A Natural Language Interface to the Unix Operating System Using Recurrent Neural Networks

A Thesis

Presented for the

Master of Science Degree

The University of Tennessee, Knoxville

Renwick L. Conrad

December 1995

Dedication

This thesis is dedicated to my parents, Mr. and Mrs. Lowell Conrad, for without their love and prayerful support this would have not been possible.

Acknowledgments

First and foremost I want to thank God for giving me the strength to finish this project and for affirming Philippians 4:13. “I can do all things through Christ which strengtheneth me” (KJV).

I also want to thank Dr. Bruce MacLennan for introducing me to neural networks and for the feedback and guidance he has given me throughout this project. I want to take this opportunity to thank the other members of my committee, Dr. David Straight and Dr. Bradley Vander Zanden for their participation on my committee and suggestions given on this project.

I would also like to thank Dian Witter for her friendship, prayers, emotional support and for the Thursday night dinners. Those evenings out meant more to me than you will ever know, Dian.

Finally, I would like to thank my parents, Lowell and Joy Conrad. Your prayers, love and support helped make this degree possible. Thank you Mom and Dad.

Abstract

This paper presents an experiment that was done in natural language processing using a recurrent neural network, specifically attempting to construct a natural language interface to the Unix operating system. Most natural language experimentation with neural networks has been done using a small vocabulary. The experiments in this paper were done to try to address what would happen using a larger vocabulary. Several different trials were run using binary inputs, bipolar inputs and a variety of different network architecture. When the input to the network was binary the network was able to correctly classify 85 percent of the sentences using the training set; when using bipolar input, the net was only able to classify 62 percent of the sentences correctly. Neither network did well with generalization.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

If you were to ask five different people to define the term **artificial intelligence** you might get as many definitions as the number of people you asked. Practical applications in the field of artificial intelligence (AI) have been at the center of controversy for quite some time. If by artificial intelligence we mean machine intelligence that such visionaries as Arthur C. Clarke, Gene Roddenberry, and Isaac Asimov have presented in their masterpieces of science fiction, the current state of AI is indeed primitive. If, however, we look at the field of AI as advanced computing then we can see that there are AI techniques in many commercial and industrial areas. Diane Crawford, executive editor for *Communications of the ACM*, in her editorial commentary for the March 1994 issue, has this to say about artificial intelligence applications:

> AI is everywhere. It streamlines manufacturing operations, propels cars and subway trains, sustains hospitals and schools, tracks financial markets, makes buildings "smart", and even runs some of our favorite household appliances [Cra94].

In order to stay competitive, implementation of correct AI techniques is often a must. In fact "truly profitable AI techniques are even kept secret [Mun94, p. 23]. It is also a field that is very active in the research and development area, and advances are being made almost daily in its various fields.

1.1 Approaches to AI

There are two major approaches to the field of artificial intelligence. The symbolic approach has been historically dominant and has sometimes been called the traditional approach to AI. This approach is "characterized by high levels of abstraction and a macroscopic view" [Mun94, p. 25]. Parallel distributed processing (PDP) has recently emerged as an alternative to the symbolic approach [Mii93, p. 17]. It is an approach that focuses on the subsymbolic rather than the symbolic. This approach models more closely the biological aspect of the world, similar in emphasis to the emphasis of physiology or genetics. Neural networks and genetic algorithms are excellent examples of this latter approach. Although the neural network models that we use seldom resemble their biological counterparts (nor were they meant to), they are inspired and motivated by the apparent distributed representations in the brain [Mii93] and they are an area from which many people expect to see significant practical applications in the near future.

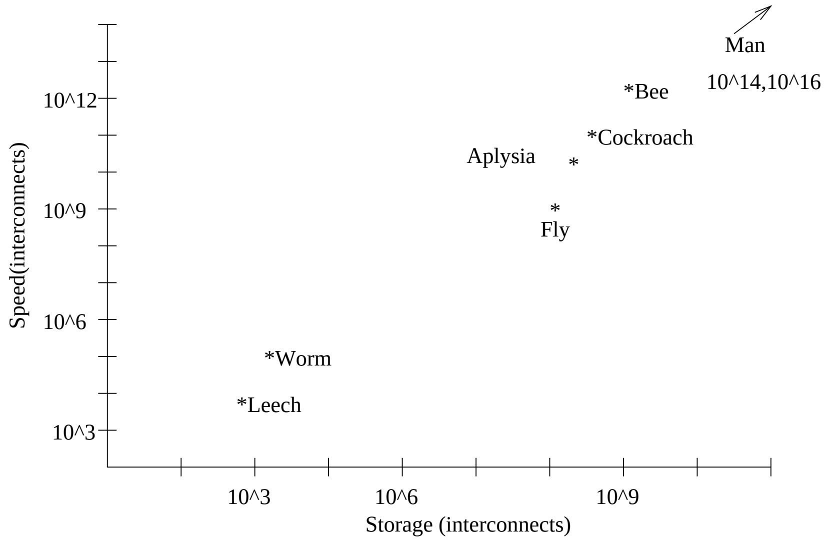
Neural networks go by many aliases. Parallel distributed processing is one that was just mentioned. Other aliases that are often heard for this type of network are connectivist/connectionist models, self-organizing systems, neurocomputing and neuromorphic systems [NI91]. It is realized that not all connectionist networks are parallel and that not all PDP models are created by using a neural network, but for the scope of this paper the terms can be thought of as interchangeable. The focus of this paper will be on this approach to the field of artificial intelligence. More specifically it will concentrate on neural networks in the field of Natural Language Processing.

What is it that makes Parallel Distributed Processing models so appealing? One reason is that they are more closely tied to the physiology of the brain than other kinds of information processing. But another, and maybe more important reason, is that they offer the possibility of giving us a computationally sufficient and psychologically accurate mechanistic account of human cognition, which has elusively avoided successful explanations in conventional computation formalisms. They have, however, "radically altered the way that we think about the time-course of processing, the nature of representation, and the mechanisms of learning" [RMH86a, p.11].

1.2 Overview of Connectionism

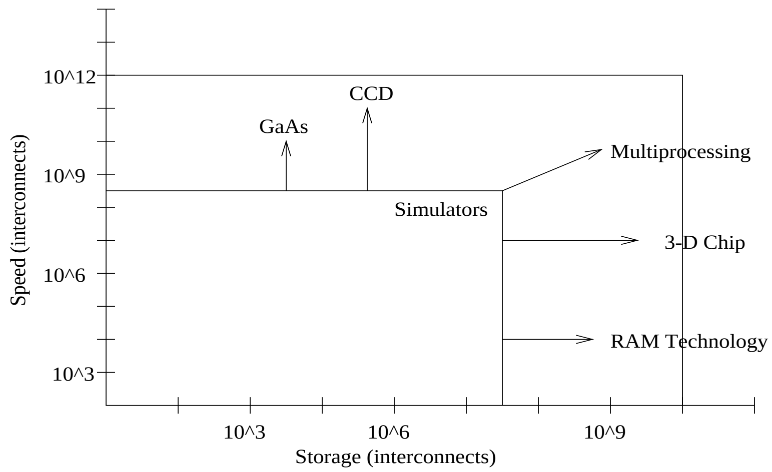
An artificial neural network is capable, after training, to address problems whose solutions have not been explicitly formulated. In contrast the computer systems that most people are familiar with are designed to solve problems, some of which are extremely complex, whose solutions have been made explicit.

Topologically, a neural network is "a logical (and sometimes physical) structure in which multiple nodes communicate with each other through synapses that interconnect them" [Che93, p. vii]. This structure imitates its biological counterpart in the nervous system with the nodes of a neural network corresponding to neurons and the connections between the nodes corresponding to the synaptic regions [Che93, p. 5]. The brain is estimated to contain 10^{11} neurons (comparable to the number of stars in the galaxy) and 10^{16} synaptic interconnections. Figures 1.1 and 1.2 show capability measures for biological and electronic neural networks.



This figure shows the estimated computational capabilities of various life forms in terms of speed and storage capabilities. Human capability is estimated at 10-^16 interconnects per second and storage of 10^14 interconnects (redrawn from a DARPA study [Che93, p. 7]).

Figure 1.1: Computational Capabilities (life forms)



This figure, when used in combination with Figure 1.1, shows that according to the estimates of the late 1980's that technology available to artifical neural networks makes it possible to surpass leeches and worms in storage and rate parameters. The inner area shows the past, established technology, while the outer contour shows the boundary toward which the technology is moving (Redrawn from a DARPA study [Che93, p. 7]).

Figure 1.2: Computational Capabilities (technology)

These figures show that artificial neural networks fall short of matching the capabilities of the human brain; it also shows that current technologies surpass the leech and the worm and are gaining on the computational ability of the cockroach [Che93, p.7].

1.3 Aspects of a PDP model

Rumelhart, Hinton and McClelland [RMH86b] have defined eight major aspects to a parallel distributed processing model:

- A *set of processing* units
- A *state of activation*
- A *output function* for each unit
- A *pattern of connectivity* among units
- A *propagation rule* for propagating patterns of activities through the network of connectivities
- An *activation rule* for combining the inputs impinging on a unit with the current state of that unit to produce a new level of activation for the unit.
- A *learning rule* whereby patterns of connectivity are modified by experience
- An *environment* within which the system must operate

We will look closer at these requirements in Chapter 2, when we will examine the specifics of how a neural network, or PDP model, learns.

1.4 Applications using Neural Networks

Neural Networks are seen in almost every area today. Though it is true that a good number of the applications are found in the academic world we also find Neural Networks in many other diverse areas too. Included here are just a few of the many places where Neural Networks are found.

1.4.1 Autonomous Vehicles

One area where Neural Networks are being used, is in vision based, autonomous driving. ALVINN, which stands for Autonomous Land Vehicle In a Neural Net, is part of such a system. It is a "fully connected three-layer backpropagation network, whose input is image information from a video camera and whose output is the vehicle heading required to stay on the road" [KRW94, p. 59]. ALVINN is trained as the vehicle is driven down the highway. Road images from the camera are the input to the system and the vehicle heading, as steered by the human driver, provides the feedback for the System. "The most recent experiment achieved autonomous driving at the speed of 55mph for a stretch of more than 90 miles on a highway near Pittsburgh" [KRW94, p. 60].

1.4.2 Deforestation Monitoring

Neural networks are also used in dealing with the problem of automating the interpretation of satellite images of the Amazon region for deforestation monitoring. The approach is based on a combination of image segmentation and classification techniques, the latter using a neural network architecture that works on a fuzzy model of classification. The architecture combines a relaxation technique on top of a feedforward neural network to take advantage of the interrelations among neighboring image segments [BMdSL94, p. 321].

1.4.3 Dog Races

A division manager at the MathWorks had an idea that pari-mutuel betting would be a good application for a neural network. He purchased old racing forms from one of the nearby dog tracks and programmed an OCR scanner to read the forms. Using this as input to a backpropagation neural network, he trained the net to predict the running time of the eight dogs in the race. He then figured out his standard error of estimate and used that to turn the eight times into odds estimates. In the long run, he ended up making some money at this venture.[1]

1.4.4 Pattern Classification

Credit Card Fraud

Since one area where neural networks shine is in the field of pattern classification, there are several banks and credit card companies that are using neural networks to assist in credit card fraud detection. The neural networks are used to study the patterns of credit card usage and are trained to detect the transactions that are potentially fraudulent. Some of the places are using software that was developed in-house while others are using commercial products developed by Nestor, HNC and other companies [WRL94, p. 94].

[1] Told by J. Michael Hammond of the University of Tennessee, Knoxville and also published as part of a marketing series "fun things that people do with MATLAB"

Handwriting Recognition

Nestor is the developer of NestorWriter, a handwriting recognizer partially based on a neural network pattern recognizer. Input is written on a digitized pad. After being trained on typical handwriting, NestorWriter can interpret handwriting it has not previously seen regardless of changes in size, style or distortions.

Another usage of pattern classification can be seen at Avon's order processing center and the state of Wyoming's Department of Revenue. They are using HNC's Quickstrokes Automated Data Entry Systems to recognize handwritten forms. Before using this system, it was estimated that Wyoming was losing $300,000 per year in interest income because so many checks were being deposited late. One software firm offers a product which uses Nestor's hand-printed character recognition system to convert a fax machine into an Optical Character Recognition (OCR) scanner [WRL94, p. 95].

