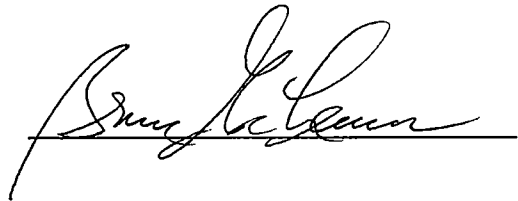


To the Graduate Council:

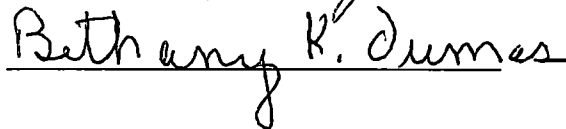
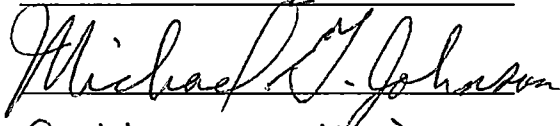
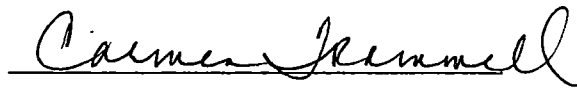
I am submitting herewith a dissertation written by Karlyn Ammons Barilovits entitled "Hebbian Learning in Recurrent Neural Networks for Natural Language Processing." I have examined the final copy of this dissertation for form and content and recommend that it be accepted in partial fulfillment of the requirements for the degree of Doctor of Philosophy, with a major in Computer Science.



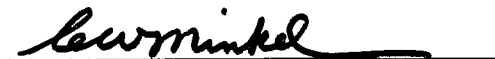
Bruce J. MacLennan, Major Professor

We have read this dissertation

and recommend its acceptance:



Accepted for the Council:



Associate Vice Chancellor and

Dean of the Graduate School

**HEBBIAN LEARNING IN
RECURRENT NEURAL NETWORKS
FOR NATURAL LANGUAGE
PROCESSING**

A Dissertation

Presented for the

Doctor of Philosophy Degree

The University of Tennessee, Knoxville

L. Karlyn Ammons Barilovits

May 1999

Copyright ©Karlyn Ammons Barilovits, 1999

All rights reserved

Dedication

This dissertation is dedicated to

my parents

Joyce Boone Ammons and Lewis Woodson Ammons, Jr.

for giving me a love of learning and

a dedication to complete the task

and to

my husband

Paul Barilovits

for the unwavering love, support, and encouragement

that he has shown me throughout the entire pursuit.

Acknowledgements

There are many people to whom I am grateful for helping me to successfully complete my dissertation and my degree. I am particularly grateful to Dr. Bruce MacLennan whose time and patience were in great demand throughout the process. I would like to thank my PhD committee members: Dr. Bethany Dumas, Dr. Mike Johnson, Dr. Carmen Trammell, and Dr. Brad Vander Zanden; for taking the time to evaluate my research and for providing valuable suggestions in producing my dissertation.

I would like to thank the Appalachian College Association for the two Mellon Fellowships which allowed me to have the time off from work to finish my research.

I would like to thank Rachel Hoffman for providing the results to the two and one half year old vocabulary survey. I would like to thank Tyler Hoffman, Tate Robinson, and Megan Clough for participating in the survey.

I would like to thank Martin Do and Asim Yarkhan for their continual support throughout the program and for technical advise. I would like to thank Vivie Babb for the emotional support and many hours spent editing the different drafts. I would like to thank Dr. Wayne Lang for convincing me that I could do this.

The greatest debt that I owe, however, is to my family. I would like to thank my parents, Joyce and Woody Ammons, for their constant support and encouragement. I would like to thank Juree, Rachel, Lee, Sarah, and Tyler for their

support and for the much needed time away from the degree. I would like to thank my daughter, Hannah, for reminding me what is important in life and why I have worked so hard. Finally, I would like to thank my husband, Paul, without whom I would never have reached this moment.

Abstract

This research project examines Hebbian learning in recurrent neural networks for natural language processing and attempts to interpret language at the level of a two and one half year old child. In this project five neural networks were built to interpret natural language: a Simple Recurrent Network with Hebbian learning, a Jordan network with Hebbian learning and one hidden layer, a Jordan network with Hebbian learning and no hidden layers, a Simple Recurrent Network with back propagation learning, and a nonrecurrent neural network with back propagation learning. It is known that Hebbian learning works well when the input vectors are orthogonal, but, as this project shows, it does not perform well in recurrent neural networks for natural language processing when the input vectors for the individual words are approximately orthogonal. This project shows that, given approximately orthogonal vectors to represent each word in the vocabulary the input vectors for a given command are not approximately orthogonal and the internal representations that the neural network builds are similar for different commands. As the data shows, the Hebbian learning neural networks were unable to perform the natural language interpretation task while the back propagation neural networks were much more successful. Therefore, Hebbian learning does not work well in recurrent neural networks for natural language processing even when the input vectors for the individual words are approximately orthogonal.

Contents

1	Introduction	1
2	Background	12
3	Project Environment	25
3.1	The World Model	27
3.2	The Robot Control System	29
4	The Vocabulary	38
5	The Natural Language Processor	46
5.1	Introduction	46
5.2	Neural Networks with Hebbian Learning	51
5.2.1	Introduction to Neural Networks with Hebbian Learning .	51
5.2.2	Tests on Input Vectors	55
5.2.3	Tests on Target Vectors	57

5.2.4	Tests on Initial Parameters	58
5.2.5	Tests on Updating Weight Matrix	60
5.2.6	Tests to Solve Unlearning Problem	62
5.2.7	Tests on Hidden Layer Size	65
5.2.8	Tests on Weight Update Rule	66
5.2.9	Tests on Activation Function	69
5.3	Recurrent Neural Network with Back Propagation Learning . . .	70
5.4	Nonrecurrent Neural Network with Back Propagation Learning . .	75
6	Results and Conclusions	78
6.1	Introduction	78
6.2	First Question	81
6.3	Second Question	82
6.4	Third Question	85
6.5	Other Results	87
6.6	Conclusion	89
7	Future Work	91
	Bibliography	100
A	Items Allowed in House	123
B	Actions Allowed on Items	128

C	Survey used to compute weights	129
D	Words in Input Vocabulary	135
E	Words Two and one half year old child understands	141
F	Words Natural Language Processor Understands	144
G	SRN with One Hidden Layer	147
H	Jordan Network with No Hidden Layer	149
I	Jordan Network with One Hidden Layer	151
J	Tests on Neural Networks with Hebbian Learning	153
K	Simple Recurrent Network with Back Propagation Learning	155
L	Nonrecurrent Neural Network with Back Propagation Learning	157
M	SRN with Hebbian Learning	159
M.1	Euclidean Distance for Question Four	159
M.2	Covariance Matrix for Question Four	160
N	Other Results	161
N.1	SRN with Hebbian Learning - Learning rate vs. Error rate	161
N.2	SRN with Hebbian Learning - Hidden layer size vs. Error rate . .	162

N.3	SRN with Back Propagation Learning - Learning rate vs. Error rate	163
N.4	SRN with Back Propagation Learning - Hidden layer size vs. Error rate	164
N.5	Nonrecurrent Net with Back Propagation Learning - Learning rate vs. Error rate	165
N.6	Nonrecurrent Net with Back Propagation Learning - Hidden layer size vs. Error rate	166
O	Robot Control System Data	167
O.1	House 1	167
O.2	House 2	169
O.3	House 3	170
O.4	House 4	171
O.5	House 5	172
O.6	House 6	174
O.7	House 7	176
O.8	House 8	177
O.9	House 9	178
O.10	House 10	180
O.11	House 11	182
O.12	House 12	183

Appendices

123

Vita

185

List of Figures

3.1	System Diagram	28
3.2	Robot Control System	37
5.1	Natural Language Processor	48

Chapter 1

Introduction

The purpose of this project is to study Hebbian learning in recurrent neural networks for natural language processing. The Hebb rule was the first and is the best known learning method for neural networks. Natural language is a language which is natural to humans as opposed to a language such as binary which is natural to computers. Language is learned by humans at such an early age that it has become a fascinating field of study for many, myself included, in the artificial intelligence community.

There have been thousands of papers written on the subject of natural language processing. Even before the advent of computers, linguistics researchers were studying natural language. According to Websters Dictionary, "Linguistics is the study of human speech including the units, nature, structure, and modification of language, languages, or a language". Linguistics includes the study of

phonetics, phonology, morphology, syntax, semantics, and pragmatics. Initially, linguistics' research focused on the syntax and grammar of a particular language, as well as how different languages related to one another. However, "the study of language from a pragmatics perspective has come to be the most recent movement in linguistics" [88]. Computational linguistics is the study of computer processing of human language. Computational linguistics is a very active area of research. At the IBM Watson Research Center one of the major areas of research is Natural Computing. Natural Computing attempts to allow users to issue commands to the computer in a manner which is "natural", using speech and gesture, as opposed to using keyboard and mouse. While linguistics researchers are doing work with computers, the field of linguistics is much larger than the use of computers for natural language processing.

From linguistics and many other fields of study came the study of artificial intelligence. "How can we make the computer behave intelligently?" was the question that researchers asked. One of the areas of artificial intelligence where much attention was focused was natural language processing. In artificial intelligence the focus of natural language processing is on the knowledge necessary to understand language rather than the language elements themselves. Researchers in artificial intelligence were interested in making a computer behave intelligently, they were not interested in how natural language was carried out in the brain, and these systems were not fault tolerant nor did they generalize well. The traditional

artificial intelligence approach, as this is called, is interested in the outcome, not the process. The goal is to get a computer to output what you would like it to output, for example, natural language. The goal is not to get the computer to process language in the same manner in which the brain processes natural language. This approach is the symbolic approach and is discussed in greater detail later.

In the 1950's and 1960's some work was done toward neural modeling for artificial intelligence. The goal in neural modeling was to get the computer to learn without the use of explicit rules. In other words, to get the computer to learn the way that our brain learns. The following comes from An Introduction to Neural Networks by Anderson [3]:

The first person who seems explicitly to have phrased a learning rule in terms of synaptic change was Donald Hebb (1949) in his influential book, *Organization of Behavior*. Hebb said, in a much-quoted sentence, that what is critical for synaptic learning is a coincidence between excitation in the presynaptic and postsynaptic cells:

When an axon of cell A is near enough to excite a cell B and repeatedly or persistently takes part in firing it, some growth process or metabolic change takes place in one or both cells such that A's efficiency as one of the cells firing B, is increased.

The study of neural networks was a new branch of artificial intelligence which sought to improve on the performance of the symbolic approach by building models which more closely resembled the brain. This is called the subsymbolic, or connectionist, approach. During the 1970's and most of the 1980's little work was done using neural networks for natural language processing, but this field has had a dramatic resurgence in the past ten years, beginning mostly with the work of McClelland and Rumelhart [133], [96], and [95]. This is due to the introduction of back propagation methods to neural networks, as will be discussed in Chapter 5.

Most of the previous work that has been done in natural language processing has been done using the symbolic approach. In the symbolic approach, for the most part, rules and look-up tables are applied to syntactic language units. An alternative to the symbolic approach is the subsymbolic, or connectionist approach. In the connectionist approach, language units emerge from a subsymbolic representation of the data which does not necessarily correspond to syntax. There are no explicit rules, but the computer learns the language by recognizing patterns it has seen before, much like we learn language. Language units are connected to one another through the activations that they send to one another to "learn" to interpret the language. There are no specified rules of behavior, but many patterns are presented to the system so that over time it learns the appropriate output for a given input and hopefully learns to generalize to similar inputs that it

has not seen before. Such a system is called a neural network. A neural network, according to Rumelhart and McClelland [133], is described as follows:

- A set of processing units
- A state of activation
- An 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

Neural networks are attractive because they have properties in common with the brain, such as massive parallelism, neuron-like computing units, synapse-like weights, fault tolerance and adaptability. [13] They have shown themselves to be more fault tolerant, error correcting, generalizable, and to degrade more gracefully than corresponding symbolic systems. It is for these reasons that the approach taken in this project uses the connectionist method of learning language. In

this project I have used a constraint satisfaction neural network to implement the robot control system (to be described in more detail in Chapter 3). I have used a recurrent neural network with Hebbian learning to implement the natural language processor (to be described in more detail in Chapter 5). I have also used a recurrent neural network with back propagation learning and a nonrecurrent neural network with back propagation learning to implement the natural language processor (also to be described in more detail in Chapter 5).

As mentioned previously, much work has been done recently in the field of natural language processing. The artificial intelligence research in natural language processing has been through the use of three methodologies: the symbolic approach, the subsymbolic approach, and a combination of the symbolic and subsymbolic approaches. In the following paragraphs, I will discuss some of the most notable research which has been done using each of the three methodologies. I will then narrow the focus down to those works which have been most influential to this research project.

Timothy Potts' [121] symbolic approach made use of frames to represent linguistic expressions. Potts' research is similar to mine in that he uses the rooms of a house to show his point. Among other symbolic natural language processing works are those of Cottrell [29] and Ng and Lee [115] for word sense disambiguation. Also on the symbolic side of research is NPtool [151] which is able to extract noun phrases from English texts. Much of the symbolic natural language process-

ing research in recent years has centered around the MUC systems. These systems are a collaboration of many researches, mostly at the University of Massachusetts, including Lehnert, Cardie, Fisher, McCarthy, and Soderland [82]. This project takes a copy of text, such as newspaper print, and searches for specific topics. The project has been quite successful and is continuously being improved upon.

A number of researchers have concluded that a combination of the symbolic and subsymbolic methodologies is the solution to computers being able to solve the natural language processing problem. Among them are Kwasny and Faisal and Lehnert. Kwasny and Faisal [76] have developed CDP (Connectionist Deterministic Parser). CDP uses a winner-take-all neural network and a parsing stack to parse sentences. Lehnert [84] has developed CIRCUS which uses stack control for syntactic analysis of sentences and a combination of marker passing and numeric relaxation for semantic analysis of sentences.

Finally, as previously mentioned, the subsymbolic approach to natural language processing has had a dramatic resurgence in the past twelve years. This approach has been the main focus of this research as it is the approach that I am taking. The first major triumph of the subsymbolic approach to processing natural language was by Rumelhart and McClelland [134]. They built a neural network to form regular and irregular past tenses of English verbs. This triumph set off an explosion of research in the area. Lawrence, Giles, and Fong [79] built a neural network which could classify natural language sentences as grammati-

cal or ungrammatical. McClelland and Kawamoto [94] used semantic features to build a knowledge driven sentence interpreter with constraints from the world and not linguistic representations. This is similar to this research project in that I am using microfeatures and knowledge from the world rather than linguistic representation. Elman [40] used a simple recurrent neural network, as I have used in this project, to predict the next word in script-based stories. Wermter and Lehnert [154] built NOCON (NOun phrase analysis with CONnectionist networks) which learns semantic relationships between nouns. Berwick [11] used constraints, as I am using in the robot control system, to learn the meanings of verbs. Hinton [60] developed BoltzCONs which creates and manipulates symbol structures. Ferrell [42] used a recurrent neural network to perform grammatical inference. This is just a sampling of the tremendous amount of research that has been carried out in subsymbolic natural language processing. I will now focus on those works which most strongly influenced this research project.

The most notable recent natural language processing work is Risto Miikkulainen's DISCERN [103]. DISCERN is a modular neural network system that learns to process simple stereotypical narratives. He uses a script-processing system for natural language. He also uses hierarchical trace feature maps to recognize and store the scripts.

In addition to DISCERN, other works contributed greatly to the inception of this particular project. The first is Winograd's "blocks world" SHRDLU [158].

SHRDLU was a system built upon a microworld where the world is a set of blocks of different shapes and sizes. The user entered commands to a simulated robot to move the blocks around. The system was built entirely within the symbolic framework and as such did not generalize well nor was it shown that it could be expanded to solve more complicated problems. It did, however, show the usefulness of a microworld for learning natural language.

Another project that contributed to the ideas behind this project is McClelland and Rumelhart's rooms problem [95]. They built a neural network system using constraint satisfaction which would take certain features of a given room and would then fill in other features that fit that room description. This system did not learn what kind of room it was nor did it provide natural language to discuss the rooms. It did, however, provide the concept that information about an environment such as a room could best be learned by constraint satisfaction systems.

Finally, a project of significance in the recent history of natural language processing is the CLUES project by Robert Allen [2]. In CLUES a microworld was introduced as part of the input and the neural networks were trained to generate descriptions of the objects in that microworld. The system then gave one word answers to questions about the objects in the microworld. The CLUES project provides further evidence of how a microworld facilitates the learning of language. However, this project had several problems: it was unstable, it did not generalize well, and was shown not that it could be expanded to solve more complicated

problems, i.e. scale up. The author suggests that future projects in this area should be built with a hierarchical or modular approach.

In this project I have used a real world environment in which knowledge of the world is embedded. The purpose of this "world" is to show simulated situated intelligence. I am showing that the robot not only can talk about the environment, but has knowledge of the environment. It is by limiting the subject matter to a small area that we are able to address how language is used. Having a suitably confined environment upon which to build the language facilitates the learning of language. Knowledge representation is difficult without such a world. It has not, however, been shown that such a world is a necessity for building language. My research helps to reinforce the notion that having a confined subject area in which there is existing knowledge, i.e. a world, facilitates the learning of language.

I have developed a modular neural network system which has knowledge of the real world environment in which it is situated, as in CLUES and SHRDLU. The real world environment in this project is a house. The interaction with the user is accomplished first by allowing the user to enter into the computer the items that are present in a given house. Then the user is allowed to issue commands to a simulated robot system which will move items about the house. The project is described in much more detail in Chapter 3.

This project uses artificial intelligence in that it simulates the behavior of a robot and, therefore, the program should respond to the user as a robot (or person)

would given the same amount of information. For example, if a robot were built to move things about a house it would first need to see the items in the house and classify each of the rooms in some manner. It would then be able to go from room to room and carry out the commands that were issued by the user. Sometimes this would require additional interaction with the user. By first learning the rooms of the house, the simulated robot is building its knowledge base of the world. It then has the ability to use that knowledge to carry out the commands that the user enters. This is situated intelligence. The robot is showing intelligence in this particular situation or environment.

The contribution that this project represents to the field of Computer Science, in particular to the Neural Network and Natural Language Processing communities, is in the study of Hebbian learning. It shows that Hebbian learning does not work well in recurrent neural networks for natural language processing when the input vectors for the words of the language are approximately orthogonal.

After making the decision to study natural language using the connectionist approach, I turned my attention to the limitations of the current neural network projects in natural language processing. I needed to pick a topic which would improve upon the existing research. In order to do this, I had to determine the shortcomings of the current projects in natural language processing. Chapter 2 presents a detailed description of those shortcomings.

