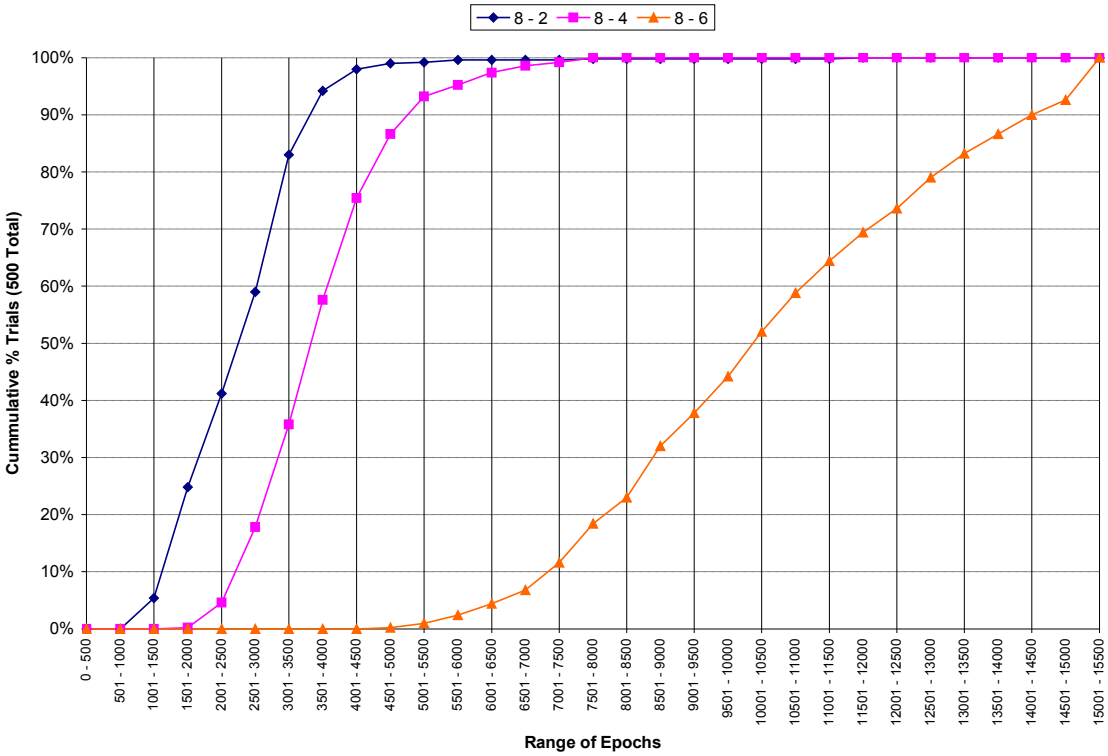

Figure D.4.8: Empirical CDF of Epochs for Hidden Layer Size 8 Networks by ρ Value (All Data)

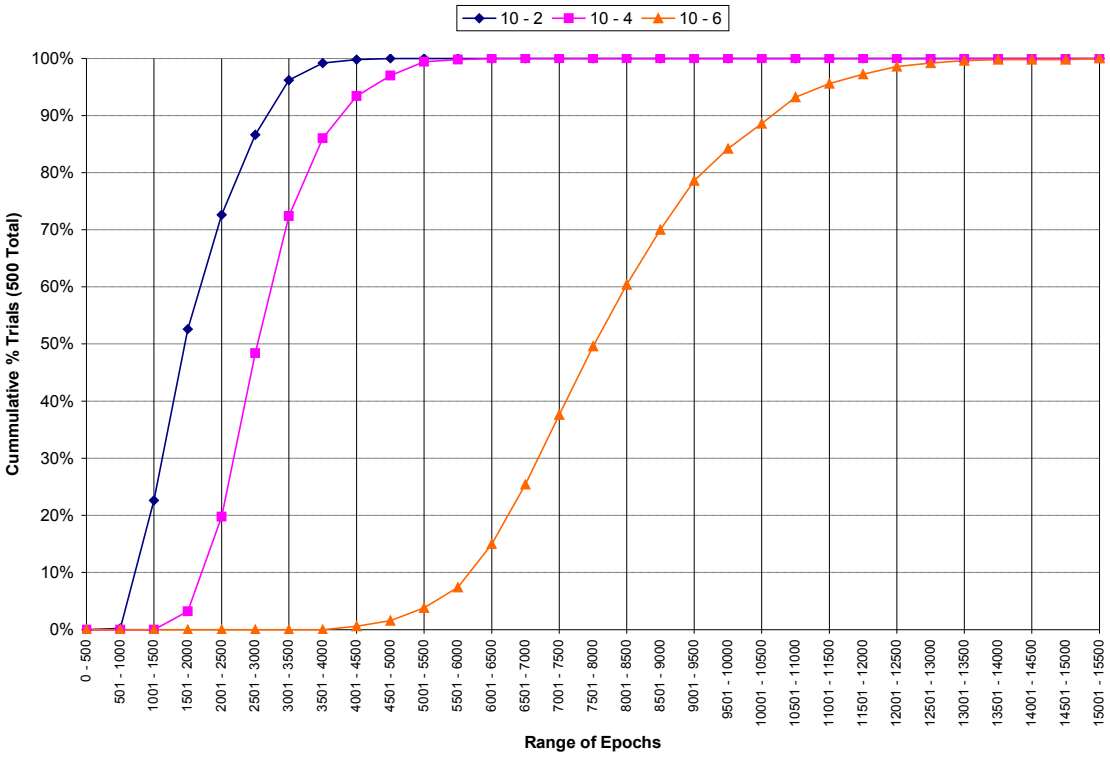

Figure D.4.9: Empirical CDF of Epochs for Hidden Layer Size 10 Networks by ρ Value (All Data)

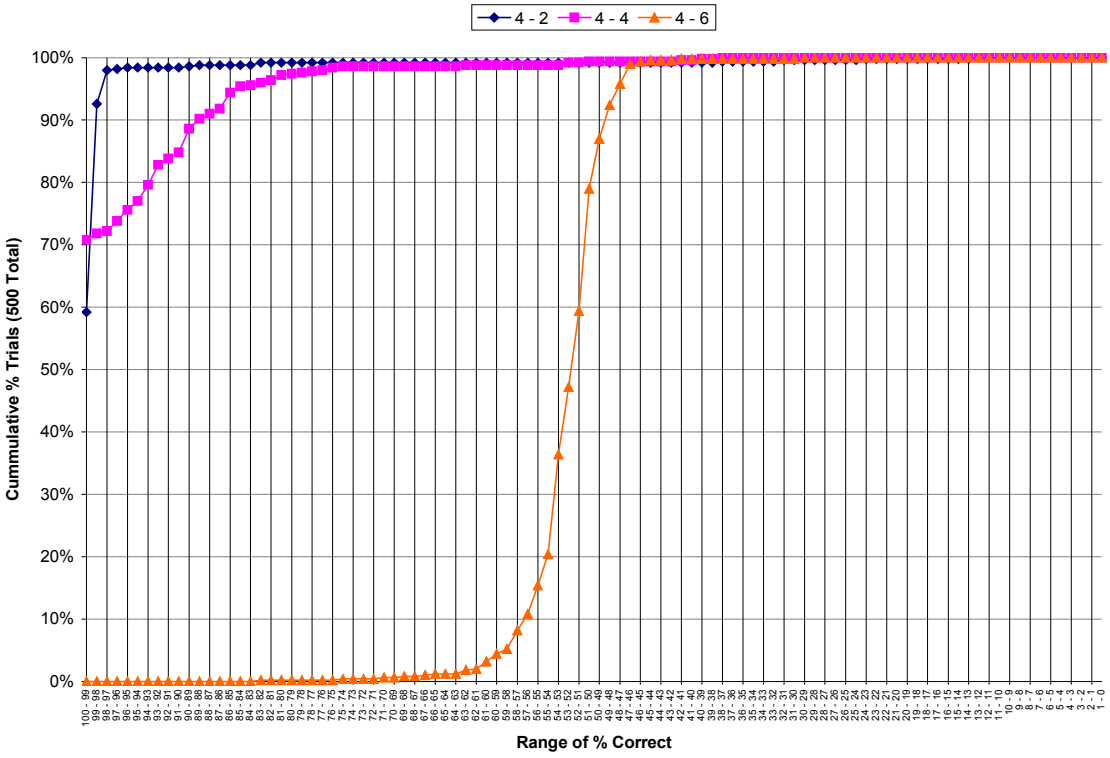

Figure D.4.10: Empirical CDF of Percent Correct for Hidden Layer Size 4 Networks by ρ Value (All Data)

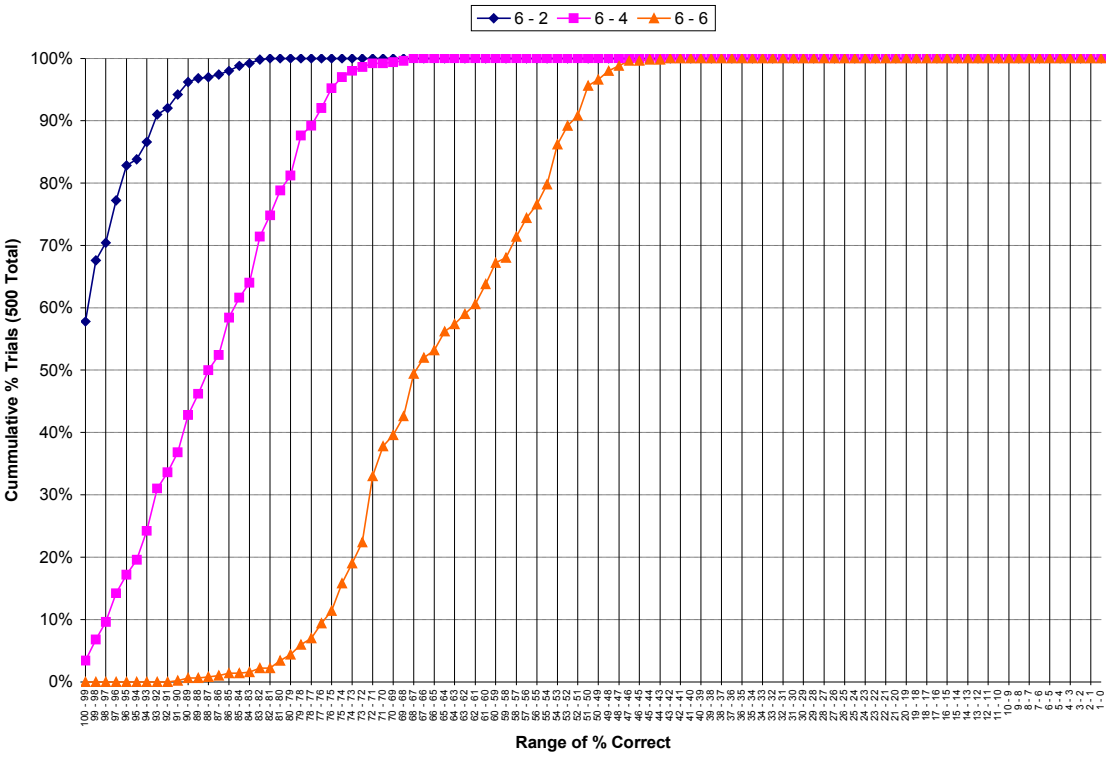

Figure D.4.11: Empirical CDF of Percent Correct for Hidden Layer Size 6 Networks by ρ Value (All Data)

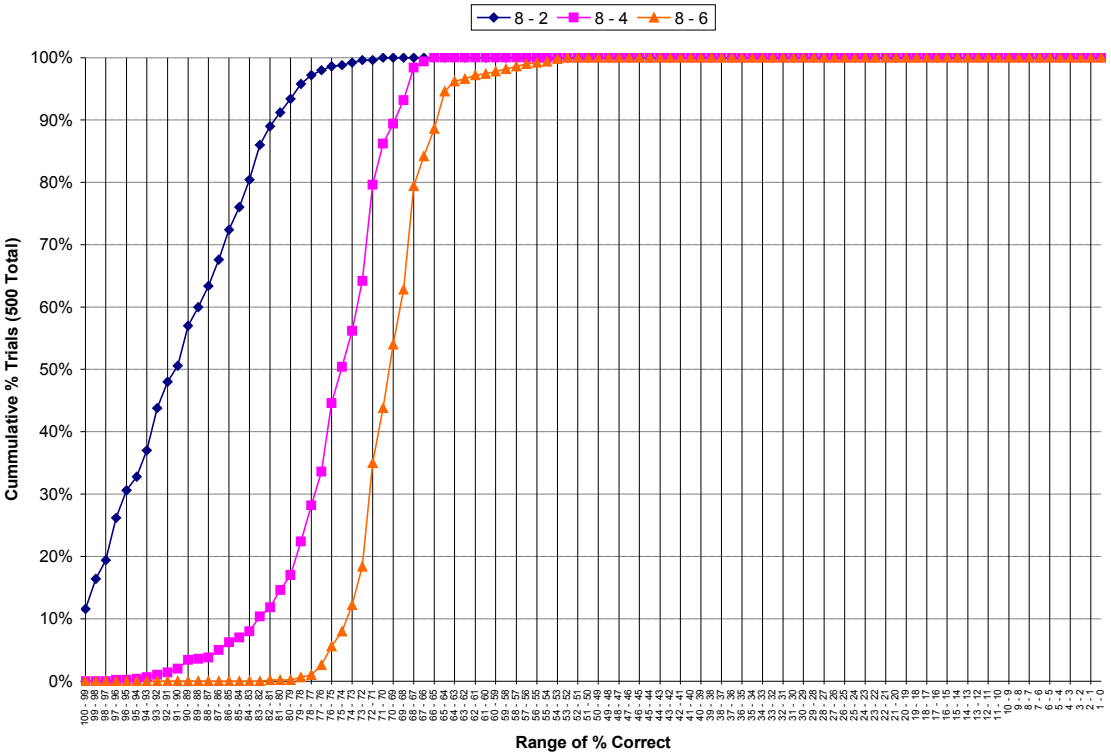

Figure D.4.12: Empirical CDF of Percent Correct for Hidden Layer Size 8 Networks by ρ Value (All Data)

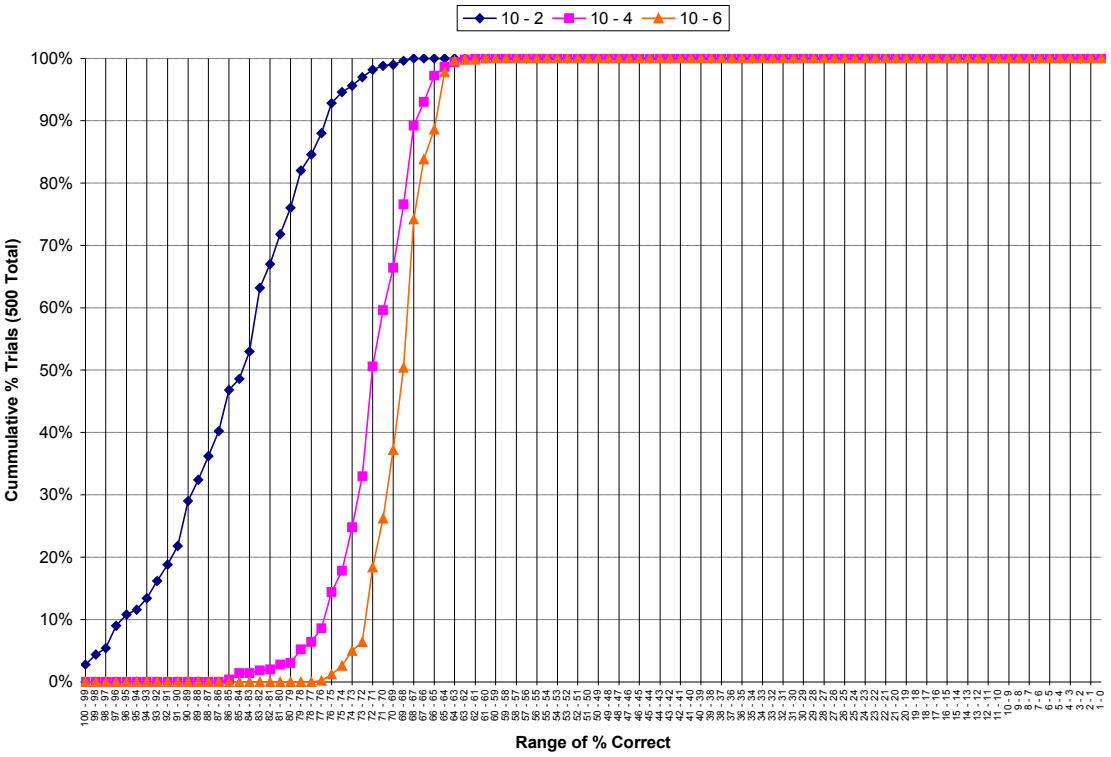

Figure D.4.13: Empirical CDF of Percent Correct for Hidden Layer Size 10 Networks by ρ Value (All Data)

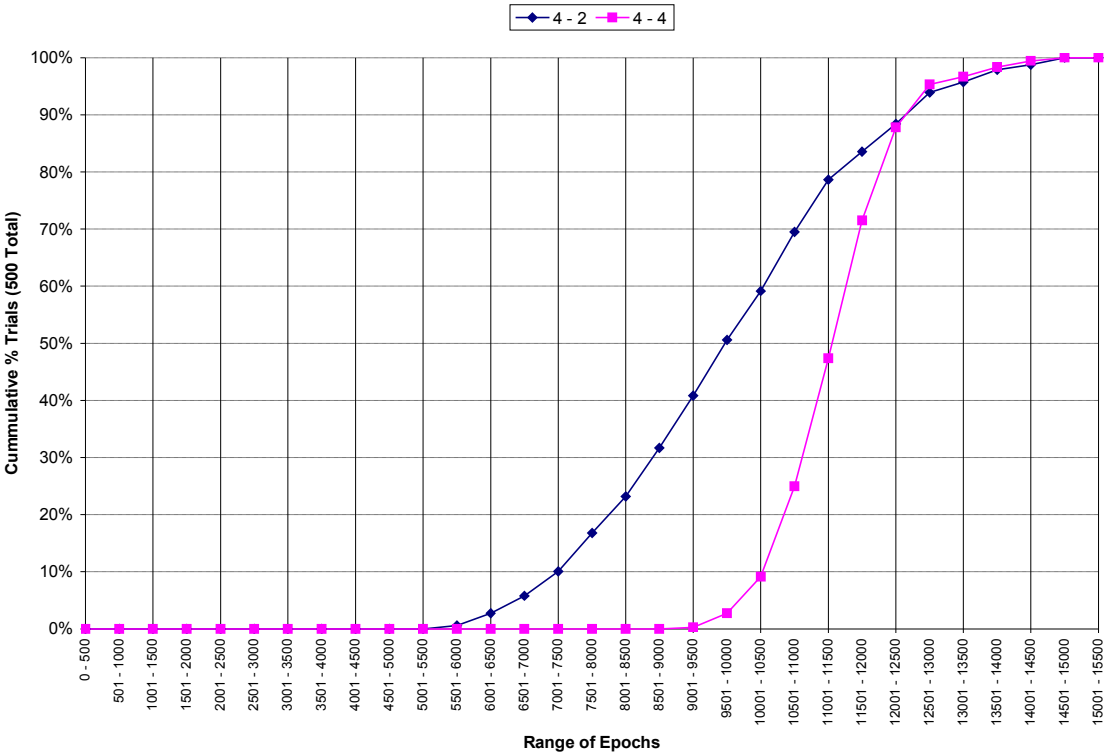

Figure D.4.14: Empirical CDF of Epochs for Hidden Layer Size 4 Networks by ρ Value (Convergent Trials Only)