Sonar Classification

One major concern of the United States Navy is keeping our costal waters and harbors secure. The ability to detect mines in relatively shallow water is essential. Paul Gorman of Bendix Aerospace has developed a neural network that has been trained to recognize underwater targets by sonar. The training set consisted of sonar "shots" taken by divers at each target from various angles. The network was able to correctly classify 90 percent of the targests correctly on a test set of data it had not seen. On the training data, it achieved a 99.8 percent accuracy. This is better than either a trained human listener or the traditional technique that is called a nearest neighbor classifier [NI91, pp. 8-9].

1.4.5 Applications in Natural Language Processing

DISCERN

There has been some excitement among cognitive scientests about the sybsymbolic approach to natural language processing. Part of the reason for this, is that the subsymbolic approach seems to resemble more closely how we humans process information such as learning by example, context sensitivity, generalization, and intuitive reasoning. Risto Miikkulainen at the University of Texas, Austin, has developed a connectionist natural language processing system that operates entirely at the subsymbolic level. This system uses the idea of scripts and tracks. A tract is an established minor deviation of a script (fast food vs. fancy restaurant). DISCERN reads the stories sequentially and forms an internal representation. Then, using questions posed to it as a cue, the appropriate story representation is retrieved from the episodic memory, where it was stored, and the answer is generated sequentially. DISCERN can also generate a full paraphrase of the story, filling in details that might have been left out of the original story but could be inferred. The inferences are statistical and learned from the number of similiar stories that it has read previously [Mii93].

NETtalk

NETtalk, developed in 1986 by Terrence Sejnowski of Johns Hopkins University and Charles Rosenberg a graduate assistant at Princeton, is a program that teaches itself to perform speech synthesis from written text. Using the text as input, the output codes are converted to sounds by using DECtalk, a complex rule-based expert system voice synthesizer. Initially, what the experimenters heard was like the babbling of a child that had not learned how to speak. It was producing sounds remarkably like the sounds of early speech in children. In later successive stages they heard the sounds of a child learning to speak and finally speaking clearly. The net was trained using not only backpropagation, but also through another paradigm — the Boltzmann machine [Che93, NI91, Sal89].

Learning Past Tense of Verbs

In 1986, Rumelhart and McClelland developed a network that transformed the phonetic representation of a verb in its root form, to the phonetic representation of its past tense form. Rumelhart and McClelland proposed that instead of an *explicit inaccessible rule* view, which states that the rules for language are stored explicitly but can only be understood by the language subsystem because they are written in a special code, the performance of a language system is characterizable by rules, but that the rules are not in any explicit form within the system.

It is interesting to note that the network shows the three same stages as that of a young child learning past tense English. In stage one, not many verbs are used in the past tense. Stage two finds children using more verbs in the past tense and a few irregular verbs. In this stage the child can generate a past tense for an invented word and they now incorrectly supply regular past-tense endings for words used correctly in stage one such as "comed" or "camed". This shows that the child has actually learned a linguistic rule even if it has never been explicitly stated. In stage three the regular and irregular forms coexist [RM86, Sal89].

1.4.6 Summary

These are just a very few of the applications where Neural Networks are found. Some other areas where Neural Networks are found are doing marketing analysis, airline seating allocation, semiconductor process control, loan approval, fighter flight and battle pattern guidance, speech recognition, biomedical applications and control of copiers [WRL94]. More applications for Neural Networks are discovered every day. It has been suggested that the inpact of neural network technology could be similar to the impact of the invention of the transistor. Robert White, when announcing the Draper Prize in 1989, said, "The development of the integrated circut was the single most important event that helped usher in the Information Age ... the creation of the integrated circut has fundamentally changed our lives [NI91, p. 271]. We will just have to wait and see if neural networks will have the same impact as the transistor but it is a definite possibility.

Chapter 2

How Neural Networks Learn

In chapter one, we examined the two main approaches to artificial intelligence: we were introduced to connectionism and the aspects of a PDP model, and examined several applications in the world of neural networks.

In this chapter we will examine, in a little more detail, the eight requirements, or attributes, that Rumelhart, Hinton and McClelland [RMH86b] set forth for a parallel distributed processing model. We will also study how a neural network learns and will examine in more depth a backpropagation network. We will also look at a special type of backpropagation network, the recurrent neural network.

2.1 Aspects of a PDP Model

The first attribute of a PDP model is that it consist of a set of processing units. In our diagram, Figure 2.1, these processing units are represented by circles. The job of each processing unit is to receive input from each of its neighbors to which it is connected. It then computes an output and sends that output to its neighbors. A system like this is inherently parallel because many units can carry out their computations at the same time. There are three types of processing units found in a neural network: input, output and hidden. Input units receive their inputs from sources external to the system while output units send their output to sources external to the system. The hidden units receive their input from sources inside the system and send their outputs to other locations within the system. These units are not "visible" outside the system.

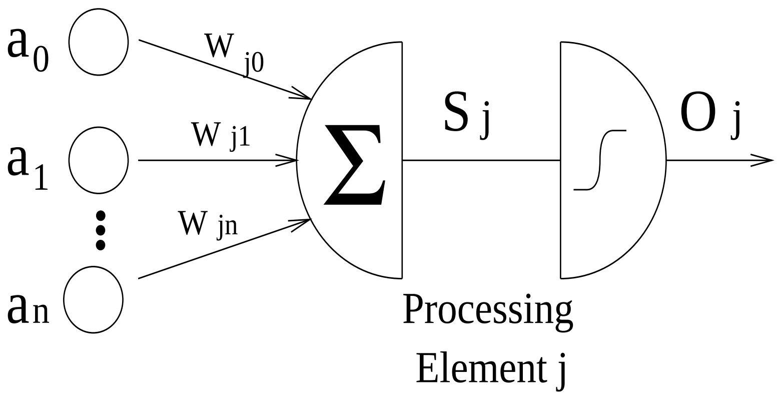


Figure 2.1: Basic Components of a PDP System

Activation Value At each point in time, each unit has an activation value, shown by S_j in Figure 2.1. It is the activation value over the set of all the units that captures what the system is representing at any given time. The activation value may be continuous or discrete. If the activation is continuous, it may be unbounded or bounded and if discrete, it may take on binary values or any of a small set of values. Most often, when the values are discrete they are binary values though in certain cases they will be bipolar values (-1, 1). The networks that were used in this project were simpler that what is described here. In our units the activation value was simply the sum of the inputs times their weights plus μ times the previous activation value.

Output The activation value is passed through a function in order to produce an output value (O_j) for each unit. It is in this way that the units interact. They transmit their signals to their neighbors. The degree in which they affect their neighbors is determined by the strength of the activation. In some models the output value is equal to the strength of the activation, more often, though, the activation is passed through some continuous or discontinuous threshold function so that a unit has no affect on other units unless its activation value surpasses a certain value.

Pattern of Connectivity The pattern of connectivity is very important to the system. It is this pattern that determines what each unit represents, what the system knows and how it will react to an arbitrary input. The output value from each unit is passed to the next unit in line through unidirectional connections. Associated with each connection is a real number that is called a weight. A positive weight indicates an excitatory connection while a negative weight indicates an inhibitory connection. One important issue that may decide how much information a given network can store and the amount of serial processing the network must perform is the fan-in, or the number of elements that either excite or inhibit a given unit, and the fan-out, which is the number of units that is directly affected by any given unit.

Propagation Rule The rule of propagation takes an output vector from a previous layer and combines it with the connectivity matrices, or weights connecting each unit, to produce a net input for each type of input into the unit. This rule is usually straightforward. One propagation rule that is used frequently is the sum of the outputs from the previous layer multiplied by the weights that lead to the unit referred to, $\sum W_{ji}O_i$. In figure 2.1, this is shown by the sigma (Σ) located in the first half circle of the processing element.

Activation Rule The activation rule is some function that takes the net input from the current time step t and combines the activation from the previous time step $t-1$, to produce a current activation for the processing element. The function can be deterministic or stochastic, though usually it is assumed that it is deterministic. It is sometimes useful to add the constraint that the function be a differentiable or semi linear function.

Learning Rule The learning rule is used to modify the patterns of connectivity as a function of experience. According to Rumelhart [RMH86b], this can involve three types of modifications:

1. The development of new connections.

2. The loss of existing connections.

3. The modification of the strengths of connections that already exist.

Very little work has been done on the first two, however they can be considered special cases of number three. If a weight is moved away from zero, this has the same effect as growing a new connection and if we change the weight to zero this is the same as losing a connection. Thus we will only look at rules that modify the strengths of the connections through experience.

Most of the learning rules are a variant of Hebbian Learning. The basic idea of Hebbian Learning is that "if a unit, u_i, receives a input from another unit, u_j; then if both are highly active, the weight w_{ij}, from u_j to u_i should be strengthened"[RMH86b, p. 53]. This idea has been modified and extended so it can be more generally stated as:

$$\Delta W_{ij} = g[S_i(t), t_i(t)]h[O_j(t), W_{ij}] \tag{2.1}$$

where $t_i(t)$ is some type of teaching input to u_i. This function says that the change in the weights from u_j to u_i is the product of the function $g()$ and $h()$, where $S_i(t)$ is the current activation level of u_i, O_j is the output of u_j and W_{ij} is the weight from u_j to u_i. In its simplest form the Hebbian rule can be stated as

$$\Delta W_{ij} = \eta S_i O_j \tag{2.2}$$

where η is the learning rate. In this form there is no teaching input and the functions are proportional to their first arguments. A third common variation of the learning rule is the Widrow–Hoff rule [RMH86b, p. 52] often called the delta rule. One form of the delta rule is

$$\Delta W_{ij} = \eta(t_i(t) - S_i(t))O_j \tag{2.3}$$

which is a generalization of the perceptron learning rule.

Environment In the development of any model we must have a clear model of the environment in which it will exist. For a PDP model, this environment is represented "as a time-varying stochastic function over the space of input patterns" [RMH86b, p. 53]. This environment is typically characterized by a stable probability distribution over the set of all input patterns independent of previous inputs and past responses of the system. Some PDP models are restricted in what they can learn. Some are able to respond only to input vectors that are linearly independent, others are only able to respond to orthogonal vectors while still others are able to learn to respond to what appears to be arbitrary patterns of input[RMH86b].

2.2 Basic Networks

The main distinction between network topologies are *feed forward* and *recurrent* networks. In the feed forward network data flows from input to output, through the various levels of the network. The flow of information goes in one direction and there are no connections from the outputs of units to the inputs of units in the same layer or previous layers [KvdS93]. The recurrent network, on the other hand, does contain connections of this type. We will look more in depth at the recurrent network in a little bit.

Like network topologies, the learning paradigms are divided into two distinct categories, *supervised* and *unsupervised* learning [NI91, KvdS93]. In supervised or associative learning, the network is trained by providing it with input patterns and associated desired output patterns. These pattern pairs can be presented to the network by an outside source or by the system that contains the network. With unsupervised learning, in contrast, the network is presented with a set of patterns but is not given a target answer for each category. It is the networks job to organize the training pattern into classes or clusters of patterns on its own. With this paradigm the network "is suppose to discover statistically salient features of the input population" [KvdS93, p. 18].

In order to understand better how a Neural Network learns, we will first examine the Perceptron before moving on to the Backpropagation Network and finally to

Recurrent Neural Networks.

2.2.1 Perceptrons

The perceptron, the earliest of the neural network paradigms, was developed in 1958 by Frank Rosenblatt, a psychologist turned neurophysicist. Rosenblatt believed that the perceptron should be able to handle noisy data and variable inputs, to self organize and to associate stimulus with response. The one drawback of the one layer perceptron is that the patterns that it is learning must to be linearly separable in order to be classified. This drawback did not allow the Perceptron to learn the XOR problem. In 1969, Marvin Minsky and Seymour Papert exploited this fact in their book *Perceptrons* and they charged that it could not solve any "interesting" problem. Their attack was "clinical and precise — but was undoubtedly overinterpreted in the field as being a nullification of all neural network approaches" [Che93, p. 25]. This caused research funds to be withdrawn and the study of Neural networks was severely limited until the early 80's [NI91].

A perceptron learns to classify patterns through the use of supervised learning. Usually the patterns it learns to classify are binary-valued (0/1) vectors and the output of the perceptron is usually a binary valued vector. It is limited to two layers of processing units with one layer of changeable weights between them. Though the Perceptron can have more than one layer of weights, only one layer is adaptable [Day90].