Chapter 2

Background

Neural networks, which implement the connectionist approach to Artificial Intelligence, have been studied for some years now. However, we are still very far from being able to represent anything close to human knowledge using neural networks. If neural networks are to be the answer to representing human knowledge on a computer, what are the issues that currently need to be addressed? There are seven areas where improvement is needed in the use of neural networks for modeling human behavior in the processing of natural language. These seven areas are: neural network hardware and software availability, knowledge representation, ability of neural nets to generalize or deal with exceptions, ability of neural networks to scale up to solve real world problems, more knowledge of how the brain actually works, ability to program and understand neural networks, and the amount of prewiring necessary for neural networks to do anything significant. Aparicio and

Levine [86] alluded to several of these areas by saying "Many commentators view the body of neural network theory as a monolithic architecture, the Multilayer Perceptron (MLP), in its most simplistic form. MLPs have poor scaling properties, are notoriously slow to learn, are sensitive to initial conditions, and require 'black arts' for proper network configuration." In the following paragraphs I will discuss each of the seven areas where further work is necessary and in particular how it relates to implementing natural language processing using neural networks.

In order for major advances to be made in connectionist models there have to be major improvements in the hardware and software that is available to program and implement neural networks. There are many software packages available today to program neural networks, but most of those packages are only available for PCs or Unix work stations, not for massively parallel architectures. Natural language processing with neural networks has not been attempted to a large extent on massively parallel processors. Implementing the connectionist approach in parallel is very important because the brain, according to current knowledge, does not operate as does a serial processor. Why then, if we are implementing neural networks, which are obviously parallel, are we doing it on serial processors? There are several reasons for this. One is the fact that few machines have been built specifically for neural network implementation. A second reason is that the generic parallel machines that are available have few, if any, software tools available for programming neural networks. Finally, the size and cost of using

massively parallel processors has held many researchers back from utilizing their potential power. In order to build neural networks which have large vocabularies and solve real problems (i.e. show human understanding) we need to implement them on hardware which is comparable to the concept of connectionism.

While the connectionist approach requires massively parallel processors, there is also evidence that much of the activity in the brain is continuous, not discrete. Therefore, if we wish to model the brain perhaps we should use processing that is continuous, i.e. analog computers. So far, the research in this area has been slow and approached by few in the neural network community. Carver Mead (1989) is one of the few who have approached the topic. He is building hardware devices with components which try to mimic the circuits of the brain. Another researcher who is taking the continuous approach is Jonathan Mills. Jonathan Mills has designed a chip which is a field computer. A field computer, as defined by MacLennan [90], is a computer which deals with information processing in terms of fields, i.e. continuous distributions of data. Mills' chip is a combination of discrete and continuous processing but is a step in the continuous processing direction.

The second area where improvement is needed in the future of neural networks for natural language processing is in knowledge representation. This is possibly the most vital area where improvements can be made. "...AI finally had to face the problem of representing everyday knowledge— a difficult, decisive, and philo-

sophically fascinating task with which it is still struggling today.” [38] We have not had adequate data structures to use, possibly because we do not know how data is structured in the brain. I will address this issue later. For now, I will discuss problems with knowledge representation. In natural language processing, it is difficult to determine how to represent the meaning. Sutcliffe [146] says:

Any program which is capable of processing natural language must incorporate some scheme for representing meaning, yet, despite a great deal of intensive research in this area, workers within artificial intelligence (AI) have failed to come up with any general method for capturing the underlying content of even simple passages of English.

We must first determine how the data is going to be presented to the neural network. Miikkulainen describes how knowledge representation is an obstacle to further progress in natural language processing. “Representing and learning knowledge structures appears to be the most important research problem in our way toward artificial neural network intelligence.” [103] We must be able to represent data in a form that will hold large amounts of information and yet have them be retrievable in an acceptable amount of time without using tremendous computer resources. MacLennan [89] says: “Our goal is to develop a theoretical framework for connectionist knowledge representation that fills a role analogous to the theory of formal systems in symbolic knowledge representations.” This goal

is very important to the future of neural networks and is being sought after by many in the field today.

Even though there are many methods being used to represent a word, such as Miikkulainen's feature maps [98] and Plate's holographic memories [120], most of the current research seems to have one thing in common— their method of representing the structure of the input sequence. The methods generally used are frames and case-role assignment. Since both methods operate on the same principle, I will present only one of the two here. Frames are used by Plate [120], Miikkulainen [103], and McClelland and Rumelhart [95]. The concept of frames is derived from symbolic artificial intelligence where there are a set of rules and each word fits into a corresponding slot in the rule. With frames, an input sequence is scanned to find objects to fill each slot within the frame:

	agent		action	
			patient	
			recipient	

Where an agent is a noun doing action, the action is the verb, the patient is the thing being used to do the action (animate or not), and the recipient is the noun receiving the action. The particular slots within the frame vary from application

to application, but the concept is the same. One of the reasons that frames and scripts have crossed over to connectionist artificial intelligence is that many believe that humans have thousands of scripts in their brain and apply them to everyday situations without thinking. For example, our restaurant script might call to mind being seated, looking at the menu, ordering, eating, and paying. Whether or not our brains operate using scripts is a topic of debate.

Independent of this debate is that there are several problems with using the frame representation; a lack of flexibility and a lack of appropriate meaning are among those problems. First, frames do not allow for multiple agents, actions, recipients, etc. Frames allow a sentence to only be represented in a very rigid structured form with no embedded clauses, etc. Also, this method degrades (works less effectively) when some of the slots within the frame are left unfilled. The second problem with frames is a lack of appropriate meaning. When we as human beings use language, it is for a purpose or to produce an effect. When we speak, hear, or read words, we do not think of the meaning of each word and whether it is a verb or not. We concentrate on the effect of the words on our behavior. It is a matter of semantics versus pragmatics. "A case frame approach to meaning, based as it is on list structures, cannot readily be adapted to capture subtle differences in meaning." [146] Finally, it is very difficult to determine how many slots there should be, how many possible different scripts will be allowed within the frames so as not to cause confusion, etc.

A second method used to represent the input sequences in recent research is using features or microfeatures. Features and microfeatures may be used alone or in conjunction with frames or case-role assignments, depending on the application. Features and microfeatures are alike in that both use a list of characteristics to classify each input item. Features and microfeatures are different in their granularity. Features are usually things like big, small, winged, human, etc. Microfeatures are characteristics on a smaller scale, for example, has edges, has certain orientation, etc.

Features were used by Sutcliffe [146] as well as McClelland and Kawamoto [94] and others. With features, each input item is classified according to a certain preset list of features. Features might be: big, small, commonly used, used at a certain time of day, hard, soft, etc. There are two problems with this approach. First, according to Sutcliffe, a representation based on features alone cannot capture the structured aspects of language. Features lack the recursive or hierarchical features necessary to build adequate language structures. [81] Recursion is needed to build language structures so that embedded clauses can be used within sentences. A second problem with this approach is that each time a new feature is added, all previously input items must be reevaluated based on that feature. This is a very costly and time consuming process and shows that features do not hold much promise in being used to solve large, real-world problems.

The third area where there is room for improvement in the field of neural networks is the inability of current neural networks to generalize well or to deal with exceptions. Neural networks are capable of dealing with incomplete data and perform well when filling in what is missing, provided that they have seen the original object previously. However, when it comes to taking something they have never seen before and generalizing to place it in an appropriate category, they do not perform quite so well. Most of the networks today can only generalize well when the regular class is much larger than the number of exceptions. Since neural networks work on the concept of statistical regularities, they do not perform very well when there are deviations from the norm. "Unless the class of possible generalizations is restricted in an appropriate a priori manner, nothing resembling human generalizations can be confidently expected." [38] Further study of how to make neural networks better able to make generalizations and deal with exceptions is needed.

The fact that no one has built a neural network for natural language processing with a significant vocabulary is just a small indication of the problem that neural networks do not scale up to solve real-world problems well. Most of the networks that have shown any sort of human intelligence have been those built with a small vocabulary (between 50 and 75 words), Miikkulainen's DISCERN for example. In order for the connectionist approach to be taken seriously in the research community it is important that we start to build machines with a vocabulary on the order

of 1000 words or more. "One needs a model with a realistically large vocabulary and range of possible senses and interpretations, in order to convincingly argue that the model is appropriate and adequate." [152] On this same note, we have got to build neural networks that solve "real" problems. No natural language processing problem of any significance has been solved with neural networks. Many connectionist proponents have said that neural network models scale up, but no one has shown that neural network models can be built which can be expanded to solve increasingly complicated problems. One possibility lies in building modular networks, where each subnetwork has a specific function and the many subnetworks work together. "Adaptive control of cognitive processes cannot be done by a single network with homogeneous structure. Rather it depends on a variety of interacting subnetworks, each with definable functions." [86] This method has been tried by Miikkulainen with DISCERN [102], Much more research in this area is needed.

The fifth area where there is room for improvement in the building of neural networks that demonstrate human knowledge is the study of the brain. This is not a project that is recommended to be carried out by a Computer Scientist, but it is important to the field nonetheless. We hope to be better able to build computer programs which show human understanding when we are better able to understand how the brain actually functions. We know very little of how the brain actually performs natural language processing [13] [97] [139].

The brain is proving very difficult to understand and may never be fully understood. It is the same for neural networks: neural networks are very difficult to program and to understand. It is very difficult to program a neural network and much preprogramming must be done. Also, it is difficult to interpret the output of a neural network and understand exactly what is going on while the program is being run through many cycles. Better methods of programming must be introduced and output generated that is more easily interpreted. Finally, much of the “knowledge” shown by connectionist models must be preprogrammed into the computer before the network ever starts to learn. The network architecture must be designed and the weight matrices set up. Most of the information that is going to be used later must be set up beforehand. If you add a new aspect to your program, you have to retrain the neural network. “Much preprogramming is needed to create successful neural network models today. A major direction for future research is to find ways to develop more of the necessary structures automatically.” [103] When someone solves this problem, it will be a major leap in the acceptance of neural networks for natural language processing as well as for other problems.

The shortcomings, which were discussed above, being addressed by this project are: lack of efficient knowledge representations, poor generalization, and inability to scale up. The knowledge representation problem is dealt with by building the system with emphasis on pragmatics, the study of how context impacts the use of

language. I am more concerned with the effects of the words on the robot than with whether a particular word is a noun or not. "Complex knowledge representation is difficult without a microworld." [2] The "knowledge" is structured around elements in the world and manipulation of those elements. This system is much more flexible and scalable than the case frame system, where the user is severely limited by the number and types of inputs allowed. The generalization problem of neural networks is addressed by having a large vocabulary, a better method of knowledge representation, and a simple network architecture which does not require a large amount of preprogramming.

Through this project, the goal was to show that we can build modular neural networks for natural language processing which will not only have a larger vocabulary than systems seen thus far, but which will also work with even larger vocabularies. According to Guthrie [55],

A group of AI researchers of natural language processing (NLP) admitted in public how many words there really were in the vocabularies of their systems. Of the answers, the average was 36.

The vocabulary problem is addressed in this project by having a large number of items in the world, a number of actions that will be allowed on those items, and a significant number of features which are associated with those items.

The scaling up problem in neural networks is addressed in this project by the simplicity of the algorithms used. The algorithm used is Hebbian Learning, which is less complex therefore faster than other learning algorithms. Hebbian learning works best with orthogonal input vectors, i.e. vectors whose scalar product is zero, however, orthogonal inputs grow with the length of the input and could be thousands of bits long. It is for that reason that I have used approximately orthogonal input vectors, which will be described in Chapter 5.

Another aspect of this project that makes it possible to scale up is the use of recurrent neural networks. These networks present the input to the neural network one word at a time and, therefore, the input does not grow with the size of the input sentence. Recurrent neural networks and Hebbian Learning are described in more detail in Chapter 5.

It is true that we have a long way to go to prove to the world that neural networks are capable of showing human understanding. However, I (and many other connectionist researchers) think that it is possible to do so. "As far as I can tell the most exciting thing happening in AI these days is the invasion of the brain people (a.k.a. the connectionists)." [83] When the seven areas discussed above have been addressed (many of them are being addressed as this is being written) there is no doubt that the connectionist approach will prove to be a useful approach for artificial intelligence in the future. Even with its flaws it has been shown to generalize better than symbolic approaches, to be more fault

tolerant, to show more promise of scaling up, and to be more robust against noise than symbolic approaches. “It seems highly likely that the rationalist dream of representationalist AI will be over by the end of the century,” [38]

Chapter 3

Project Environment

In this research project, I address three of the shortcomings which are discussed in the previous chapter: lack of efficient knowledge representations, poor generalization, and the inability of neural networks to scale up to solve larger problems. I implement an approach to natural language processing which uses a simulated robot that attempts to show situated intelligence. The robot exists in a “world” of which it is knowledgeable and in this case the “world” is a house. The robot learns what is in each of the rooms of a house. This project is similar to the CLUES project [2] except that the “world” is much larger, and it attempts to interpret natural language in a much more sophisticated manner. The rooms which can be included in a house are kitchen, dining room, office, bathroom, bedroom, and living room. The number of rooms and what is in each room will depend on the input.

This "world" is based on the rooms example presented by Rumelhart, McClelland, et al. [133] which is built on the Schema model. In their model, a list of input values representing the characteristics of a room are given, and constraint satisfaction is used to determine the other characteristics. "The Schema model has the ability to combine knowledge from different contexts, in this case rooms, to produce novel interpretations of the input." [133] In other words, more information is given in the output than a simple restatement of the input. In this research project, parallel constraint satisfaction is used to determine what items classify a particular room. Parallel constraint satisfaction seeks to satisfy as many constraints as possible concurrently. Parallel constraint satisfaction is a very efficient method for the schema model of processing because it has a goodness of fit measure and produces the outcome that is maximally consistent with both the external inputs and the internal knowledge. The words in the input for this project are used as evidence for complete interpretations. [133] The neural network learns what is in each of the rooms, then instructions are given to the robot as follows:

Input:

Move the coffee cup from the living room to the kitchen.

Input:

Where is the china?

Input:

Put the toothpaste in the bathroom.

This project is implemented using three separate modules. The modules are the World Model, the Robot Control System, and the Natural Language Processor. The three modules of this research project are interconnected as shown in figure 3.1. The first two modules (the World Model and the Robot Control System) are described in detail in the following sections of this chapter. The third module, the Natural Language Processor, is the major focus of this research and is described in Chapter 5. The vocabulary used for this project was a topic of much discussion. It is important that the vocabulary go beyond any that has been interpreted by other natural language processing systems. It is also important that the vocabulary not be a set of hand-picked words, but have a sound foundation behind why it was chosen. Therefore, the vocabulary warrants its own chapter (Chapter 4).

3.1 The World Model

The world model represents the virtual physical world in which the robot operates. The reason for this module is to simulate the environment of the robot. I have chosen this method because I would like to study natural language processing in a realistic environment, but without having to build a robot. The world model

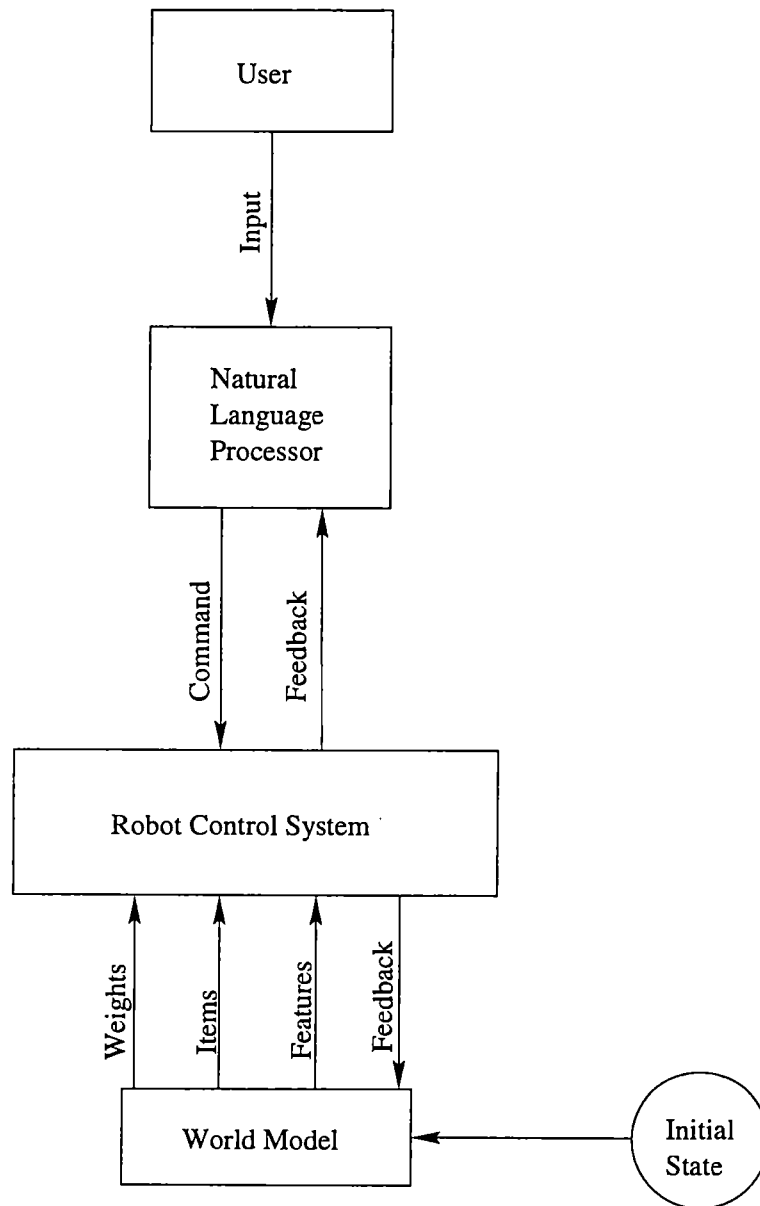


Figure 3.1: System Diagram

reads the input from the initial state and stores it in an appropriate data structure. This information includes: the number of rooms, the types of items in each of the rooms, and the features for the items in each room. The features include: color, texture, size (height, width, length), manufacturer or author, usability (length of time), and location on a two dimensional map.

Therefore the world model provides information to the robot control system as to where to find a particular item and the types of features associated with a particular item. This information may be provided on demand, via data structures, or one time only by use of an input file.

The world model is implemented in C++. There is a class called rooms which holds the room type, the item names, and the features for each of the items (as mentioned above). The actions allowed on this class are: list room types, add item, delete item, list item, check if item is present in a particular room, list the features of a particular item, and set up the output file for the middle (Robot Control System) module.

3.2 The Robot Control System

The robot control system is the second module in this project. The robot control system models, at a high level of abstraction, the robot's interaction with the physical world, in this case the world model. The robot control system first

takes input from the world model (items and features of the items) along with the necessary weights between them and learns what each of the rooms are. The weights between the items and features are predetermined by a survey as described below and give general information as to what items are typically found in a particular room. These weights represent the background knowledge of the robot about the “world” that it lives in. The specific items in each room (as determined by the initial state and the user) will determine what type of room it is. No item will be necessary nor sufficient for determining the type of room. For instance, I have run tests which show that the program learns a bedroom even if it does not have a bed, if the other items in it point to it being a bedroom.