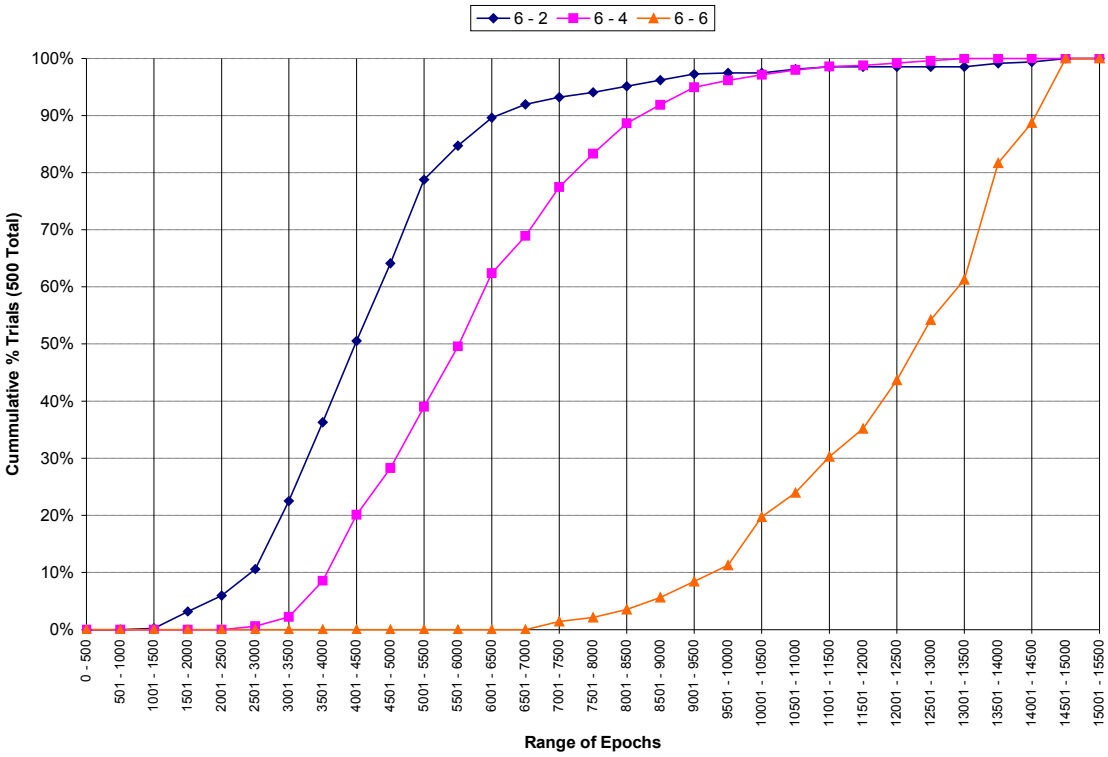

Figure D.4.15: Empirical CDF of Epochs for Hidden Layer Size 6 Networks by ρ Value (Convergent Trials Only)

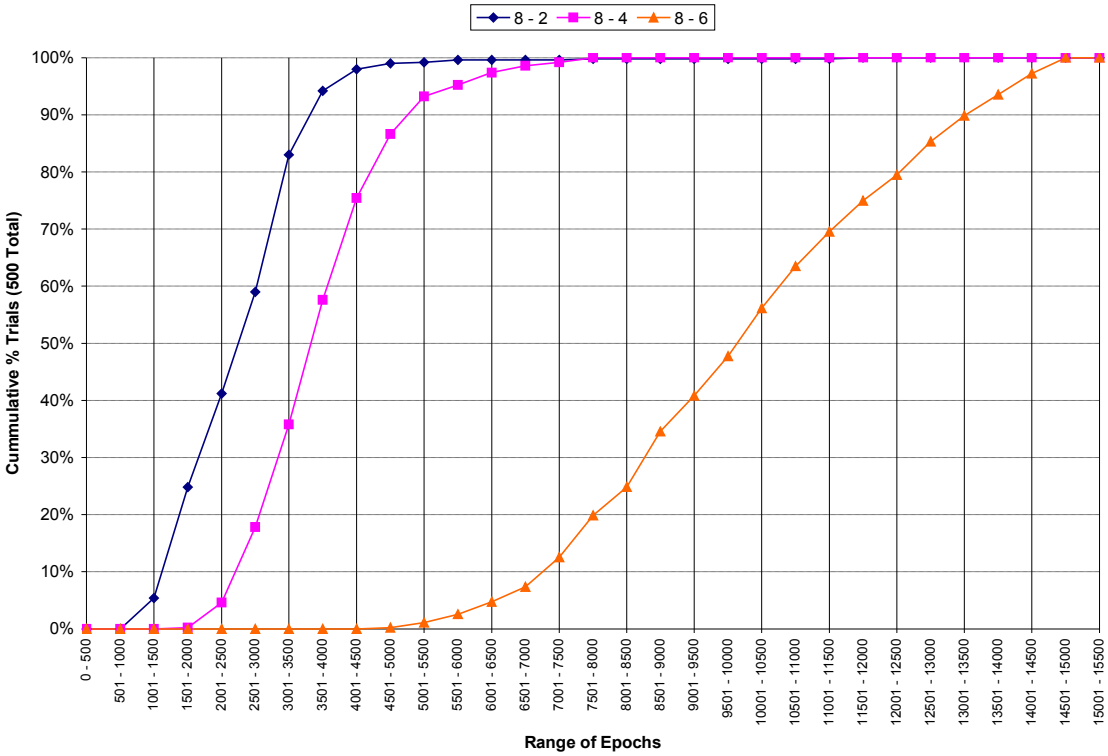

Figure D.4.16: Empirical CDF of Epochs for Hidden Layer Size 8 Networks by ρ Value (Convergent Trials Only)

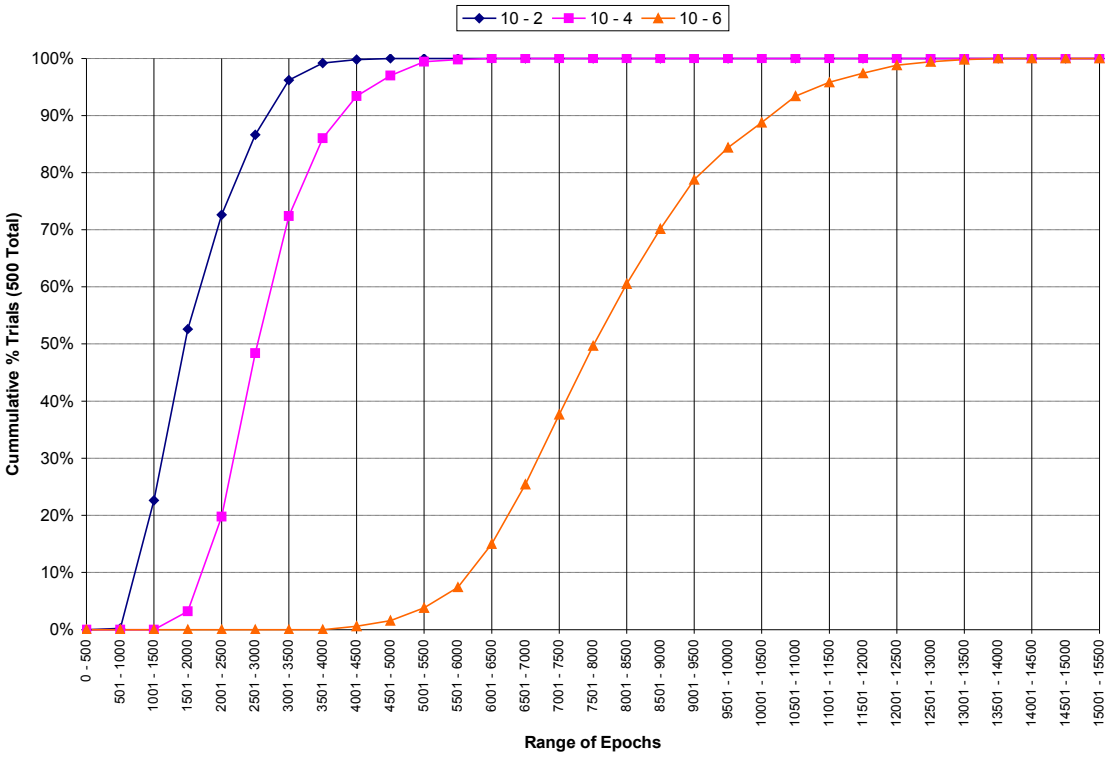

Figure D.4.17: Empirical CDF of Epochs for Hidden Layer Size 10 Networks by ρ Value (Convergent Trials Only)

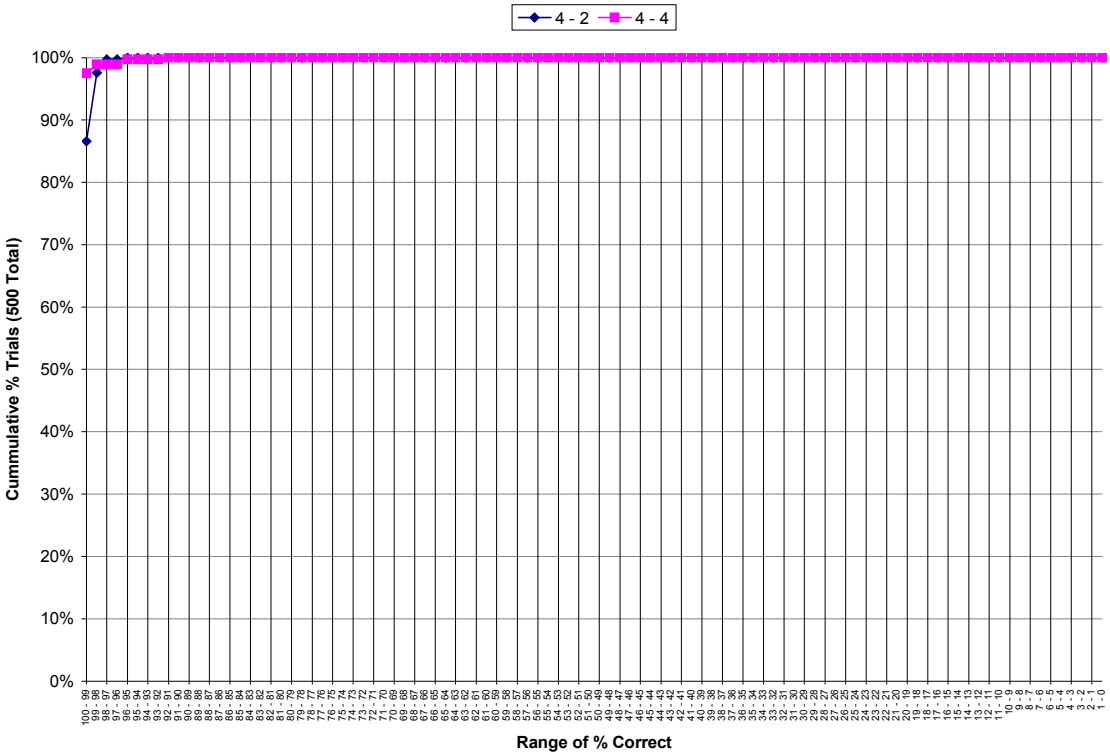

Figure D.4.18: Empirical CDF of Percent Correct for Hidden Layer Size 4 Networks by ρ Value (Convergent Trials Only)

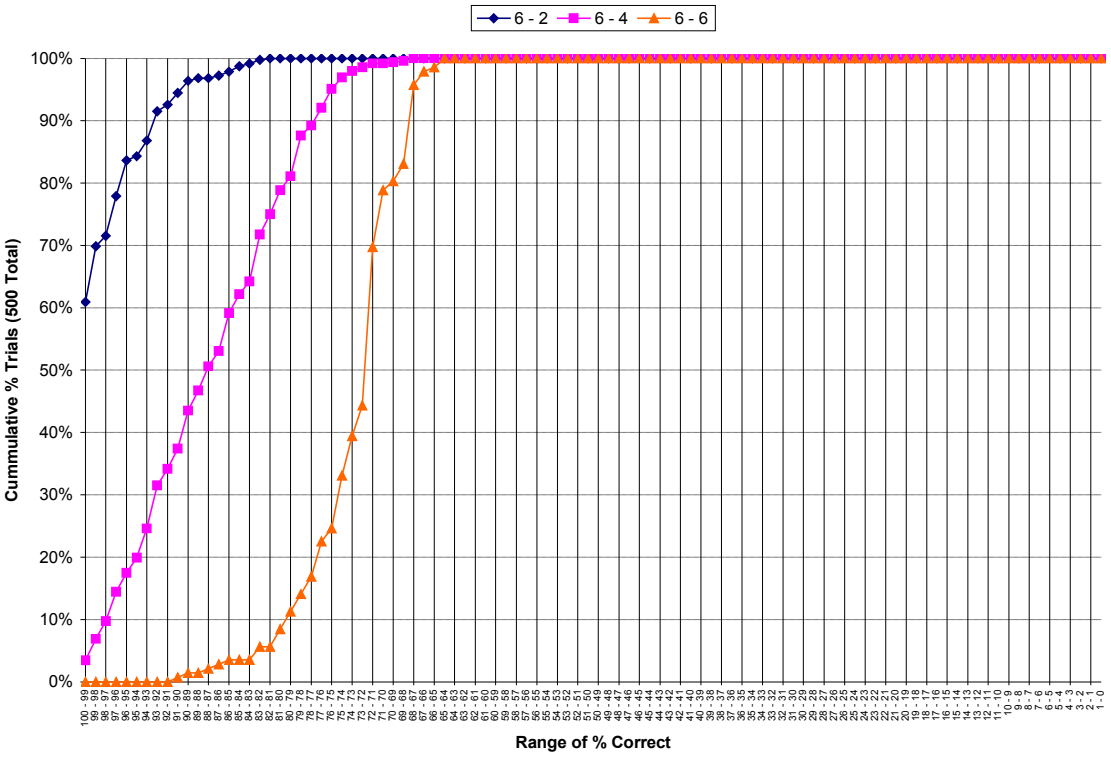

Figure D.4.19: Empirical CDF of Percent Correct for Hidden Layer Size 6 Networks by ρ Value (Convergent Trials Only)

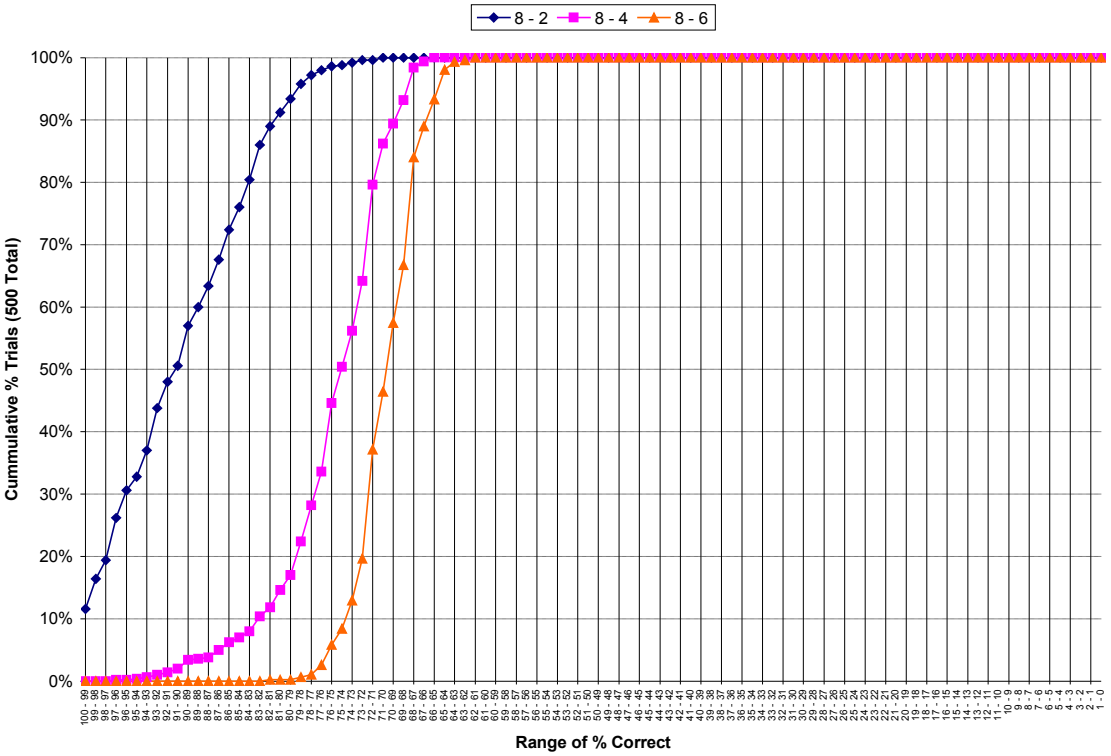

Figure D.4.20: Empirical CDF of Percent Correct for Hidden Layer Size 8 Networks by ρ Value (Convergent Trials Only)

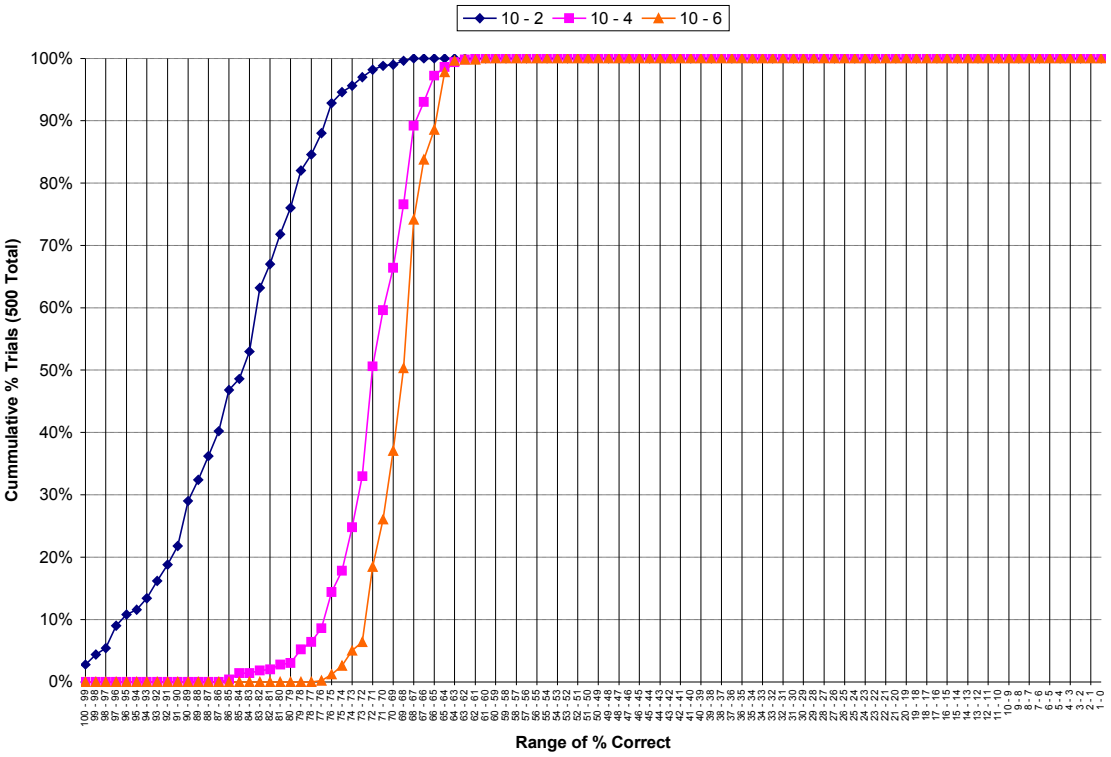

Figure D.4.21: Empirical CDF of Percent Correct for Hidden Layer Size 10 Networks by ρ Value (Convergent Trials Only)

D5: C-G Bit Counting

Data Set	Total Trials	Convergent Trials	Min Epochs	Mean Epochs	Median Epochs	Max Epochs	StdDev Epochs
4 - 2	500	347	215	6,837	3,227	15,000	6,241
4 - 4	500	370	235	4,776	905	15,000	6,194
4 - 6	500	373	251	5,277	1,927	15,000	5,931
6 - 2	500	476	105	3,323	1,775	15,000	3,903
6 - 4	500	498	105	786	327	15,000	1,643
6 - 6	500	500	199	1,353	646	13,128	1,569
8 - 2	500	499	74	1,166	684	15,000	1,520
8 - 4	500	500	115	445	258	2,958	435
8 - 6	500	499	153	934	636	15,000	1,013
10 - 2	500	500	70	584	384	4,033	560
10 - 4	500	500	101	435	270	3,604	401
10 - 6	500	500	99	761	590	4,337	655

Table D.5.1: Summary Statistics for by Data Set (All Data)

Data Set	Total Trials	Convergent Trials	Min % Correct	Mean % Correct	Median % Correct	Max % Correct	StdDev % Correct
4 - 2	500	347	0	93.8	100.0	100.0	20.2
4 - 4	500	370	89.1	98.5	98.4	100.0	1.8
4 - 6	500	373	60.9	97.6	98.4	100.0	3.3
6 - 2	500	476	89.1	97.9	98.4	100.0	2.7
6 - 4	500	498	75.0	93.2	93.8	100.0	4.6
6 - 6	500	500	73.4	89.8	90.6	100.0	6.3
8 - 2	500	499	75.0	92.5	93.8	100.0	5.5
8 - 4	500	500	73.4	86.6	85.9	100.0	6.2
8 - 6	500	499	0.8	81.7	79.7	98.4	7.4
10 - 2	500	500	71.9	87.4	87.5	100.0	5.8
10 - 4	500	500	70.3	81.5	79.7	98.4	5.6
10 - 6	500	500	68.8	78.7	78.1	95.3	4.9