We can use Figure 2.1 as an example of a single element perceptron. This perceptron has N inputs (a_0 through a_n) and one output (O_j). The inputs arrive from the left and each incoming connection has a weight that is associated with it, W_{jn}. The processing element performs a weighted sum of its input values and takes the form

$$S_j = \sum_{i=0}^{n} a_i w_{ji} \tag{2.4}$$

where w_{ji} is the weight associated with the connection from processing element i to processing element j and a_i is the output of the input unit. Instead of the sigmoid that is shown in the second half of our figure, it is more common for a perceptron to use a threshold function such as

$$O_j = \begin{cases} 1 & \text{if } S_j > 0 \\ 0 & \text{if } S_j \leq 0 \end{cases} \tag{2.5}$$

to determine the output (O_j) of the processing unit.

The adaptable weights can be adjusted by any of numerous learning rules but using one of the simplest rules, the difference between the target, or desired output and the actual output of the net is computed first $(T_{jp} - O_{jp})$ where T_{jp} is the target output and O_{jp} is the actual output of the network. After this is computed, a constant is added or subtracted from the appropriate weights during learning

$$W_{ji_{new}} = W_{ji_{old}} + C(t_j - x_j)a_i \tag{2.6}$$

where C is a small constant or the learning rate, a_i is the value of input unit i and $(t_j - x_j)$ is the difference in the desired output and the actual output of the perceptron.

During training each member of the training set is presented to the network many times and upon each presentation the weights are adjusted. Each time the patterns are presented to the network the performance of the network should improve. Eventually the performance of the network stops improving and the network is said to have "converged". One measurement of a network's performance is using the root-mean-squares (RMS) method which is computed by the equation

$$\sqrt{\frac{\sum_p \sum_j (t_{jp} - x_{jp})^2}{n_p n_o}} \tag{2.7}$$

where n_p is the number of patterns in the training set, n_o is the number of units in the output layer, the first sum is taken over all the patterns in the training set and the second sum is taken over all the output units [Day90]. Another measurement is the total sum of squares (TSS), which is the measurement that was used during the experiments for this paper. The tss is computed by first computing the pattern sum of squares (PSS), which is equal to the squared error terms summed over all of the output units. Then this value is added to the tss which is, in fact, just the cumulative sum of the pss for all patterns processed within the current epoch, which is shown by the equation $\sum_p \sum_j (t_{jp} - x_{jp})^2$.

2.2.2 Backpropagation Networks

The chief functional limitation of the one layer perceptron is that it can only classify patterns that are linearly separable. The reason for this limitation is that the perceptron does not allow for more than one layer of adaptive weights. With more than one layer, it was not obvious how to assign credit (or blame) for the error for each of the hidden units and therefore difficult to know how to adjust the weights so the error would decrease.

Back Propagation, or backprop as it is often called, seems to be an algorithm "whose time had come. It arrived on the neural network scene at approximately the same time from several independent sources (Werbos; Parker; Rumelhart, Hinton, and

Williams)" [NI91, p. 121]. The backprop network is a feed forward network with supervised learning which "first adjust weights connected to the output layer. Then, working backward toward the input layer, ... adjust weights in each successive layer to reduce the errors at each level" [NI91, p. 109]. Figure 2.2 shows the structure of an n layer fully connected network. The Back Propagation algorithm belongs to a class

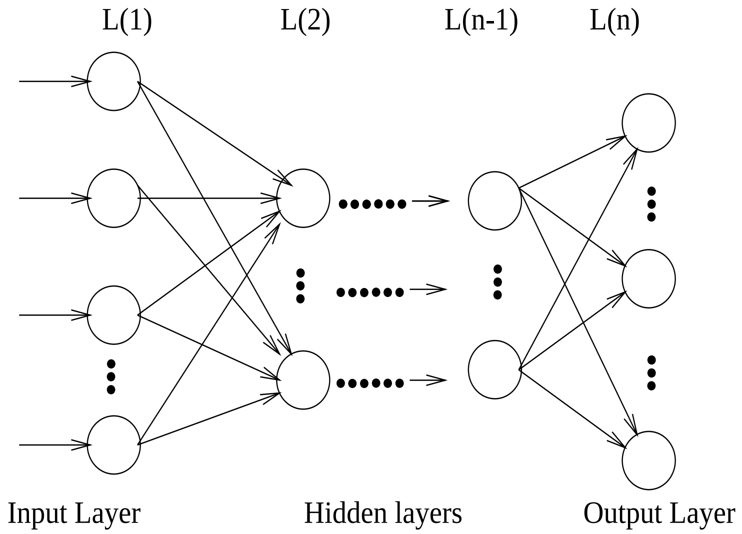


Figure 2.2: A Multi-Layer Feed Forward Network With n Layers

of algorithms that perform gradient decent, which is a descendant of the Widrow-Hoff or delta rule. For a more in depth study of this learning rule, it is suggested that the paper *An Analysis of the Delta Rule and the Learning of Statistical Associations* by G. O. Stone be read. This paper can be found in Volume 1 of *Parallel Distributed Processing* by Rumelhart, McClelland and the PDP Research Group, as Chapter 11.

The goal of the backprop algorithm is to adjust the weights of the connections in such a manner that the overall error of the network is reduced. As with the Perceptron, the root-mean-squared error is usually what is used to calculate how well the network is learning. The error surface can be thought of as looking like very rugged terrain, containing peaks, valleys, craters, ridges and slopes. This "terrain is the graph of the RMS value as a function of all of the weights in the network" [Day90, p. 72]. In essence the algorithm is an explorer looking, by the use of gradient decent, for a crater or minimum (ideally a global minimum) in this complex, multidimensional, but fixed error surface.

It is important to recognize that this algorithm does not always find a global minimum but may indeed get stuck in a local minimum that is far from the desired minimum. The appearance of a local minimum is not always the problem that it seems to be since backpropagation networks typically converge to a good RMS when the training examples are clearly distinguishable. There are, however, a number of techniques that can be used to assist the network in avoiding these local minima. The learning rate can be adjusted or the number of hidden units can be changed. Sometimes adding small random values to the weights can move the network from a local minimum to some nearby random point. If this new position is far enough away from the valley, then convergence may proceed in a new direction allowing the network to avoid that same local minimum [Day90].

During the training phase of a network the weights can be updated either after each pattern has been presented to the network, or after all patterns have been presented to the network, when the weights are updated according to the cumulative effect of all the vectors in the training set. This latter method, known as batch learning, approximates true gradient decent because each input pattern sees the same network state and hopefully takes the network directly down the slope to the global minimum. In the first approach each training pattern sees a slightly different state vector thus introducing a small amount of error. "However, if the learning rate constant is sufficiently small, the process will still approximate gradient descent" [Che93, p. 57].

2.2.3 Recurrent Neural Networks

A recurrent backpropagation network combines both feedforward and feedback mechanisms. There can be connections from processing units in one layer to processing units in a previous layer, to other units in the same layer or even to itself. These connections allow the network to compute complex time-dependent calculations. These networks were first studied by Hopfield and Tank on combinatorial optimization problems such as the traveling salesman problems [Sal89].

Figure 2.3 shows a sequential recurrent network which is very much like the network that was used for this paper. The memory units, sometimes called the current state units, receive their inputs from the hidden units. The input is given to this network sequentially, or one word at a time until the full sentence is presented to the network. When the first word is presented to the network, the memory units receive no input. As the second word, and each subsequent word, the memory units retain the previous state of the hidden units. "Actually, the activation of each current-state unit at the beginning of the next processing sweep is set equal to the activation of the corresponding output unit plus μ (mu) times the unit's previous activation" [MR88, pp. 156-157].

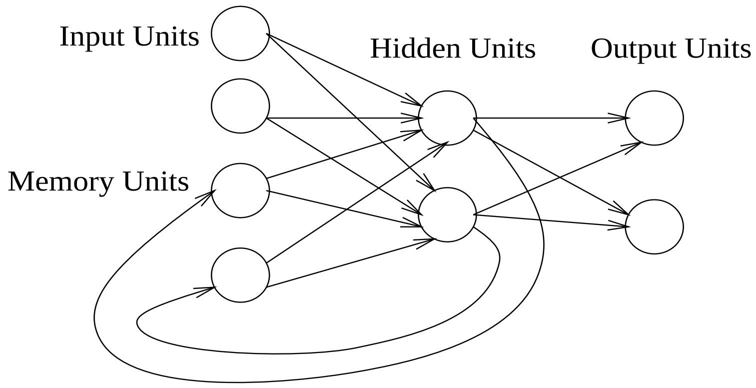


Figure 2.3: A Sequential Recurrent Network

2.3 Summary

In this section we have examined the aspects of a parallel distributed model as set forth by Rumelhart, Hinton and McClelland. We also looked at the basic workings of a neural network through the study of a Perceptron, before moving on to the Backpropagation network. Next we briefly looked at the main differences between the Backpropagation Network and the Sequential Recurrent Network. Next we will look at the actual experimentation that was done for this project and examine the results of that experimentation.

Chapter 3

Experimentation and Results

3.1 Setup of Project

One of our goals with this project was to work with a large vocabulary. In order to realize this goal, we needed to select a project that, by its very nature would result in a large vocabulary. In the computer industry there has been an increasing interest in vocal interfaces to computers. With the current level of technology, however, you still need to know exactly what command you want to give the computer in order to use the voice interface. The next logical step in the development of command interfaces, whether spoken or typed, would be a natural language interface where the user would be able to type, in plain English, what he or she wanted the computer to do, and the computer would be able to act upon that request. This, combined with my experience as a teaching assistant for the introduction to computer science class, prompted the idea of trying to develop a natural language interface to the Unix operating system.

One of the first things that needed to be done was to generate a data set, consisting of Unix commands and possible ways to represent them in everyday language. A file containing most of the standard Unix commands was located in our system and using that as a starting point I started writing sentences for each command. In order to have a wider variety of ideas for the commands I used several resources. I drew on my own knowledge of working with the introductory computer science students, I asked different people in the department how they might state different commands and I asked several of my friends on the Internet, some who did not know Unix, how they would go about trying to complete different tasks if they didn't know the exact command. A file containing approximately 10,000 sentence-command pairs was finally generated.

In order to simplify things somewhat, each Unix command is considered a string in and of itself. Rather than generating the command "man lpr" as two words, it appears in the file as "–man lpr" (the actual command is preceded by a dash). Also commands that can take variable arguments such as the print command is of the form "lpr <file1>", where <file1> is considered a variable that a rear end processor would be able to recognize and replace with the original filename that the user had typed. This part was not implemented in this project.

The next step in the project was to generate a unique random vector code for each word. We decided that a 100 bit vector would be used for this project. Looking ahead to future work, one thing that I did not want was for the system to crash just because a user happened to type in a legitimate word that I had not thought of when generating the sentences. I located a file that was from one of the spell checking programs that contained around 25,000 words and after combining that with my file of the commands I ended up with a file containing over $26,000$ words and symbols.

In order to generate the codes, I wrote a program that used a randomly generated binary tree to produce the vectors for each word in the data set. The program generated random values of zero and one and traversed the tree accordingly. When the last bit had been generated it checked to see if the leaf node had already been added to the tree. If it had, then that particular binary code was not a unique code, and a new code was produced for that particular word.

The work up to this point was implementation independent. The vectors for the words and the sentence–command pairs could be ported to any neural network program that we decided to use. The next step was to make a decision on which neural network package to use or if it would be necessary to write my own. After looking at several, including the DISCERN package and the Rumelhart and McClelland PDP software, we decided to go with the PDP software in particular the back propagation (bp) program.

The sequential recurrent form of the bp software was used for the experiments in this project. With this program the "memory units" are not differentiated from the input units in the network definition file and the feedback connections are specified as part of the pattern file. A negative input into an input unit, $-m$, specifies that that particular input unit is to receive its input from unit number m. When using sequential input, in order to keep the network from trying to learn until the whole sentence has been presented, a target value of -1 is specified in the pattern file. This tells the network to ignore the output of the net and proceed with the next pattern in the file. For a more complete explanation on this, see pp. 156-158 in

Explorations in Parallel Distributed Processing by J. L. McClelland and David E. Rumelhart. Appendix C lists the network definitions that were used in this project and Table 3.1 shows a summary of the networks.

Table 3.1: Networks Used for Project

NETWORK DEFINITIONS						
Network	Total Units	Input Units	Memory Units	Hidden Layers	Hid Units Per Layer	Starting Unit For Memory Input
Net 1	300	100	50	1	50	150
Net 2	350	100	50	2	50	150
Net 3	1000	100	400	1	400	500
Net 4	400	100	100	1	100	200
Net 5	600	100	200	1	200	300

3.2 Training

The training and testing phase of the project was lengthy and driven by problems that were encountered and by the attempt to find reasons or solutions to those problems. Most of the training of the neural networks was done on a Sun SPARCstation 5, Model 70. This is a 70 MHz machine with 32 MB of memory, rated at 100 MIPS and 13.1 MFLOPS. My first attempt at training was using Net 1 and the full data set. I quickly realized that using the full data set would not be possible at the start since the first epoch of training took over four hours to complete. I reduced the set to 500 sentence pairs and tried the training again. I was still learning the nuances of the bp program and learning how to capture the information that would be needed to see if the network was learning properly so I was unable to obtain any good results from the first experiments with this data set. I did, however, learn some valuable pointers that would assist me during the rest of the training.