The network weight matrix contains correlations measuring how well the individual items are associated within a “typical” house. The correlations are calculated from the results of a poll of one hundred people as to what rooms they think a particular item is found in. In the survey, which is shown in Appendix C, forty of the participants were male and sixty were female. Fifty two of the participants were single and forty eight were married. The oldest person who took the survey was sixty three and the youngest nine. The average age for persons participating in the survey was twenty nine. The point here is not to demonstrate that I have produced a statistically accurate survey, but to show that the method in which the weight matrix was produced was through the use of a survey in which some statistics have been gathered. I am certain that a much more sophisticated survey

could have been taken, but this one seems to serve the purpose, the purpose being to determine, in general, what items people believe belong in what rooms of a house.

The weight matrix is calculated using the following linear correlation coefficient as found in [124]:

$$r = \sum_i (x_i - \bar{x})(y_i - \bar{y}) / \sqrt{\sum_i (x_i - \bar{x})^2} \sqrt{\sum_i (y_i - \bar{y})^2} \quad (3.1)$$

where the value r gives the measure of association between variables. The value is a real number between -1 and 1, inclusive. The value is large and positive (i.e. closer to 1) if the values have a high positive correlation. In our model this means that the items or room and item are likely to be found together in a house. The value is large and negative (i.e. closer to -1) if the values have a high negative correlation. This means in our model that the items are not likely to be found together in a house. The value is close to zero if the values are uncorrelated. This means that there is no strong indication that they would or would not be found together in a house. Each individual x or y value corresponds to one entry in the survey. The mean values over x and y are the mean across that particular item for every room. For example, with x being the value for sofa and y being the value for easy chair, r represents how likely it is that sofas and easy chairs would be in the same room of a house.

The robot control system is implemented using C++. This choice was made through an investigation which led in several different directions. I first attempted to implement the model through building a Hopfield Network using the Stuttgart Neural Network Simulator. However, since I have eighty three inputs and eighty three outputs it had to build an eighty three by eighty three network, and given six hours on a Sparc 5 it could not do so. I abandoned this idea. If the Stuttgart Neural Network Simulator could not even build an eighty three by eighty three network in six hours, it was not going to solve the problem of neural networks not scaling up well.

I next attempted to build the model using NevProp (Nevada Propagation) which uses back propagation and quick propagation. After running several tests and getting some but not all to work, I determined that supervised learning was not appropriate for this module of this project.

Since I first got the idea for this project from the rooms example in the PDP Explorations book [95], I decided that the solution to this portion of the project was back in the PDP software. I first attempted to use the cs (constraint satisfaction) program that is presented in Chapter 3 of the PDP Explorations book [95]. I was successful at getting it to learn a particular room, but could not find a method to extend it to learn and remember multiple rooms of a given house.

After determining that the PDP constraint satisfaction program was not powerful enough for this project, I moved to the Interactive Activation and Competition (iac) program. It seemed to fit my model in some ways because there was interaction between the features, items, and rooms. There is also competition between the rooms. Once again I was successful at getting the iac model to learn one room at a time, but like the cs program it was not powerful enough to learn and remember all of the rooms. In Chapter 7 of the PDP Explorations book [95] the authors present a program which is an expansion of the iac model. They call it the Interactive Activation (ia) model. After studying the model and determining that it did seem to parallel this module of my project in some ways, I attempted to learn the C code that was used to implement it and to change it to fit this project. Through this process, I learned that their model did not fit my model as closely as I had thought. This led to the realization that I was probably not going to be able to use someone else's neural network code.

At this point, I made the decision to write my own neural network code. The neural network in the program performs constraint satisfaction using the schema model to classify each of the rooms of the given house. It is similar to the model presented in Chapter 3 of the PDP Explorations book [95]. This model is particularly appropriate for this project because what it attempts to do is to take a set of constraints and simultaneously satisfy them. Typically there is no perfect solution. The solution chosen is the one which satisfies as many of the constraints

as possible. The model is well suited for the room classification problem since there is no perfect answer and many times a number of the rooms will be activated, but the one which is most highly activated is the one which is chosen. This is more appropriate than, say, a supervised learning model in which a target output is present and over hundreds of trials the neural network learns the type of room based on past examples. We, as humans, typically learn to recognize a particular room long before we have seen hundreds of them.

Therefore, constraint satisfaction was chosen and implemented. The reason that this program was successful where the PDP cs program was not is that when the program receives the initial state from the bottom (world) module, it receives an entire house (not just one room) and learns each of the rooms in the house concurrently. The formula used to update the activations on each item and room from [95] is:

$$netinput_i = \sum_j w_{ij} \times a_j + input_i + bias_i \quad (3.2)$$

if $netinput_i > 0$

$$a_i = a_i + netinput_i(1 - a_i) \quad (3.3)$$

else

$$a_i = a_i + netinput_i a_i \quad (3.4)$$

for each unit i in the neural network. In these equations j represents all units other than i , w_{ij} represents the weight matrix value between units i and j , and $input$ represents the external input. The $bias_i$ is a constant input which is negative if the unit is usually off and positive if the unit is usually on. In this program, the $bias_i$ was set to zero so it had no effect. The external input in this case shows the absence or presence of a particular item in a given room. The goodness measure (or measure of how well the necessary constraints are being met) is then calculated by:

$$goodness_i = netinput_i a_i \quad (3.5)$$

When the program runs, the constraint satisfaction continues to run until the goodness stabilizes (reaches a peak) for each of the rooms in the house. This method has been tested on eighty-three rooms in twelve different houses and the neural network learned the appropriate room in all except one case. In that case, the room was a dining room which had several items that are not typically found in dining rooms, such as pantry, stereo, CDs, and tapes. The program incorrectly classified the room as a kitchen. The tests run on the robot control system are shown in Appendix O.

The neural network written for this module is the knowledge base of this project and manipulates the items in the house based on the user's requests passed

down from the natural language processing module. The robot control module does the following tasks: displays room activations, displays item activations, resets the screen for new tests, displays the weight matrix, displays the items in the house, adds an item to a room, deletes an item from a room, locates an item in the house, checks if an item is present in a given room, learns the room types, lists the items in a given room, determines what is near a specified location (x and y coordinates given as features), and prints the names of the learned rooms. The robot control system performs the operations listed above through interaction with the world model as described earlier.

The robot control system is a vital part of this project and is the knowledge base upon which the natural language is built. It is implemented with a neural network which successfully classifies, remembers, and modifies the rooms as given by the initial state in the world model. A diagram of the robot control system is shown in figure 3.2.

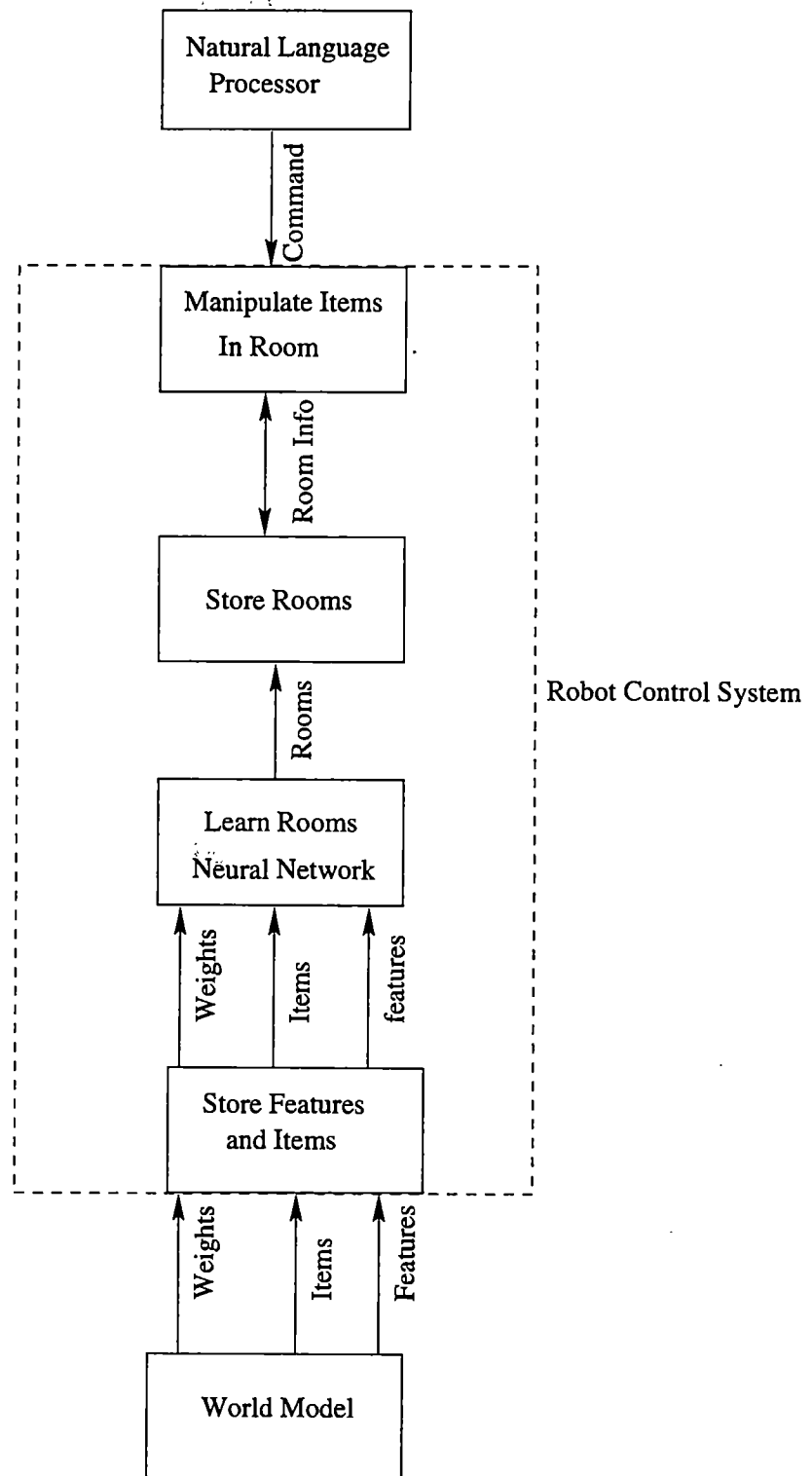


Figure 3.2: Robot Control System

Chapter 4

The Vocabulary

The size and content of the vocabulary for this project was an area of much concern. In order to show that this project can process natural language it was important to have a representative vocabulary. How large? As mentioned previously, the average size of the vocabulary for previous natural language systems was thirty-five words [55]. DISCERN [103], which is considered by many to be the state of the art in natural language systems with neural networks, was about fifty-five words. It was important to have a vocabulary that was large enough to be able to argue that this system actually processes natural language and is not just another “toy” system. How many words are enough? After implementing much of the project with a small, hand-picked vocabulary, it was determined that a much larger vocabulary was needed. Several different numbers were discussed as being large enough, but I had to choose vocabulary in an objective fashion.

Finally, it was concluded that the input vocabulary would consist of the Dolch Word List [31]. The Dolch Word List is a list of one thousand words which are considered to be the first one thousand words for children's reading. These words are used in many elementary schools to teach children to read. This list may well be called "basic" because it includes the "service words" that are used in all writing, no matter on what subject. [31] I determined that this was an appropriate input vocabulary because I must first be able to interpret natural language at the level of a child before moving on to process language at the adult level. In comparing the previously mentioned hand-picked list to the Dolch Word List, there were a number of words which had previously been a part of the vocabulary which were not in the Dolch Word List. These words were not removed from the input vocabulary. Therefore, the input vocabulary consists of one thousand and sixty words. A list of the words in the input vocabulary is shown in Appendix D. These are the words that can be typed as commands to the system. In other words, the system will not crash if given these words as input. However, the list of words that the system is able to comprehend and process is a subset of these words, as explained below. All words that are not in this subset, but which are allowed as input to the system, will simply be ignored by the system.

The next task in dealing with the vocabulary was to determine which of the words in the vocabulary would be understood by the system. As previously mentioned, we must be able to understand language at the level of a child before we

can understand language at the level of an adult. Children do not comprehend all of the words that they hear and children understand language at many different levels. For instance, there is a dramatic difference between the number and type of words that a two year old child understands and the number and type of words that a four year old child understands. It was determined for this project, that I would interpret natural language at the level of a two and one-half year old child. Based on the information that I have read, the vocabulary for a two and one-half year old child is approximately four to five hundred words. More importantly, though, is the type of language that a two and one-half year old understands. Since this project focuses on natural language interpretation rather than natural language generation, I am interested in the type of language that a two and one-half year old child understands rather than the type of language that a two and one-half year old speaks.

“There are two main points to notice here. First, expectably, production lags behind comprehension for all children in all age ranges, and for both word classes (nouns and verbs). Second, within either production or comprehension, verbs lag well behind nouns.” [74] It is commonly known that one of the first things that children learn are nouns. In the beginning, they know many more nouns than other parts of speech; they know only a few verbs. This fits perfectly into the system that I have designed, since I have a large number of items that are possible with the house and a much smaller number of actions that may be taken

on the items. These items are nouns which a two and one-half year old would be able to understand.

According to Brown [19], the first aspect of syntax that children are sensitive to is word order. Therefore, I built a system that is particularly sensitive to the order of the words. This will be well represented in this research project since "Put the soap in the kitchen" and "Put the kitchen in the soap" have entirely different meanings. In fact, the latter sentence is not legal within the system because you are not allowed to place rooms within objects. The system simply ignores commands that are not legal as long as the words are in the vocabulary. If the words are not legal words within the vocabulary, the system will reject them.

I have built a system whose goal is to understand language at the Stage II or Stage III level as identified by Brown [19]. Brown discusses the five stages of language in terms of fourteen grammatical morphemes. The fourteen grammatical morphemes (in order of acquisition) are: present-progressive, preposition in, preposition on, plural, past irregular, possessive, uncontractible copula, articles, past regular, third person regular, third person irregular, uncontractible auxiliary, contractible copula, and contractible auxiliary. I am focusing on Brown's study because his study is used to analyze the language of children by many in the fields of linguistics and psychology.

Brown states, "The sentences the child makes are like adult telegrams in that they are largely made up of nouns and verbs (with a few adjectives and adverbs)

and in that they generally do not use prepositions, conjunctions, articles, or auxiliary verbs. Articles were not used at all during the first two years." [19] Of the fourteen morphemes presented by Brown [19], the definite and indefinite articles have the most semantic complexity. My system responds to commands which have the article "the" before the object or room name and it also responds if the article is omitted. The Stage II child uses generic verbs. I also try to model this behavior by using only simple, generic verbs as commands. These are verbs that a two and one-half year old child understands.

According to Brown's study, "Seventy percent of the children's speech was of one of the following forms: agent and action, action and object, agent and object, action and location, entity and location, possessor and possession, entity and attribute, and demonstrative and entity." [19] The types of sentences that are allowed within this project typically deal with commands to an agent to take some type of action on objects within the house, so that the objects will be moved from location to location; therefore, I am modeling the language of a Stage II child.

"Recognition, anticipation based on signs, the concept of the enduring object, awareness of a single space that contains the self as well as other objects, are all developed in the period of sensorimotor intelligence (birth to two years). During this sensorimotor period the child learns to discriminate between herself and the environment and between elements in the world. The fundamental knowledge that the child acquires during this period concerns the location of objects, her power to

act on them, and their effects on one another.” [21] By having all language take place in a confined environment where objects must be discerned and acted upon, I am attempting to represent the sensorimotor intelligence of a two and one-half year old child. Research in the last decade has shown the dramatic effect of context on the way language is used and interpreted, making it clear that language cannot be realistically assessed if it is isolated from what is going on with the speaker and the listener. [88] Because more researchers are finding this to be true, I find that providing a domain of discourse, such as a house in this project, provides an appropriate environment in which to study language and how a neural network can learn language.

During Stage II the prepositions “in” and “on” begin to appear. The commands in this project will use these words since most two and one-half year olds understand them. Of the fourteen grammatical morphemes presented by Brown, only present progressive, the preposition in, and the preposition on were acquired by Stage III for all three children in his study. [19] The grammatical morphemes tend to be unstressed forms of little phonetic substance and considerable grammatical and semantic complexity. This is one reason that they are not learned as early as nouns, verbs, and adjectives. [19]

According to Lund and Duchan [88], at 24 months old, the child studied could answer “where” questions with place-naming answers. My system adequately models this behavior since one of the actions is “Where is objectX?” Even though

I am not generating language, this project was set up to interpret this type of language as a two year old would.

After doing the above mentioned research to determine the type of language that a two and one-half year old understands, I put it in the context of my input vocabulary (Dolch words plus previously hand-picked words). Which of the 1060 input words would a two and one-half year old understand? I did not want to look at the list and try to guess, but I did not feel that a formal statistically correct survey was necessary either. So, I asked a childcare administrator/owner, Rachel Hoffman, to tell me which of the words her two and one-half year olds understood. She took the list of 1060 words and unscientifically determined which of them the children could understand. The children she used were Tyler (23 months), Tate (28 months), and Meagan (37 months). The answers from her informal survey provided the list of words which are shown in Appendix E. I decided that I would choose my vocabulary from the Dolch words that Tate and Meagan (the two older children) both understood. This list contains 471 words.

Finally, once I had the Dolch Words which a two and one-half year old could understand (471 words), I had to further break down the word list to fit the environment of this research project. Not all of the words that a two and one-half year old understands fit into the domain of discourse for this natural language processing system. Remember that I am talking about the items in a house and the manipulation of those items. Some of the words did not fit into that domain

and therefore were omitted. I tried to be as flexible as possible in order to have a large vocabulary. For example, I allowed words like car, truck, and airplane. Cars, trucks, and airplanes are not typically seen in houses, but two and one-half year olds often speak of them and play with toy cars, trucks, and airplanes. Therefore, I allowed these items to be in the house. The final list of words contains 218 words. These are the words that this research project will interpret, i.e. the system is programmed to carry out commands which contain these words. These words are shown in Appendix F.

Chapter 5

The Natural Language Processor

5.1 Introduction

The natural language processing module is the final module of this research project. The other modules are support modules for the language processing. The natural language processing module is where the actual communication with the user takes place and the language is interpreted for the rest of the system to use.

The natural language processor takes the input from the user. It then interprets the input and determines the instructions to be sent to the robot control system. The manner in which the interpretation takes place is as follows: the input from the user is separated into individual words (where a word is a set of characters with spaces on either side), the words from the user are converted into

approximately orthogonal vectors, the vector stream (one vector per word of input) is entered, one vector at a time, into a neural network which converts it to a single output vector. The output vector is then converted into a command for the robot control system to execute. A diagram of the natural language processor is shown in figure 5.1.

The goal of the natural language processing system is to be the interpreter between the user and the robot control system. The system takes a command such as:

Move the chair from the kitchen to the living room.