Table D.5.2: Summary Statistics for Percent Correct by Data Set (All Data)

Data Set	Convergent Trials	Min Epochs	Mean Epochs	Median Epochs	Max Epochs	StdDev Epochs
4 - 2	347	215	3,238	1,652	14,916	3,703
4 - 4	370	235	1,184	594	10,752	1,456
4 - 6	373	251	1,967	1,333	13,596	1,981
6 - 2	476	105	2,734	1,554	14,655	2,961
6 - 4	498	105	729	327	14,928	1,377
6 - 6	500	199	1,353	646	13,128	1,569
8 - 2	499	74	1,138	682	11,790	1,390
8 - 4	500	115	445	258	2,958	435
8 - 6	499	153	906	629	5,099	794
10 - 2	500	70	584	384	4,033	560
10 - 4	500	101	435	270	3,604	401
10 - 6	500	99	761	590	4,337	655

Table D.5.3: Summary Statistics for Epochs by Data Set (Convergent Trials Only)

Data Set	Convergent Trials	Min % Correct	Mean % Correct	Median % Correct	Max % Correct	StdDev % Correct
4 - 2	347	98.4	100.0	100.0	100.0	0.3
4 - 4	370	90.6	98.7	100.0	100.0	1.9
4 - 6	373	90.6	98.3	100.0	100.0	2.1
6 - 2	476	89.1	98.1	98.4	100.0	2.6
6 - 4	498	75.0	93.2	93.8	100.0	4.6
6 - 6	500	73.4	89.8	90.6	100.0	6.3
8 - 2	499	75.0	92.5	93.8	100.0	5.4
8 - 4	500	73.4	86.6	85.9	100.0	6.2
8 - 6	499	70.3	81.8	79.7	98.4	6.4
10 - 2	500	71.9	87.4	87.5	100.0	5.8
10 - 4	500	70.3	81.5	79.7	98.4	5.6
10 - 6	500	68.8	78.7	78.1	95.3	4.9

Table D.5.3: Summary Statistics for Percent Correct by Data Set (Convergent Trials Only)

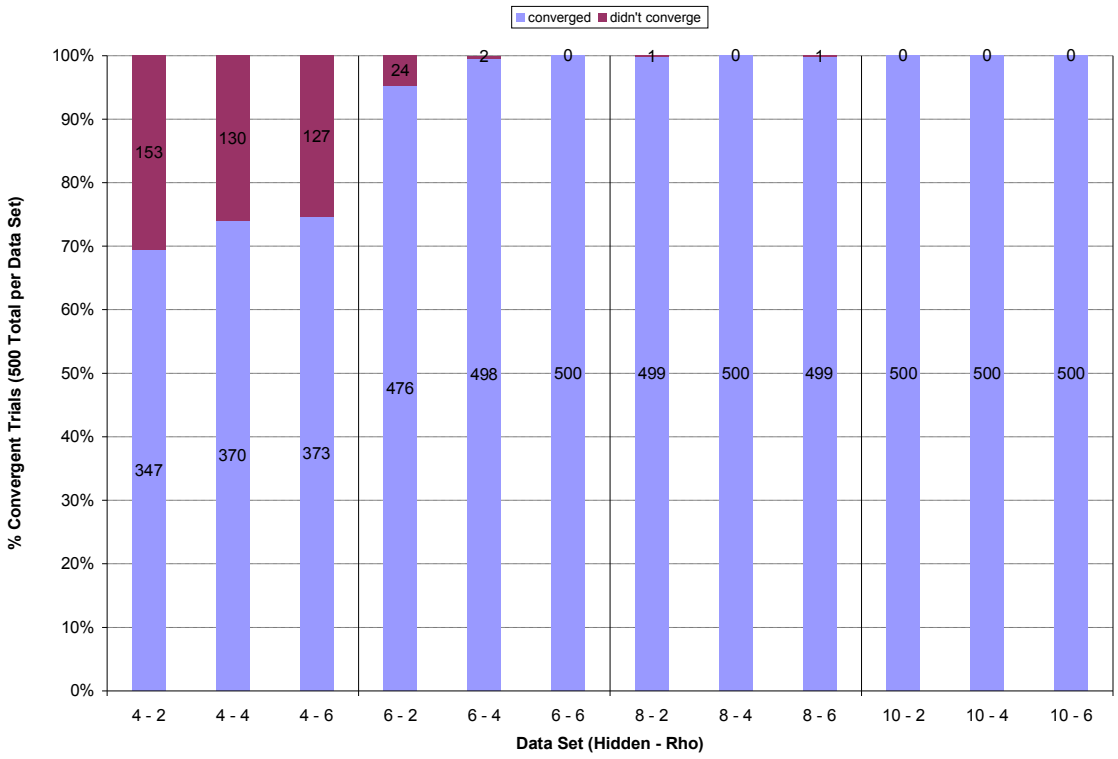

Figure D.5.1: Percent of Convergent Trials by Data Set

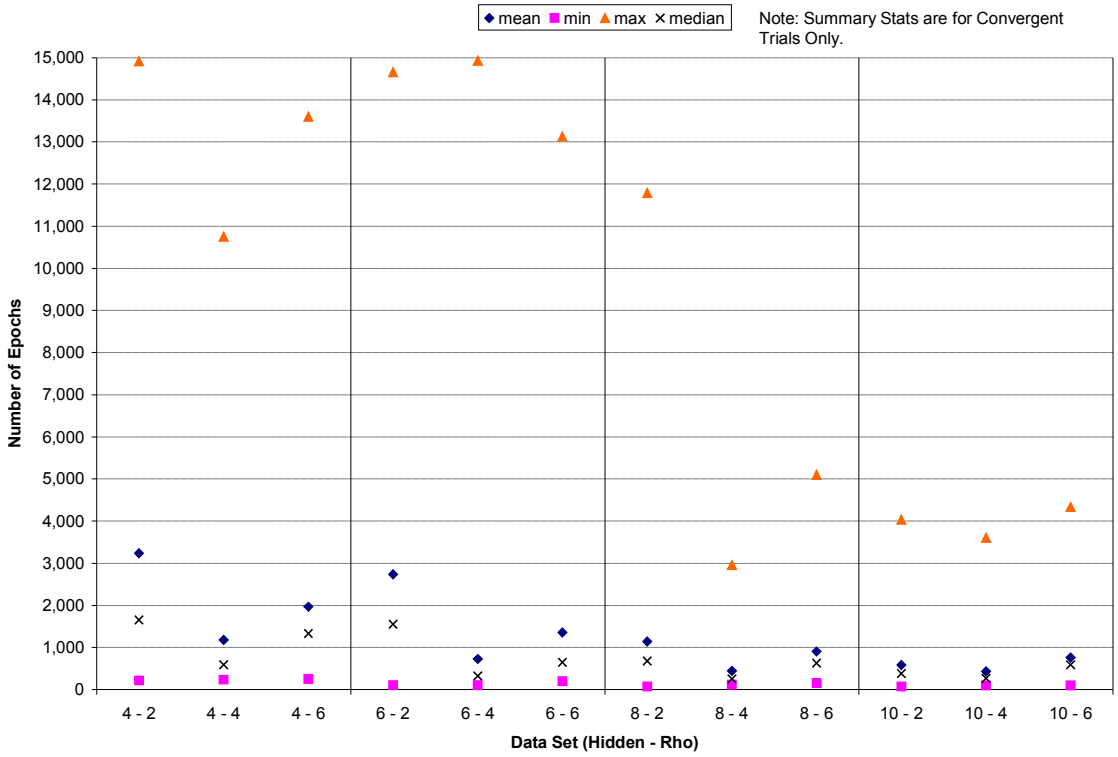

Figure D.5.2: Plot of Summary Statistics for Epochs by Data Set (Convergent Trials Only)

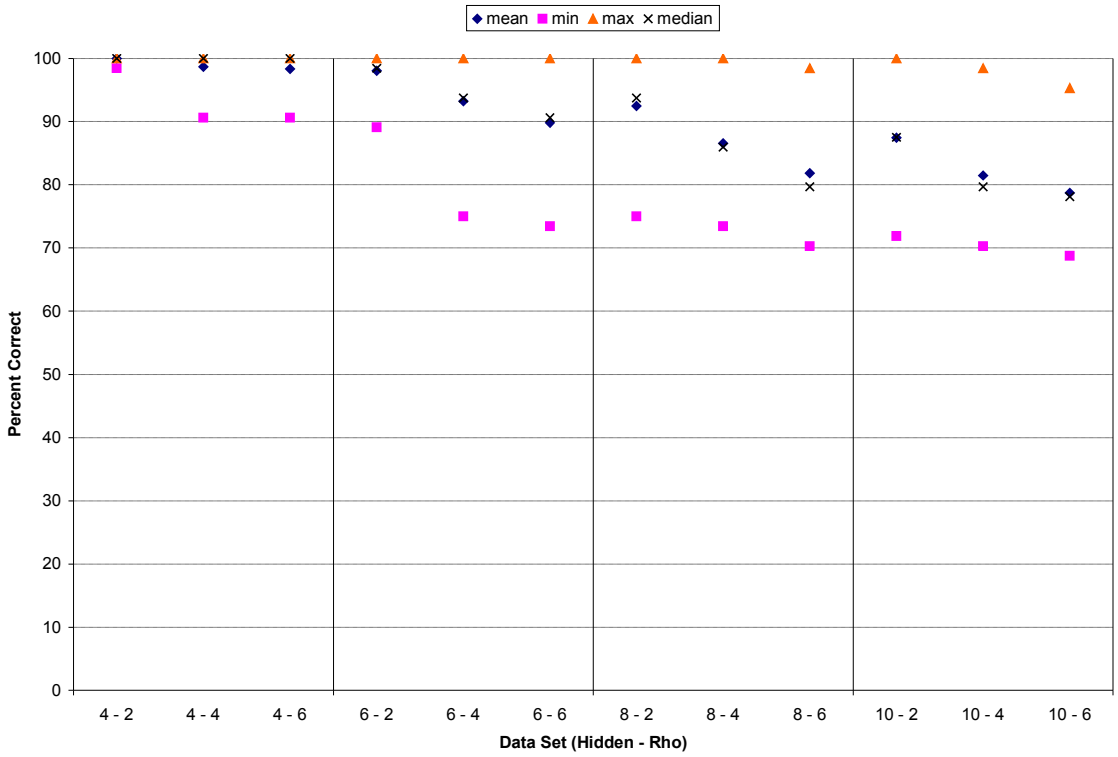

Figure D.5.3: Plot of Summary Statistics for Percent Correct by Data Set (Convergent Trials Only)

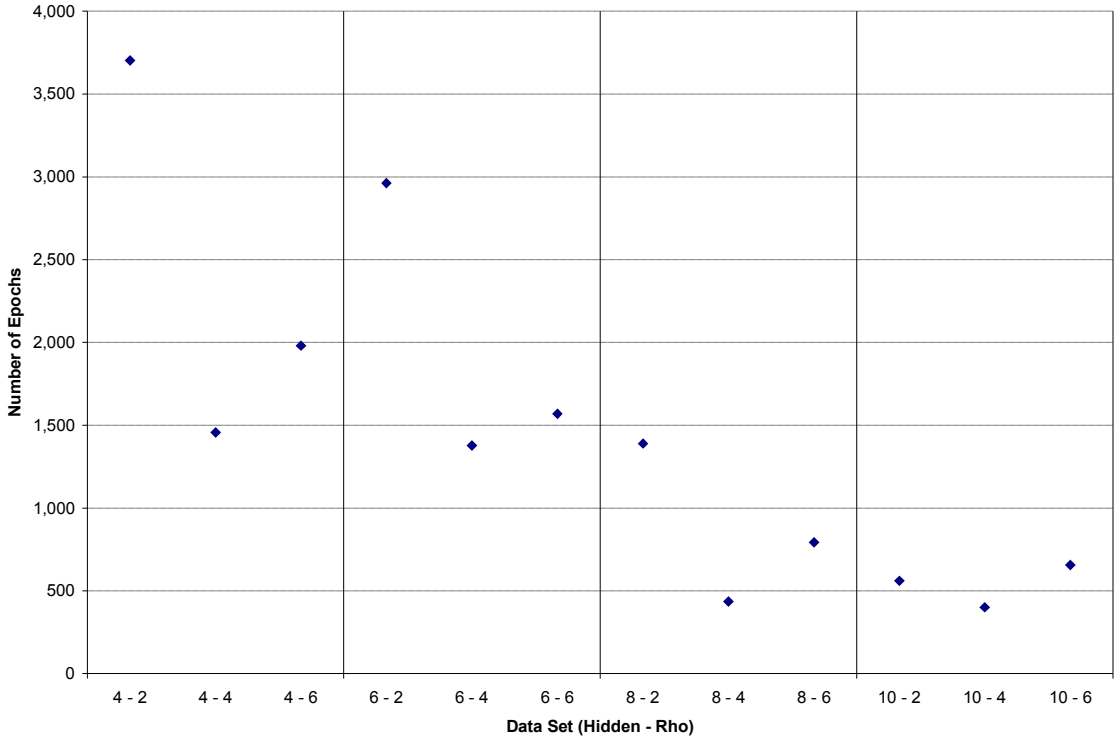

Figure D.5.4: Plot of Standard Deviations of Epochs by Data Set (Convergent Trials Only)

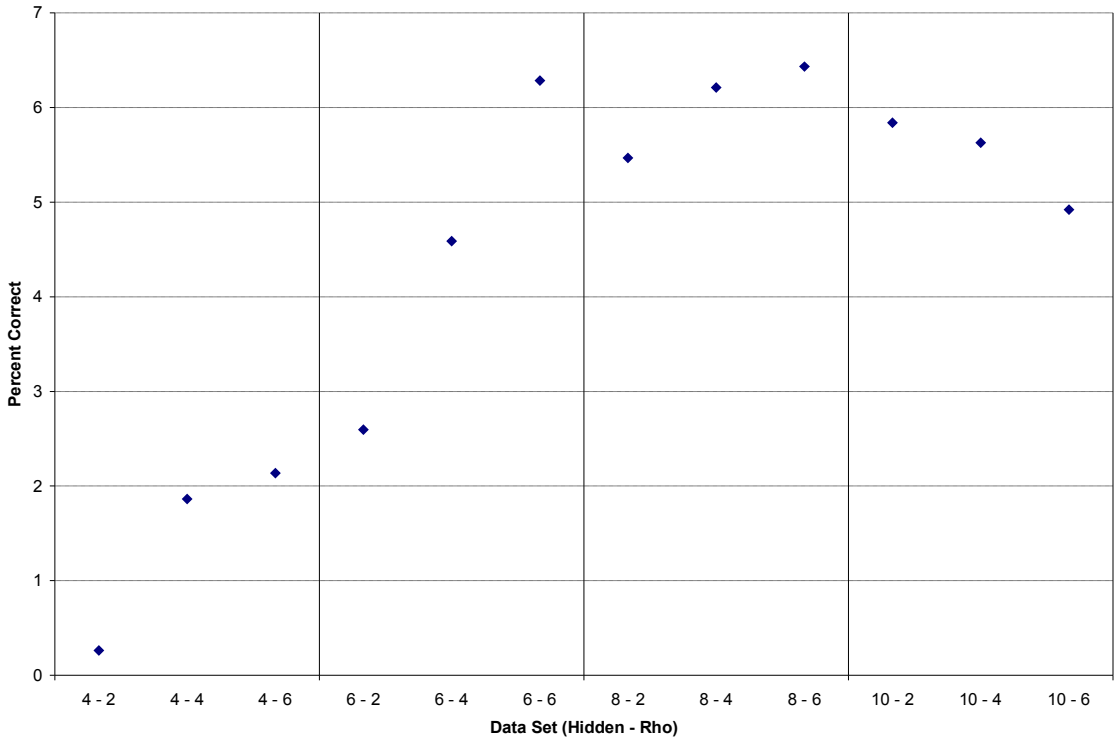

Figure D.5.5: Plot of Standard Deviations of Percent Correct by Data Set (Convergent Trials Only)

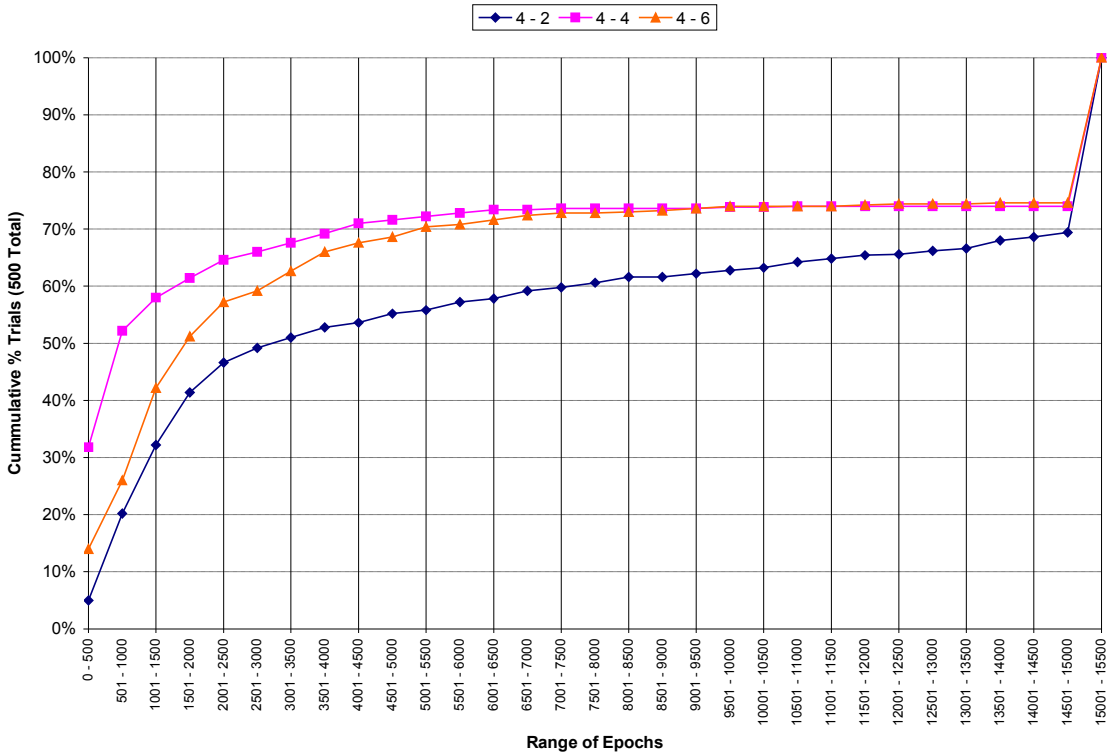

Figure D.5.6: Empirical CDF of Epochs for Hidden Layer Size 4 Networks by ρ Value (All Data)

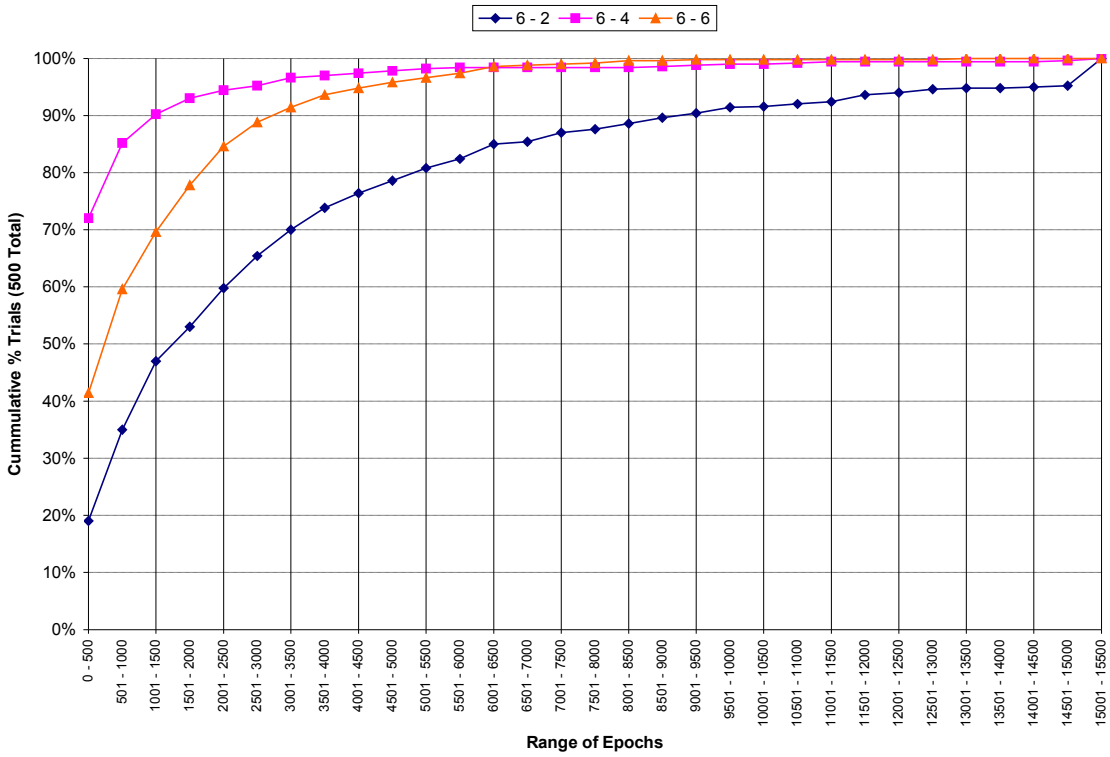

Figure D.5.7: Empirical CDF of Epochs for Hidden Layer Size 6 Networks by ρ Value (All Data)