I worked with three different networks, Net 1, Net2, and Net3 during this phase of the project. I learned that the larger the network and the larger the data set, the lower the learning rate needs to be. If the learning rate is too high, the network may start to converge but will then jump to a high tss value and plateau. Also, with this large of a data set, the learning rate needs to be adjusted downward every few hundred epochs as the tss can start to oscillate and even start to climb. One thing that would have made this project easier in terms of the training would have been a way to have the bp program check the tss value and if it was oscillating or had risen over a certain number of epochs of training, then the program could lower the learning rate.

I was gauging my results by the total sum of squares (tss) error since that is the error measure used by the PDP software program. Based on previous experience with neural networks, when the network's tss was below .1 this meant that the net had learned or nearly learned the problem. I was starting to see some convergence during training, but nothing that led me to believe that the network was anywhere close to learning the data set. The tss would start out around 40,000 and drop slightly, after a few hundred epochs, to around 29,000. Since there were still no discernible results, I decided to further reduce the training set to 100 sentence pairs. This did have the effect of having a lower tss, which was expected since the tss is the total error summed over all the patterns and if you have fewer patterns you have fewer error values to sum.

Using Net 2 I tried a comparison of bipolar and binary inputs. Net 2 had two hidden layers of fifty units each with the feedbacks from the first hidden layer to the memory layer. It is important to note, that even though I was using bipolar inputs into the network, the output was still in the binary form. This constraint was due to the way the bp program was originally designed. I tried several different runs with both the bipolar and the binary inputs, varying the learning rate and starting weights. The results from this set of trials did not show the results that I was hoping for since the lowest tss value was still up over two thousand.

I finally increased the net size once again to Net 4 and started three different tests using this net. Two of these tests were using bipolar inputs and the third was using binary inputs. All three trials were using the 100 sentence patterns. I achieved my best results from the trial that was using the binary input patterns. I started the learning rate of this trial at .01 for the first 1000 epochs. For the second 1000 epochs I lowered the learning rate to .001 and for the final 1000 epochs the learning rate was at .0001. At the end of 3000 epochs the tss value was 222.440. I decided to check the output on this network. The measure used to check the correctness of the networks throughout this project was to compare each bit string to its target. If a bit string had all 100 bits correct, then that output string was considered to be correct. At this time it appeared that this network was only producing 50 percent correct outputs on a bit by bit basis and not producing any correct output strings [1]. Figure 3.1 shows the Learning Curve of this network.

We considered several different solutions to this problem. Among them were to concentrate strictly on bipolar inputs, to reduce the size of the input patterns once again

[1] I later discovered that I had been looking at the output values of the last 50 hidden units and the first 50 output units instead of the 100 output units. This is the net that actually has an 85 percent accuracy rate on the output strings.

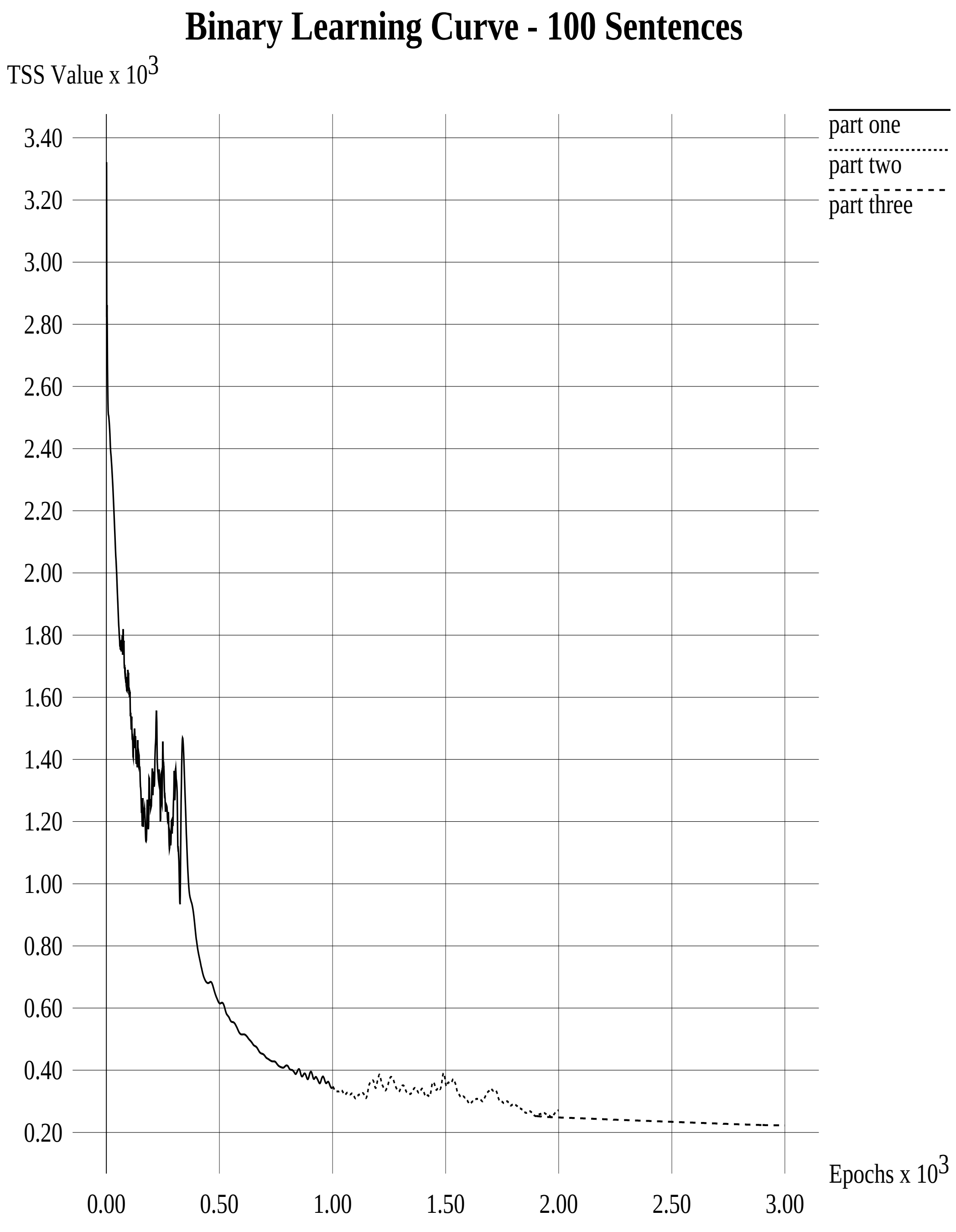


Figure 3.1: Learning Curve - Binary

and to change the architecture of the network. The first one that I did was to switch to bipolar inputs. After several trials and no results, I further reduced the number of patterns to 50 and ran several trials. I finally changed the network architecture to that of Net 5. After several more days of trying to find a good set of starting weights I had some success. It was also during this trial that I increased the mu value from zero to .5 to see if this would help in getting the network to converge. The mu value is used in calculating the output of the memory units and this change did seem to assist the network in reaching convergence since at the end of 900 epochs the tss value was 447.544. I ended up training this network for a total of 6,000 epochs in 500 and 1000 epoch increments, examining the graphs of the tss values, watching where the tss started to rise, then going back to weights saved previous to that point, lowering the learning rate and training again. I started the learning rate at .01 and the final learning rate was at .00001. Figure 3.2 shows the learning curve of the Bipolar Network through the 6000th epoch. Between epoch 5,000 and epoch 6,000 the tss was actually increasing slightly from 208.888 to 209.442.

3.3 Results

At this point, I decided to see what kind of results the network was achieving. The results were similar to the binary network, in that only about 50 percent of the bits were correct and the network was not getting any of the sentences correct. Thinking that the network should be getting better results than this, I went back and double checked everything. I discovered that the way I had my template file set up that I was using the activations from some of the units in the hidden layer and treating them as if they were the activations of the output layer. After correcting my mistake,

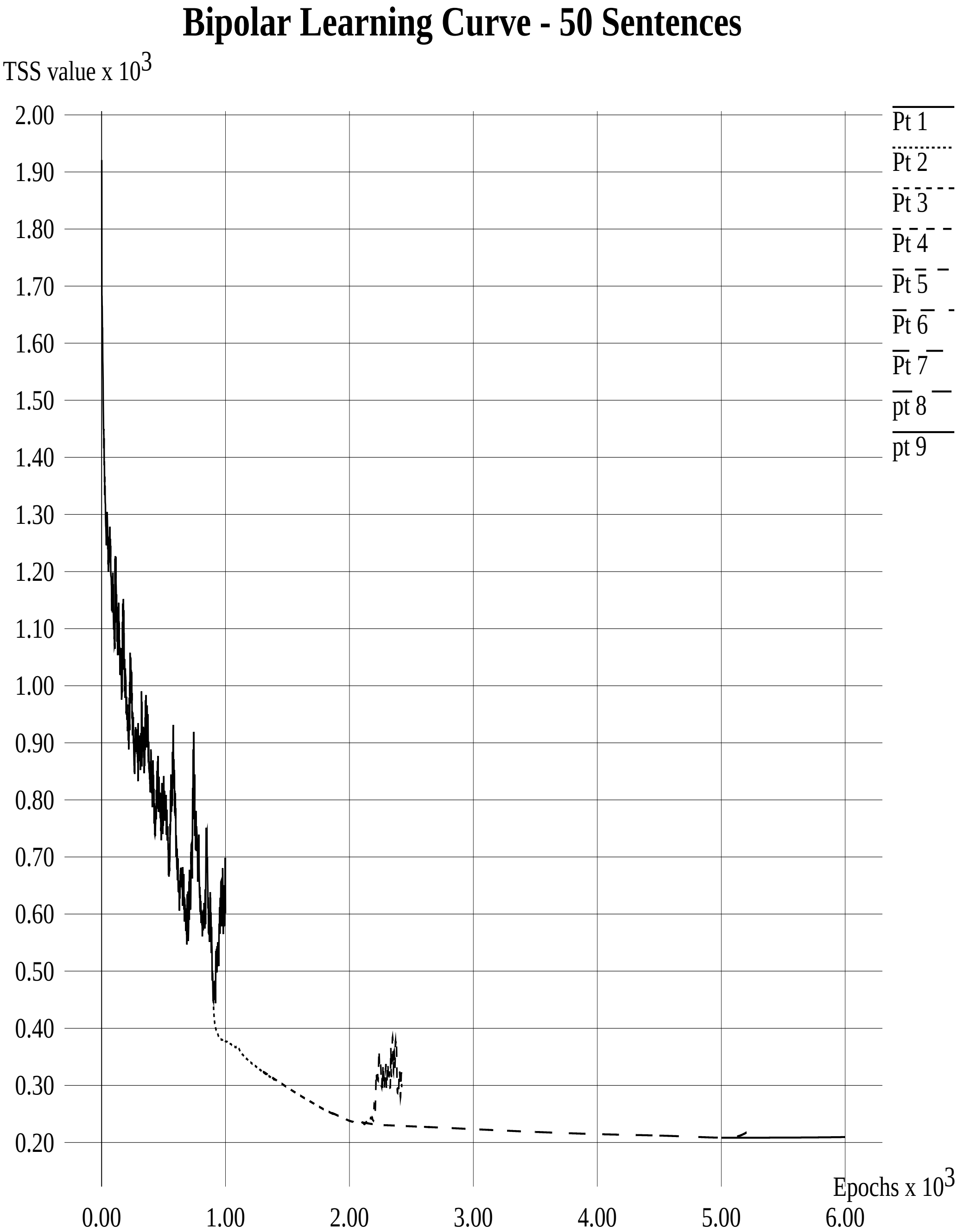


Figure 3.2: Learning Curve - Bipolar

I rechecked the output using the weights from epoch 5200 and discovered that the network was actually producing correct responses to 32 of the 50 sentences with an overall correct bit percentage of 98.28. I went back and checked the binary network and found that this network was producing 85 of the 100 sentences correct and an overall correct bit percentage of 99.82. A summary of the tss value, the percent of bits correct and the number of correct sentences for the binary data set and the same information for the bipolar data set is shown in Table 3.2 and Table 3.3. I did not include the information from the last training run of the network, labeled Part 9 in Figure 3.2, as the network was still only getting 31/50 sentences correct. It is interesting to note, that even though the tss value was increasing at 5200 epochs, the network got one more sentence correct than it did when the tss was at its lowest. Appendix B contains the sentences from each network that were missed.