The natural language processor interprets this command and sends the appropriate message to the robot control system. The program takes as input this series of input words and produces an output vector that corresponds to a command that the simulated robot in the system can interpret. The input words are approximately orthogonal bipolar, i.e. -1 or +1, vectors. Approximately orthogonal means that their inner (dot) product is approximately zero. Therefore, they have few or zero elements in common, or that the elements that they have in common tend to cancel one another. Each unique word has a unique input vector. A length of one hundred was chosen for the input word vectors because

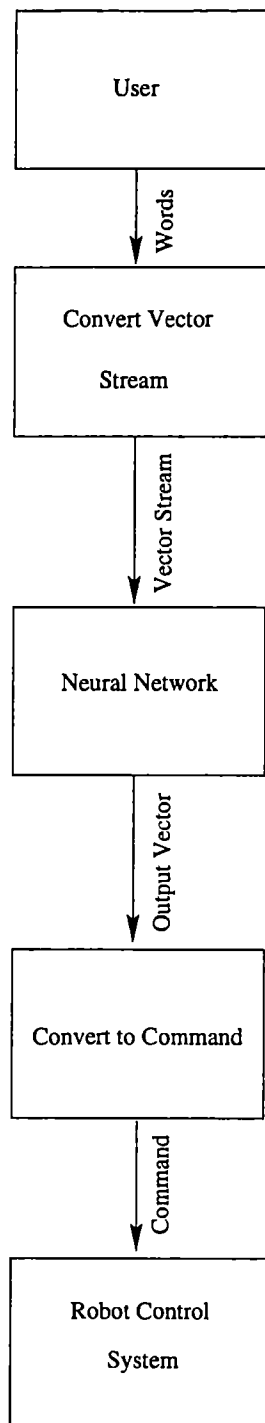


Figure 5.1: Natural Language Processor

it is sufficiently large so that the randomly generated bipolar vectors are, indeed, approximately orthogonal, and sufficiently small that it will allow the neural network model to scale up to solve larger problems. The binary output vector packet has the following four fields:

```
-----  
| Verb   | Object  | Source  | Destination |  
-----
```

For example, if the program receives as input the command:

Get the book.

The command is represented as four independent bipolar input vectors of length one hundred. One input vector represents the word "Get", one vector represents the word "the", one vector represents the word "book", and one vector represents the ".". The program would interpret this command and produce a binary output vector of length eighty-eight that is of the following format (Note: this diagram shows the format of the vector and not the vector itself, since it is not of length eighty-eight):

Verb	Object	Source	Destination	

00000000001	000000000101	000000000000	000000000000	

A length of eighty-eight was chosen because it is large enough to hold the necessary information, i.e. verb, object, source, destination, and small enough to be scalable.

The natural language processor is implemented using a neural network. Several different neural networks were implemented. Three recurrent neural networks were built using Hebbian learning, one recurrent neural network was built using back propagation, and one nonrecurrent neural network was built using back propagation. As mentioned previously, Hebbian learning is much faster than back propagation. Since one of the initial premises of this research project is to show that neural networks could scale up to solve large problems, I spent most of my research effort implementing the three recurrent neural networks using Hebbian learning. I will discuss in detail the implementation of each of the neural networks and all of the variations that were used to interpret natural language. In Chapter 6 I will give the results of this research project.

5.2 Neural Networks with Hebbian Learning

5.2.1 Introduction to Neural Networks with Hebbian Learning

The first neural networks implemented were recurrent neural networks with Hebbian Learning. Neural networks were explained in Chapter 1. In a neural network, a unit is an individual element which may take on an activation value and pass that activation value on to other units. It corresponds to a single neuron in the brain. A unit's activation value is how strongly it is being excited by other units. We refer to each set of units which perform a given task as a layer. There is an input layer, an output layer, possibly one or more hidden (between input and output) layers, and possibly a context layer. A hidden layer is such because it is invisible to the user of the neural network, only the input and output layers are visible. This is the layer where the neural network builds its internal representation of the data, i.e. builds up its mapping of the input layer to the output layer. A recurrent neural network is one in which cycles exist. In other words, certain layers (output layer or hidden layer) are copied back into previous layers. The context layer is the layer that is used to store the value of the output (or hidden) layer on the previous input. Therefore, the output (or hidden) layer is copied back to the context layer before a new input is received. The purpose of having cycles in the network is to provide a sort of memory. The neural network can "remember" what it has seen before by saving the representations created by the neural

network from previous inputs. The manner in which this "remembering" occurs is as follows. The neural network receives an input and updates the activations of the hidden and output layer units based on this input. The neural network has now build a representation of that input. This representation from the output (or hidden) layer is copied to the context layer so that it may be remembered. The neural network is then ready to receive a new input.

The first neural network with Hebbian learning investigated was the simple recurrent network (SRN). A diagram of this network is shown in Appendix G. The SRN network works as follows: The input units are connected to the hidden units. Also connected to the hidden units are the context units (sometimes called state units). The activation values of the context units are the same as the activation values of the hidden units on the previous input. The network "remembers" what it has seen before in this manner. This type of "recurrent" behavior is what defines a recurrent neural network and allows the input to consist of multiple input vectors to be associated with a single output vector. Finally, the hidden units are connected to the output units. This neural network is NN1 in Appendix J. Appendix J shows a summary of the tests that were run on the different neural networks in the form of a table.

The next neural network with Hebbian Learning investigated was the Jordan [67] recurrent network with no hidden layers. A diagram of this network is shown in Appendix H. The Jordan network with no hidden layers works as follows: The

input units are connected to the output units, the context units are also connected to the output units. For each input vector, the activation value of the context units is the activation value of the output units on the previous input. This is also a recurrent neural network, but the recurring connection is between the output layer and the context layer as opposed to being between the hidden layer and the context layer as seen in the SRN. This neural network is NN2 in Appendix J.

The final neural network with Hebbian learning investigated was the Jordan network with one hidden layer. A diagram of this network is shown in Appendix I. The input units are connected to the hidden units. The context units are also connected to the hidden units. The hidden units are connected to the output units. The activation value of the context units is the activation value of the output units on the previous input. This neural network is NN3 in Appendix J.

Hebbian learning was used for each of the three neural networks described above. The premise behind Hebbian learning is that if two connected neurons are active simultaneously, then the strength of the connection (expressed in neural network terms as the weight (w_{ij})) between them should be increased. [41] The Hebb rule is the first and best known method for learning in neural networks. The weights are initialized to a small random number. The weights are then updated using the following formula from [96]:

$$\Delta w_{ij} = \epsilon \times a_i \times a_j \tag{5.1}$$

where ϵ is the learning rate, a_i is the activation value for unit i , and a_j is the activation value for unit j . The initial activation values are described in Section 5.2.4. The activations are then updated using the following formula, also from [96]:

$$a_i = \sum_j a_j w_{ij} \quad (5.2)$$

Therefore, the activation of unit i is affected by the activations of all other units by the amount of the weight connecting them.

Hebbian learning is also called correlation learning because, using the above formulas, the weight is proportional to the correlation between the activations of units i and j . In other words, if the two units are simultaneously active, i.e. have high activation values at the same time, then the weight is large. Otherwise, the weight is small or negative.

The following tests were run on each of the three recurrent neural networks with Hebbian learning. However, since many of the results were the same, the tests will each only be described once. Where the neural networks differed in their results, it will be noted with the appropriate tests. Obviously, any of the tests done with hidden layers were not done with the Jordan network with no hidden layer. So that my explanations will be clear, I will discuss the neural network in the singular even though the tests were performed on all three neural networks.

When the term “neural network” is used it can be assumed that I am discussing the one in which a particular test is being run.

All of the neural network programs were written in C++. Once the neural network architecture (i.e. SRN or Jordan) was designed and the learning algorithm (i.e. Hebb rule) implemented, many variations of parameters were used to produce the appropriate response from the network. The following sections describe the different parameters that were used.

5.2.2 Tests on Input Vectors

Several tests were run which pertained to having the correct set of input vector values. As mentioned previously, the input vectors consist of a bipolar vector of size one hundred for each word (Test A in Appendix J). For a given command there may be as few as two or as many as nine of these words. It is because of this method of input that a recurrent neural network was chosen. Otherwise, it would have been necessary to have one input string that is the length of a command, rather than the length of one word. Originally, I had decided that the input vectors to the system would be bipolar (i.e. -1 or 1) vectors which were randomly generated for each word. The reason being that sufficiently large bipolar vectors which are randomly generated can be considered approximately orthogonal because the elements that any two of them might have in common can be cancelled out by the elements that they do not have in common. In order

for the Hebb learning rule to work, the input vectors must be approximately orthogonal. Therefore, large randomly generated bipolar vectors were used. After having difficulty getting the neural network to learn, I used binary input vectors, even though large randomly generated binary vectors are not orthogonal because the elements that they have in common cannot be cancelled out by the elements that they do have in common (Test B from Appendix J). The results achieved with these input vectors were that the weights for the neural network never became negative. Clearly, it happened because the only signed value going into the weight update rule is the input vector value (as shown in Equation 5.1). With this value always being zero or positive, the weight matrix values never became negative. Since it is desirable in some cases for the weight matrix values to become negative, I decided to stay with the bipolar input vectors instead of the binary input vectors.

A second test was run with the input vectors. Since the activation values were getting out of the acceptable range (0 to 1) very quickly, I scaled down the input vectors. I took the bipolar input vectors and normalized them (Test C from Appendix J). The normalization value is a scaled down version of the original value, but kept in the same proportion to the other values. The formula used to normalize [41] the input vectors is as follows:

$$a_i = s_i \times 1 / \sqrt{\sum_i s_i^2} \quad (5.3)$$

where s_i is the i^{th} element (unit) of the input vector and a_i is the value of the initial activation for unit i . Normalizing the input vectors did not help the neural network to learn the training patterns better, only more slowly. Therefore, the input vectors were not normalized.

5.2.3 Tests on Target Vectors

Two tests were run pertaining to the target vectors. Since the neural network learns using Hebbian learning, the appropriate weight matrix is built using the input vectors along with their corresponding target vectors. This is called a supervised learning technique. With supervised learning, during the training phase the neural network has knowledge of what the target output is, i.e. it knows the correct answer and has to train to get the weight matrix values such that during the testing phase, given an input it will produce the correct output without being given the answer. The target vector in this system is a binary vector of length eighty eight which represents the correct output of the neural network for a given command. I investigated several options as to what the target vector should be, but only the final version (described in Section 5.1) was used to run the tests on the neural network.

The first test that was run pertaining to the target vectors was to determine how often a target vector should be presented to the neural network in order for it to learn the correct answer. Two options were studied: the target being

presented to the system once per word (Test D from Appendix J) and the target being presented once per command (Test E from Appendix J). The neural network should not produce the target vector for the entire command after being given only a portion of the input words, therefore, the target vector is presented only once per command. Also, the target does not change as the input words are entered.

A second test was to use binary (ranging from 0 to 1) target vectors versus bipolar (ranging from -1 to 1) target vectors. When the neural network was given bipolar target vectors (Test G from Appendix J), it did not learn the input patterns better than with binary target vectors (Test H from Appendix J). Therefore, I used binary targets.

5.2.4 Tests on Initial Parameters

I ran tests on the neural networks with Hebbian learning by changing the setting of initial parameters before training the neural network. I first initialized all of the values in the weight matrix to zeros before training the neural network (Test I from Appendix J). Due to the type of mathematics used to update the activations and the weight matrix (see Equations 5.1 and 5.2), the weights remained at zero for the entire duration of the training. The reason is that the activation function is applied before the weight value is updated and becomes zero since one of the factors is the weight. The weight then becomes zero because the activation is zero.

I then initialized the weight values using a priori knowledge of the network, i.e. I set up a bias value to guide the weight matrix values toward the correct answer (Test J from Appendix J). In some ways this helped the neural network to keep from going too far astray, but it did not steer the neural network in the right direction. So I initialized the weight matrix to small random values before training the neural network (Test K from Appendix J). By small, in this instance, I mean between -0.1 and 0.1, inclusive. This method of initialization received the best results.

A second test involving the initialization of parameters before training of the neural network was the initialization of the activation values for each of the units in the neural network. Two different values were used as the initial activations, 0.0 (Test L from Appendix J) and 0.5 (Test M from Appendix J). The outcome of the two were the same because the only activation values that are not changed before being used are the activation values of the context units. Initializing the input activation values is useless due to the fact that the input unit activation values are set based on the input word with values of -1 or +1. Similarly, the hidden unit activation values are calculated using the weight matrix and the input unit activation values. The output unit activation values are calculated using the weight matrix and the hidden unit activation values. The exception to the above statement is the Jordan neural network with no hidden layer. In this network,

there are no hidden unit activation values and the output activation values are calculated using the weight matrix and the input unit activations.

I initialized the activation values to the mid-range value of the context units for each particular network. In the SRN, the context units are copied from the hidden layer and have the range -1 to +1, so they were initialized to 0.0. In the two Jordan networks, the context units are copied from the output layer and have the range 0 to +1, so they were initialized to 0.5.

5.2.5 Tests on Updating Weight Matrix

This test involved determining how often the weight matrix values would be updated. In recurrent networks there are three possible solutions to this problem: update the weight matrix once per input word (Test N from Appendix J), update the weight matrix once per input command (Test O from Appendix J), and update the weight matrix once per trial (i.e. after all input commands have been seen once) (Test P from Appendix J). When updating the weight matrix values once per input word, the overall error would decrease initially and then increase (as you will later see this seems to be the general tendency for the neural networks in all of the tests that were run). The difference in this test and the others was that updating the weights on every input word makes the neural network run very slowly and the results obtained were very dependent on the particular commands being tested.

The next test which was run was to update the weight matrix values once for every input command. This method of updating did not produce the correct output, it runs slowly, and the results depend on the particular commands being tested. In this case, the neural network cannot help but to learn later commands in the sequence better.

The final test to determine when to update the weight matrix values was to update the values once per trial. In other words, compute the change in the weight matrix on each input word received, but actually change the weight matrix values when all input commands had been seen once. Then reset the change in weight matrix value to zero and compute it over all input commands again. This method ran faster than the others but did not produce significantly different results. It is for these reasons the weight matrix was updated once per trial.

While running the above mentioned tests, it was difficult to determine the size of the learning rate, given the number of input patterns. Each time that a new test was executed, the learning rate would have to be updated a number of times to get it to produce the correct results, based on the size of the input, i.e. the number of commands for which the neural network was being trained (Test Q from Appendix J). I produced a method for updating the learning rate through a linear interpolation formula based on the number of input commands that are being learned (Test R from Appendix J). This simplified the running of tests and the mathematics is not complex enough to slow down the neural network.

5.2.6 Tests to Solve Unlearning Problem

After running many tests I determined that regardless of the neural network used, the initial values, the learning rate, the method of updating the weight matrix, the type of input vector, or the type of target vector, the Hebbian learning neural networks produced the same results. The error rate over all input commands goes down quickly at the beginning of training and then slowly moves upward for the rest of the training. The output results are such that two of the three neural networks, the two with hidden layers, will learn all commands given to them. However, it should be noted that the tests run were on twenty or fewer input commands. Even with this few commands, it seemed promising.

The problem being that the neural network will forget one or more commands as it learns the others. It will never learn more than a few commands at any given time. When two similar commands are given as input, the neural network will learn the first of the two commands. But then, as the neural network proceeds to learn the other similar command, the weak activations from the first command become so strong over many trials that they change the output of the neural network for the first command, and so it now will learn the second command but no longer "remembers" the first command. I will give an example. In this example, specific inputs and outputs are not identical to those given to or produced by the neural network. They have been simplified to make the explanation more clear.

Assume that the neural network receives the two input commands:

Get the sofa.

Get the chair.

and assume that the desired output vectors for these two commands are very similar, say

001001 Get the sofa.

001010 Get the chair.

The neural network will learn the first command, "Get the sofa.", i.e. it will produce the desired output vector when given the input vectors corresponding to the words of the command. However, as the neural network learns the second command, "Get the chair.", the response that the neural network gives to the first command becomes a combination of the two desired output vectors, i.e. 001011. It should be noted that the order in which the input commands are presented to the neural network is irrelevant. The neural network would perform the same with learning "Get the chair." before learning "Get the sofa.".

From now on I will refer to the problem described above as the "unlearning" or "forgetting" problem of the neural network. All three Hebbian learning neural networks which were tested suffered from this problem. This problem is the reason that three different recurrent neural networks with Hebbian learning were tested. Later in this section, I will give conclusions as to why the neural networks were

forgetting learned commands. First, I will describe the tests that were run to solve the problem.

A number of solutions were attempted to solve this problem. The first is the scaling of the weight matrix values. As noted in many neural network books, the smaller the weight matrix values, the better the generalization of the neural network. The method used was to multiply the weight matrix value by the reciprocal of the number of units in the network (Test S from Appendix J). By scaling the weight matrix values in this manner, they stay in a reasonable range, otherwise, they get large (close to 1) very quickly (Test T from Appendix J). This did not solve the unlearning problem.

The next method tested to solve the unlearning problem of the Hebbian neural networks was the use of different output ranges for the hidden unit activation functions. Since an activation function was used that was similar to a simple linear threshold function, this was a simple task. The activation is computed as in Equation 5.2. If the value computed stays within the range, then it is left unchanged. If it does not stay within the range, it is simply brought back to the boundary of the range. The ranges that were tested were the obvious ones: binary (Test U from Appendix J) and bipolar (Test V from Appendix J). The neural network learns more patterns and remembers them for a longer period of time when bipolar values are used as the output of the hidden units. Therefore, the threshold values are set for the bipolar range.

5.2.7 Tests on Hidden Layer Size

Another set of tests that were run on the two neural networks with the hidden layers and not on the neural network without the hidden layer was that of determining what size the hidden layer should be (Test W from Appendix J). This is a particularly important question for the SRN because the context layer is the same size as the hidden layer, so the decision made here will greatly affect the overall size and speed of the neural network. Since neural networks have a large amount of mathematics associated with each unit in the network, it is desirable to have as few units as possible in order to speed up the execution time of the network.

The two networks were tested with hidden unit layer sizes of from seventy to three hundred and fifty units in increments of thirty. The sizes chosen to test were simply through trial and error. A neural network with a hidden layer size smaller than seventy did not learn any of the commands. A neural network with a hidden layer size larger than three hundred and fifty did not learn more commands than those with smaller hidden layers, but slower. The results obtained were not drastically different from one end of the spectrum to the other. The best results for the SRN were achieved with a hidden layer size of two hundred units. The best results for the Jordan network were achieved with a hidden layer size of one hundred and thirty units.

5.2.8 Tests on Weight Update Rule

It was possible that the neural network was finding a local minimum (value of lowest local error), and was not jumping out of it. In neural networks of this type, you train until you reach the minimum error value for this training set. As the neural network moves down the gradient curve of the error, it sometimes gets stuck in an area where all of the values surrounding it are larger, so that it is the lowest local error, but not the lowest overall error. This is a local minimum. Neural networks will sometimes find these local minima and then move on to find the global minimum, but often they are unable to “jump” out of these local minima. To solve this problem a momentum term was added to the updating of the weight matrix values (Test X from Appendix J). This term “remembers” the previous value of the weight matrix and tries to keep the overall system moving in the right direction by using this previous value to compute the new value. The updated weight matrix formula is as follows:

$$\Delta w_{ij} = \varepsilon \times a_i \times a_j + momentum \times w_{ij} \quad (5.4)$$

The last term uses a coefficient (*momentum*) to determine how much the previous value should affect the overall weight matrix value. I used coefficients from 0.1 to 1. For obvious reasons, adding the momentum term to the weight matrix update added complexity and, therefore, training time to the neural network. However,

once the neural network was moving in the right direction, it moved faster. This made the neural network learn each of the input commands faster, but did not solve the unlearning problem, i.e. the neural network still forgot commands that it had learned before learning others.