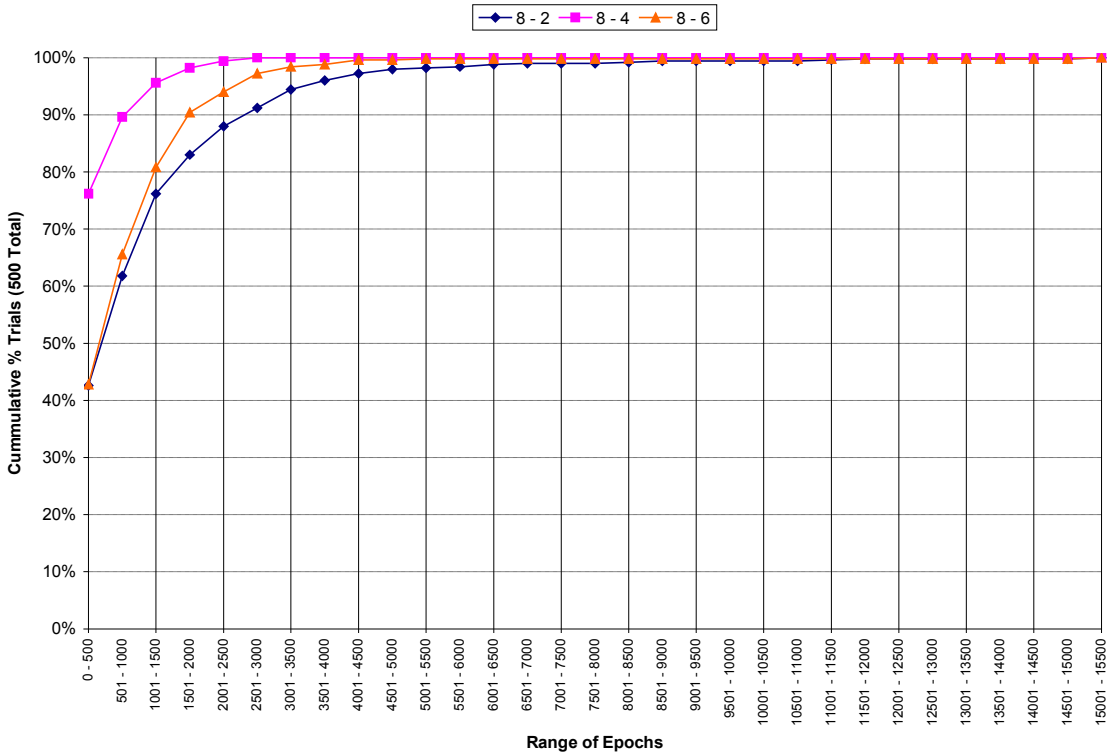

Figure D.5.8: Empirical CDF of Epochs for Hidden Layer Size 8 Networks by ρ Value (All Data)

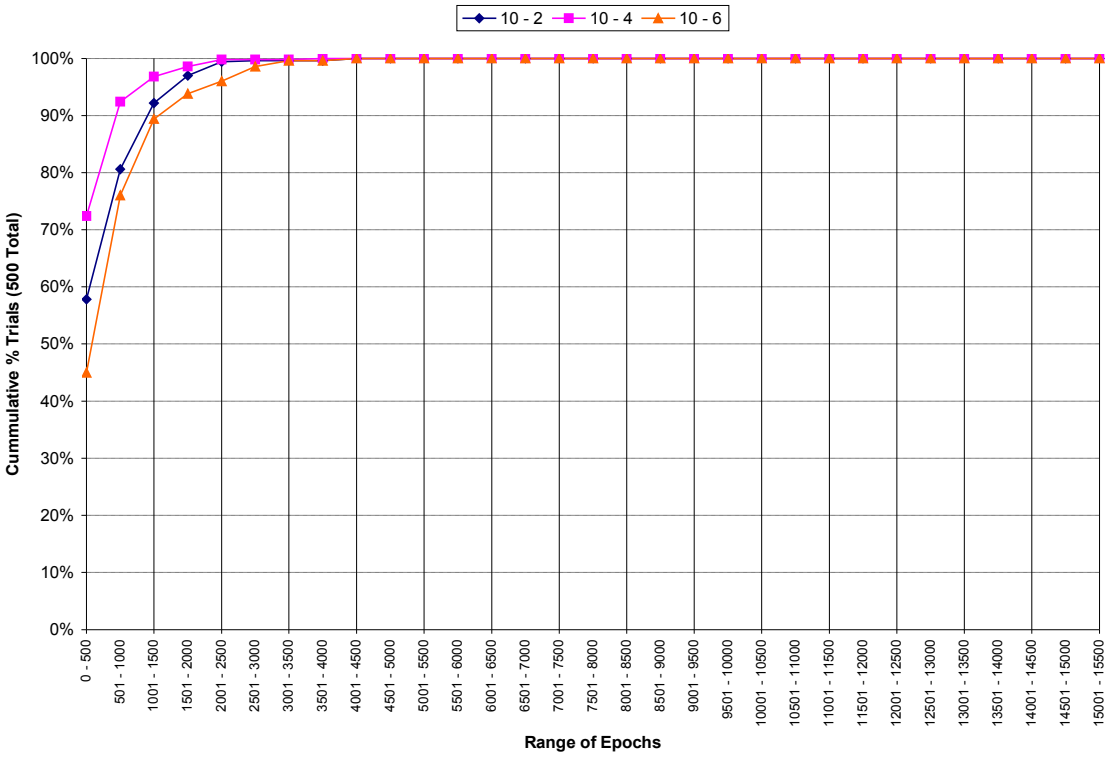

Figure D.5.9: Empirical CDF of Epochs for Hidden Layer Size 10 Networks by ρ Value (All Data)

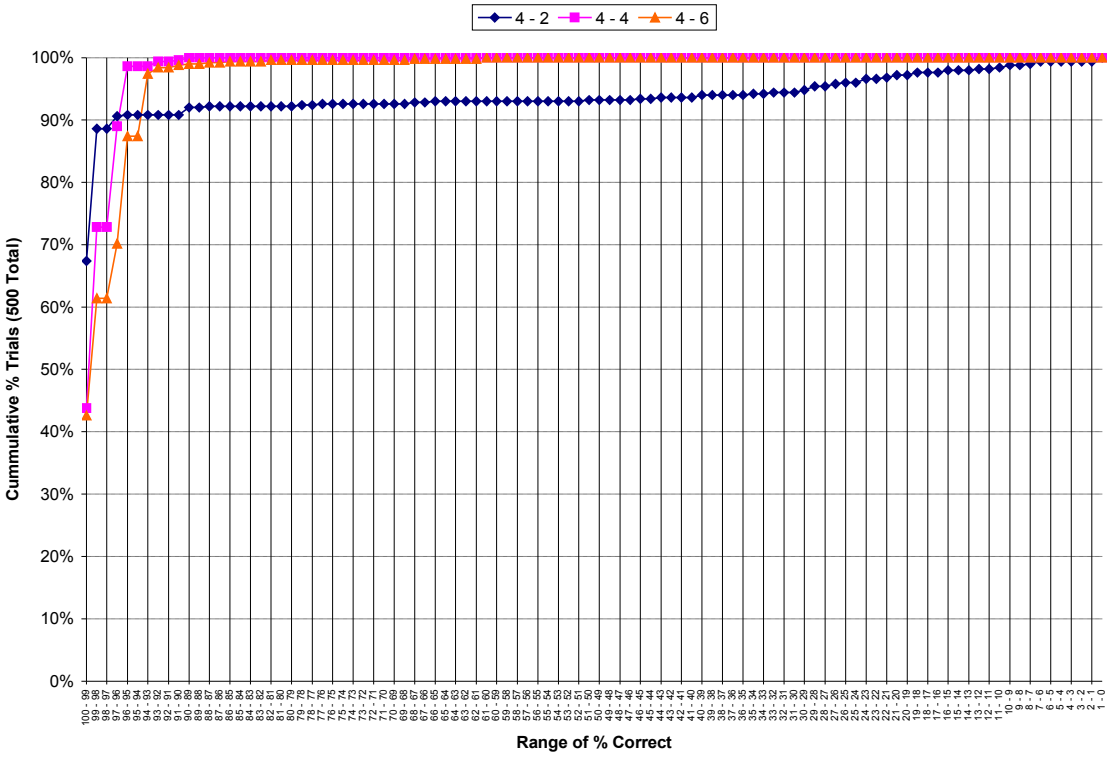

Figure D.5.10: Empirical CDF of Percent Correct for Hidden Layer Size 4 Networks by ρ Value (All Data)

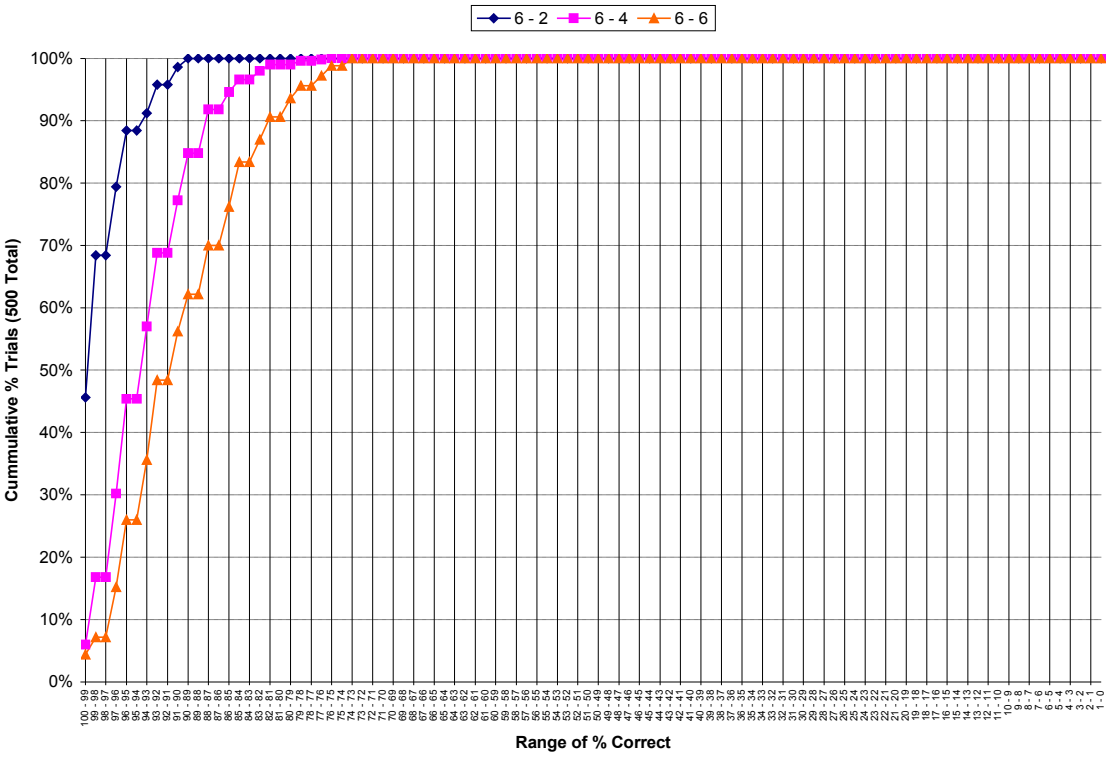

Figure D.5.11: Empirical CDF of Percent Correct for Hidden Layer Size 6 Networks by ρ Value (All Data)

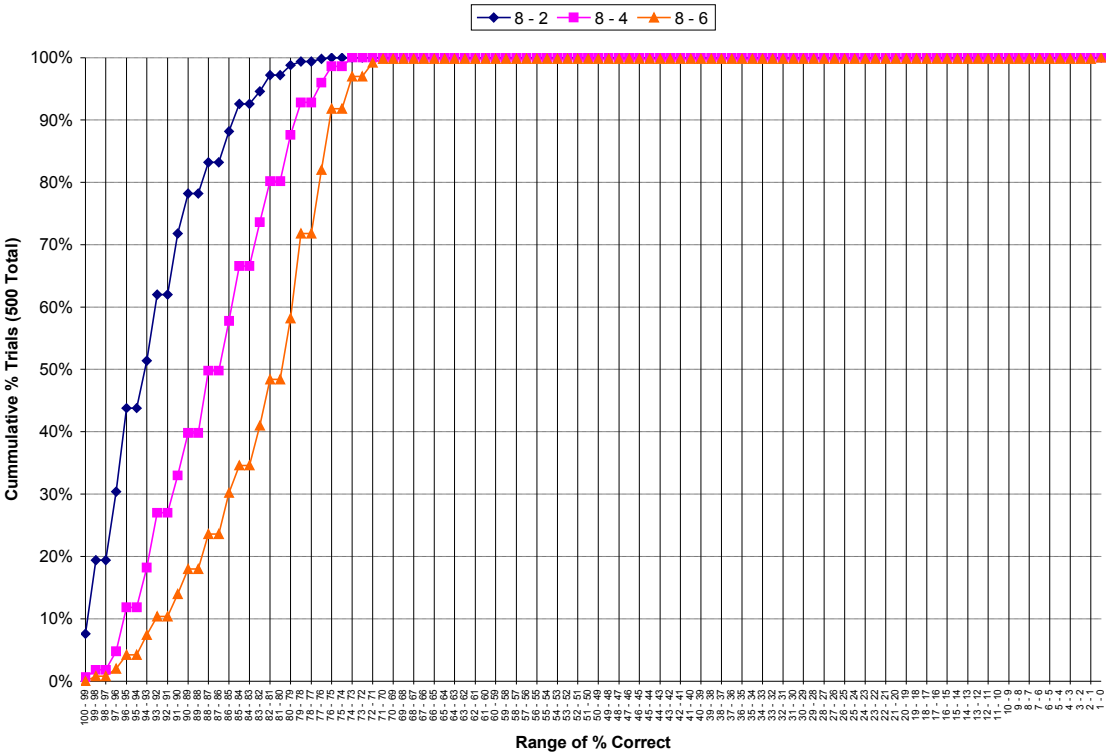

Figure D.5.12: Empirical CDF of Percent Correct for Hidden Layer Size 8 Networks by ρ Value (All Data)

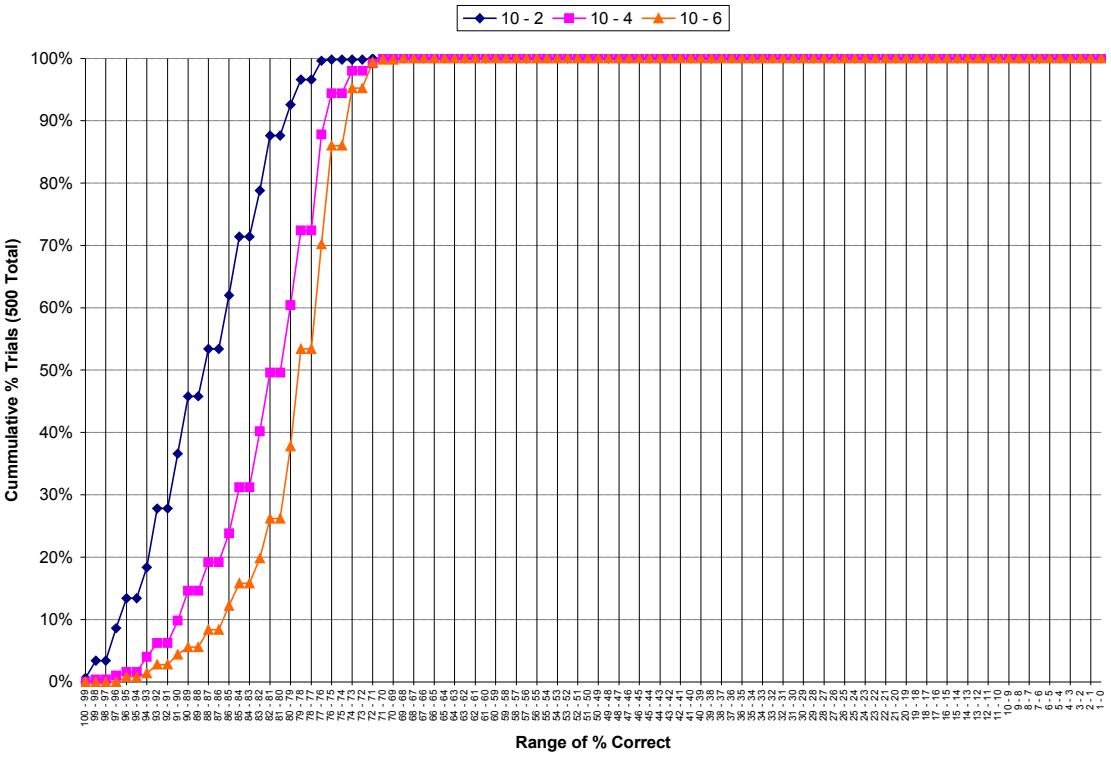

Figure D.5.13: Empirical CDF of Percent Correct for Hidden Layer Size 10 Networks by ρ Value (All Data)

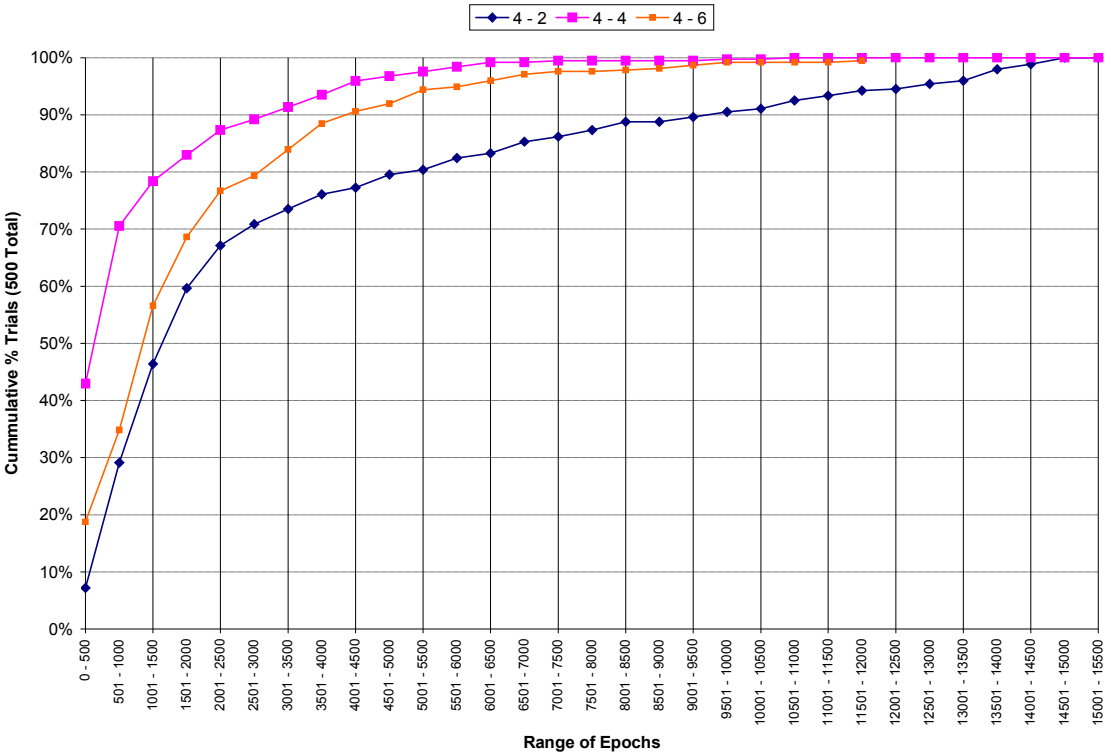

Figure D.5.14: Empirical CDF of Epochs for Hidden Layer Size 4 Networks by ρ Value (Convergent Trials Only)