3.4 Activation Tolerance

For both the bipolar and the binary nets, an activation of 0.5 or above was treated as a 1 and an activation below 0.5 was treated as a zero. Next, I wanted to determine how strongly activated the output units were. Whether they were closer to the midpoint of 0.5 or actually if they were closer to their respective target values of 0 or 1. I modified the program that took the activation values and adjusted them to either a zero or a 1 so a person could enter the upper and the lower bounds rather than using the defaults. Tables 3.4 through 3.7 show the results of this trial. The first two columns show the upper and the lower bounds that were used. The third column shows the number of bits that the network got correct. The fourth column, Number of Question Marks, are the the number of bits that fell between the upper and the

Table 3.2: Partial Training Record - Binary

PARTIAL TRAINING RECORD - Binary				
Number of Epochs	**TSS Value**	**% Bits Correct**	**Number of Sentences Correct/100**	**Number of Wrong Bits**
2000	247.920	99.72	77	28
2100	245.285	99.73	76	27
2200	242.323	99.73	77	27
2300	239.433	99.74	79	26
2400	236.731	99.77	83	23
2500	233.983	99.77	83	23
2600	231.237	99.80	83	20
2700	228.571	99.81	83	19
2800	225.921	99.82	84	18
2900	223.650	99.82	85	18
3000	222.440	99.82	85	18

Table 3.3: Partial Training Record - Bipolar

PARTIAL TRAINING RECORD - Bipolar				
Number of Epochs	**TSS Value**	**% Bits Correct**	**Number of Sentences Correct/50**	**Number of Wrong Bits**
3000	223.249	98.28	30	86
3100	222.227	98.32	30	84
3200	221.216	98.32	30	84
3300	220.204	98.32	31	84
3400	219.260	98.34	31	83
3500	218.332	98.34	31	83
3600	217.524	98.36	31	82
3700	216.684	98.38	31	81
3800	215.903	98.40	31	80
3900	215.185	98.42	31	79
4000	214.535	98.42	31	79
4100	213.976	98.46	31	77
4200	213.493	98.46	31	77
4300	213.036	98.46	31	77
4400	212.535	98.46	31	77
4500	211.994	98.48	31	76
4600	211.311	98.50	31	75
4700	210.523	98.48	31	76
4800	209.670	98.50	31	75
4900	208.839	98.50	31	75
5000	208.290	98.48	31	76
5100	209.249	98.40	31	80
5200	217.025	98.28	32	86

Table 3.4: Activation Tolerance using Training Set (Bipolar) - 3,300 Epochs

ACTIVATION TOLERANCE Bipolar Data Set – 3,300 Epochs					
Upper Bounds	Lower Bounds	Number of Bits Incorrect	Number of Question Marks	Percent Bits Correct	Percent Sentences Correct
>= 0.50	< 0.50	84	0000	98.32	62
0.55	0.45	61	0051	97.76	58
0.60	0.40	42	0124	96.68	50
0.65	0.35	26	0206	95.36	42
0.70	0.30	21	0298	93.62	32
0.75	0.25	13	0439	90.96	22
0.80	0.20	12	0637	87.02	12
0.85	0.15	09	0952	80.78	06
0.90	0.10	05	1303	73.84	00
0.95	0.05	03	1769	64.56	00

Table 3.5: Activation Tolerance using Training Set (Bipolar) - 5,200 Epochs

ACTIVATION TOLERANCE Bipolar Data Set – 5,200 Epochs					
Upper Bounds	Lower Bounds	Number of Bits Incorrect	Number of Question Marks	Percent Bits Correct	Percent Sentences Correct
>= 0.50	< 0.50	86	0000	98.28	64
0.55	0.45	58	0051	97.82	60
0.60	0.40	40	0107	97.06	54
0.65	0.35	32	0194	95.48	42
0.70	0.30	20	0302	93.56	30
0.75	0.25	13	0444	90.86	20
0.80	0.20	09	0628	87.26	12
0.85	0.15	07	0911	81.64	04
0.90	0.10	06	1274	74.40	00
0.95	0.05	04	1736	65.20	00

Table 3.6: Activation Tolerance using Training Set - 2,400 Epochs

ACTIVATION TOLERANCE Binary Data Set – 2,400 Epochs					
Upper Bounds	Lower Bounds	Number of Bits Incorrect	Number of Question Marks	Percent Bits Correct	Percent Sentences Correct
>= 0.50	< 0.50	23	0000	99.77	83
0.55	0.45	18	0019	99.63	73
0.60	0.40	14	0056	99.30	61
0.65	0.35	13	0126	98.61	43
0.70	0.30	13	0233	97.54	24
0.75	0.25	12	0457	95.31	14
0.80	0.20	13	0531	94.56	04
0.85	0.15	08	1383	86.09	00
0.90	0.10	07	2196	77.97	00
0.95	0.05	07	3291	67.02	00

Table 3.7: Activation Tolerance using Training Set - 3,000 Epochs

ACTIVATION TOLERANCE Binary Data Set – 3,000 Epochs					
Upper Bounds	Lower Bounds	Number of Bits Incorrect	Number of Question Marks	Percent Bits Correct	Percent Sentences Correct
>= 0.50	< 0.50	18	0000	99.82	85
0.55	0.45	15	0011	99.74	81
0.60	0.40	14	0038	99.48	66
0.65	0.35	12	0103	98.85	50
0.70	0.30	11	0220	97.69	29
0.75	0.25	10	0438	95.52	13
0.80	0.20	10	0788	92.02	06
0.85	0.15	09	1349	86.42	00
0.90	0.10	07	2144	78.49	00
0.95	0.05	07	3230	67.63	00

lower bounds, bits that could not be translated into either a one or a zero. The last two columns show the percent of bits and the percent of sentences that the network got correct. Figure 3.3 is a graph of the percentage of sentences that the network correctly identified. The x–axis is the percent of sentences correct and the y–axis shows the gap between the upper and the lower bounds.

3.5 Generalization

One thing that we wanted to check on was how well these networks generalized, since that it is one of the benefits of using a neural network over a program or network using conventional means. A small program was written that would randomly add or delete a word, switch around two words that were in the query, or replace one word with a different word. The program was set up so the experimenter could enter the random number seed for the random number generator, enter the percent chance per sentence that a change would take place, enter the file that contained a list of legal words to be used for the replacement or the addition of words, and enter whether the experimenter wanted to perform specifically one of the above choices, or let the program randomly choose which of the above actions to initiate on the chosen sentence.

Tables 3.8 through 3.11 show the results of this experiment. This first column shows the action that was taken, the second column shows the number of sentences that the program reported that it changed, and the last three show the percent of bits that was correct, the number of sentences correct and the percent of sentences correct. It is important to note that although it appears that the networks were able to correctly generalize on one or more of the sentences in any given trial, examination of the actual sentences that were used as input into the network, shows that the sentence had not

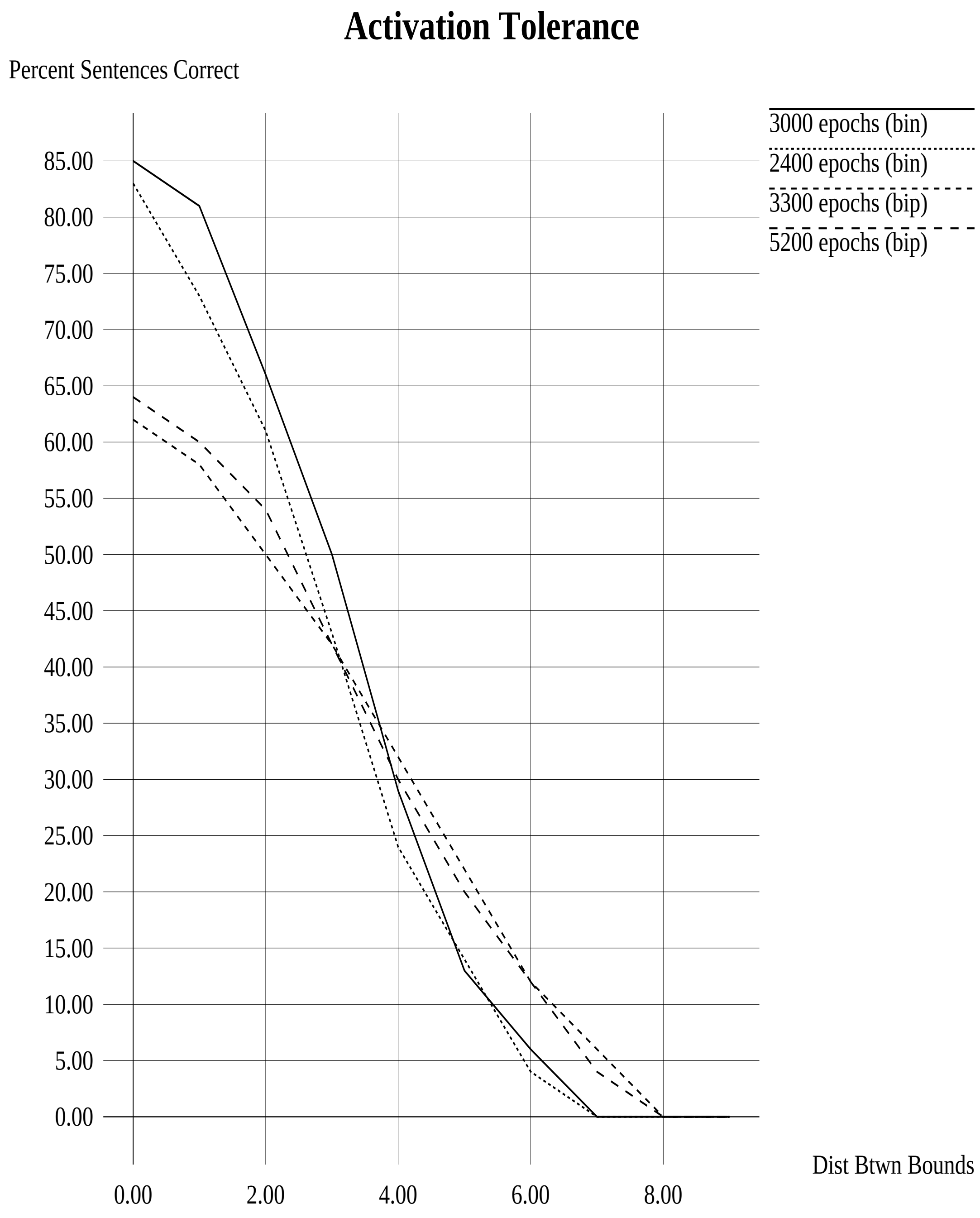


Figure 3.3: Activation Tolerance

Table 3.8: Random Changes to Sentences – 3,000 Epochs – Binary

Error Tolerance in Sentences 85 Sentence Set – 3,000 Epochs					
Action Taken	Number of Sentences Changed	Number of Bits Wrong	Percent of Bits Correct	Number of Sentences Correct	Percent of Sentences Correct
Add Word	8	130.00	98.47	77	90.59
Delete Word	8	159.00	98.13	77	90.59
Change Word	9	114.00	98.66	76	89.41
Switch Word	9	163.00	98.08	77	90.59
Random Change	9	182.00	97.86	76	89.41
Control Set	0	0.00	100.00	85	100.00

Table 3.9: Random Changes to Sentences – 2,400 epochs – Binary

Error Tolerance in Sentences 83 Sentence Set – 2,400 epochs					
Action Taken	Number of Sentences Changed	Number of Bits Wrong	Percent of Bits Correct	Number of Sentences Correct	Percent of Sentences Correct
Add Word	10	178.00	97.85	73	87.95
Delete Word	10	210.00	97.47	73	87.95
Change Word	9	113.00	98.63	75	90.36
Switch Word	9(6)	110.00	98.67	77	92.77
Random Change	8	183.00	97.79	75	90.36
Control Set	0	0.00	100.00	83	100.00

Table 3.10: Random Changes to Sentences – 5,200 Epochs – Bipolar

Error Tolerance in Sentences 32 Sentence Set – 5,200 Epochs					
Action Taken	Number of Sentences Changed	Number of Bits Wrong	Percent of Bits Correct	Number of Sentences Correct	Percent of Sentences Correct
Add Word	2	23.00	99.28	30	96.875
Delete Word	2	34.00	99.94	30	96.875
Change Word	3	38.00	98.81	29	90.625
Switch Word	3	40.00	98.75	30	96.875
Random Change	3	26.00	99.19	29	90.625
Control Set	0	0.00	100.00	32	100.00

Table 3.11: Random Changes to Sentences – 3,300 Epochs – Bipolar

Error Tolerance in Sentences 31 Sentence Set – 3,300 Epochs					
Action Taken	Number of Sentences Changed	Number of Bits Wrong	Percent of Bits Correct	Number of Sentences Correct	Percent of Sentences Correct
Add Word	2	35.00	98.87	29	93.548
Delete Word	2	53.00	98.29	29	93.548
Change Word	3	49.00	98.41	28	90.322
Switch Word	3	33.00	98.94	30	96.774
Random Change	3	39.00	98.74	28	90.322
Control Set	0	0.00	100.00	31	100.00

been changed from its original form. This was either due to replacing a word with itself or in the case of switching two words, the word was apparently switched with itself. If time allowed these experiments would be repeated and the cases just mentioned would be disallowed. It is unknown why the network was unable to generalize. It could be due to overtraining and that the network has actually memorized the responses to the given queries or another possibility is that the architecture used for the networks are not large enough for this problem set. Further experimentation in this area using different architectures might be able to answer this question.