The next possible solution to the unlearning problem was to send the weights through a hyperbolic tangent function in order to keep them small (Test Z from Appendix J). The formula used is from [41]:

$$h(x) = 1 - \exp(-2x)/1 + \exp(-2x) \quad (5.5)$$

This caused the neural network to learn more slowly, in terms of computation time as well as the number of trials needed to learn a given input command. This addition of the hyperbolic tangent function to the update of the weight matrix values did not solve the unlearning problem.

After considering the unlearning problem and the fact that weakly activated values were making the neural network forget commands over time, I implemented a method to keep these “weak” values from affecting the activations so strongly. I first set to zero any weight that was close to zero (Test AA from Appendix J). The problem that arises here is: What does it mean to be “close to zero”? Is 0.1 close to zero or is 0.000001 close to zero? Many parameters go into this determination, in particular the learning rate and the scaling of the weight matrix values. What

it means to be close to zero is dependent on these as well as other factors. None of the values used solved the unlearning problem of the neural network.

Another, similar solution was that of suppressing weak correlations (Test AB from Appendix J). In other words, I drew an invisible line on either side of zero on the x-y axis. All correlations within this boundary were subtracted from the overall change in the weight matrix and all correlations outside of this boundary were added to the overall change in the weight matrix. I name this boundary the threshold.

The reasoning behind this test is that it is the weak correlations that are causing the unlearning. Therefore, by taking the weak correlation values and subtracting them rather than adding them, it may solve the problem. These weak correlations will not be allowed to build up over time and cause the neural network to produce ones where the output should be zeros. The problem is similar to the previous problem: What is a weak correlation? Again, this depends on many factors. Primary among these factors is the learning rate. It took five hours of tweaking numbers to move from learning six input commands to seven input commands. Finding the correct threshold to go with the correct learning rate to get eighty thousand input commands learned was like searching for a needle in a haystack. Solving the problem in this manner would also make it difficult to argue that the solution was scalable to larger problems.

All of the above mentioned tests were run in an effort to change the weight matrix update rule to make the neural network learn more commands. Another method to make the neural network learn more commands is to change some of the parameters in the activation function. As previously mentioned, the activation function used is similar to the linear threshold function. With this function, you simply compute the new activation, based on the inputs to that unit and the weight matrix values. After this computation is done, you check to make sure that the activation values stay within a certain range. Therefore, you compute a linear activation function and then do not let it go beyond a certain threshold on each end of the range.

5.2.9 Tests on Activation Function

The first thing added to the activation function shown in Equation 5.2 (Test AC from Appendix J) was a bias. With a bias, you look at the data and determine a priori what bias values might help to move the network toward the correct outputs. This is basically a guessing game. A certain set of bias values worked for some input commands, but then they would not work for other input commands. I determined that it is best to let the neural network train on its own.

The next change to the activation function in order to improve the learning of the neural network was to send the activations, once computed, through a binary sigmoid (logistic sigmoid) in order to keep them from getting too large, too quickly

(Test AD from Appendix J). A binary sigmoid function is a function which scales the values down while keeping them in the same proportion to the other values. This might help the neural network to solve the unlearning problem by keeping the weak activations even smaller. It only made the neural network train much more slowly. This was not the solution to the unlearning problem.

Finally, in an attempt to keep the activations from growing too large as a result of weak correlations, I applied a decay term to the activation values (Test AE from Appendix J). This term is meant to make the values slowly get smaller over time, i.e. decay. With this term added to the activation function, none of the neural networks were able to learn a single input command.

Results from the running of these neural networks and the conclusions drawn from the test data are presented in Chapter 6. The tests include learning rate versus error rate and hidden layer size versus error rate.

5.3 Recurrent Neural Network with Back Propagation Learning

After being unsuccessful with the Hebbian learning, I implemented back propagation learning because it has been successful in solving a number of neural network problems. Back propagation, in fact, is the reason behind the resurgence of neural networks for solving artificial intelligence problems in the 1980's and 1990's [41].

The following is an explanation of the recurrent neural network with back propagation learning implemented for the natural language processing portion of this research project. The network implemented is a SRN (Simple Recurrent Network). As discussed previously, a SRN is a recurrent neural network where the recurrence is between the hidden units and the context units. A diagram of an SRN with back propagation learning is shown in Appendix K. As with the SRN with Hebbian learning, the SRN with back propagation learning has a single hidden layer.

The inputs to the neural network are bipolar vectors representing each of the individual words within a given command. The vectors are of length one hundred, therefore the input layer is of size one hundred. Even though orthogonal input vectors are not necessary for back propagation learning as they are with Hebbian learning, they were used because it was more consistent to stay with the same type of input. The input vectors are not normalized before being sent to the neural network.

The target values presented to the network are binary vectors of length eighty eight. The outputs from the network are also binary vectors of length eighty eight. The neural network has a single hidden layer. Experiments were done with the hidden layer size, and thus the context layer size. The results of these experiments are shown in Chapter 6.

As with the recurrent neural networks with Hebbian learning, the target vector is presented to the network once per input command, as opposed to once per input word. The weight matrix is initialized to small random values (-0.1 to 0.1, inclusive) before learning begins and the context units are initialized to 0.5 before training begins for each input command. The weight matrix is updated once per trial. A momentum value is used to update the weight matrix and the weights are also scaled by the reciprocal of the number of units in the network each time they are updated to keep them small. Now that I have presented the architectural information about the neural network, I will present the details associated with back propagation learning.

As mentioned previously, back propagation learning is much slower than Hebbian learning. That is the reason a back propagation network was not chosen in the first place. Back propagation learning works as follows: the activations for all units are computed, the error measure for the output layer units is computed (as shown in Equation 5.7), this error measure is then sent back to the hidden layer units and used to compute the change in weight value for that layer, then the hidden layer must be copied to the context layer. All of this must be done for each input vector, i.e. word of input. Once an entire command has been entered, the process begins again for the next command. It is easy to see why this method of learning is slower than Hebbian learning as the error value has to be “back propagated” to the other layers before their weights can be updated.

Back propagation, according to [41], is a gradient descent method to minimize the total squared error of the output computed by the neural network. Gradient descent is where the algorithm continues to move along the error gradient to the lowest point in the error field. “The basic idea of the back propagation method is to combine a nonlinear perceptron-like system capable of making decisions with the objective error function of least means squared and gradient descent.” [96] In other words, we are using an error measure to perform gradient descent. We are moving along the gradient to the point where the error is least. The way that the back propagation algorithm works is to move through the neural network computing the new activation values for each unit in the network, as follows:

$$a_i = \sum_j a_j \times w_{ij} \quad (5.6)$$

where a_i is the activation value for unit i , a_j is the activation value for unit j , and w_{ij} is the weight between units i and j . Once the activations are computed, the total of the least means squared error (tss) is computed for each output unit, as follows:

$$tss = \sum_p \sum_i (t_i - a_i)^2 \quad (5.7)$$

where p is a given input pattern (or command), t_i is the target value for output unit i , and a_i is the actual activation value for output unit i . Finally, the error

values are passed back through the network in the opposite direction in order to compute the change in the weight matrix values, as follows:

$$error_i = t_i - a_i \Delta w_{ij} = error_i \times a_i \times (1 - a_i) \times a_j \quad (5.8)$$

where $error_i$ is the error value for unit i , w_{ij} is the weight between units i and j , a_i is the activation value for unit i , and a_j is the activation value for unit j . The Δw_{ij} computed above contains the partial derivative of the error with respect to a change in the net input of the unit.

The above process continues for all input patterns and then the weight matrix values are updated for each trial, as follows:

$$w_{ij} = \epsilon \times \Delta w_{ij} + momentum \times w_{ij} \quad (5.9)$$

where ϵ is the learning rate, w_{ij} is the weight between units i and j , and *momentum* is the term which remembers the previous value of the weight and helps the neural network to move in the right direction. Momentum was discussed in more detail in section 5.2.

Several tests were run with the SRN with back propagation learning. The tests include learning rate values versus error rate and hidden layer size versus error rate. The results of these tests and the conclusions drawn from those results are presented in Chapter 6.

5.4 Nonrecurrent Neural Network with Back Propagation Learning

The final neural network that was implemented for natural language processing was a nonrecurrent neural network with back propagation learning. A diagram of this network is shown in Appendix L. The neural network has a single hidden layer. The input vector is of length eight hundred. It is made up of bipolar values. The input vector is simply a concatenation of the vectors for each of the words of the command. For example, if the command is "Where is the book?" The input vector will have five hundred meaningful values, with the first one hundred being the vector for the word "Where", and the last being the vector for the "?". Since the commands are of different lengths, the rest of the input vector is filled with -1 values since it is a bipolar vector (which corresponds to 0 values in a binary vector). It is possible to use 0 values, but -1 values made the statistics which are presented in Chapter 6 easier to understand and to explain.

The output vector is of length eighty eight. It is made up of binary values. It is organized as described in Section 5.1 and is the same as the output vectors for the other neural networks. The target vectors are also of length eighty eight. Since this is not a recurrent network, there was no decision to be made as to how often to present the target vector. It was presented to the neural network once for each command.

The weight matrix is initialized to small random values (-0.1 to 0.1, inclusive). As this is not a recurrent neural network there are no context units. The weight matrix is updated once per trial which means that as each input command is presented the change in the weight matrix is computed, but the weights are not updated until all commands have been presented. A momentum value is used to update the weight matrix. The weights are also scaled by the reciprocal of the number of units in the network in order to keep them small. A bipolar sigmoid is used to scale the activation values to the appropriate range (-1 to +1). The formula used is from [41] and is as follows:

$$a_i = 2/(1 + e^{-\sigma a_i}) - 1 \quad (5.10)$$

where σ is a steepness parameter which is set to one.

Learning in this neural network occurs just as it does in the recurrent back propagation network described in Section 5.3. The only difference being at the end of each step in the learning process, the output value of the hidden layer units is not copied back onto context layer units because this is not a recurrent neural network. Equation 5.6 is used to update the activation unit values, this is used in combination with the bipolar sigmoid in Equation 5.10. Equation 5.7 is used to compute the total sum of squares error value. Equation 5.8 is used to compute

the change in the weight matrix value for each command entered. Finally, Equation 5.9 is used to update the weight matrix on each trial.

Several tests were run with the nonrecurrent back propagation network. The tests include learning rate versus error rate and hidden layer size versus error rate. The results of these tests and the conclusions drawn from those results are presented in Chapter 6. As you will notice, the best results were achieved with this network.

Chapter 6

Results and Conclusions

6.1 Introduction

Four of the five neural networks implemented were not performing the natural language processing task that I had originally set out to perform. Regardless of the Hebbian neural network used, the initial values, the learning rate, the method of updating the weight matrix, or the type of target vector the neural networks will forget one or more similar commands as they learn new ones. When two similar commands are given as input, the neural networks will learn one of them and as it learns the other similar command, it will unlearn the previous one. This problem was discussed in greater detail in Section 5.2.6.

I focused on three questions that arose as to why this was the case. The questions focus on Hebbian learning. The reason for this is that the goal of

this research project was to solve the natural language interpretation problem with Hebbian learning. Therefore, since the neural networks did not successfully interpret natural language, I use the results of the tests performed in Chapter 5, along with the following data, to show that Hebbian learning does not work in recurrent neural networks for natural language processing even when the input vectors for the words of the language are approximately orthogonal.

In the tests done in the following sections, I calculate the covariance matrix for vectors as well as the Euclidean distance between vectors in order to measure their similarity. Therefore, I will define those terms here so that I do not have to revisit them in each section.

The covariance matrix is computed by calculating the inner product of each two vectors to produce the corresponding element of the covariance matrix. For example, the inner product of the 17th vector with the 22nd vector will produce element [17,22] in the covariance matrix. The covariance matrix value for two identical, bipolar vectors should be the length of the vector. The covariance matrix value for two orthogonal, bipolar vectors should be zero. Therefore, the number computed should be between zero and the length of the vector. The closer to zero, the more different the two vectors. The closer to the length of the vector, the more alike the two vectors.

The Euclidean distance is the distance between two points (vectors) in a Cartesian space. The Euclidean distance was computed using the following formula:

$$D = \sqrt{\sum_i (x_i - y_i)^2} \quad (6.1)$$

where x_i is the value of the i^{th} element of vector x and y_i is the value of the i^{th} element of vector y . The Euclidean distance between two identical, bipolar vectors is zero. The Euclidean distance between two orthogonal, bipolar vectors is 1.4 times the square root of the length of the vector, based on the following derivation.

$$D^2 = (x - y)^2 = ||x||^2 - 2xy + ||y||^2 = ||x||^2 + ||y||^2 = 2 ||x||^2 \quad (6.2)$$

and since

$$||x|| = \sqrt{\sum_n x^2} = \sqrt{n} \quad (6.3)$$

then

$$D = \sqrt{2} ||x|| \quad (6.4)$$

$$D = \sqrt{2} \sqrt{n} \quad (6.5)$$

Therefore, when these distances are computed for a pair of vectors the values should be between zero and 1.4 times the square root of the length of the vector. The closer to zero, the more alike the two vectors. The closer to the 1.4 times the square root of the length of the vector, the more different the two vectors.

6.2 First Question

The first question was “Are the input vectors for the individual words strongly correlated?” If so, then Hebbian learning will not work well. “If the input vectors are uncorrelated (orthogonal), then the weight matrix found by the Hebb rule will give perfect results for each of the training input vectors.” [41]

In order to determine if the input vectors for the individual words are strongly correlated, I built the covariance matrix for the input vectors. The covariance matrix, as previously mentioned, was computed by calculating the inner product of each two vectors to produce the corresponding element of the covariance matrix. The covariance matrix included all 1060 words (input vectors) in the vocabulary, which are listed in Appendix D.

I anticipated that the answer to the question would be “no”. The reason for this is that I used large, randomly generated bipolar vectors as the input vectors for the individual words in the language. Large, randomly generated bipolar vectors are approximately orthogonal, therefore, for any two chosen words in the language, the input vectors are approximately orthogonal. For two vectors to be orthogonal, their inner (scalar) product must be zero. Below I will show that the inner products of any two vectors are approximately zero and, therefore, approximately orthogonal. This means that they are not strongly correlated.

After creating the covariance matrix, I took the average across several regions. The average value of the covariance matrix (all elements) was 0.0897. As the inner product of any two “different” vectors should be approximately zero, but the inner product of a single bipolar vector with itself should be the length of the vector, I computed the values for the covariance matrix for the diagonal elements (a vector with itself) and the nondiagonal elements (a vector with another vector) separately. The values of the covariance matrix diagonal elements were all 100, i.e. the length of the vector, as they should be. The average value of the covariance matrix nondiagonal elements was 0.0046. The value of the covariance relative to the length of the vector is, therefore, 4.6×10^{-5} . With this value I can conclude that the input vectors for the individual words are approximately orthogonal and not strongly correlated.

If the input vectors for the individual words of the vocabulary are not strongly correlated, then Hebbian learning should work. The results discussed above were encouraging as they showed that Hebbian learning could solve the natural language processing problem.

6.3 Second Question

The second question was “Are the input vectors for a given command strongly correlated?” If so, then Hebbian learning will not work. This is simply an exten-

sion of the first question as it pertains to nonrecurrent networks since recurrent neural networks have multiple input vectors for a single input.

In order to run this test I could not use the input vectors that I used for the recurrent neural networks. The reason being that the input vectors for the recurrent neural networks only represent a single word in the command, not the entire command. However, I still needed to determine if the input for a given command was strongly correlated with the input for another command. I created large vectors which were made up the concatenation of the vectors for each of the words in the command. You will notice that the content of this vector is exactly the content of the vectors for the individual words of the command. However, the meaning (and use) is much different. Because the commands may have one (or usually more) words in common, these vectors are no longer orthogonal.

The tests done in this section involve the input vectors used for the nonrecurrent back propagation network described in Section 5.4. In this section, the input vectors are the long input vectors (size eight hundred). The input vector looks like the following:

`word1...word2...word3...wordn`

where the words are the individual words of a given command.

As mentioned previously, with the number of items in the input vocabulary and the number of actions allowed on those items, I am able to come up with eighty

eight thousand commands that the natural language processor should understand. However, it has taken approximately five days to train one of the neural networks on one hundred commands using a 166 MHz machine. I determined that the test results that I would realistically be able to obtain would have to involve far fewer than eighty eight thousand commands. Therefore, I chose one thousand and thirty six commands randomly and ran all of the following tests on those commands in order to have consistent results.

Given the above information, I had to answer the question, "Are the input vectors for a given command strongly correlated?" In order to do so I took the information in Section 6.2 about how to determine if vectors are strongly correlated and used it on the vectors which were input to the nonrecurrent back propagation neural network. I computed the covariance matrix for the vectors.

The average value of the covariance matrix for all elements 501.6059. As the inner product of any two "different" vectors should be approximately zero, but the inner product of a single vector with itself should be more than zero, I computed the average values of the covariance matrix for the diagonal elements (a vector with itself) and the nondiagonal elements (a vector with another vector) separately. In this case the average value for the diagonal elements should be eight hundred. The reason being that there are eight hundred elements in the vectors and the vectors have bipolar values.

The average value of the covariance matrix diagonal elements was 800. This is as it should be for the diagonal values. The average value of the covariance matrix nondiagonal elements was 501.3144. This is not close to zero and, therefore, the input vectors are not approximately orthogonal. In other words, they are strongly correlated. This suggests that Hebbian learning will not work to solve this problem.

The point is that even though the input vectors for the individual words are not strongly correlated (as shown in Section 6.2), the input vectors for the commands are strongly correlated. This gives evidence that Hebbian learning should not be able to solve this natural language processing problem well. However, this type of input was not used for the Hebbian neural network. It is significant due to the idea that if the representation that we are able to build from the approximately orthogonal word vectors are strongly correlated, chances are that the representation that the recurrent neural network with Hebbian learning builds is going to be strongly correlated.

6.4 Third Question

The third question was “Are the internal representations similar for different commands?” Neural networks cannot learn well for patterns that have similar internal representations for different input patterns, therefore, if the internal representa-

tions are similar then neural networks cannot solve this problem. Internal representations are a result of the training of the neural network. Therefore, if the training of the neural network is successful, we expect that the internal representations for the input patterns would be different. However, the training of these neural networks has not been successful so I tested whether or not the internal representations were similar.

The test used to determine whether or not the internal representations were similar is described below. It should be noted that neural network used in this test was the Simple Recurrent Network with Hebbian learning. Before describing the test it is necessary to describe how the inputs to the tests were generated, i.e. what an internal representation is. The internal representation is the representation of the input that the neural network builds through Hebbian learning. In this case, the internal representation is the set of values for the hidden unit activation vectors.