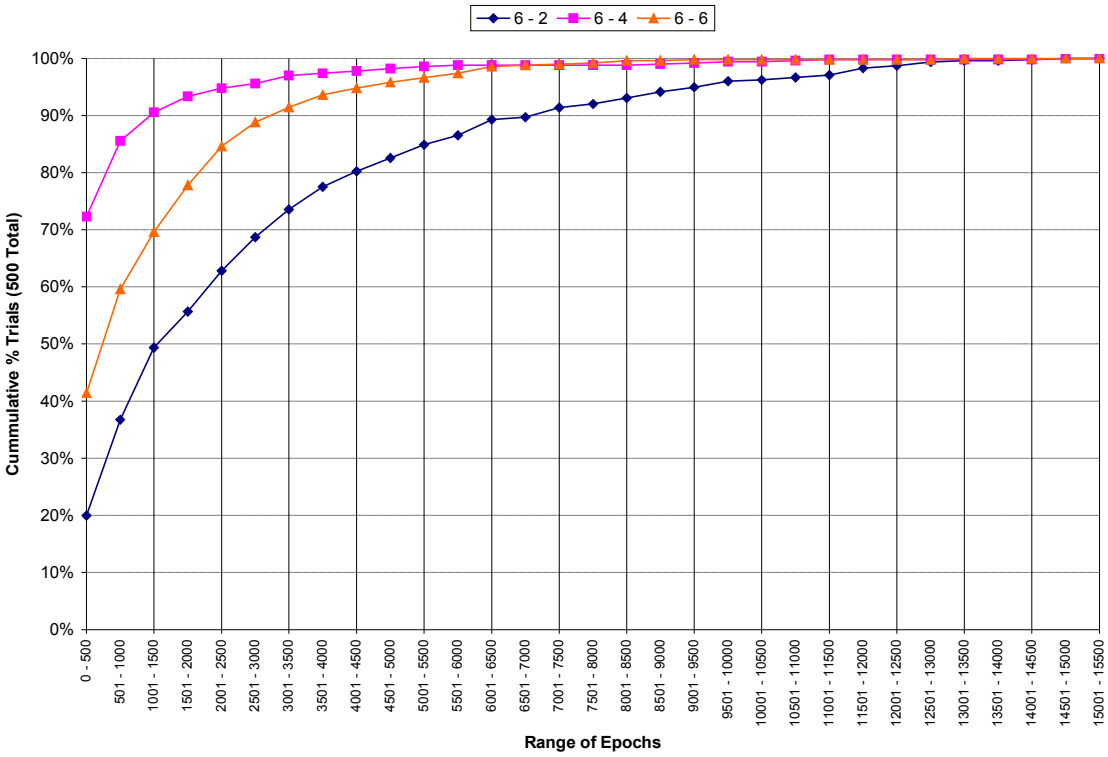

Figure D.5.15: Empirical CDF of Epochs for Hidden Layer Size 6 Networks by ρ Value (Convergent Trials Only)

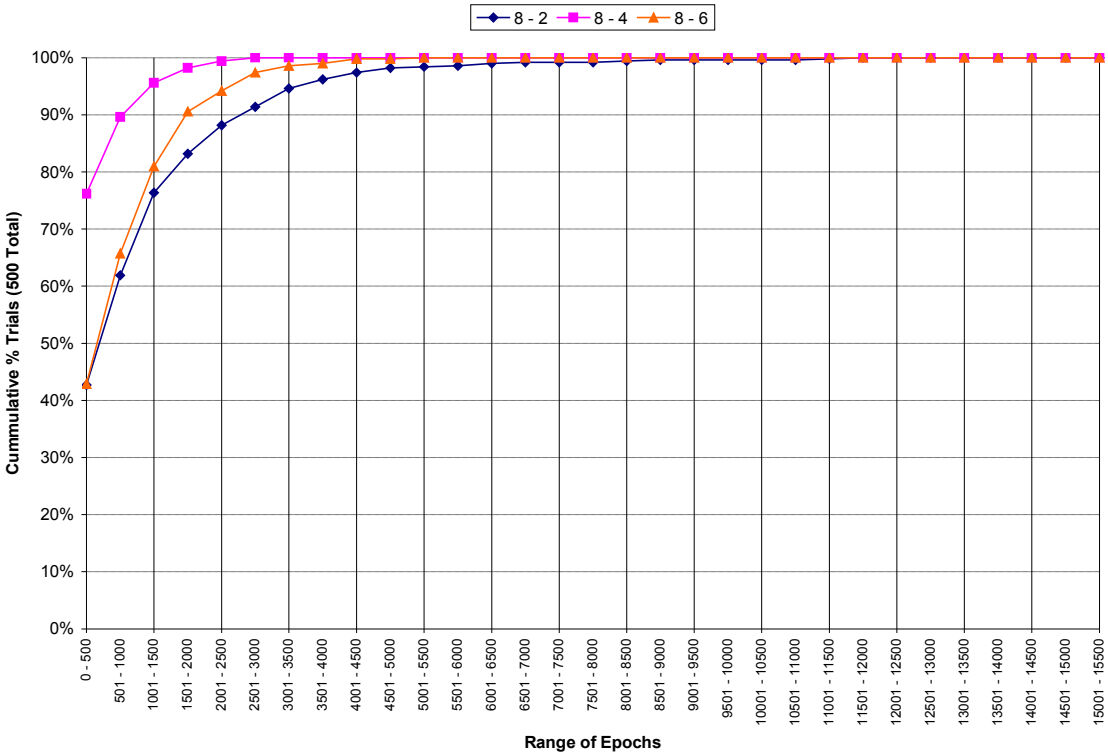

Figure D.5.16: Empirical CDF of Epochs for Hidden Layer Size 8 Networks by ρ Value (Convergent Trials Only)

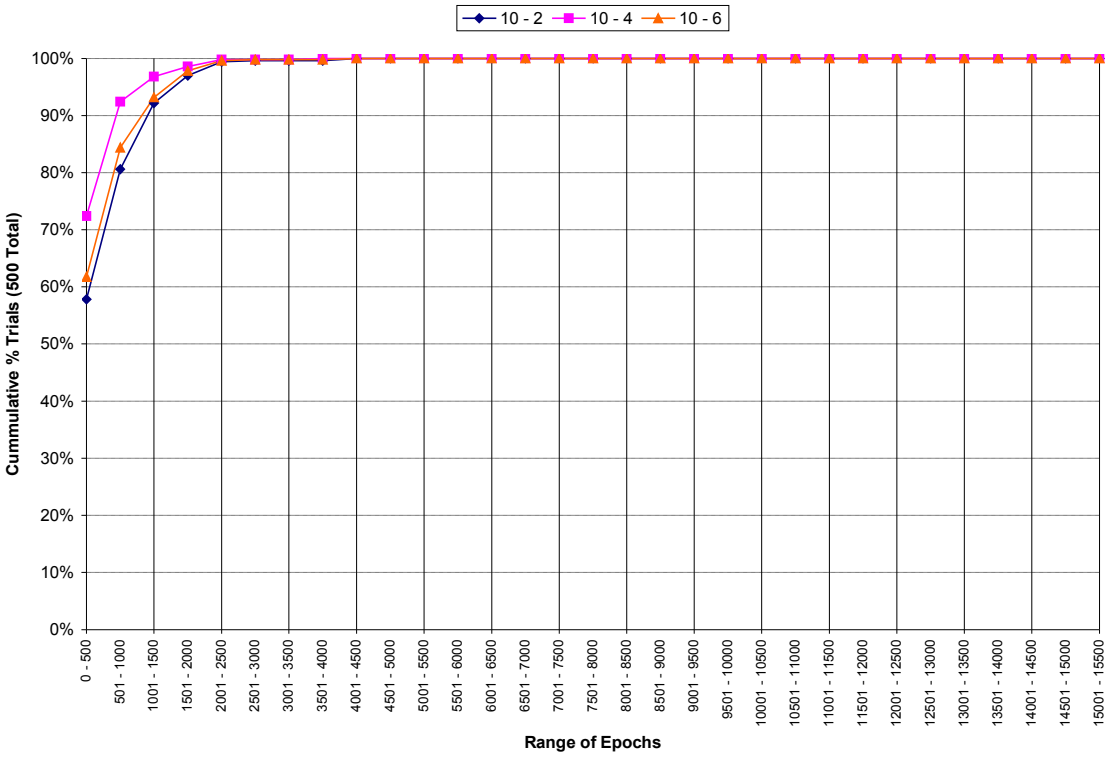

Figure D.5.17: Empirical CDF of Epochs for Hidden Layer Size 10 Networks by ρ Value (Convergent Trials Only)

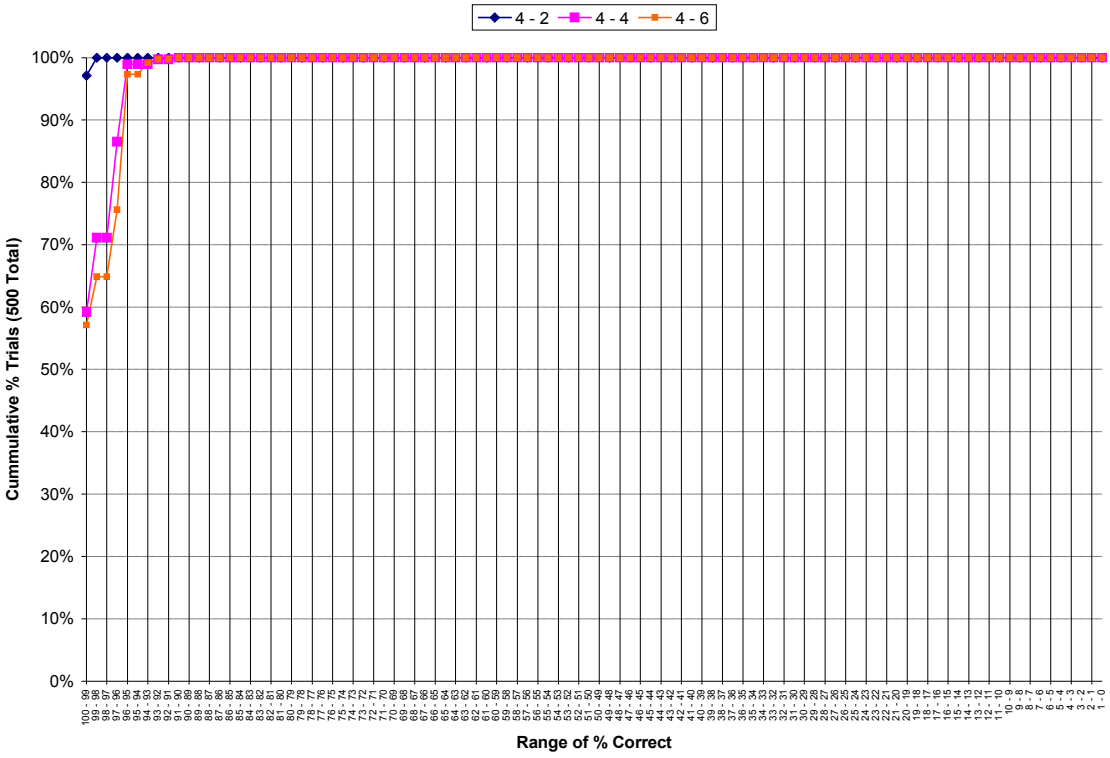

Figure D.5.18: Empirical CDF of Percent Correct for Hidden Layer Size 4 Networks by ρ Value (Convergent Trials Only)

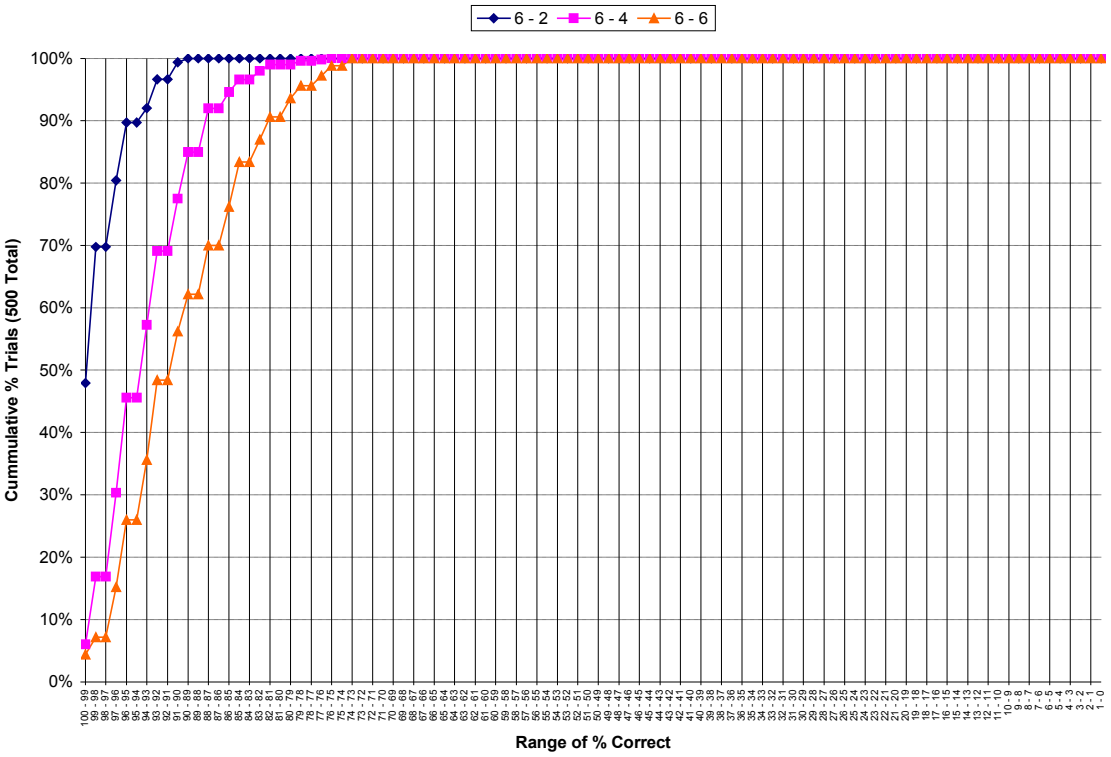

Figure D.5.19: Empirical CDF of Percent Correct for Hidden Layer Size 6 Networks by ρ Value (Convergent Trials Only)

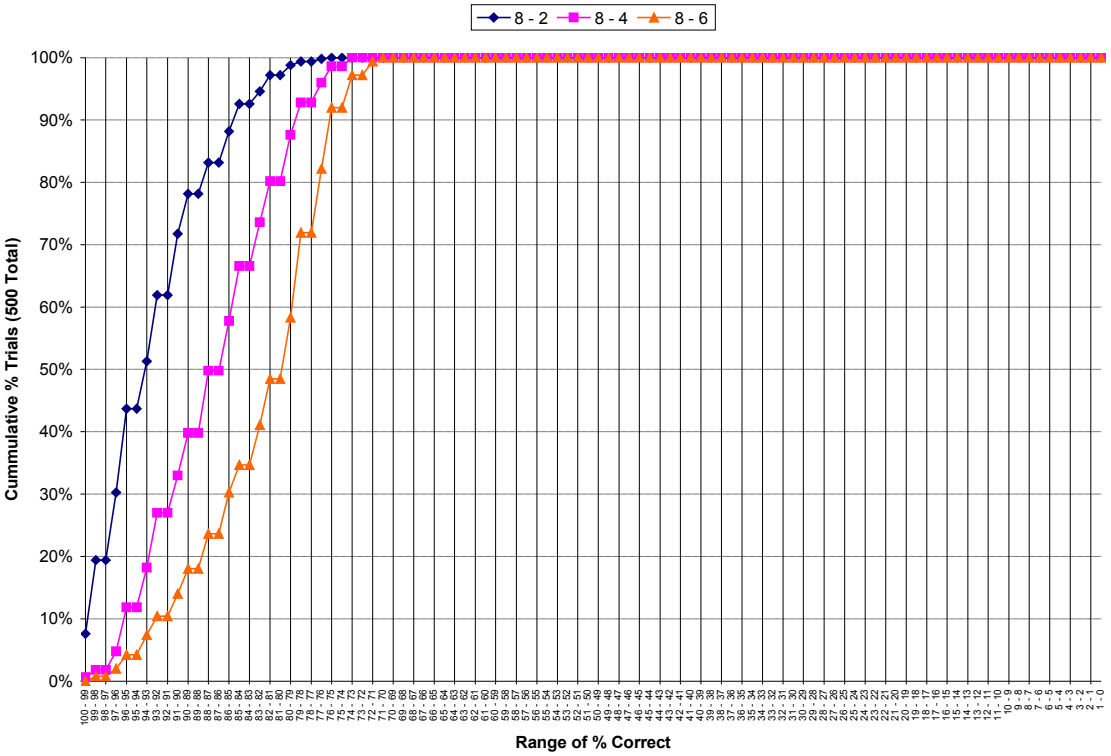

Figure D.5.20: Empirical CDF of Percent Correct for Hidden Layer Size 8 Networks by ρ Value (Convergent Trials Only)

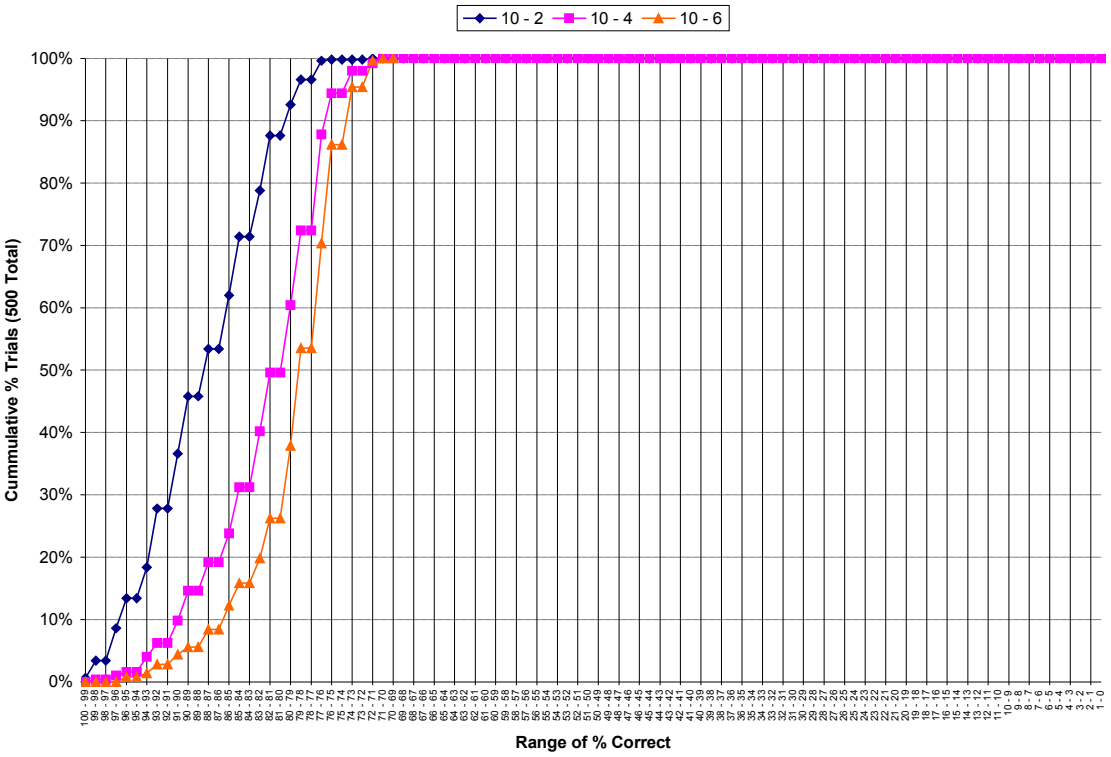

Figure D.5.21: Empirical CDF of Percent Correct for Hidden Layer Size 10 Networks by ρ Value (Convergent Trials Only)

D6: Finite-Difference Bit Counting

Data Set	Total Trials	Convergent Trials	Min Epochs	Mean Epochs	Median Epochs	Max Epochs	StdDev Epochs
4 - 2	500	21	2,148	14,649	15,000	15,000	1,821
4 - 4	500	0	15,000	15,000	15,000	15,000	0
4 - 6	500	0	15,000	15,000	15,000	15,000	0
6 - 2	500	250	908	10,116	14,949	15,000	5,322
6 - 4	500	55	2,058	14,037	15,000	15,000	2,920
6 - 6	500	10	5,608	14,862	15,000	15,000	992
8 - 2	500	413	826	5,822	3,676	15,000	4,792
8 - 4	500	226	1,530	10,828	15,000	15,000	4,973
8 - 6	500	87	3,492	13,511	15,000	15,000	3,381
10 - 2	500	484	801	3,263	2,332	15,000	2,879
10 - 4	500	356	1,561	7,940	5,846	15,000	5,036
10 - 6	500	145	2,573	12,163	15,000	15,000	4,558