3.6 Similarity Measure

Examining the results of the experimentations left us with several questions, one of which was why did the trials with the Binary input get better results than the trials with the Bipolar inputs.

In order to try to answer this question, I looked at the comparison of the absolute values of the dot products of the bipolar data sets and the values of the binary data sets. The reason for examining the dot products is that a neural network computes the dot product between the input vector and the weight vector entering a specific neuron. If two input vectors are orthogonal their dot products will be zero, while if they point in the same direction then their dot products, for the bipolar data set will be 100 while the binary data set could have a dot product of any value between 0 and 100. Therefore with orthogonal vectors this gives the neural network the ability to do an equal, not equal testing. Since the neural network is searching, or trying to match a specific output vector to a specific input vector, or in the case of sequential input a series of input vectors, examining the dot products of the words that were used in the input and output vectors tells us how similar the words are to each other. If the words are too similar the neural network will have a harder time differentiating between them and therefore will not learn the desired function as easily, if at all.

Table 3.12 shows the comparison of the average dot products for both the bipolar and the binary input data. The first two columns are the values for every word in the original data set while the second two columns are the values for just the 245 words that were used in the experimentations done for this paper. The first and third column include the dot products of the words with themselves while the second and fourth column did not include those values. Since each vector in the input set contained 100 bits, the dot product of the bipolar inputs with themselves would equal 100. Appendix E contains the 245 words and the value of the word when dotted with itself. Figures 3.4 through 3.6 are graphic representations of the frequency of the various dot products from the 245 word data set.

Table 3.12: Comparison of binary and bipolar dot products and average arccos

	BIPOLAR (Absolute Values)			
	All Words		Partial Words	
	Self	**Not Self**	**Self**	**Not Self**
Max Abs Dot Prod	100.0	52.0	100.0	48.0
Min Abs Dot Prod	0.0	0.0	0.0	0.0
Avg Abs Dot Prod	8.020936	7.960264	8.346322	7.970693
Avg Arc Cos	1.553733	1.552766	1.570151	1.563776

	BINARY			
	All Words		Partial Words	
	Self	**Not Self**	**Self**	**Not Self**
Max Abs Dot Prod	66.0	47.0	63.0	42.0
Min Abs Dot Prod	7.0	7.0	9.0	9.0
Avg Abs Dot Prod	25.288298	25.271816	24.812778	24.710739
Avg Arc Cos	1.553512	1.552523	1.568912	1.562518