The neural network was trained on a set of one thousand and thirty six commands. The neural network was trained for one thousand trials (one thousand presentations of the set of inputs to the network). Every one hundred trials the values for the hidden unit activation vectors for each command was saved to an output file. These hidden unit activation vectors were then used to calculate the covariance matrix and the Euclidean distance between all of them. Here, I am considering the hidden unit activation values for the neural network as the internal

representations for the commands. All of the vectors used to produce this data are bipolar vectors. The vectors are of length 200.

The average values for the covariance matrix of the different vectors grew larger with more trials and moved closer to the size of the vectors being compared. This could mean that the internal representations for the commands chosen are very similar. This gives evidence as to the reason that the neural network was unable to learn the commands given to it.

The Euclidean distance between the different vectors continued to decrease with more trials. It moved closer and closer to zero. This gives evidence that with more trials the internal representations for the vectors became more similar. This too suggests that the neural network should not be able to learn the commands.

Appendix M shows a table of the Euclidean distance average values computed as well as the covariance matrix average values for the simple recurrent network with Hebbian learning.

6.5 Other Results

Different types of information were calculated for each of the neural networks which were implemented. This was done in order to compare the Hebbian learning SRN with the two back propagation neural networks. Each of the neural networks was given the randomly chosen 1036 commands as input and trained for one

thousand trials. The data was collected every one hundred trials. The results of these tests are shown in Appendix N.

For the SRN with Hebbian learning the following tests were conducted: learning rate versus error rate and hidden layer size versus error rate. As the data shows, this neural network performed the worst of the three tested. It did, however, perform faster. The speed with which this algorithm runs is the major reason for choosing it. The most commands that this neural network learned during training was 8.

For the SRN with back propagation learning the following tests were conducted: learning rate versus error rate and hidden layer size versus error rate. As the data shows this neural network performed much better than the neural network with Hebbian learning, but not as well as the nonrecurrent neural network. It did, however, perform the slowest of the three neural networks tested. The most commands that this neural network learned during training was 121. This took 630 trials.

For the nonrecurrent neural network with back propagation learning the following tests were conducted: learning rate versus error rate and hidden layer size versus error rate. As the data shows, this neural network performed the best of the three neural networks tested on the chosen commands. It also learned them faster than the recurrent neural network with back propagation. The most commands that this neural network learned during training was 710. This took 55,000

trials. It should be noted that at the time of the writing of this dissertation, this neural network was still learning new commands. I have no evidence to support a claim that this nonrecurrent back propagation neural network could not solve this natural language processing task. Further study must be done, as discussed in Chapter 7.

6.6 Conclusion

The data presented in this and the previous chapter gives evidence that Hebbian learning does not work well in recurrent neural networks for natural language processing even when the input vectors to the network are approximately orthogonal. There are three reasons for this conclusion.

The first and foremost reason for this conclusion is the empirical data presented in Chapter 5. Three different Hebbian learning neural networks were implemented. Different types of input and output vectors were used. All available activation functions were implemented. A number of different methods of initializing, changing, and updating the weight matrix were implemented. A large range of hidden layer sizes were used. With all of these diverse approaches, none of the three Hebbian learning neural networks was able to interpret the natural language.

The second reason for this conclusion is that the input vectors for any two given commands are strongly correlated. In other words, if you randomly pick two commands and look at the input vectors which correspond to the command, you will see that they have many elements in common. This makes it difficult for the Hebbian learning rule to distinguish between them.

The third reason for this conclusion is that the internal representations are similar for different commands. If the internal representations, i.e. the representation that the neural network builds to distinguish the patterns, are similar then the neural network will have difficulty distinguishing the patterns.

So it is that I have found that the Hebb learning rule does not work well in recurrent neural networks for natural language processing even when the input vectors are approximately orthogonal. In Chapter 7 I will discuss ways that we may use this research to move forward in solving the natural language processing task.

Chapter 7

Future Work

Hinton states, “Most connectionist researchers are aware of the gulf in representational power between a typical connectionist network and a set of statements in a language. They continue to develop the connectionist framework not because they are blind to its limitations, but because they aim to eventually bridge the gulf by building outwards from a foundation that includes automatic learning procedures and/or massively parallel computation as essential ingredients. Subject to these hard constraints, they aim to progressively improve representational power.” [60]

This project has sought to bring the application of automatic learning procedures to natural language closer to fruition. While the Hebbian neural networks of this research project were not able to interpret natural language at the level that was expected, this project has made a contribution to connectionist research. This project has shown that Hebbian learning does not work in recurrent neural

networks for natural language processing even when the input vectors are approximately orthogonal. Now we should take this information and go on to build better neural networks. This project in no way implies that the task cannot be done with neural networks, as is shown by the nonrecurrent back propagation neural network.

In determining how the research community can learn from this project and go on to build neural networks that can process natural language, I believe there are three areas. The first is in choosing the input vocabulary. The second is in using orthogonal input vectors instead of approximately orthogonal if the Hebb learning rule is used. The third is in the use of learning algorithms other than the Hebb rule in conjunction with massively parallel machines for implementing neural networks. I will address each of those in more detail.

Considering that the most successful connectionist approach to natural language processing has had a vocabulary of fifty-five words, this project may have been slightly on the ambitious side. The input vocabulary to the natural language processor was one thousand and sixty words. It would be worthwhile to start with a subset of the input vocabulary and get it to work and then build up from there. An interesting project would be to take the same set of actions (verbs) and a small subset of the number of items (nouns) and build a neural network to interpret them.

It is my conclusion that the neural networks would perform better if the input commands were not so similar to each other. I could have chosen commands that were not as much alike as those that I presented to the neural networks. Consider that "Put the dish on the table." and "Put the apple on the refrigerator." are both commands of the type that occur often in the commands given to the neural network. These commands have four of their six words in common. It would be very difficult to get a neural network to distinguish between them because of the fact that they are very similar, but also very dissimilar. Changing the type of input commands to the system presents two problems: a small scale problem and a large scale problem.

The small scale problem is that it was necessary for me to have a small number of verbs and a large number of items (nouns) that the verbs operate on in order to stay with the two and one half year old child framework of language. If I had changed the type of input, it would have defeated that goal. Having such a large number of items (nouns) emphasized the small number of verbs and made the commands more difficult to learn.

The large scale problem is that in natural language many of the same words are used in many different sentences. This means that even if I changed the input commands, it might make the neural networks perform better, but would it bring us closer to being able to perform natural language processing with neural networks? It is true that natural language is not as repetitive as the commands

used in this project are, but count how many times in one day you use the word "the", or "a", or "I".

In spite of the two problems discussed above, it would be interesting to take an input vocabulary of roughly the same size and the same type of language, but with a different type of statement. In other words, get rid of the entire "robot moving about a house" concept and make a set of statements as input that are not all commands. Make them different types of statements, but keep with the two and one half year old concept. For example, the following might be statements that are interpreted by the system:

I see the dog.

What is your name?

The cup is blue.

It is probable that the neural networks can learn this language more successfully than a language in which the statements are too similar, as in this project. Sometimes the similarity of the input commands is the problem and sometimes the dissimilarity (orthogonality) is the problem. They might have many words in common so that the neural network thinks that they are the same, but at the same time they have several key distinguishing words that the neural network is unable to pick up on. I think that this combination was one of the key reasons that the neural networks were unable to interpret the language in this project.

It is possible that this natural language processing problem could be solved using Hebbian learning if the input vectors were orthogonal. As this project shows, approximately orthogonal input patterns cannot be learned well with Hebbian learning, but nothing in the data collected for this project has shown that leaving everything the same and changing the inputs to orthogonal vectors will not work. There are two ways to create orthogonal vectors.

The first way to create orthogonal vectors is to create binary input vectors with only a single element (unit) being one. All of the other values would be zero. This is a unary vector. For example, the input vector for "get" might be:

0 0 0 0 0 0 0 0 0 0 0 1

and the input vector for porch might be

0 0 0 0 0 1 0 0 0 0 0 0

Another method of creating orthogonal input vectors is to use Walsh functions, as described by [3]. Every element is either 1 or -1, but the vectors are not unary, i.e. more than one unit is "on" at a given time. The values for the elements of the vectors are chosen such that there is always a method of distinguishing between any two vectors, i.e. a pattern exists. Since there is a pattern to the input, it would now be possible to do a mapping between the input and output units. Therefore, a neural network could possibly be built such that a strong correlation between these inputs and a given set of outputs would exist. In other words, since

there is no "cross-talk" between the input vectors, the neural network can now learn, for example, that if there is a 1 in position 7 in the input vector and a -1 in positions 12 and 22, then the output should be 1100.

A problem with using orthogonal input vectors is that "in a system of n dimensions, there can be at most n mutually orthogonal vectors" [3]. Also, the dimensionality of the system must be a power of two. To create either unary input vectors or input vectors using Walsh functions, the vector length would have to be the size of the input vocabulary (or possibly larger for Walsh functions if the vocabulary size is not a power of two). In this case the vector length would be one thousand and sixty. This would increase the size of the input vectors more than ten times. It would also dramatically increase the amount of time necessary for the neural network to learn the commands. However, it could allow Hebbian neural networks to learn natural language.

In a different direction there is the use of learning algorithms other than the Hebb rule in conjunction with massively parallel hardware. As discussed in Section 6.6, the nonrecurrent back propagation neural network which was implemented for this project looks promising. The problem occurs in that it took the neural network 55,000 trials to learn 710 commands. These 55,000 trials took 110 days of training on a Sparc 5. Therefore, it is important to combine this approach with the use of massively parallel machines.

As mentioned in Chapter 2, one of the keys to solving large "real world" natural language processing tasks is going to be the use of massively parallel hardware. The training time for neural networks is just so large on a problem of any magnitude that it is not productive to try to get them to work on serial machines. They are just too slow. I believe that I could have made much more progress toward trying to build a neural network to solve this problem if I had not had to spend so much time running each of the tests. It would have made the decision to use Hebbian learning a much less significant one. If we ever hope to solve large problems in real time, we are going to have to execute them on massively parallel machines. This is becoming less of a pipe dream as parallel machines become more available and hardware costs go down.

Many different aspects of natural language processing were addressed in this project: including some of the recent work; how two and one-half year olds interpret language; how the connectionist approach differs from the symbolic approach; and how five different neural networks perform on a natural language interpretation task. Also, many of the shortcomings of current neural network technologies were presented, including knowledge representation, poor generalization, and the inability of neural networks to scale up to solve larger problems. This research project gives evidence to show that Hebbian learning does not work well in recurrent neural networks for natural language processing, even when the input vectors are approximately orthogonal. This in no way should discourage those of us who

believe in the connectionist approach. The ideas presented in this chapter, along with other ideas generated by this research, can be investigated as we continue to move closer to finding the right combination of architecture and algorithm to get computers to understand natural language at a two and one half year old level and beyond.

BIBLIOGRAPHY

Bibliography

- [1] James F. Allen, Lenhart K. Schubert, George Ferguson, Peter Heeman, Chung Hee Hwang, Tsunseaki Kato, Mark Light, Nathaniel G. Martin, Bradford W. Miller, Massimo Poesio, and David R. Traum. The trains project: A case study in building a conversational planning agent. Technical report, The University of Rochester, 1994.
- [2] Robert B. Allen. Connectionist language users. In *Connectionist Natural Language Processing: Readings from Connection Science*. Kluwer Academic Publishers, 1992.
- [3] James A. Anderson. *An Introduction to Neural Networks*. MIT Press, 1995.
- [4] James A. Anderson and Edward Rosenfield. *Neurocomputing: Foundations of Research*. MIT Press, 1989.
- [5] Moshe Anisfeld. *Language Development from Birth to Three*. Lawrence Erlbaum Associates, 1984.

- [6] M.A. Arbib and J.A. Robinson. *Natural and Artificial Parallel Computation*. MIT Press, 1990.
- [7] Martin Atkinson. *Explanations in the Study of Child Language Development*. Cambridge University Press, 1982.
- [8] Elizabeth Bates and Jeffrey Elman. Learning rediscovered. *Science*, 274, December 1996.
- [9] Thomas Bayer, Ingrid Renz, Michael Stein, and Ulrich Kressel. Domain and language independent feature extraction for statistical text categorization. Technical report, Daimler Benz AG- Research and Technology, July 1996.
- [10] M. Susan Beck. *KIDSPEAK: How Children Develop Language Skills*. Plume: New American Library, 1982.
- [11] Robert C. Berwick. Learning word meanings from examples. In David Waltz, editor, *Semantic Structures: Advances in Natural Language Processing*. Lawrence Erlbaum Associates, 1989.
- [12] Justine Blackmore and Risto Miikkulainen. Visualizing high-dimensional structure with the incremental grid growing neural network. In A. Prieditis and Russel S., editors, *Machine Learning: Proceedings of the 12th International Conference*. Department of Computer Science, The University of Texas, 1995.

- [13] H. Tad Blair. Evaluating connectionist models in psychology and neuroscience. In Mozer, Smolensky, Touretsky, Elman, and Weigand, editors, *Proceedings of the 1993 Connectionist Summer School*, volume 1. Lawrence Erlbaum Associates, 1994.
- [14] Lois Bloom and et al. *Language Development from Two to Three*. Cambridge University Press, 1991.
- [15] Gary C. Borchardt. *Thinking Between the Lines: Computers and the Comprehension of Causal Descriptions*. MIT Press, 1994.
- [16] P. J. Braspenning, F. Thuijsman, and A.J.M.M. Weijters. *Artificial Neural Networks: An Introduction to ANN Theory and Practice*. Springer, 1995.
- [17] Ted Briscoe. Language acquisition: The biogram hypothesis and the baldwin effect. Technical report, University of Cambridge, January 1997.
- [18] Eric W. Brown. An approach for improving execution performance in inference network based information retrieval. Technical report, University of Massachusetts, 1994.
- [19] Roger Brown. *A First Language: The Early Stages*. Harvard University Press, 1973.
- [20] E.R. Caianiello. *Neural Nets Wirn Vietri-93*. World Scientific, 1993.

- [21] Helen Smith Cairns. *The Acquisition of Language*. Pro-ed, second edition, 1996.
- [22] Claire Cardie. A case-based approach to knowledge acquisition for domain-specific sentence analysis. Technical report, University of Massachusetts, Department of Computer Science, 1993.
- [23] Claire Cardie and Wendy Lehnert. A cognitively plausible approach to understanding complex syntax. Technical report, University of Massachusetts, Department of Computer Science, 1991.
- [24] Jean-Pierre Chanod and Pasi Tapanainen. Tagging french - comparing a statistical and constraint-based method. Technical report, Rank Xerox Research Centre, Grenoble Laboratory, 1995.
- [25] Martin Charles. *Advances in Artificial Intelligence*. Springer-Verlag, 1990.
- [26] Jennifer Chu-Carroll and Sandra Carberry. Generating information-sharing subdialogues in expert-user consultation. Technical report, University of Delaware, January 1997.
- [27] Axel Cleeremans. *Mechanisms of Implicit Learning: Connectionist Models of Sequence Processing*. MIT Press, 1993.

- [28] Renwick Conrad. A natural language interface to the unix operating system using recurrent neural networks. Master's thesis, University of Tennessee at Knoxville, 1995.
- [29] Garrison W. Cottrell. *A Connectionist Approach to Word Sense Disambiguation*. Morgan Kaufmann Publishers, 1989.
- [30] Hamish Cunningham, Yorick Wilks, and Robert J. Gaizauskas. New methods current trends and software infrastructure for nlp. Technical report, University of Sheffield, UK, July 1996.
- [31] Edward William Dolch. *Teaching Primary Reading*. The Garrard Press, third edition, 1960.
- [32] E. Domany, J.L. van Hemmen, and K. Schulten. *Models of Neural Networks III: Association, Generalization, and Representation*. Springer, 1996.
- [33] Georg Dorffner. Radical connectionism for natural language processing. Technical report, Department of Medical Cybernetics and Artificial Intelligence, University of Vienna, and Austrian Research Institute for Artificial Intelligence, 1991.
- [34] Georg Dorffner. Winner-take-more a mechanism for soft competitive learning. Technical report, Department of Medical Cybernetics and Artificial

- Intelligence, University of Vienna, and Austrian Research Institute for Artificial Intelligence, 1992.
- [35] Georg Dorffner. Words, symbols, and symbol grounding. Technical report, Department of Medical Cybernetics and Artificial Intelligence, University of Vienna, and Austrian Research Institute for Artificial Intelligence, 1993.
- [36] Georg Dorffner. Why connectionism and language modeling need dete: A commentary on nenov and dyer. Technical report, Department of Medical Cybernetics and Artificial Intelligence, University of Vienna, 1994.
- [37] Georg Dorffner, Peter Rappelsberger, and Arthur Flexer. Using self-organizing feature maps to classify eeg coherence maps. Technical report, Department of Medical Cybernetics and Artificial Intelligence, University of Vienna and Austrian Research Institute for Artificial Intelligence, 1993.
- [38] H.L. Dreyfus. *What Computers Still Can't Do*. MIT Press, 1994.
- [39] Jeffrey L. Elman. Distributed representations, simple recurrent networks, and grammatical structure. *Machine Learning*, 7:195–225, 1991.
- [40] Jeffrey L. Elman. Learning and development in neural networks: the importance of starting small. *Cognition*, 48:71–99, 1993.
- [41] Laurene Fausett. *Fundamentals of Neural Networks: Architectures, Algorithms, and Applications*. Prentice Hall, 1994.

- [42] Michael Peter Ferrell. Neural grammatical inference using network-adapted encoding. Master's thesis, University of Tennessee at Knoxville, 1994.
- [43] Arthur Flexer. Connectionists and statisticians, friends or foes. Technical report, The Austrian Research Institute for Artificial Intelligence, 1995.
- [44] Norbert Fuchs and Rolf Schwitter. Specifying logic programs in controlled natural language. Technical report, Department of Computer Science, University of Zurich, 1995.
- [45] Michael Gasser. Acquiring receptive morphology: A connectionist model. Technical report, Computer and Linguistics Department, Indiana University, May 1994.
- [46] Michael Gasser. Modularity in a connectionist model of morphology acquisition. Technical report, Computer and Linguistics Department, Indiana University, May 1994.
- [47] Michael Gasser. Transfer in a connectionist model of the acquisition of morphology. Technical report, Computer and Linguistics Department, Indiana University, July 1995.
- [48] Dale Gerdemann and Paul John King. The correct and efficient implementation of appropriateness specifications for typed feature structures. Technical report, University of Tuebingen, Germany, August 1994.