Table D.6.1: Summary Statistics for by Data Set (All Data)

Data Set	Total Trials	Convergent Trials	Min % Correct	Mean % Correct	Median % Correct	Max % Correct	StdDev % Correct
4 - 2	500	21	16.4	73.5	81.3	100.0	23.9
4 - 4	500	0	0.0	18.9	14.8	89.8	19.5
4 - 6	500	0	0.8	15.9	16.4	74.2	10.5
6 - 2	500	250	8.6	89.9	96.1	100.0	14.2
6 - 4	500	55	0.8	33.8	23.4	100.0	30.2
6 - 6	500	10	0.8	19.1	18.0	86.7	14.6
8 - 2	500	413	21.9	90.8	93.0	100.0	9.4
8 - 4	500	226	0.8	55.2	69.9	100.0	32.1
8 - 6	500	87	0.8	28.0	21.9	80.5	22.2
10 - 2	500	484	50.0	85.6	86.3	100.0	8.5
10 - 4	500	356	1.6	65.8	73.4	99.2	24.2
10 - 6	500	145	0.0	36.4	26.6	89.8	24.4

Table D.6.2: Summary Statistics for Percent Correct by Data Set (All Data)

Data Set	Convergent Trials	Min Epochs	Mean Epochs	Median Epochs	Max Epochs	StdDev Epochs
4 - 2	21	2,148	6,624	5,309	14,139	3,486
4 - 4	0	0	0	0	0	0
4 - 6	0	0	0	0	0	0
6 - 2	250	908	5,231	4,075	14,896	2,971
6 - 4	55	2,058	6,236	5,880	13,133	3,027
6 - 6	10	5,608	8,058	7,824	10,471	1,436
8 - 2	413	826	3,888	3,121	14,594	2,505
8 - 4	226	1,530	5,768	5,276	14,260	2,816
8 - 6	87	3,492	6,436	5,978	14,393	2,243
10 - 2	484	801	2,875	2,312	14,249	1,962
10 - 4	356	1,561	5,084	4,411	14,148	2,692
10 - 6	145	2,573	5,216	4,790	13,259	1,882

Table D.6.3: Summary Statistics for by Data Set (Convergent Trials Only)

Data Set	Convergent Trials	Min % Correct	Mean % Correct	Median % Correct	Max % Correct	StdDev % Correct
4 - 2	21	96.9	98.7	98.4	100.0	1.2
4 - 4	0	0	0	0	0	0
4 - 6	0	0	0	0	0	0
6 - 2	250	82.8	98.0	100.0	100.0	3.5
6 - 4	55	74.2	87.2	85.9	100.0	7.0
6 - 6	10	66.4	71.5	71.1	78.9	3.3
8 - 2	413	62.5	92.5	94.5	100.0	636
8 - 4	226	66.4	82.0	81.3	100.0	7.9
8 - 6	87	61.7	69.7	69.5	80.5	3.8
10 - 2	484	57.8	86.0	86.7	100.0	8.0
10 - 4	356	62.5	77.6	76.6	99.2	7.7
10 - 6	145	61.7	69.2	68.8	89.8	3.8

Table D.6.3: Summary Statistics for Percent Correct by Data Set (Convergent Trials Only)

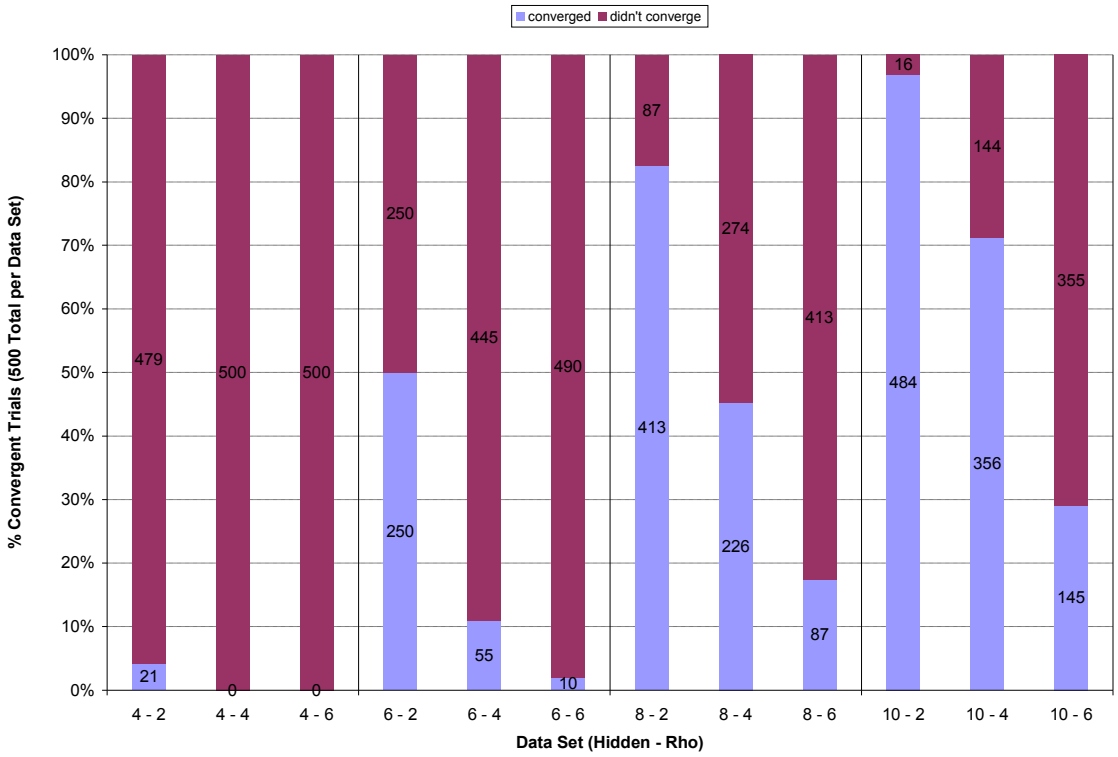

Figure D.6.1: Percent of Convergent Trials by Data Set

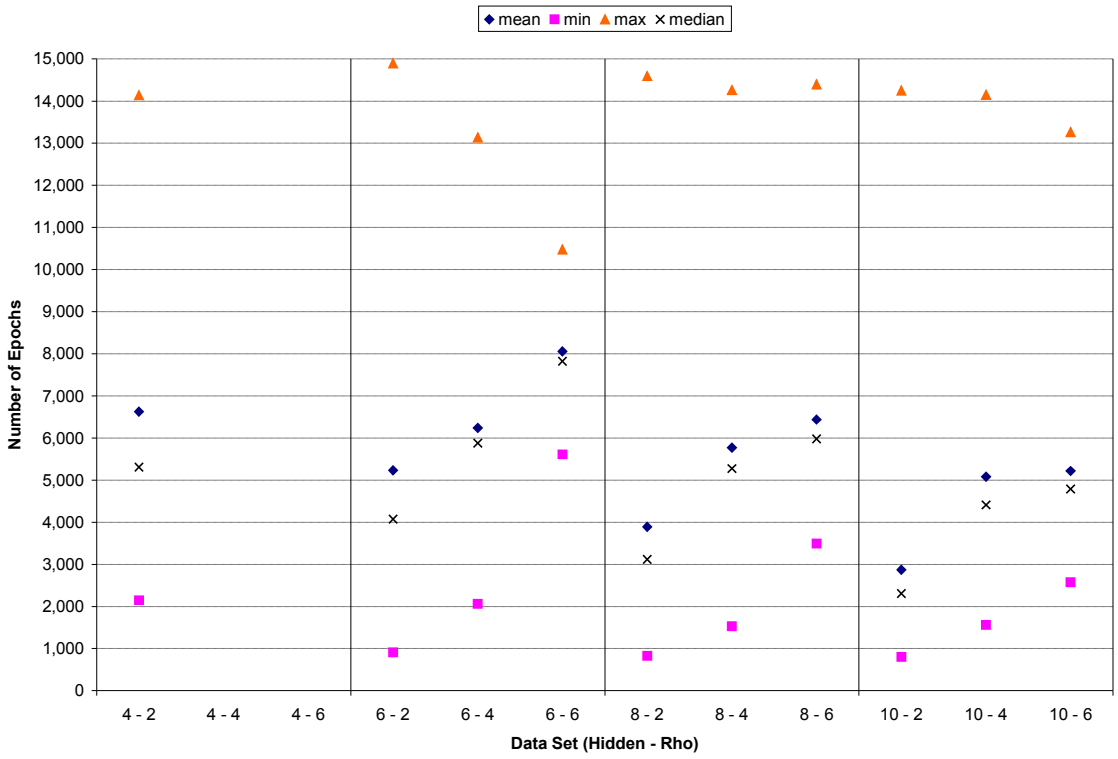

Figure D.6.2: Plot of Summary Statistics for Epochs by Data Set (Convergent Trials Only)

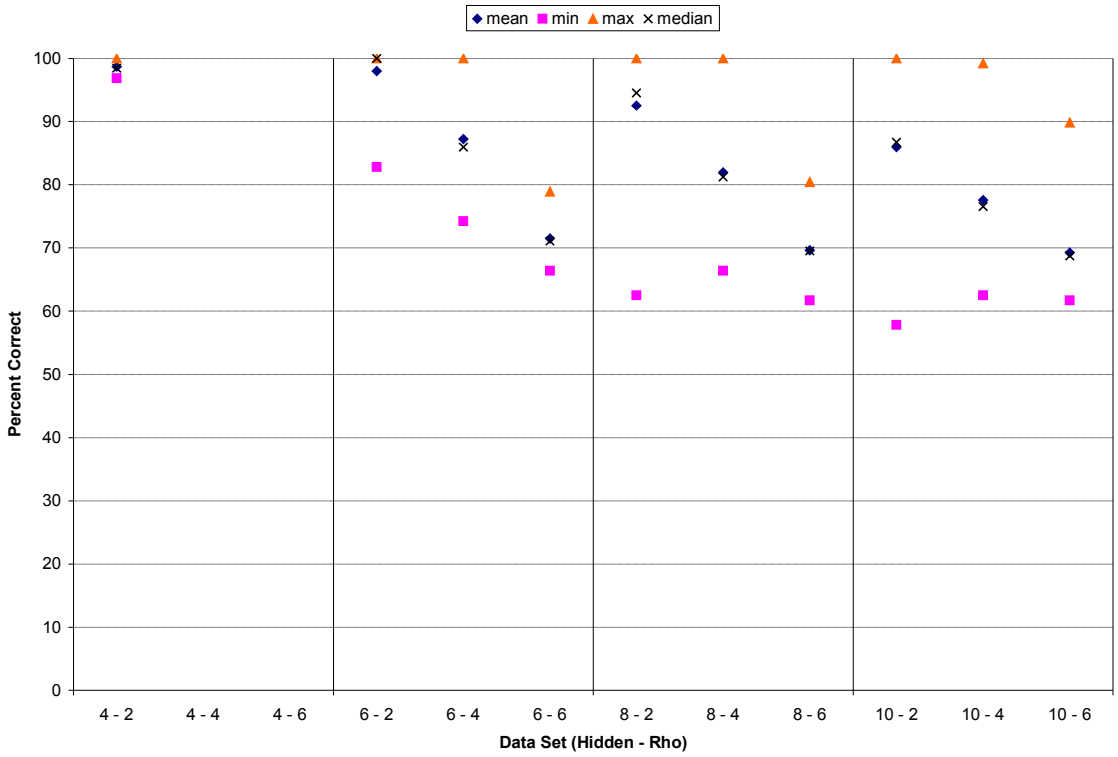

Figure D.6.3: Plot of Summary Statistics for Percent Correct by Data Set (Convergent Trials Only)

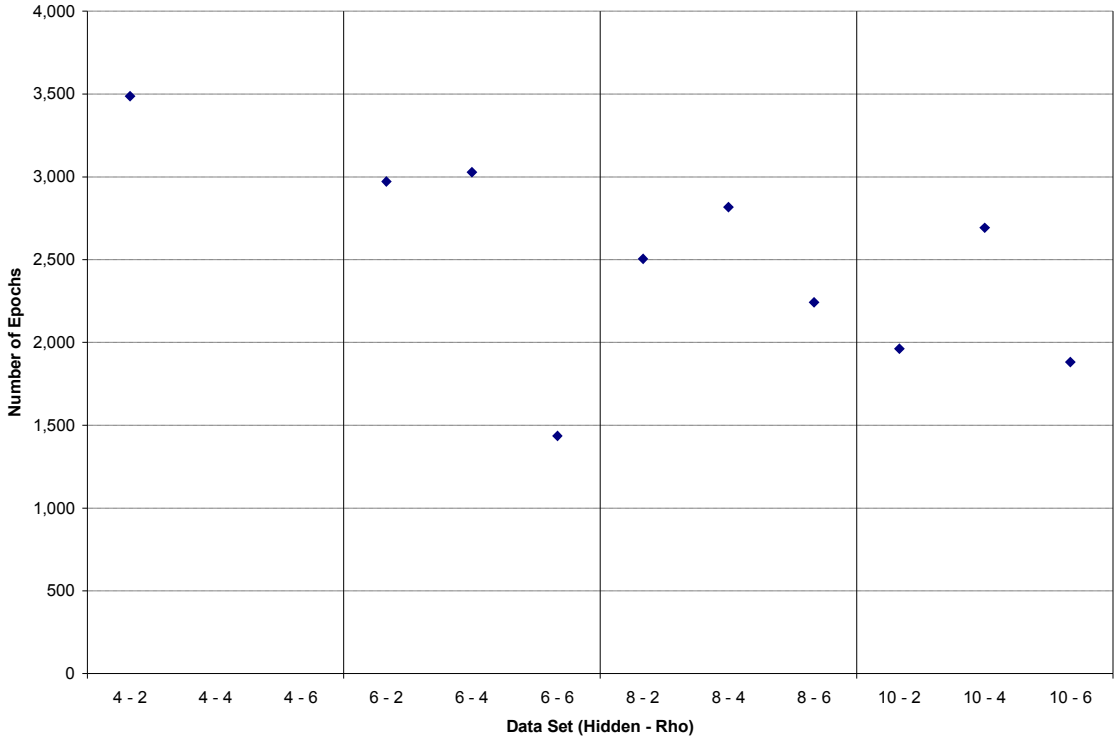

Figure D.6.4: Plot of Standard Deviations of Epochs by Data Set (Convergent Trials Only)

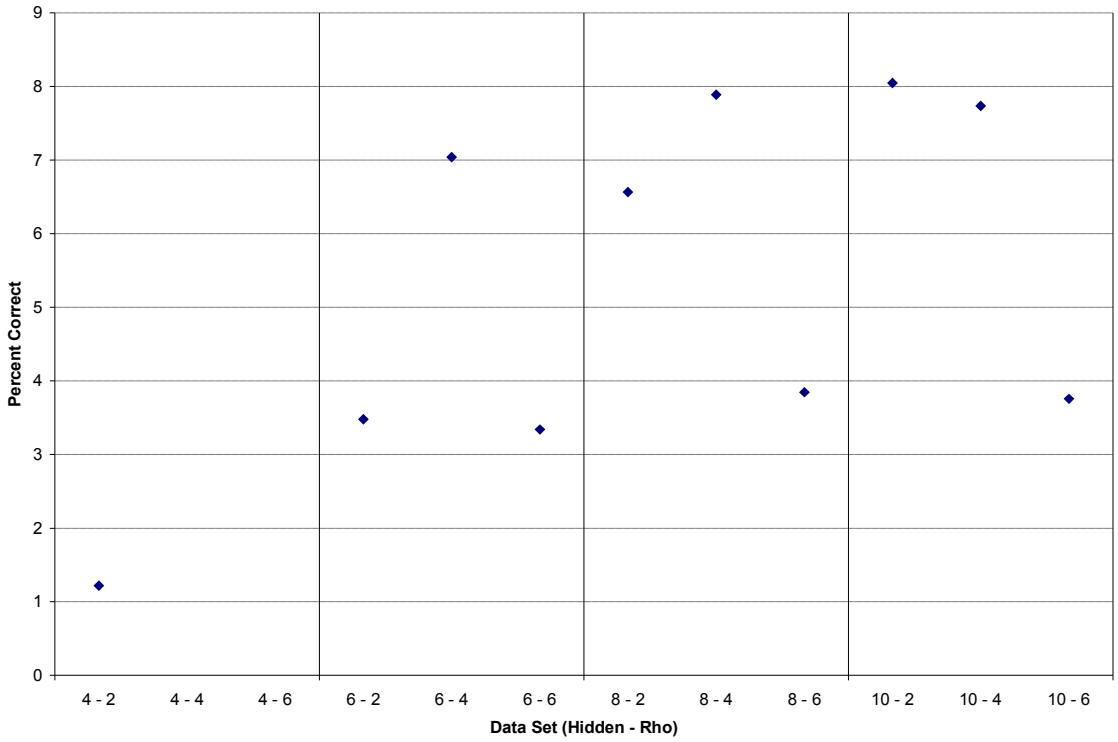

Figure D.6.5: Plot of Standard Deviations of Epochs by Data Set (Convergent Trials Only)

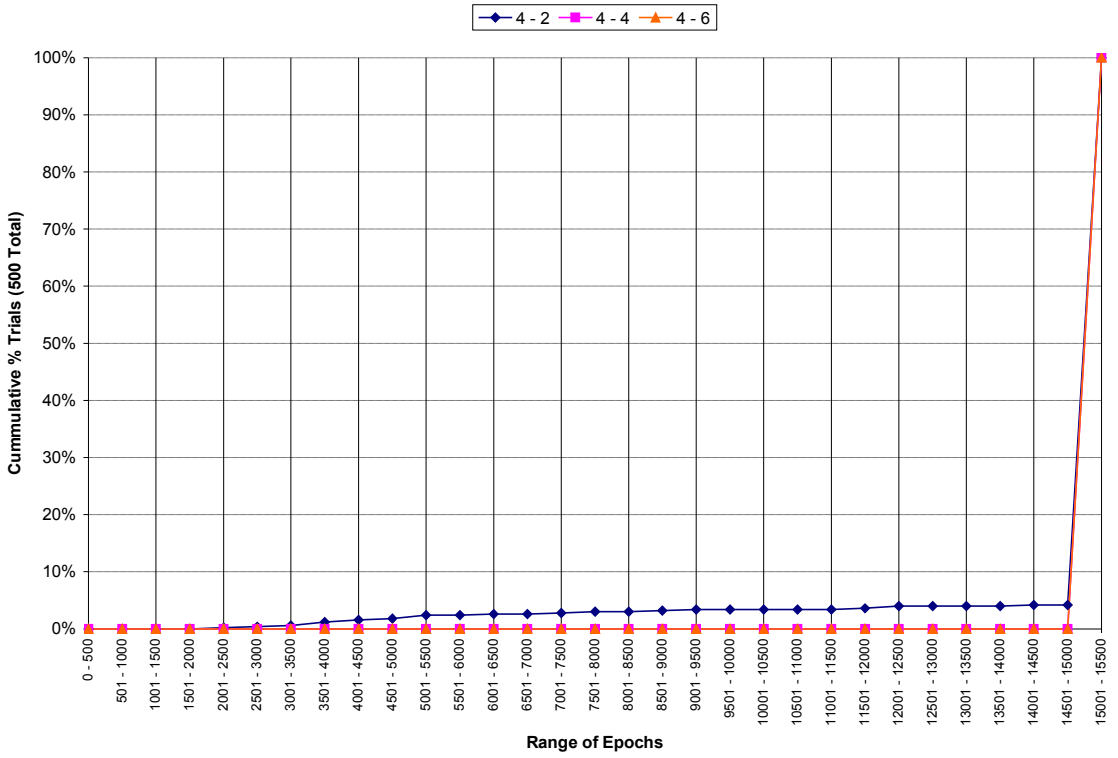

Figure D.6.6: Empirical CDF of Epochs for Hidden Layer Size 4 Networks by ρ Value (All Data)