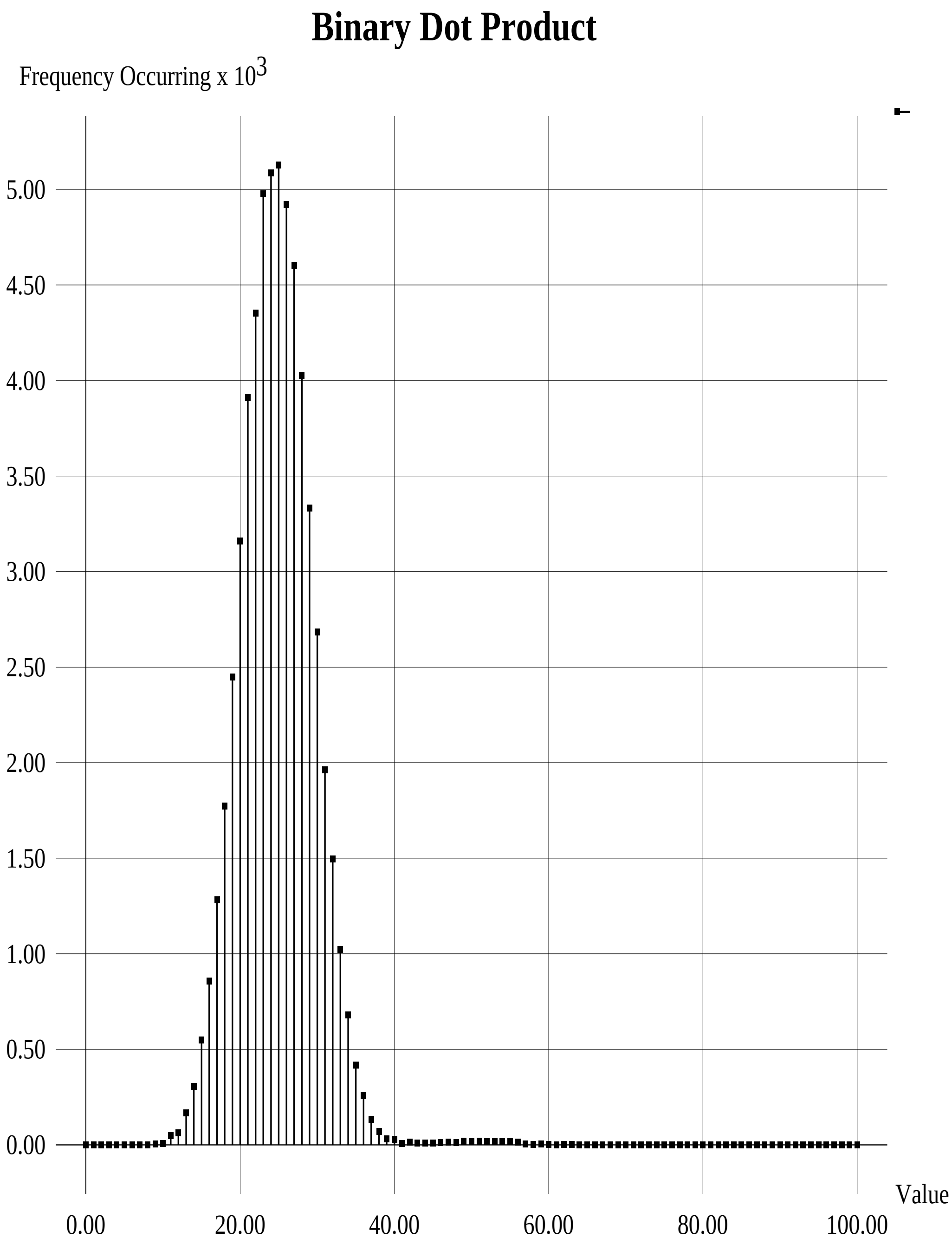


Figure 3.4: Dot Products of the Binary Data Set

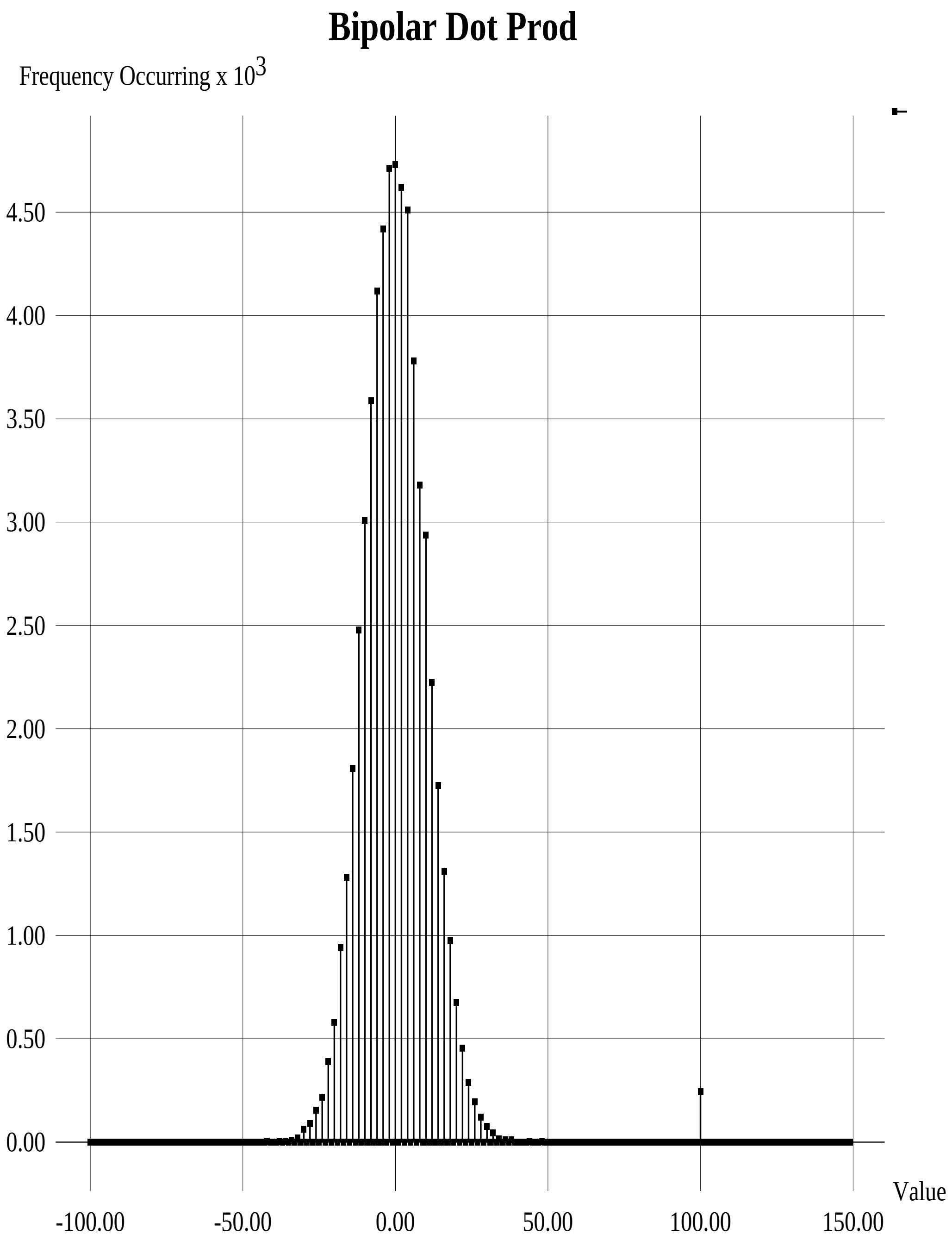


Figure 3.5: Dot Products of the Bipolar Data Set

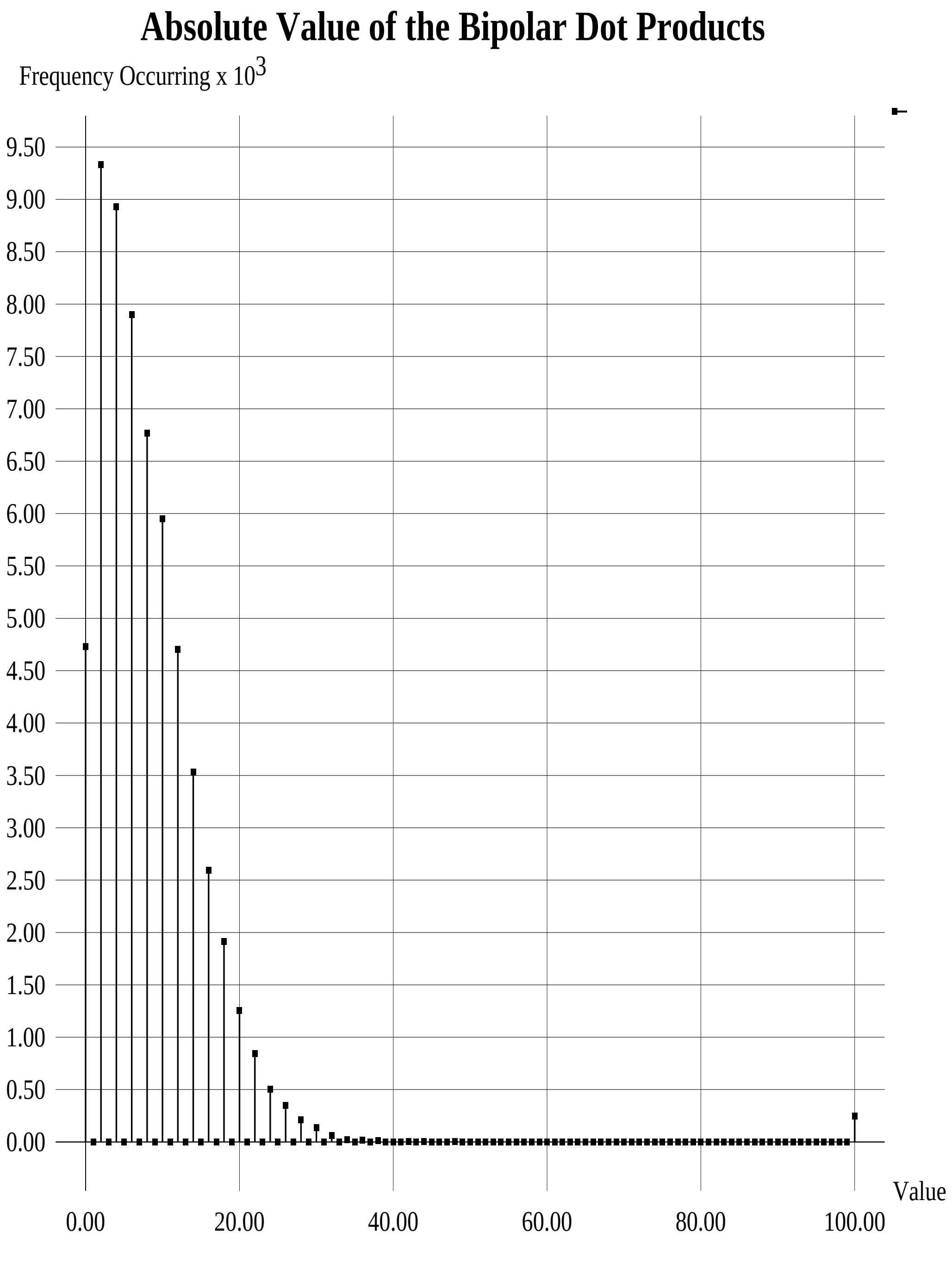


Figure 3.6: Absolute Value of the Dot Products of the Bipolar Data Set

3.7 Summary

In this chapter I have presented an outline of what I was trying to accomplish with this project, the steps taken during the implementation of the project and a summary of the results of the project.

I started by developing a database of sentences that a person who did not know the Unix command set might try to use to get a specific task accomplished, such as printing a file. Then, taking the words that were generated from this process and combining them with a dictionary file from one of the dictionary programs, I generated a 100 bit code for each word using a binary tree structure.

After deciding to use the recurrent backpropagation program from the PDP Software by Rumelhart and McClelland, the next step was to try to find a correct combination of architecture, starting weights, and learning rate to successfully train the network on the data set being used. This phase of the project was driven by problems that we encountered during the training and by trying to find solutions or reasons for each of these problems.

I then presented the results from this project. This project has presented more questions than answers or results, but it is a good foundation for someone to pick up where I left off and continue the work.

Chapter 4

Future Work

There is a great need for better understanding of how neural networks actually work. Currently the design of a neural network for any given problem is a combination of trail and error and some guesswork based on past experience with neural networks. This project is an exemplification of this statement.

During the course of working on this project, several things have come to mind that might be implemented at a future date on this project. One thing that would speed up the training of the network would be the ability to train in parallel. Using a neural network with an adaptive learning rate would save many wasted epochs of training and wasted time in having to back up and restart the training, after lowering the learning rate, to the point where the tss started to rise. Something else that could be tried would be to use a different learning algorithm, such as Recurrent Cascade–Coorilation [Fah91].

A different measure of "correctness" might also be tried at a future date. Instead of comparing the outputs to their targets on a bit by bit basis as was done in this paper, you could take the output code and compare it to the word codes and select the word code that is nearest to the output code and then ask if this is correct.

There is also the need to address the possibility of certain words in the data set causing problems in being too similar to other words so the network has a difficult time in differentiating between them. In this case we can go back and regenerate a new code for that particular word and see if this makes a difference in the ability of the network to learn.

Bibliography

Bibliography

[BMdSL94] Valmir C. Barbosa, Ricardo J. Machado, and Frederico dos S. Liporace. A neural system for deforestation monitoring on landsat images of the amazon region. *International Journal of Approximate Reasoning*, 11(4):321–359, November 1994.

[Che93] Michael Chester. *Neural Networks A Tutorial.* Prentice Hall, 1993.

[Cra94] Diane Crawford. Editorial pointers. *Communications of the ACM*, 37(3):5, March 1994.

[Day90] Judith E. Dayhoff. *Neural Network Architectures.* Van Nostrand Reinhold, 115 Fifth Avenue New York, New York 10003, 1990. ISBN 0-442-20744-1.

[Fah91] Scott E. Fahlman. The recurrent cascade-coorelation architecture. availabe by anonymous ftp: archive.cis.ohio-state.edu/pub/neuroprose/, May 1991. CMU-CS-91-100.

[KRW94] Takeo Kande, Michael L. Reed, and Lee E. Weiss. New technologies and applications in robotics. *Communications of the ACM*, 37(3):58–67, March 1994.

[KvdS93] Ben J. A. Krose and P. Patrick van der Smagt. *An Introduction To Neural Networks*. University of Amsterdam, The Netherlands, January 1993.

[Mac93] Bruce MacLennan. Characteristics of connectionist knowledge representation. *Information Sciences*, 70:119–143, 1993.

[Mii93] Risto Miikulainen. *Subsymbolic Natural Language Processing: An Integrated Model of Scripts, Lexicon, and Memory*. The MIT Press, Cambridge, Massachusetts, 1993. ISBN 0-262-13290-7.

[MR88] James L. McClelland and David E. Rumelhart. *Explorations in Parallel Distributed Processing: A Handbook of Models, Programs, and Exercises*. MIT, Press, Cambridge, Massachusetts, 1988. ISBN -0262-63113-X.

[Mun94] Toshinori Munakata. Commercial and industrial AI. *Communications of the ACM*, 37(3):23–25, March 1994.

[NI91] Marilyn McCord Nelson and W. T. Illingworth. *A Practical Guide to Neural Nets*. Addison-Westly Publishing Company, Inc., Route 128 Reading, Massachusetts, March 1991.

[RM86] David E. Rumelhart and J. L. McClelland. On learning the past tenses of english verbs. In *Parallel Distributed Processing: Exploriatons in the Microstructure of Cognition*, volume 2, chapter 18, pages 216–271. MIT, Press, Cambridge, Massachusetts, 1986.

[RMH86a] David E. Rumelhart, J. L. McClelland, and G. E. Hinton. The appeal of parallel distributed processing. In *Parallel Distributed Processing: Exploriatons in the Micros tructure of Cognition*, volume 1, chapter 1, pages 3–44. MIT, Press, Cambridge, Massachusetts, 1986.

[RMH86b] David E. Rumelhart, J. L. McClelland, and G. E. Hinton. A framework for PDP. In *Parallel Distributed Processing: Exploriatons in the Microstructure of Cognition*, volume 1, chapter 2, pages 45–76. MIT, Press, Cambridge, Massachusetts, 1986.

[Sal89] Fermin Soriano Salkjelsvik. Natural language processing with recurrent neural networks. Masters thesis, University of Tennessee, Knoxville, Computer Science Department, December 1989.

[WRL94] Bernard Widrow, David E. Rumelhart, and Michael A. Lehr. Neural networks: Applications in industry, business and science. *Communications of the ACM*, 37(3):93–105, March 1994.

Appendices

Appendix A

Glossary

connectionism – The use of neural networks for knowledge representation and inference [Mac93, p. 119].

epoch – A training cycle where all patterns of the training set are presented and the weights are adjusted.

hidden units – The units in a neural network that are neither input units or output units. They receive their input from input units or other hidden units.

input units – The units that receive their input pattern from a source outside the network.

input pattern – An n bit vector that encodes the input sentence, where n is the number of bits needed to encode the sentence.

learning rate – In the back propagation algorithm, the learning involves a change of the weight proportional to the weight error derivative. The learning rate is the constant of proportionality.

memory units – Units that contain feedback connections from other units within the network. The activations of these units represent the prior history of activations within the network.

momentum – A parameter that determines the effect of past weight changes on the current direction of movement in the weight space.

μ **(mu)** – A parameter that determines the effect of past activations on the memory units.

novel set – A set of input and target patterns that have not been used during the training of the network.

output pattern – An n bit vector that encodes the output produced by the network, where n is the number of bits needed to encode the output pattern.

output units – The units whose activations represent the output pattern.

pattern sum of squares (PSS) – The squared error terms summed over all the output units.

script – A stereotypic sequence of events one might expect to encounter when engaging in such activities as visiting a restaurant, traveling by airplane, or shopping at a supermarket.

target pattern – An n bit vector that represents the correct output pattern, where n is the number of bits needed to encode the output pattern.

total sum of squares (TSS) – The cumulative sum of the PSS for all patterns thus far processed within the current epoch.

training set[1] – (sometimes called the **learning set**) A set of input and target patterns that is used during the training of the network.

[1] This Glossary, with minor modifications, was taken from Fermin Salkjelsvik's Masters Thesis, Chapter 3, Section 3.1 pp. 11-12[Sal89]

Appendix B

Incorrect Sentences

The incorrect sentences from the 5,200 epoch Bipolar training set.

1. what does defaults_to_indentpro mean	–man defaults_to_indentpro
8. show me help on unalias	–man unalias
16 Please give me help with paxcpio	–man paxcpio
18 Please give me help with sortm	–man sortm
19 show help on toolplaces	–man toolplaces
23 help using rmdel	–man rmdel
28 help with bc	–man bc
32 help with ap	–man ap
35 show help on unload	–man unload
37 help using roffbib	–man roffbib
38 I need help with loadkeys	–man loadkeys
40 help on unexpand	–man unexpand
43 I need help using keylogin	–man keylogin
44 run <command> on <host>	–rsh –l <username> <host> <command>
45 help with true	–man true
47 display help on eval	–man eval
48 make <file1> bigger via gunzip	–gunzip <file1>
50 show help on u3b5	–man u3b5

The incorrect sentences from the 3,300 epoch Bipolar training set.

1. what does defaults_to_indentpro mean	–man defaults_to_indentpro
3 show help on traffic	–man traffic
8. show me help on unalias	–man unalias
16 Please give me help with paxcpio	–man paxcpio
18 Please give me help with sortm	–man sortm
19 show help on toolplaces	–man toolplaces
23 help using rmdel	–man rmdel
25 show help on arch	–man arch
28 help with bc	–man bc
32 help with ap	–man ap
35 show help on unload	–man unload
37 help using roffbib	–man roffbib
38 I need help with loadkeys	–man loadkeys
40 help on unexpand	–man unexpand
43 I need help using keylogin	–man keylogin
44 run <command> on <host>	–rsh –l <username> <host> <command>
45 help with true	–man true
48 make <file1> bigger via gunzip	–gunzip <file1>
50 show help on u3b5	- -man u3b5

The incorrect sentences from the 2,400 epoch Binary training set.

15 show help on jobs	–man jobs
33 show me help on uptime	–man uptime
37 show help on traffic	–man traffic
49 Please give me help with tbl	–man tbl
51 help on the command logger	–man logger
55 help on the command ypmatch	–man ypmatch
56 help on the command adb	–man adb
57 help using rmdel	–man rmdel
64 display a count of lines and words in <file>	–wc <file>
66 help with ap	–man ap
67 help on the command wall	–man wall
68 show me the man page on rehash	–man rehash
70 I need help using w	–man w
83 what does uuto mean	–man uuto
91 help on biff	–man biff
92 help with setenv	–man setenv
96 Please give me help with defaults_to_mailrc	–man defaults_to_mailrc

The incorrect sentences from the 3,000 epoch Binary training set.