- [49] Mark A. Gluck and David E. Rumelhart. *Neuroscience and Connectionist Theory*. Lawrence Erlbaum Associates, 1990.
- [50] Georgia M. Green. *Pragmatics and Natural Language Understanding*. Lawrence Erlbaum Associates, 1996.
- [51] John Griffith. Modularizing contexted constraints. Technical report, University of Tuebingen, Germany.
- [52] H.W. Guesgen and J. Hertzberg. *A Perspective of Constraint-Based Reasoning: An Introductory Tutorial*. Springer-Verlag, 1992.
- [53] Ahmed Guessoum and Torulf Mollestad. A knowledge-based approach to goal recognition. Technical report, Department of Computer Systems and Telematics, The Norwegian Institute of Technology.
- [54] Frank J. Guszak. *Diagnostic Reading Instruction in the Elementary School*. Harper and Row, 1972.
- [55] Louise Guthrie, James Pustejovsky, Yorick Wilks, and Brian M. Slator. The role of lexicons in natural language processing. *Communications of the ACM*, 39(1):63–72, January 1996.
- [56] Stephen Jose Hanson and Carl R. Olson. *Connectionist Modeling and Brain Function: The Developing Interface*. MIT Press, 1990.

- [57] S. Haykin. *Neural Networks: A Comprehensive Foundation*. Maxwell Macmillan Canada, Inc., 1994.
- [58] Renate Henschel. Traversing the labyrinth of feature logics for a declarative implementation of large scale systematic grammars. Technical report, Darmstadt, Germany.
- [59] J. Hertz, A. Krogh, and R.G. Palmer. *Introduction to the Theory of Neural Computation*. Addison-Wesley Publishing Company, 1991.
- [60] Geoffrey Hinton. *Connectionist Symbol Processing*. MIT Press, 1991.
- [61] Wassim Hossari. Counterpropagation for object recognition and defect detection in images. Master's thesis, University of Tennessee at Knoxville, 1992.
- [62] Christopher C. Huckle. Grouping words using statistical context. Technical report, Department of Psychology, University of Edinburgh, February 1995.
- [63] R. Jackendoff. *Semantic Structures*. MIT Press, 1991.
- [64] Glenn James and Robert C. James. *Mathematics Dictionary*. D. Van Nostrand Company, Inc., 1960.
- [65] Julia E. Johnsen. *Basic English*. H. W. Wilson Company, 1944.
- [66] Mark Johnson. Memoization in constraint logic programming. Technical report, Brown University, April 1994.

- [67] M.I. Jordan. Attractor dynamics and parallelism in a connectionist sequential machine. In *Proceedings of the Eighth Annual Meeting of the Cognitive Science Society*, 1986.
- [68] Bert Kappen and Stan Gielen. *Neural Networks: Artificial Intelligence and Industrial Applications*. Springer, 1995.
- [69] Andre Kempe. Probabilistic tagging with feature structures. Technical report, University of Stuttgart, Institute for Computational Linguistics, Stuttgart, Germany, October 1994.
- [70] Simon Kirby and James Hurford. Learning, culture and evolution in the origin of linguistic constraints. Technical report, University of Edinburgh.
- [71] Kevin Knight, Ishwar Chander, Matthew Haines, Vasileios Hatzivassiloglou, Eduard Hovy, Masayo Iida, Steve K. Luk, Akitoshi Okumura, Richard Whitney, and Kenji Yamada. Integrating knowledge bases and statistics in mt. Technical report, USC/Information Sciences Institute, July 1994.
- [72] Kevin Knight and Steve K. Luk. Building a large-scale knowledge base for machine translation. Technical report, USC/Information Sciences Institute, September 1994.
- [73] Teuvo Kohonen. *Self-Organization and Associative Memory*. Springer-Verlag, third edition, 1989.

- [74] Stan II Kuczaj. *Language Development: Language, Thought, and Culture*, volume 2. Lawrence Erlbaum Associates, 1982.
- [75] Stan II Kuczaj. *Language Development: Syntax and Semantics*, volume 1. Lawrence Erlbaum Associates, 1982.
- [76] Stan C. Kwasny and Kanaan A. Faisal. Connectionism and determinism in a syntactic parser. In Noel Sharkey, editor, *Connectionist Natural Language Processing: Readings from Connection Science*. Kluwer Academic Publishers, 1992.
- [77] Diane Law and Risto Miikkulainen. Grounding robotic control with genetic neural networks. Technical report, Department of Computer Science, The University of Texas, May 1994.
- [78] Steve Lawrence, Sandiway Fong, and C. Lee Giles. On the applicability of neural network and machine learning methodologies to natural language processing. Technical report, NEC Research Institute, 1995.
- [79] Steve Lawrence, Sandiway Fong, and C. Lee Giles. Natural language grammatical inference: A comparison of recurrent neural networks and machine learning methods. In Stefan Werner, Ellen Rilof, and Gabrielle Scheler, editors, *Symbolic, Connectionist, and Statistical Approaches to Learning for Natural Language Processing: Lecture notes in AI series*. NEC Research Institute, Springer Verlag, 1996.

- [80] Steve Lawrence, C. Lee Giles, and Sandiway Fong. On the applicability of recurrent neural networks to grammatical inference - theory and practice. Technical report, NEC Research Institute, 1996.
- [81] Guebae Lee, Margot Flowers, and Michael G. Dyer. Learning distributed representations of conceptual knowledge and their application to script-based story processing. In Noel Sharkey, editor, *Connectionist Natural Language Processing: Readings from Connection Science*. Kluwer Academic Publishers, 1992.
- [82] W. Lehnert, Cardie C., J. Fisher, D. and McCarthy, E. Riloff, and S. Soderland. University of massachusetts: Muc-4 test results and analysis. In *Proceedings of the Fourth Understanding Conference*. Department of Computer Science, University of Massachusetts, June 1992.
- [83] Wendy G. Lehnert. Possible implications of connectionism. In Yorick Wilks, editor, *Theoretical Issues in Natural Language Processing*. Lawrence Erlbaum Associates, 1989.
- [84] Wendy G. Lehnert. Symbolic/subsymbolic sentence analysis: Exploiting the best of two worlds. In John A. Barnden and Jordan B. Pollack, editors, *Advances in Connectionist and Neural Computation Theory*, volume 1, 1991.
- [85] Wendy G. Lehnert. Cognition, computers, and car bombs: How yale prepared me for the 90's. In Schank and Langer, editors, *Beliefs, Reasoning,*

and Decision Making: Psycho-logic in Honor of Bob Abelson. Department of Computer Science, University of Massachusetts, Lawrence Erlbaum Associates, 1994.

- [86] Daniel S. Levine and Manuel IV Aparicio. *Neural Networks for Knowledge Representation and Inference.* Lawrence Erlbaum Associates, 1994.
- [87] Marc Light and Lenhart Schubert. Knowledge representation for lexical semantics: Is standard first order logic enough? Technical report, University of Rochester, August 1994.
- [88] Nancy J. Lund and Judith F. Duchan. *Assesing Children's Language in Naturalistic Contexts.* Prentice Hall, second edition, 1988.
- [89] Bruce MacLennan. Continuous symbol systems:the logic of connectionism. In Daniel S. Levine and Manuel IV Aparicio, editors, *Neural Networks for Knowledge Representation and Inference.* Lawrence Erlbaum Associates, 1994.
- [90] Bruce MacLennan. Field computation in motor control. In Pietro G. Morasso and Vittorio Sanguinetti, editors, *Self-Organization, Computation Maps, and Motor Control.* Elsevier-North Holland, In-Press.
- [91] Bruce MacLennan, Dave Brumgard, Joseph Dougherty, Debra J. Holl, David E. Johnson, Phil Reynolds, Fermin Salkjelsvik, and Gerald B.

Schmidt. Studies in natural language 1989 progress report. Technical report, Computer Science Department, University of Tennessee, Knoxville, September 1990.

- [92] Bruce J. MacLennan. Synthetic ethology: An approach to the study of communication. In *Artificial Life II, Studies in the Sciences of Complexity*, volume X. Addison-Wesley, 1991.
- [93] David M. Magerman. *Natural Language Parsing as Statistical Pattern Recognition*. PhD thesis, Stanford University, May 1994.
- [94] J.L. McClelland and A. H. Kawamoto. Mechanisms of sentence processing: Assigning roles to constituents of sentences. In *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, volume 2. MIT Press, 1986.
- [95] J.L. McClelland and D. E. Rumelhart. *Explorations in Parallel Distributed Processing: A Handbook of Models, Programs, and Exercises*. MIT Press, 1988.
- [96] J.L. McClelland and D.E. Rumelhart. *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, volume 2. MIT Press, 1986.
- [97] Michael M. Merzenich, Gregg H. Recanzone, William M. Jenkins, and Randolph J. Nudo. How the brain functionally rewires itself. In M.A. Arbib and

- J.A. Robinson, editors, *Natural and Artificial Parallel Computation*. MIT Press, 1990.
- [98] Risto Miikkulainen. A distributed feature map model of the lexicon. Technical report, Computer Science Department, University of California, 1990.
- [99] Risto Miikkulainen. A pdp architecture for processing sentences with relative clauses. Technical report, Computer Science Department, University of California, 1990.
- [100] Risto Miikkulainen. Subsymbolic case-role analysis of sentences with embedded clauses. Technical report, The University of Texas, 1990.
- [101] Risto Miikkulainen. Trace feature map: a model of episodic associative memory. Technical report, Computer Science Department, University of Texas, 1991.
- [102] Risto Miikkulainen. Discern: A distributed neural network model of script processing and memory. Technical report, Department of Computer Science, University of Texas, 1992.
- [103] Risto Miikkulainen. *Subsymbolic Natural Language Processing: an Integrated Model of Scripts, Lexicon, and Memory*. MIT Press, 1993.
- [104] Risto Miikkulainen. Script recognition with hierarchical feature maps. Technical report, Computer Science Department, University of California, 1996.

- [105] Ruth Garrett Millikan. A common structure for concepts of individuals, stuffs, and real kinds: More mama, more milk, and more mouse. *Behavioral and Brain Sciences*, 1997.
- [106] Marvin Minsky. Logical vs. analogical or symbolic vs. connectionist or neat vs. scruffy. In Patrick H. Winston, editor, *Artificial Intelligence at MIT, Expanding Frontiers*, volume 1. MIT Press, 1990.
- [107] Herman Moisl. Connectionist finite state natural language processing. *Connection Science*, 4(2), 1992.
- [108] Mark Moll, Risto Miikkulainen, and Jonathan Abbey. The capacity of convergence-zoned episode memory. Technical report, Department of Computer Science, University of Twente, Department of Computer Sciences, The University of Texas, Applied Research Laboratories, 1994.
- [109] David E. Moriarity and Risto Miikkulainen. Evolutionary neural networks for value ordering in constraint satisfaction problems. Technical report, Department of Computer Science, University of Texas, 1994. May.
- [110] Michael Morreau. How to derive conveyed meaning. Technical report, Philosophy Department and UNIACS, University of Maryland.
- [111] Mozer, Smolenksy, Touretsky, Elman, and Weigand, editors. *Proceedings of the 1993 Connectionist Models Summer School*. Lawrence Erlbaum Asso-

- ciates, 1994.
- [112] B. Muller and J. Reinhardt. *Neural Networks: An Introduction*. Springer-Verlag, 1990.
 - [113] Alan F. Murray. *Applications of Neural Networks*. Kluwer Academic Publishers, 1995.
 - [114] Marilyn McCord Nelson and W.T. Illingworth. *A Practical Guide to Neural Nets*. Addison-Wesley Publishing, 1990.
 - [115] Hwee Tou Ng and Hian Beng Lee. Integrating multiple knowledge sources to disambiguate word sense: An exemplar-based approach. Technical report, Defence Science Organisation, Singapore, June 1996.
 - [116] Klaus K. Obermeier. *Natural Language Processing Technologies in Artificial Intelligence: The Science and Industry Perspective*. Ellis Horwood Limited, 1989.
 - [117] Kemal Oflazer and Okan Yilmaz. A constraint-based case frame lexicon architecture. Technical report, Bilkent University, July 1995.
 - [118] C. K. Ogden. *Basic Words: A Detailed Account of their Uses*. Kegan Paul, Trench, Trubner and Co., 1933.
 - [119] Steven Pinker. *Learnability and Cognition: The Acquisition of Argument Structure*. MIT Press, 1989.

- [120] Tony Plate. *Holographic Reduced Representations*. PhD thesis, University of Toronto, 1994.
- [121] Timothy C. Potts. *Structures and Categories for the Representation of Meaning*. Cambridge University Press, 1994.
- [122] Loreen Pratt. Transfer between neural networks to speed up learning. *Journal of Artificial Intelligence Research*.
- [123] Erich Prem, Markus Mackinger, and Georg Dorffner. Concept support as a method for programming neural networks with symbolic knowledge. Technical report, Austrian Research Institute for Artificial Intelligence, January 1993.
- [124] William H. Press, William T. Vetterling, Saul A. Teukolsky, and Brian P. Flannery. *Numerical Recipes in C: The Art of Scientific Computing*. Cambridge University Press, 1992.
- [125] A. Prieto, editor. *Artificial Neural Networks International Workshop IWANN'91*. Springer-Verlag, 1991.
- [126] Alan Prince and Paul Smolensky. Optimality: From neural networks to universal grammar. *Science*, 275, March 1997.
- [127] William Ramsey, Stephen P. Stich, and David E. Rumelhart. *Philosophy and Connectionist Theory*. Lawrence Erlbaum Associates, 1991.

- [128] Ronan G. Reilly and Noel E. Sharkey. *Connectionist Approaches to Natural Language Processing*. Lawrence Erlbaum Associates, 1992.
- [129] Burghard B. Rieger. Situation semantics and computational linguistics toward information ecology. Technical report, Department of Computational Linguistics, University of Trier, 1994.
- [130] Ellen Rilof. Using cases to represent context for text classification. In *Proceedings of the Second International Conference on Information and Knowledge Management*, 1993.
- [131] Helge Ritter, Thomas Martinetz, and Klaus Schulten. *Neural Computation and Self-Organizing Maps*. Addison-Wesley Publishing Company, 1991.
- [132] A.F. Rocha. *Neural Nets: A Theory for Brains and Machines*. Springer-Verlag, 1992.
- [133] D.E. Rumelhart and J.L. McClelland. *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, volume 1. MIT Press, 1986.
- [134] D.E. Rumelhart and J.L. McClelland. On learning the past tense of english verbs. In James L. McClelland and David E. Rumelhart, editors, *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, volume 2. MIT Press, 1988.

- [135] Jenny R. Saffran, Richard N. Aslin, and Elissa L. Newport. Statistical learning by 8-month-old infants. *Science*, 274, December 1996.
- [136] Fermin Soriano Salkjelsvik. Natural language processing with recurrent neural networks. Master's thesis, University of Tennessee at Knoxville, 1989.
- [137] Roger C. Schank and Kenneth Mark Colby. *Computer Models of Thought and Language*. W. H. Freeman and Company, 1973.
- [138] Mark S. Seidenberg. Language acquisition and use: Learning and applying probabilistic constraints. *Science*, 275, March 1997.
- [139] T.J. Sejnowski, C. Koch, and P.S. Churchland. Computational neuroscience. In Stephen Jose Hanson and Carl R. Olsen, editors, *Connectionist Modeling and Brain Function: The Developing Interface*. MIT Press, 1990.
- [140] Noel Sharkey. *Connectionist Natural Language Processing: Readings from Connection Science*. Kluwer Academic Publishers, 1992.
- [141] S.M. Shieber. *Constraint-Based Grammar Formalisms*. MIT Press, 1992.
- [142] Gert Smolka. A survey of oz. Technical report, German Research Center for Artificial Intelligence, January 1995.
- [143] Francoise Soulie and Jeanny Herault. *Neurocomputing: Algorithms, Architectures and Applications*. Springer-Verlag, 1989.

- [144] Edward P. Jr. Stabler. *The Logical Approach to Syntax*. MIT Press, 1992.
- [145] Luc Steels and Paul Vogt. Grounding adaptive language games in robotic agents. Technical report, VUB AI Laboratory and SONY Computer Science Laboratory.
- [146] Richard F. E. Sutcliffe. Representing meaning using microfeatures. In Ronan G. Reilly and Noel E. Sharkey, editors, *Connectionist Approaches to Natural Language Processing*. Lawrence Erlbaum Associates, 1992.
- [147] Kevin Swingler. *Applying Neural Networks: A Practical Guide*. Academic Press, 1996.
- [148] Jasper Taylor. Generating efficient belief models for task-oriented dialogues. Technical report, Human Communication Research Center University of Edinburgh, March 1995.
- [149] J.G. Taylor. *The Promise of Neural Networks*. Springer-Verlag, 1993.
- [150] Claudia Ulbricht. Handling sequences with a competitive recurrent network. Technical report, Austrian Research Institute for Artificial Intelligence, 1992.
- [151] Atro Voutilainen. Nptool, a detector of english noun phrases. Technical report, Research Unit for Computational Linguistics, February 1995.

- [152] David Waltz. *Semantic Structures: Advances in Natural Language Processing*. Lawrence Erlbaum Associates, 1989.
- [153] Nigel Ward. *A Connectionist Language Generator*. Ablex Publishing Company, 1994.
- [154] Stefan Wermter and Wendy G. Lehnert. A hybrid symbolic/connectionist model. In Noel Sharkey, editor, *Connectionist Natural Language Processing: Readings from Connection Science*. Kluwer Academic Publishers, 1992.
- [155] Stefan Wermter and Volker Weber. Screen: Learning a flat syntactic and semantic spoken language analysis using artificial neural networks. *Journal of Artificial Intelligence Research*, 6:35–85, 1997.
- [156] Janyce Wiebe, Graeme Hirst, and Diane Horton. Language use in context. *Communications of the ACM*, 39(1):102–111, January 1996.
- [157] Yorick Wilks. *Theoretical Issues in Natural Language Processing*. Lawrence Erlbaum Associates, 1989.
- [158] Terry Winograd. A procedural model of language understanding. In Roger C. Schank and Kenneth Mark Colby, editors, *Computer Models of Thought and Language*. W. H. Freeman and Company, 1973.
- [159] Wlodek Zadrozny. Natural language processing: Structure and complexity. Technical report, IBM Research, T.J. Watson Research Center, July 1996.

APPENDICES

Appendix A

Items Allowed in House

sofa

desk chair

desk

end table

refrigerator

stove

sink

toilet

bathtub

bed

lamp

bookshelf

china cabinet
television
chest
dresser
pantry
coffee table
dining table
work table
computer
easy chair
table chair
filing cabinet
dishwasher
kitchen cabinet
loveseat
fireplace
stereo
rocking chair
swing
books
china

tooth paste

soda

clothes

towel

washclothe

cds

tapes

diskettes

printer

printer paper

files

dishes

tooth brush

hair brush

soap

magazine

telephone

scale

clock

coffee cup

picture

ashtray
clothes hanger
toaster
pillow
curtains
walls
ceiling
trashcan
stapler
nightstand
laundry basket
washer
dryer
mirror
hair dryer
hutch
stool
oven
microwave
grill
shower

piano

Appendix B

Actions Allowed on Items

Get objectX.

Put objectX in LocationA.

Move objectX from locationA to locationB.

Where is objectX?

Is objectX present?

What is in roomQ?

What is near objectX?

Put objectX on objectY.

Put objectX in objectY.