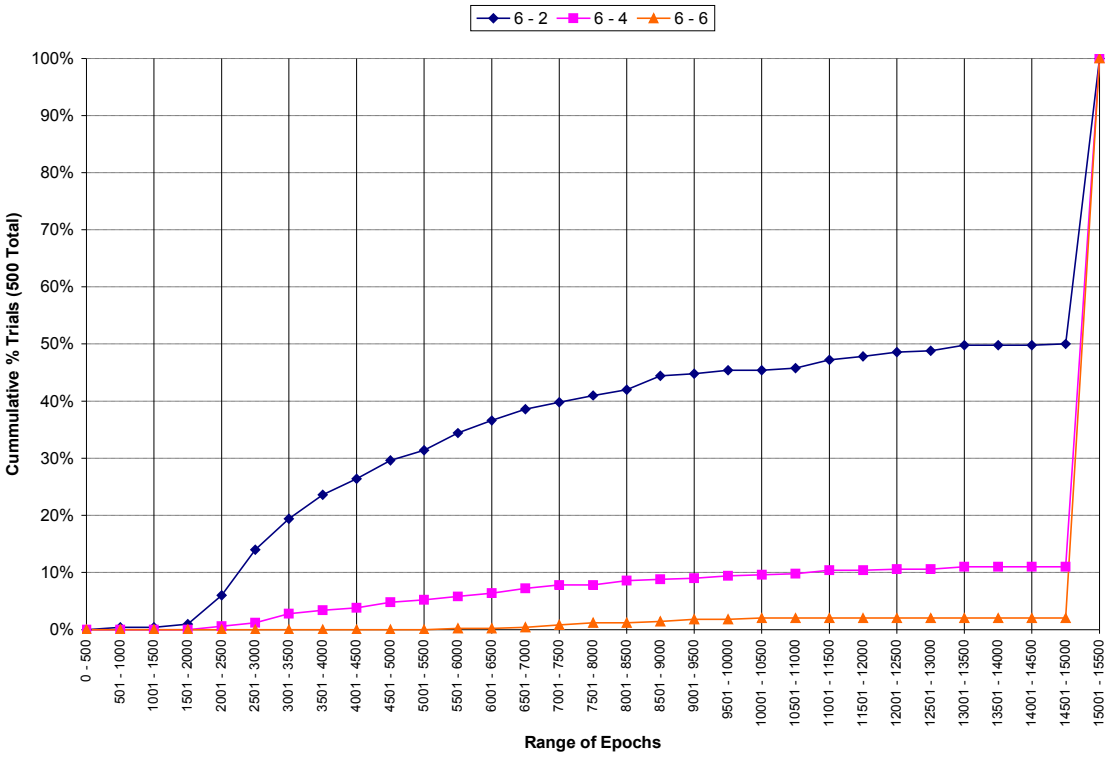

Figure D.6.7: Empirical CDF of Epochs for Hidden Layer Size 6 Networks by ρ Value (All Data)

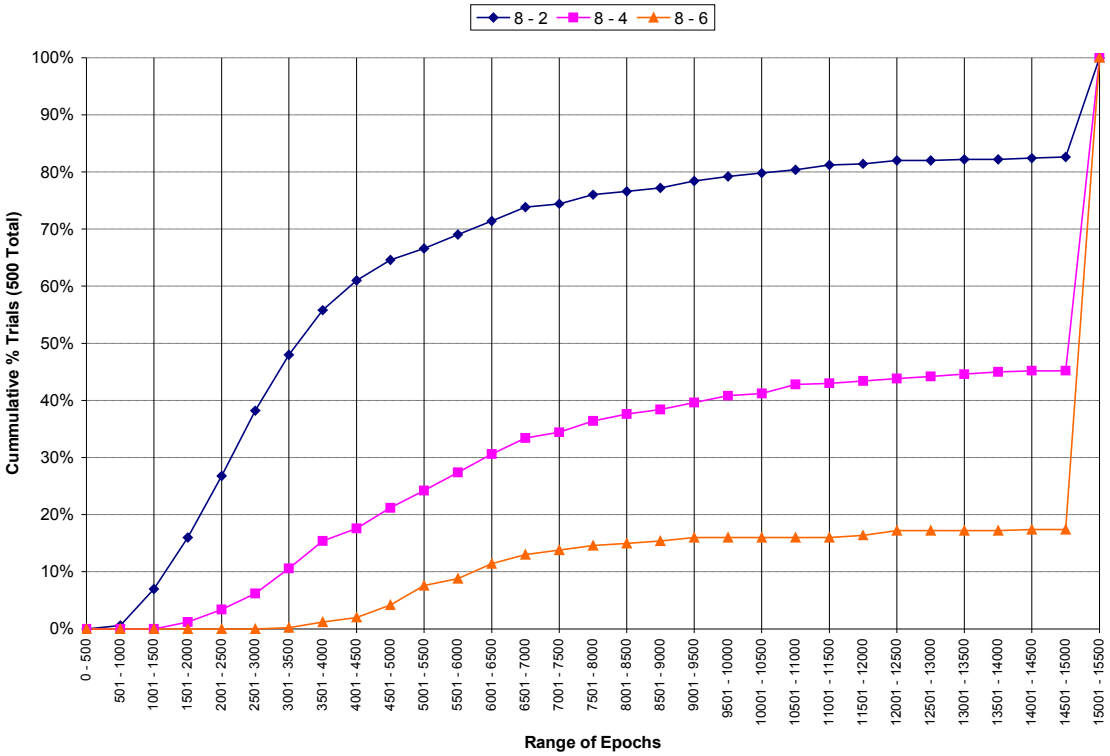

Figure D.6.8: Empirical CDF of Epochs for Hidden Layer Size 8 Networks by ρ Value (All Data)

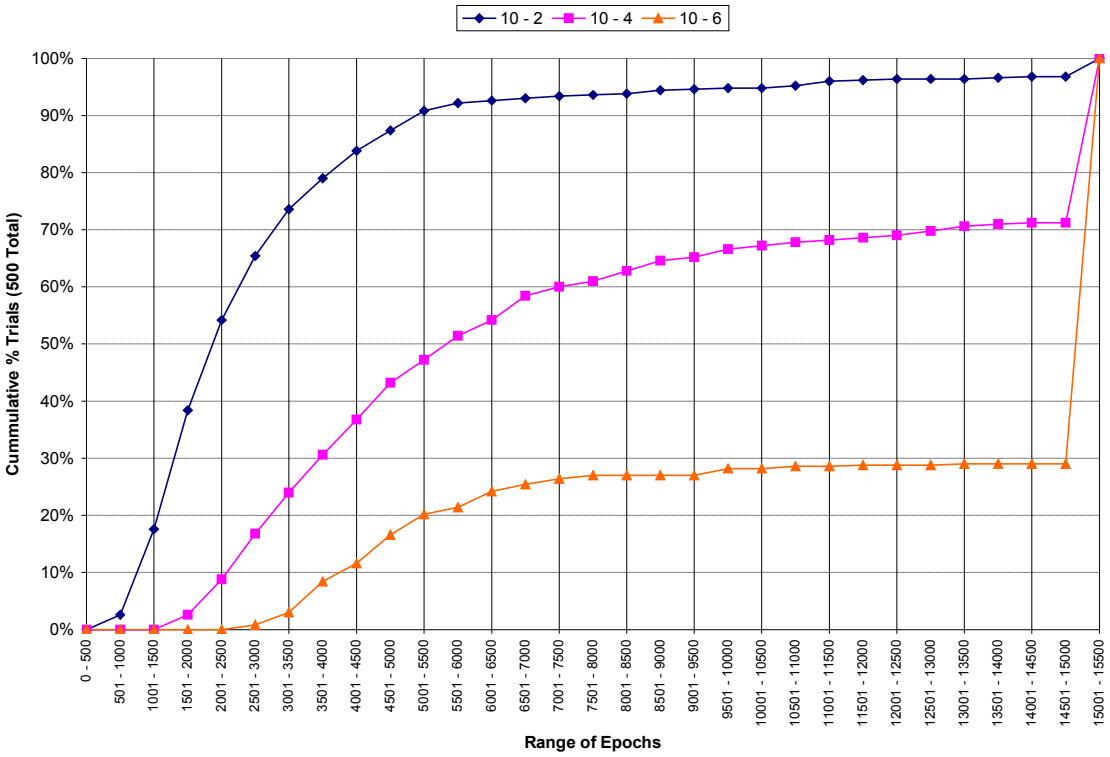

Figure D.6.9: Empirical CDF of Epochs for Hidden Layer Size 10 Networks by ρ Value (All Data)

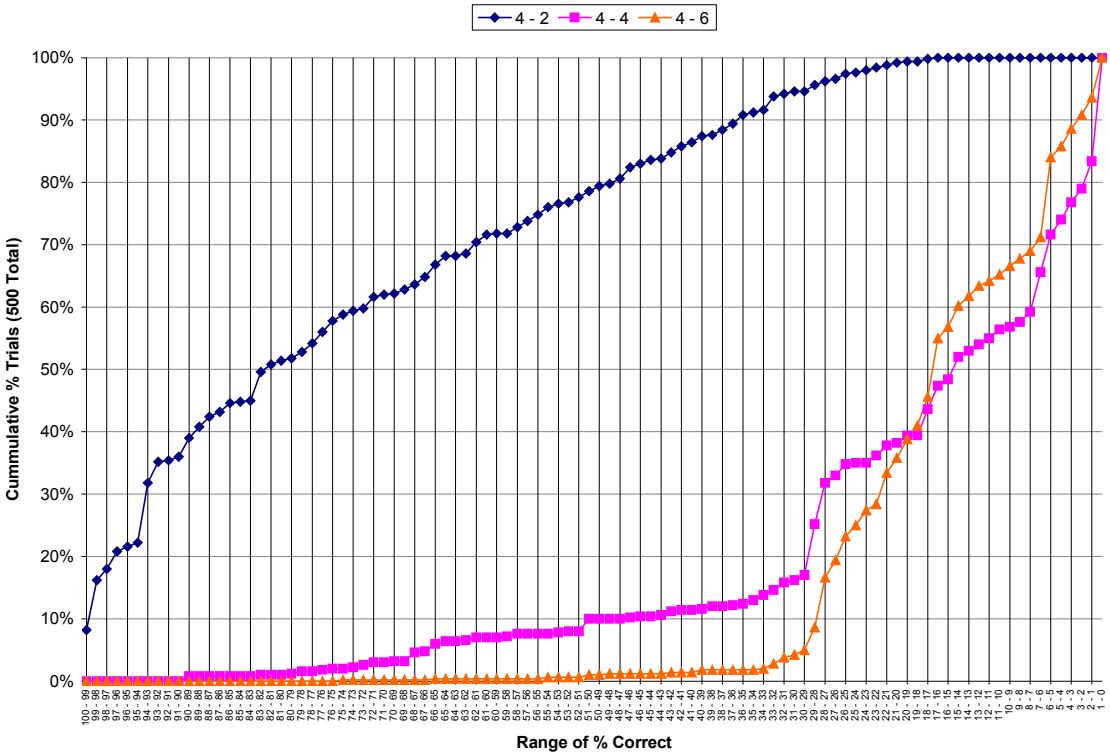

Figure D.6.10: Empirical CDF of Percent Correct for Hidden Layer Size 4 Networks by ρ Value (All Data)

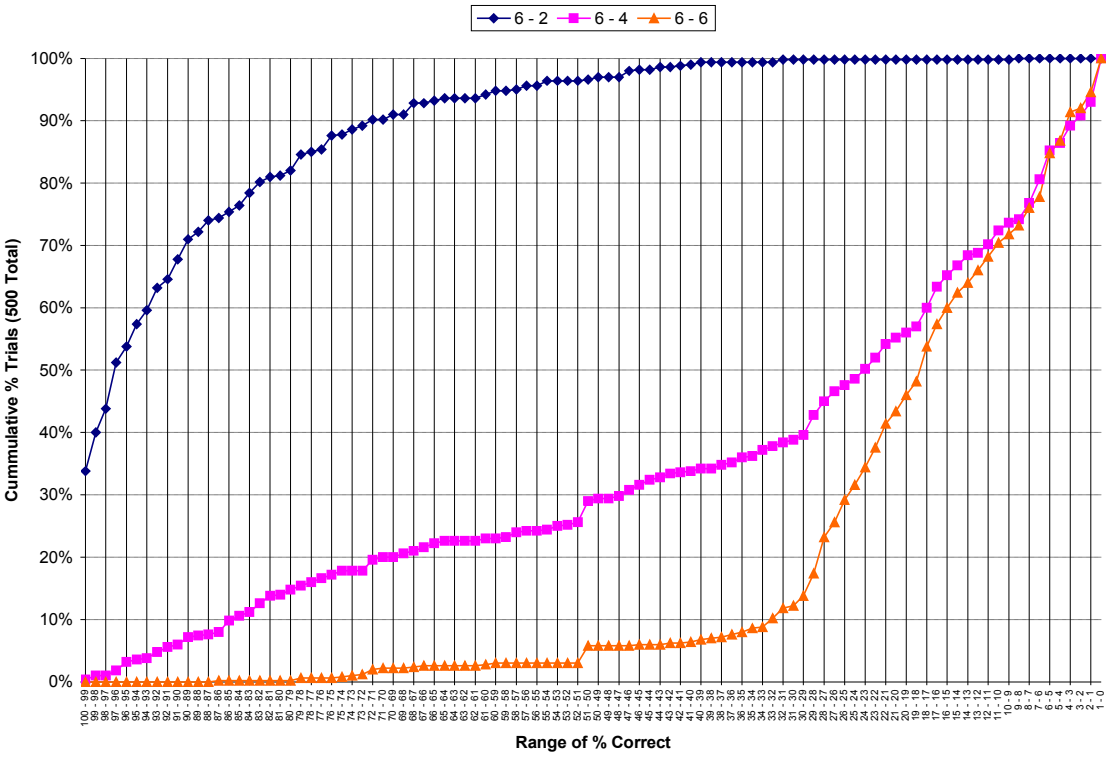

Figure D.6.11: Empirical CDF of Percent Correct for Hidden Layer Size 6 Networks by ρ Value (All Data)

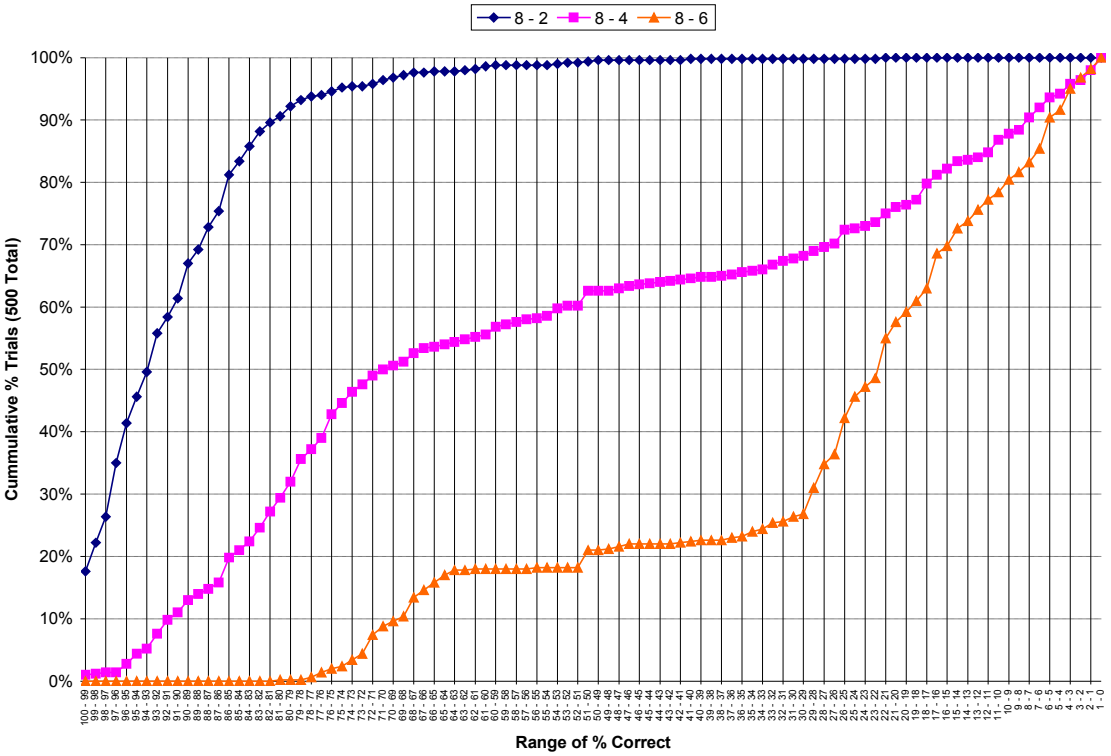

Figure D.6.12: Empirical CDF of Percent Correct for Hidden Layer Size 8 Networks by ρ Value (All Data)

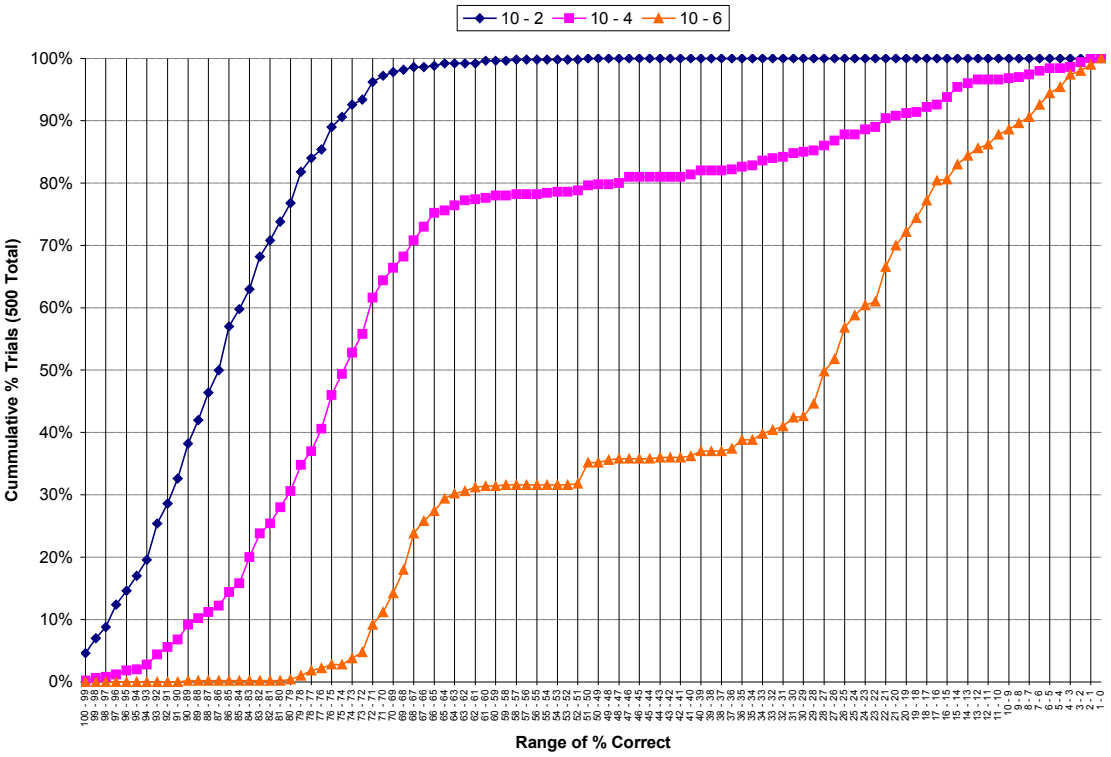

Figure D.6.13: Empirical CDF of Percent Correct for Hidden Layer Size 10 Networks by ρ Value (All Data)

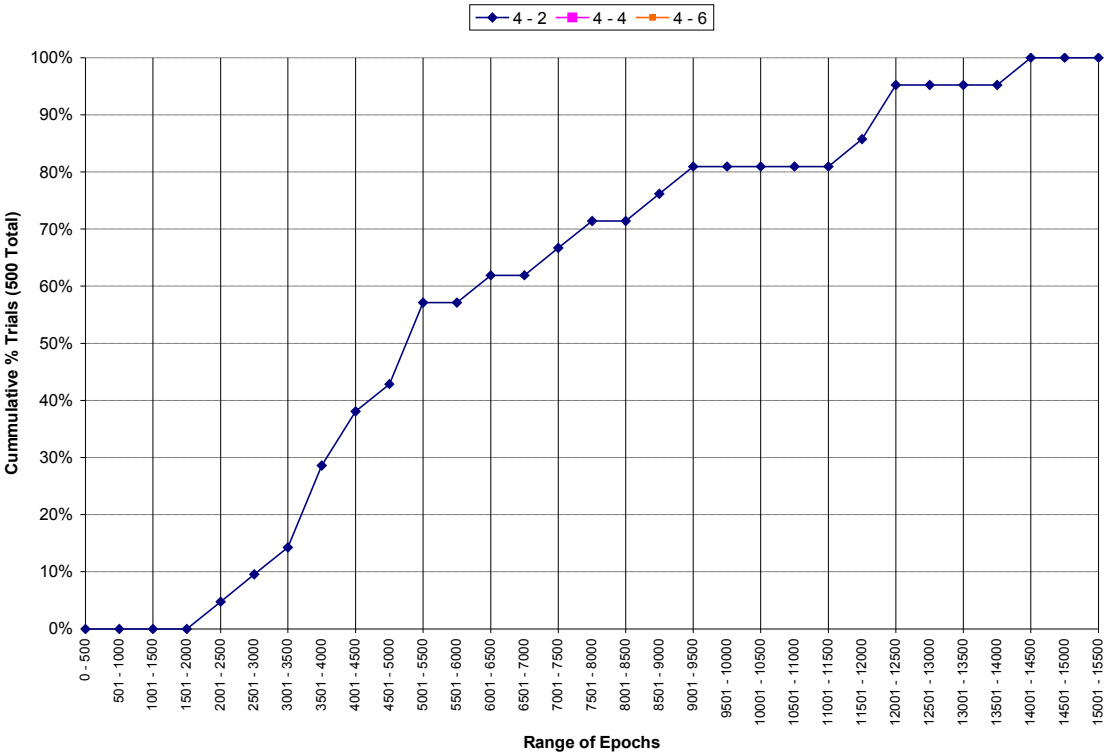

Figure D.6.14: Empirical CDF of Epochs for Hidden Layer Size 4 Networks by ρ Value (Convergent Trials Only)