14 beautify <file1> and store it in <file2>	–cb <file1> > <file2>
15 show help on jobs	–man jobs
33 show me help on uptime	–man uptime
37 show help on traffic	–man traffic
51 help on the command logger	–man logger
55 help on the command ypmatch	–man ypmatch
56 help on the command adb	–man adb
57 help using rmdel	–man rmdel
64 display a count of lines and words in <file>	–wc <file>
65 list the name of the NIS server	–ypwhich
66 help with ap	–man ap
67 help on the command wall	–man wall
70 I need help using w	–man w
91 help on biff	–man biff
92 help with setenv	–man setenv

Appendix C

Network Definition Files

The network definition file is used by the bp program to define the type of network. The definition section defines the total number of units that will be in the network (nunits) the number of input units (ninputs) and the number of output units (noutputs). The network section defines how the network will be connected. Each unit in the network has a unique identification number, starting at 0 for the 1st input unit and ending with nunits - 1. The first two numbers in each line represent which units are going to be receiving the inputs and the last two represent the units where the inputs are coming from. If we had a line: %r 150 50 0 150 this would mean that the block of units starting with unit 150 and continuing for 50 units, are connected, or will receive their inputs from the block of units starting with unit 0 and continuing for 150 units. The biases section defines which units, if any, are to have a bias input. A bias input is an input to a unit that is always one. The first number is the starting unit of the block of units that are to have a bias and the second number is how many units are in the block.

Net one:

```
definitions:
nunits 300
ninputs 150
noutputs 100
end
network:
%r 150 50 0 150
%r 200 100 150 50
end
biases:
%r 150 150
end
```

Net two:

```
definitions:
nunits 350
ninputs 150
noutputs 100
end
network:
%r 150 50 0 150
%r 200 50 150 50
%r 250 100 200 50
end
biases:
%r 150 200
end
```

Net three:

```
definitions:
nunits 1000
ninputs 500
noutputs 100
end
network:
%r 500 400 0 500
%r 900 100 500 400
end
biases:
%r 500 500
end
```

Net four:

```
definitions:
nunits 400
ninputs 200
noutputs 100
end
network:
%r 200 100 0 200
%r 300 100 200 100
end
biases:
%r 200 200
end
```

Net five:

```
definitions:
nunits 600
ninputs 300
noutputs 100
end
network:
%r 300 200 0 300
%r 500 100 300 200
end
biases:
%r 300 300
end
```

Appendix D

Data Set

Sentences used for the Binary Trials

help nroff	–man nroff
I need help with vedit	–man vedit
help on crypt	–man crypt
run <command> on <host>	–rsh –l <username> <host> <command>
show help on jobs	–man jobs
display help on sccs–comb	–man sccs–comb
display a count of lines and words in <file>	–wc <file>
format my floppy	–fdformat
help using write	–man write
Please give me help with defaults_to_mailrc	–man defaults_to_mailrc
help with tbl	–man tbl
show help on u3b5	–man u3b5
help using rmdel	–man rmdel
show help on arch	–man arch
what does uuto mean	–man uuto
what does date mean	–man date
I need help using w	–man w
I need help with loadkeys	–man loadkeys

help on the command sum	–man sum
display help on tput	–man tput
what is fold	–man fold
show me the man page on rehash	–man rehash
display help on rusers	–man rusers
show me the man page on troff	–man troff
show help on toolplaces	–man toolplaces
help on the command input_from_defaults	–man input_from_defaults
help with setenv	–man setenv
display id	–id
what does sccs–help mean	–man sccs–help
man nawk	–man nawk
help selection_svc	–man selection_svc
help on old–eyacc	–man old–eyacc
help nawk	–man nawk
check spelling in <file>	–spell <file>
help with bc	–man bc
show help on defaults_to_indentpro	–man defaults_to_indentpro
what does sortm mean	–man sortm
help with true	–man true
arch	–arch
show help on traffic	–man traffic
print the files that are queued	–atq
help on the command ypmatch	–man ypmatch
Please give me help with tbl	–man tbl
I need help with sccs–get	–man sccs–get
I need help with admin	–man admin
list who all is logged on	–who
show me help on unalias	–man unalias
I need help using keylogin	–man keylogin
clear	–clear

show help on unload	–man unload
show me the man page on input_from_defaults	–man input_from_defaults
I need help using get_selection	–man get_selection
help on the command ldd	–man ldd
what does defaults_to_indentpro mean	–man defaults_to_indentpro
converse with <username@host>	–talk <username@host>
show current directory	–pwd
show ripple pattern to check display	–lptest
show me the man page on glob	–man glob
display help on nl	–man nl
what is spellin	–man spellin
what does env mean	–man env
man arch	–man arch
Please give me help with paxcpio	–man paxcpio
display how much space I am taking up	–du
help with ul	–man ul
help on unexpand	–man unexpand
how do I use uudecode	–man uudecode
help using roffbib	–man roffbib
remotely log <username> into <host>	–rlogin –l <username> <host>
I need help using pg	–man pg
help on the command wall	–man wall
Please give me help with sortm	–man sortm
help with ap	–man ap
what is show	–man show
man unload	–man unload
help mh–sequence	–man mh–sequence
help on biff	–man biff
show me groups for <user>	–groups <user>
show help on sccs–comb	–man sccs–comb
what does ar mean	–man ar

make <file1> bigger via gunzip	–gunzip <file1>
list the name of the NIS server	–ypwhich
display the users in compact form	–users
show <file1> with vi	–vi <file1>
help on the command adb	–man adb
show me help on uptime	–man uptime
beautify <file1> and store it in <file2>	–cb <file1> > <file2>
give me the name of the yellow pages server	–ypwhich
Please give me help with fmtdump	–man fmtdump
display the clock window	–clock
help on the command ar	–man ar
how do I use show	–man show
help on the command logger	–man logger
display help on eval	–man eval
what does switcher mean	–man switcher
help on nawk	–man nawk
help on hostid	–man hostid
how do I use strings	–man strings
how do I use label	–man label
display help on m4	–man m4

Sentences used for the Bipolar Trials

what does defaults_to_indentpro mean	–man defaults_to_indentpro
show ripple pattern to check display	–lptest
show help on traffic	–man traffic
clear	–clear
give me the name of the yellow pages server	–ypwhich
what does ar mean	–man ar
help mh–sequence	–man mh–sequence
show me help on unalias	–man unalias
display help on m4	–man m4
display the clock window	–clock
what does sccs–help mean	–man sccs–help
show me the man page on troff	–man troff
show current directory	–pwd
remotely log <username> into <host>	–rlogin –l <username> <host>
Please give me help with tbl	–man tbl
Please give me help with paxcpio	–man paxcpio
help on the command logger	–man logger
Please give me help with sortm	–man sortm
show help on toolplaces	–man toolplaces
help with tbl	–man tbl
help on the command ypmatch	–man ypmatch
help on the command adb	–man adb
help using rmdel	–man rmdel
display help on tput	–man tput
show help on arch	–man arch
help on crypt	–man crypt
help on the command ldd	–man ldd
help with bc	–man bc
how do I use uudecode	–man uudecode
display a count of lines and words in <file>	–wc <file>

list the name of the NIS server	–ypwhich
help with ap	–man ap
help on the command wall	–man wall
show me the man page on rehash	–man rehash
show help on unload	–man unload
I need help using w	–man w
help using roffbib	–man roffbib
I need help with loadkeys	–man loadkeys
help nawk	–man nawk
help on unexpand	–man unexpand
I need help with admin	–man admin
I need help with vedit	–man vedit
I need help using keylogin	–man keylogin
run <command> on <host>	–rsh –l <username> <host> <command>
help with true	–man true
display id	–id
display help on eval	–man eval
make <file1> bigger via gunzip	–gunzip <file1>
what does uuto mean	–man uuto
show help on u3b5	–man u3b5

Appendix E

Binary Dot Products

This file shows the value of the dot product of each word with itself.

34 –lptest	35 rmdel	37 mh–sequence
37 nl	38 compact	38 logged
38 name	39 –man label	39 –man ypmatch
40 –man unexpand	40 –vi <file1>	40 log
40 of	41 –man show	41 <return>
41 check	41 unalias	42 –fdformat
42 –l	42 –man paxcpio	42 admin
42 clock	42 files	42 glob
42 nroff	42 wall	43 –clock
43 –man crypt	43 –man env	43 –man tbl
43 –man vedit	43 <file1>	43 logger
43 on	43 strings	44 –cb <file1> > <file2>
44 –man logger	44 –man pg	44 –man rmdel
44 –spell <file>	44 –ypwhich	44 bigger
44 current	44 print	44 troff
45 –man ap	45 –man arch	45 –man get_selection
45 –man hostid	45 –man nroff	45 –man old–eyacc
45 –rlogin –l <username> <host>	45 <user>	45 clear

45 via	46 –man glob	46 –man jobs
46 –man u3b5	46 arch	46 are
46 form	46 queued	46 run
46 traffic	46 write	46 ypmatch
47 –man loadkeys	47 –man sccs–get	47 –man sortm
47 <file2>	47 beautify	47 directory
47 do	47 make	47 my
47 rehash	47 server	47 that
47 uudecode	47 window	47 yellow
48 –man biff	48 –man fmtdump	48 –man rusers
48 –man sccs–help	48 –man spellin	48 –man uudecode
48 biff	48 help	48 is
48 man	48 paxcpio	48 to
48 words	49 –clear	49 –man adb
49 –man input_from_defaults	49 –man selection_svc	49 –man ul
49 –man w	49 and	49 converse
49 date	49 defaults_to_indentpro	49 fmtdump
49 format	49 get_selection	49 hostid
49 label	49 nawk	49 taking
49 tbl	49 uptime	50 –atq
50 –man setenv	50 –man tput	50 –wc <file>
50 <host>	50 I	50 Please
50 command	50 gunzip	50 ldd
50 old–eyacc	50 pages	50 sccs–comb
50 setenv	50 space	50 vedit
51 –arch	51 –id	51 –man keylogin
51 –man roffbib	51 –man true	51 a
51 crypt	51 display	51 give
51 how	51 into	51 list
51 much	51 need	51 sum
51 ul	51 what	51 who

52 –man bc	52 –man defaults_to_indentpro	52 –man fold
52 –man ldd	52 –man nl	52 –man rehash
52 –man sccs–comb	52 –man write	52 defaults_to_mailrc
52 eval	52 groups	52 m4
52 tput	52 u3b5	52 unexpand
52 unload	53 –man uuto	53 –pwd
53 <command>	53 <file>	53 adb
53 am	53 env	53 it
53 me	53 page	53 pattern
53 rusers	53 sortm	53 store
53 true	53 users	54 –man defaults_to_mailrc
54 –man eval	54 –man m4	54 –talk <username@host>
54 –users	54 NIS	54 all
54 does	54 floppy	54 input_from_defaults
54 keylogin	54 loadkeys	54 mean
54 ripple	54 sccs–help	54 switcher
54 toolplaces	55 –du	55 –gunzip <file1>
55 –man date	55 –man switcher	55 –man toolplaces
55 –man troff	55 –man uptime	55 –man wall
55 ar	55 bc	55 –rsh –l <username> <host> <command>
55 pg	55 the	55 up
55 w	55 with	56 –man admin
56 –man mh–sequence	56 –man nawk	56 –man sum
56 <username@host>	56 fold	56 id
56 in	56 remotely	56 roffbib
56 sccs–get	56 spellin	56 spelling
56 use	57 –man unalias	57 –man unload
57 <username>	57 ap	57 jobs
57 uuto	58 –man strings	58 >
58 show	59 –groups <user>	59 –who
59 count	59 selection_svc	59 vi

60 –man ar	60 –man traffic	62 for
63 lines	63 using	

Vita

Renwick Lowell Conrad lived with his parents, Lowell and Reba Conrad, in the small town of Osborne, Kansas until December, 1965 when he and his family moved to North Platte, Nebraska. He graduated from North Platte Senior High School in May of 1973 and then went on to attend one year at the North Platte Junior College before attending a year at Grace Bible Institute in Omaha, Nebraska. He worked as a Night Auditor for ten years, first at the Ramada Inn of North Platte, then later, at the Holiday Inn of North Platte. He returned to College full time in August of 1988 and graduated from the University of Nebraska, Kearney in May of 1992 with a Comprehensive Degree in Computer Information Systems and a Comprehensive Degree in Business Administration with an emphasis in Management Information Systems. He then moved to Knoxville, Tennessee where he received his Masters in Computer Science.