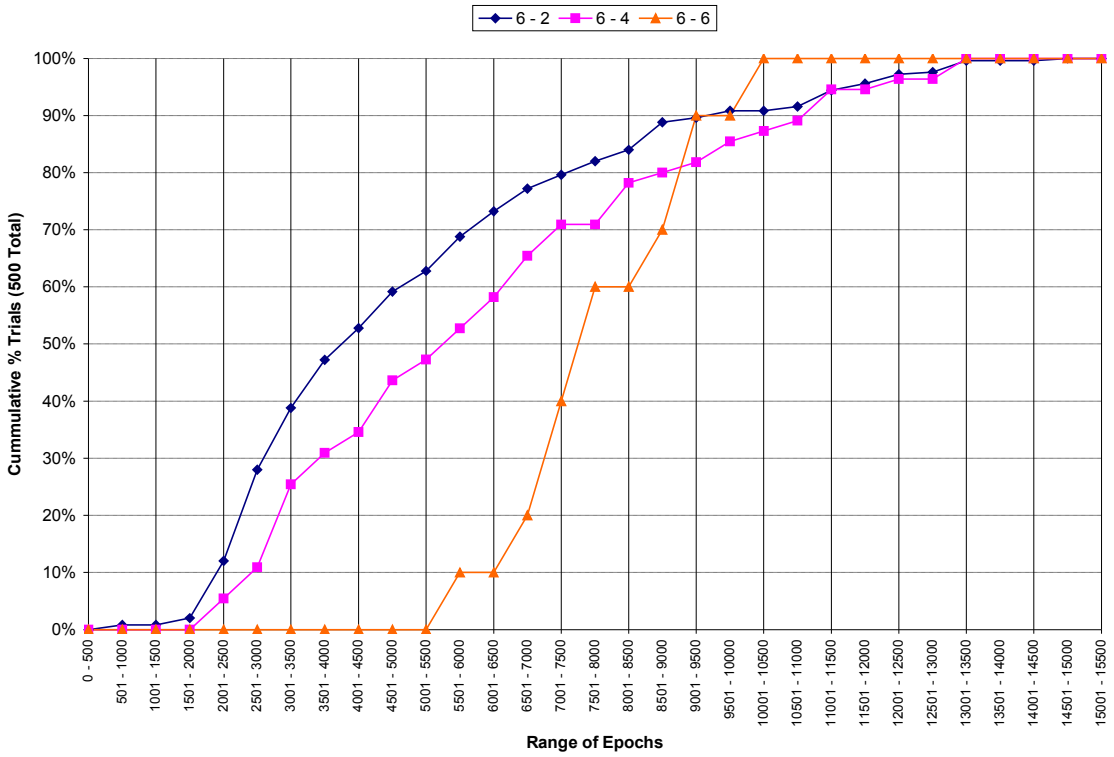

Figure D.6.15: Empirical CDF of Epochs for Hidden Layer Size 6 Networks by ρ Value (Convergent Trials Only)

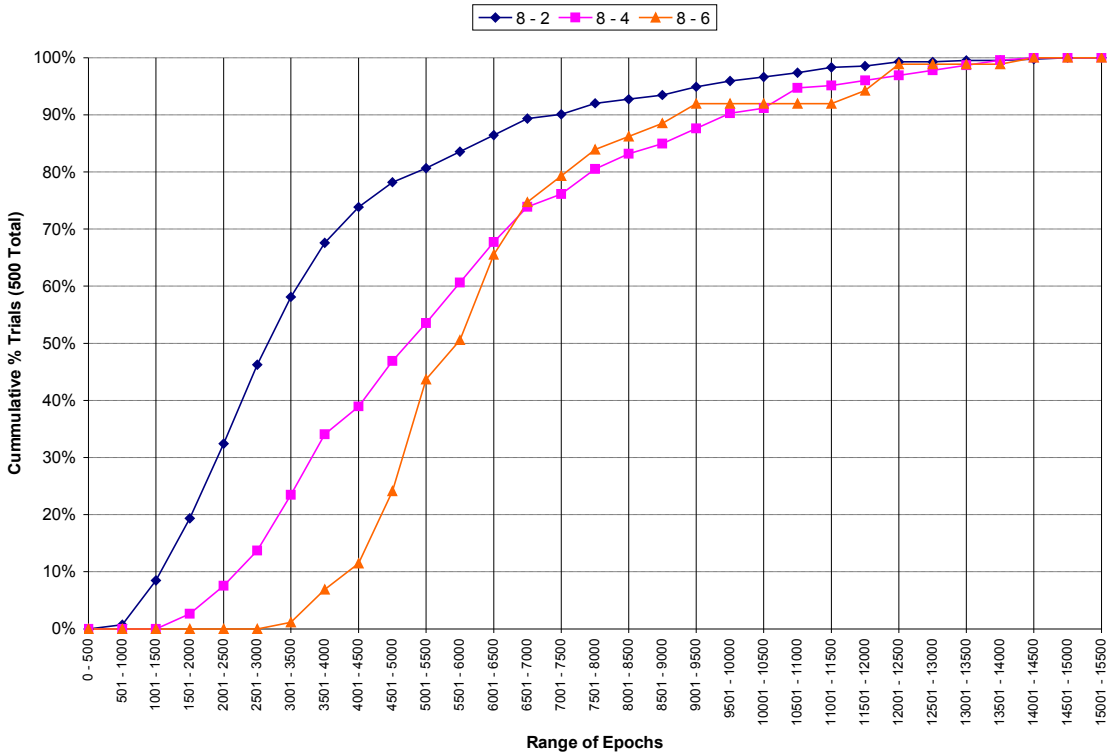

Figure D.6.16: Empirical CDF of Epochs for Hidden Layer Size 8 Networks by ρ Value (Convergent Trials Only)

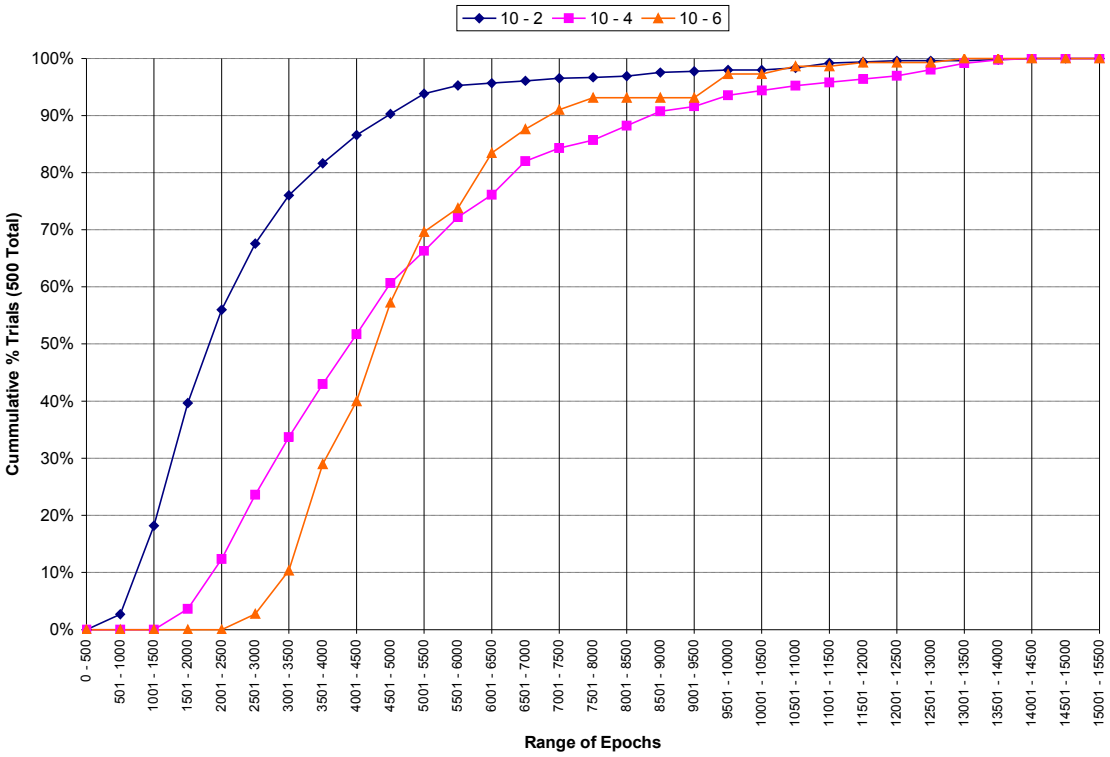

Figure D.6.17: Empirical CDF of Epochs for Hidden Layer Size 10 Networks by ρ Value (Convergent Trials Only)

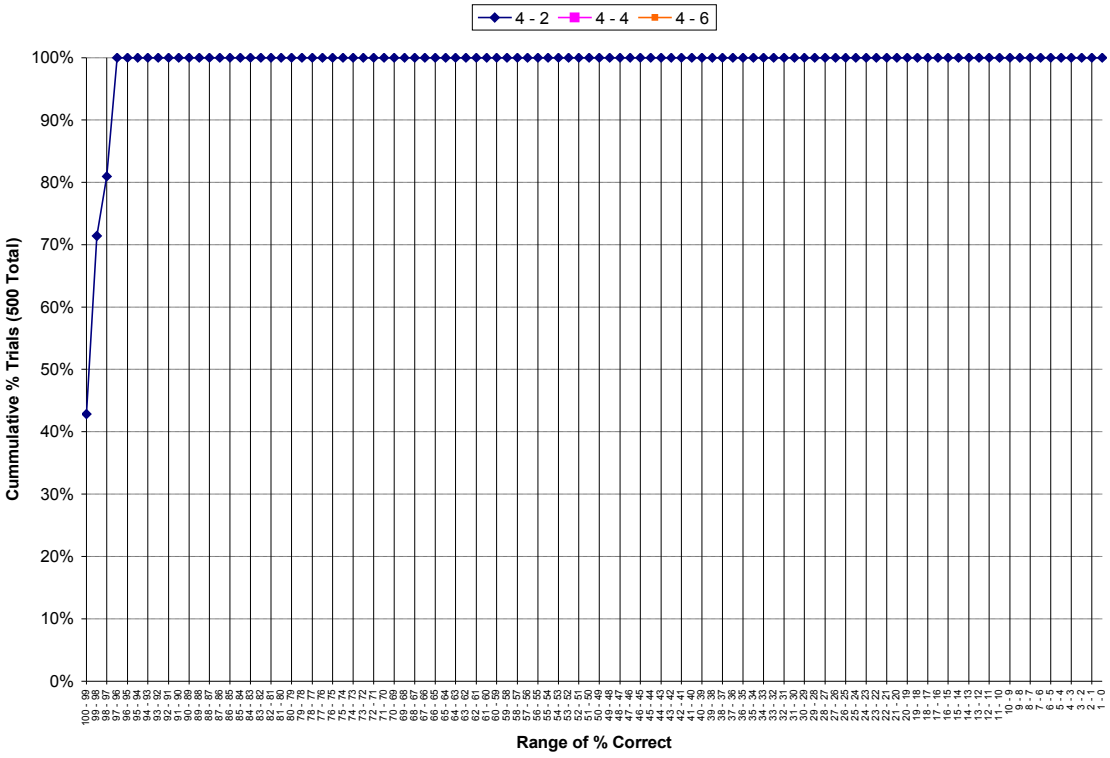

Figure D.6.18: Empirical CDF of Epochs for Hidden Layer Size 4 Networks by ρ Value (Convergent Trials Only)

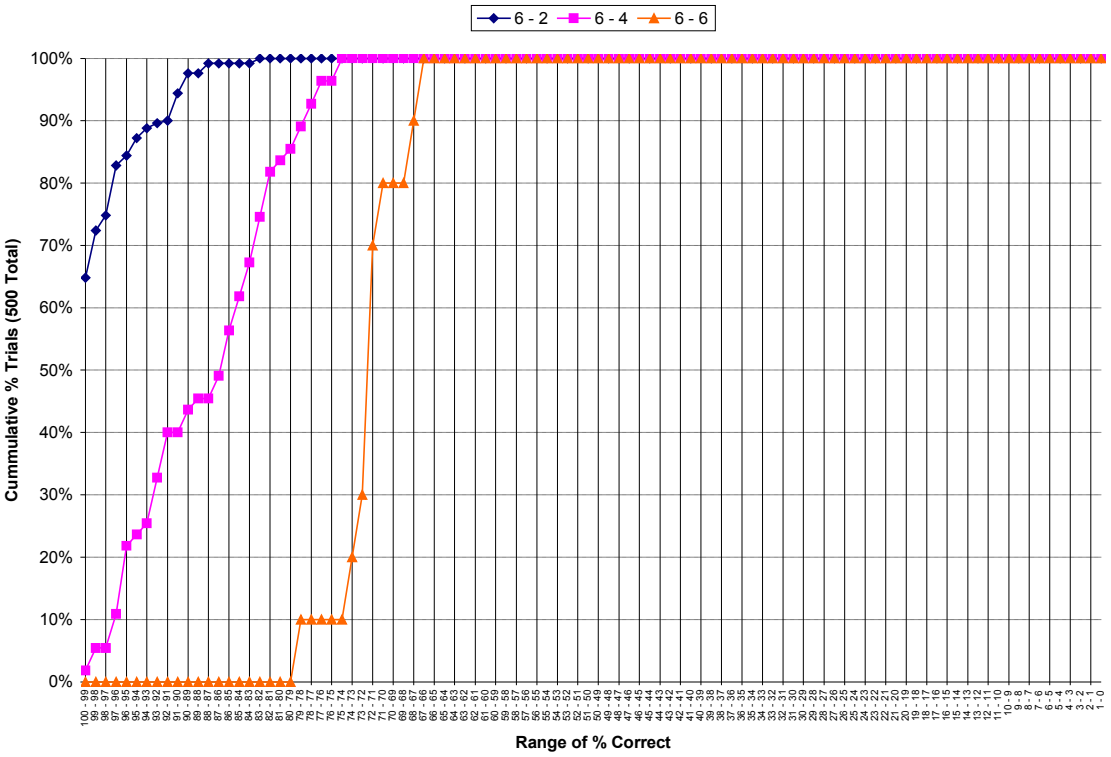

Figure D.6.19: Empirical CDF of Epochs for Hidden Layer Size 6 Networks by ρ Value (Convergent Trials Only)

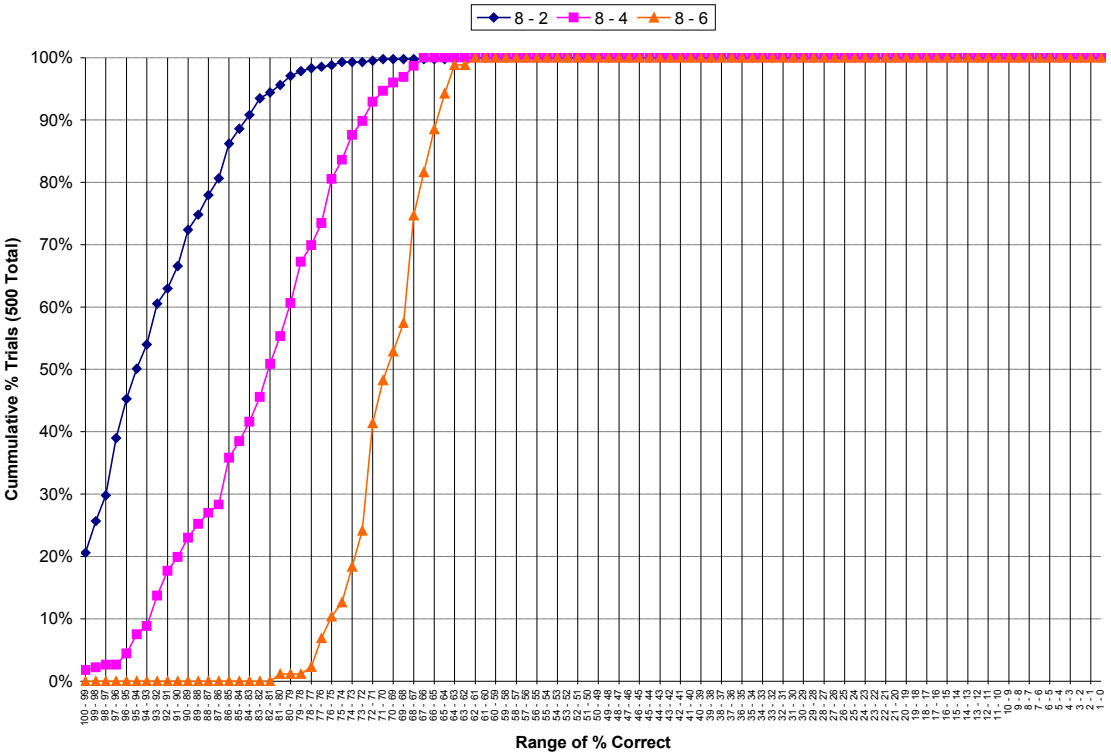

Figure D.6.20: Empirical CDF of Epochs for Hidden Layer Size 8 Networks by ρ Value (Convergent Trials Only)

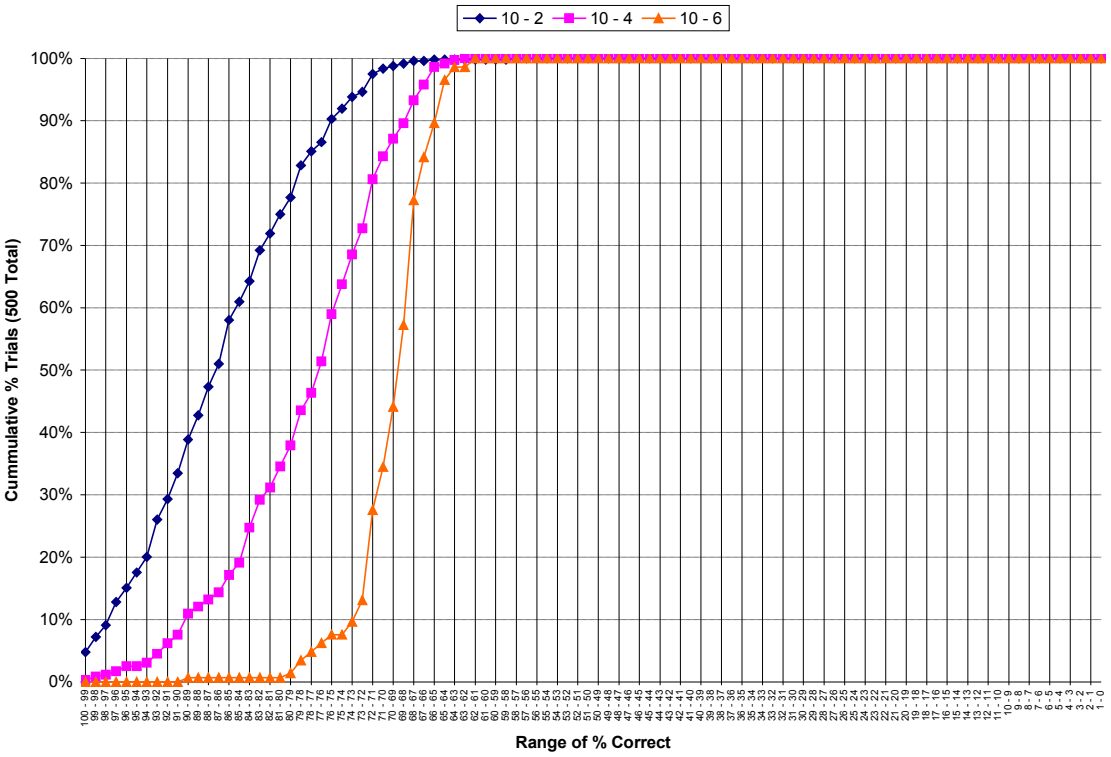

Figure D.6.21: Empirical CDF of Epochs for Hidden Layer Size 10 Networks by ρ Value (Convergent Trials Only)