Appendix C

Survey used to compute weights

ROOMS SURVEY

I am currently working on my dissertation for my PhD in Computer Science at the University of Tennessee at Knoxville. My project involves Artificial Intelligence. I am creating a simulated robot who will move about the rooms of a house and perform the actions asked of it by the user. In order to do so, I have to have some idea of what belongs in each room of a house. In creating these rooms, I would like to be as accurate as possible. Therefore, I am asking one hundred people to fill out this survey of what items they think belong in what rooms. I appreciate your time and energy in filling out this survey to the best of your ability. This will greatly help me to continue with the rest of my research project.

Survey Instructions

To fill out the survey, simply place a check in each box of the table if you think that the item on that row typically is seen in the room listed at the top of that column. Note that one particular item may be seen in more than one room.

Item	Office	Bedroom	Dining	Living	Kitchen	Bathroom	Porch
sofa							
desk-chair							
desk							
end-table							
refrigerator							
stove							
sink							
toilet							
bathtub							
bed							
lamp							
book							
bookshelf							
china-cabinet							
television							
chest							
dresser							
pantry							
coffee-table							
dining-table							
work-table							
computer							
easy-chair							
table-chair							
filing-cabinet							
dishwasher							
kitchen-cabinet							
loveseat							
fireplace							

Item	Office	Bedroom	Dining	Living	Kitchen	Bathroom	Porch
stereo							
rocking-chair							
swing							
books							
china							
toothpaste							
soda							
clothes							
towel							
washcloth							
cds							
tapes							
diskettes							
printer							
printer-paper							
files							
dishes							
toothbrush							
hair-brush							
mirror							
soap							
magazine							
telephone							
scale							
clock							
coffee-cup							
picture							
ashtray							
clothes-hanger							

Item	Office	Bedroom	Dining	Living	Kitchen	Bathroom	Porch
toaster							
pillow							
curtains							
walls							
ceiling							
trashcan							
stapler							
nightstand							
laundry-basket							
washer							
dryer							
mirror							
hair-dryer							
hutch							
stool							
oven							
microwave							
grill							
shower							
piano							

Appendix D

Words in Input Vocabulary

The following list is the Dolch Word List plus the words that were previously in the vocabulary. This is the input vocabulary to the natural language processor, i.e. the set of words that are legal as input.

office	bedroom	dining	living	kitchen	bathroom
porch	sofa	desk	refrigerator	stove	sink
toilet	bathtub	bed	lamp	book	bookshelf
cabinet	television	chest	dresser	pantry	coffee
work	computer	filig	dishwasher	loveseat	fireplace
stereo	rocking	swing	china	toothpaste	soda
clothes	towel	washclothe	cds	tapes	diskettes
printer	files	dish	toothbrush	hairbrush	mirror
soap	magazine	telephone	scale	clock	picture
ashtray	hanger	toaster	pillow	curtain	wall
ceiling	trash	stapler	laundry	washer	dryer
hutch	stool	oven	microwave	grill	shower
piano	get	put	in	move	from
to	where	is	what	near	find
return	by	big	small	white	yellow
orange	red	blue	green	brown	black
soft	hard	on	the	one	a
about	above	across	accident	ache	act
address	afraid	after	afternoon	again	ago
ahead	air	airplane	all	almost	alone
along	already	also	always	am	an
and	angry	animal	another	answer	ant
any	anywhere	apple	are	arm	around
as	ask	at	ate	aunt	automobile
awake	away	baby	back	bad	bag
bake	ball	balloon	banana	band	bandage
bank	bark	barn	basket	bath	bathe
be	beans	bear	beat	beautiful	because
become	bee	been	before	began	begin
begun	believe	bell	belong	bend	beside
best	better	between	bicycle	bill	bird
birthday	bit	bite	blackboard	bleed	bless
blew	blind	blood	blow	board	boat
body	bone	born	both	bottle	bottom
bow	bowl	box	boy	branch	brave
bread	break	breakfast	brick	bridge	bright
bring	broke	broom	brother	brought	bug
build	built	bump	burn	bus	busy
but	butcher	butterfly	butter	button	buy
cake	calf	call	came	camp	can
candy	cap	captain	car	card	care
careful	careless	carry	case	cat	catch

caught	cause	cent	center	chain	chair
chalk	chance	change	cheek	chicken	chief
child	chimney	chin	chocolate	choose	church
circle	circus	city	class	clean	clear
climb	close	cloth	cloud	clown	close
cloth	cloud	clown	coal	coat	cocoa
cold	color	come	company	cook	cookie
cool	copy	corn	corner	cost	cough
could	count	country	course	cousin	cover
cow	crackers	crayons	cream	creek	cross
crowd	crown	cry	cup	cupboard	cut
dance	danger	dark	date	day	dead
dear	deep	deer	dentist	did	die
different	dig	dime	dine	dinner	dirt
do	doctor	dog	doll	dollar	done
door	double	down	draw	drawer	dream
dress	drink	drive	drop	drug	dry
duck	dust	each	ear	early	earth
easy	eat	edge	egg	eight	either
elephant	eleven	else	empty	end	engine
enough	eraser	even	evening	ever	every
except	expect	eye	face	fair	fall
family	far	farm	fast	fat	father
feather	feed	feel	feet	fell	fellow
felt	fence	few	field	fight	fill
find	finger	finish	fire	first	fish
fit	five	fix	flag	flew	floor
flower	fly	follow	food	foot	for
forget	forgot	fork	forth	found	four
fresh	friend	frog	front	fruit	full
fun	fur	furniture	game	garage	garden
gate	gave	gift	girl	give	glad
glass	go	gold	gone	good	goodbye
got	grade	grain	grandfather	grandmother	grass
gray	great	grew	grocery	ground	grow
guess	had	hair	half	hall	hammer
hand	handkerchief	hang	happen	happy	has
hat	have	he	head	hear	heard
heart	heavy	held	hello	help	hen
her	here	hid	hide	high	hill
him	his	hit	hold	hole	home
hope	horse	hot	hour	house	how

hundred	hung	hungry	hunt	hurry	hurt
i	ice	if	indoors	inside	instead
into	iron	it	juice	jump	just
keep	kept	kick	kill	kind	kiss
kitten	knee	knew	knife	knock	know
lady	laid	lake	lamb	land	lap
large	last	late	laugh	lay	lead
leaf	learn	leather	leave	leaves	led
left	leg	lemonade	lesson	let	letter
lettuce	lie	life	lift	light	like
line	lion	lip	listen	little	live
load	long	look	lost	lot	loud
love	low	lunch	maid	magic	mail
make	man	many	march	mark	market
matter	may	maybe	me	mean	measure
meat	medicine	meet	men	mend	met
middle	might	mile	milk	mill	mind
minute	miss	money	monkey	month	moon
more	morning	most	mother	mountain	mouse
mouth	much	music	must	my	nails
name	nap	napkin	miss	money	monkey
month	moon	more	morning	most	mother
mountain	mouse	mouth	much	music	must
my	nails	name	nap	napkin	neck
need	neighbor	neither	nest	never	new
next	nice	nickel	night	nine	no
noise	none	noon	nor	north	nose
not	note	nothing	now	number	nurse
nut	oak	ocean	of	off	often
oh	old	once	only	onto	open
or	other	ought	our	out	outdoors
outside	over	overalls	own	page	pail
pain	paint	pair	pan	pants	paper
parade	part	party	pass	past	paste
path	pay	peach	peas	pen	pencil
penny	people	pet	pick	picnic	pie
piece	pig	place	plain	plant	plate
play	please	pocket	point	policeman	pond
pony	poor	post	pot	potatoes	pound
present	press	pretty	pull	puppy	push
puzzle	quarter	queen	question	quick	quite
quiet	rabbit	race	radio	rag	rain

ran	rather	reach	read	ready	real
reason	remember	rest	rich	ride	right
ring	river	road	robin	rock	roll
roof	room	rooster	root	rope	rose
round	row	rub	rubber	rug	ruler
run	sad	said	sail	salt	same
sand	sandwich	sang	sat	save	saw
say	school	scissors	scooter	sea	season
seat	second	see	seed	seem	seen
self	sell	send	sent	serve	set
seven	several	shadow	shake	shell	shape
she	sheep	shine	ship	shirt	shoe
shook	shop	short	should	shoulder	show
shut	sick	side	sign	silk	silver
sing	sir	sister	sit	six	size
skates	skin	skirt	sky	sleep	slip
slow	smart	smell	smile	smoke	snow
so	socks	sold	soldier	some	song
soon	sore	sorry	sound	soup	south
space	speak	spoken	spoon	spot	spread
spring	square	squirrel	stairs	stand	star
start	station	stay	step	stick	still
sting	stocking	stomach	stone	stood	stop
store	storm	story	straight	street	strike
string	strong	such	sudden	sugar	suit
summer	sun	supper	suppose	sure	surprise
sweater	sweep	sweet	swim	table	tale
take	talk	tail	taste	teach	tear
teeth	tell	ten	tent	than	thank
that	their	them	then	there	these
they	thick	thin	thing	think	third
thirsty	this	those	though	thought	thousand
three	threw	throat	through	throw	thumb
ticket	tie	till	time	tire	tired
today	toe	together	told	tomatoes	tomorrow
tongue	tonight	too	took	tooth	top
touch	toward	town	toys	trade	train
tree	trick	trip	truck	true	try
tub	turn	turtle	twelve	twenty	two
ugly	umbrella	uncle	under	until	up
upon	us	use	valley	very	visit
wagon	wait	wake	walk	want	war

warm	was	wash	waste	watch	water
wave	way	we	wear	weather	week
well	went	were	west	wet	wheat
wheel	when	whenever	whether	which	while
whisper	who	whole	whom	whose	why
wide	wife	wild	will	win	wind
window	wing	winter	wise	wish	with
without	woman	women	wonder	wood	wool
word	wore	work	world	would	wrap
write	wrong	yard	year	yes	yesterday
yet	you	young	your	zipper	

Appendix E

Words Two and one half year old child understands

The following is the list of words which two of the three children surveyed understood out of the input vocabulary.

a	above	across	accident	afraid	after
again	airplane	all	almost	already	am
and	angry	animal	another	ant	any
apple	arm	ask	at	ate	awake
away	baby	back	bad	bag	ball
balloon	banana	band	bandage	bark	barn
basket	bath	beans	bear	beat	beautiful
bed	bee	been	bell	bicycle	big
bird	bite	black	bleed	blood	blow
blue	boat	book	both	bottle	bow
bowl	bread	break	brick	bridge	bring
broke	broom	brought	brown	bug	bump
bus	butterfly	butter	button	by	cake
calf	call	came	can	candy	cap
car	card	careful	carry	cat	catch
caught	chair	chalk	chicken	chin	chocolate
clean	clear	climb	clock	close	clothes
cloud	clown	coat	cold	color	come
cool	corn	cough	count	cover	cow
crackers	crayons	cry	cup	cut	dance
danger	dark	deer	desk	did	dig
dinner	dirt	do	doctor	dog	doll
done	door	down	draw	drink	drive
drop	dry	duck	ear	easy	eat
egg	elephant	empty	eye	face	fall
fast	father	feed	feel	feet	fell
fence	fight	fill	find	finger	finish
fire	fish	fit	fix	flag	floor
flower	fly	food	foot	for	fork
found	friend	frog	fruit	full	fun
game	gate	gave	get	give	go
gone	good	goodbye	got	grass	great
green	ground	had	hair	hammer	hand
hang	happen	happy	hard	has	hat
have	head	hear	heard	heavy	hello
help	here	hide	high	hill	his
hit	hold	hole	home	horse	hot
house	hungry	hunt	hurry	hurt	I
ice	in	inside	instead	juice	jump
kick	kiss	kitchen	kitten	knee	knife
know	laid	lamp	lap	laugh	lay
leaf	leave				

leaves	leg	let	lift	light	like
lion	lip	listen	little	look	lost
loud	lunch	made	mail	make	man
maybe	me	mean	medicine	men	might
milk	mirror	miss	monkey	moon	more
mother	mouse	mouth	move	much	music
my	name	nap	napkin	neck	need
next	nice	night	no	noise	nose
now	nut	ocean	of	off	oh
on	one	open	orange	out	outside
over	own	pain	paint	pants	paper
party	peach	peas	pen	pencil	people
pet	pick	picnic	picture	pig	pillow
plate	play	please	pocket	point	policeman
pony	present	pull	puppy	push	put
quiet	rabbit	radio	rain	reach	read
ready	red	rest	ribbon	ride	river
road	rock	roll	room	rooster	run
sad	salt	sand	sat	say	school
scissors	seat	see	shadow	shake	sheep
ship	shirt	shoe	show	shut	sick
sing	sister	sit	skin	skirt	sky
sleep	slow	small	smell	smile	smoke
snow	soap	socks	soft	song	sore
sorry	soup	spoon	squirrel	stairs	stand
star	stay	step	stick	still	stomach
stop	store	storm	story	stove	string
strong	sun	supper	surprise	sweep	sweet
swim	table	take	talk	taste	teeth
tell	tent	that	the	there	these
thirsty	this	throat	throw	thumb	tired
to	toe	tomatoes	tongue	tooth	touch
towel	toys	train	tree	truck	tub
turtle	umbrella	under	up	wait	wake
walk	wall	want	warm	wash	watch
water	wave	wear	wet	whisper	white
who	wind	window	work	write	yellow
yes	you				

Appendix F

Words Natural Language

Processor Understands

The following is the list of words from the input vocabulary and the words which two of the three children understood that fit into the domain of discourse for this project. These are the words which the natural language processor is able to understand, i.e. comprehend commands from. The words are alphabetized within part of speech.

ARTICLES

the

		NOUNS			
airplane	animal	ant	apple	ashtray	bag
ball	balloon	banana	bandage	barn	basket
bathroom	bathtub	beans	bear	bed	bedroom
bee	bell	bicycle	bird	boat	book
bookshelf	bottle	bow	bowl	bread	broom
bug	bus	butter	butterfly	button	cabinet
cake	can	candy	cap	car	card
cat	cds	ceiling	chalk	chest	chicken
china	chocolate	clock	clothes	clown	coat
coffee	computer	corn	cow	crackers	crayons
cup	curtain	deer	desk	dish	dishwasher
diskettes	dog	doll	door	dresser	dryer
duck	egg	elephant	files	fireplace	fish
flag	floor	flower	food	fork	frog
fruit	game	gate	grill	hairbrush	hammer
hanger	hat	horse	hutch	ice	juice
kitchen	kitten	knife	lamp	light	lion
loveseat	magazine	mail	microwave	milk	mirror
monkey	mouse	napkin	nut	office	oven
pantry	pants	paper	peach	peas	pen
pencil	piano	picture	pig	pillow	plate
pony	porch	printer	puppy	rabbit	radio
refrigerator	ribbon	rock	room	rooster	salt
scale	scissors	sheep	ship	shirt	shoe
shower	sink	skirt	soap	socks	soda
sofa	soup	spoon	squirrel	stairs	stapler
star	stereo	stool	stove	string	swing
tapes	telephone	television	tent	toaster	toilet
tomatoes	toothbrush	toothpaste	towel	toys	train
truck	tub	turtle	umbrella	wall	washcloth
washer	water				

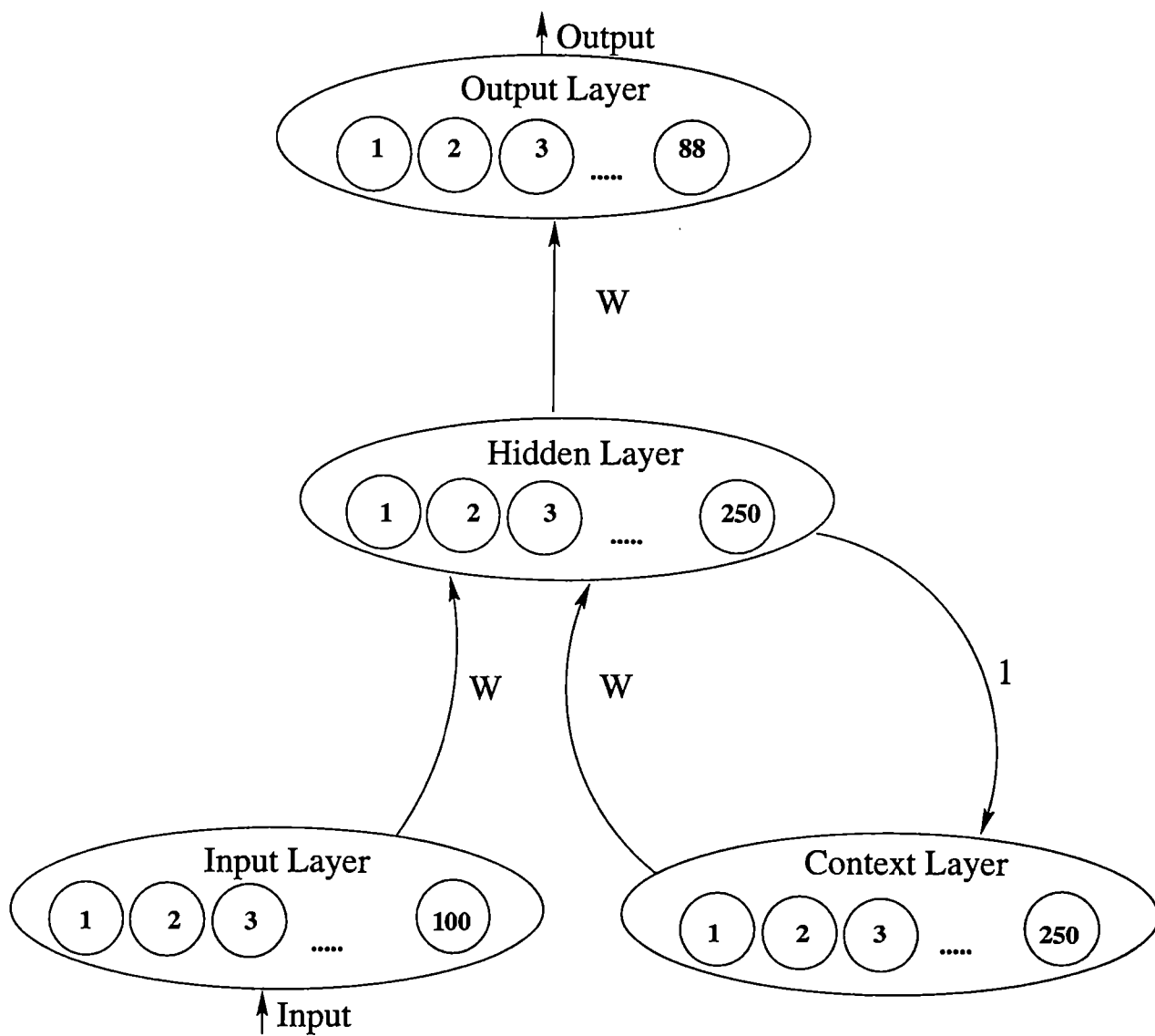
		VERBS			
bring	find	get	is	move	put
what	where				

		PREPOSITIONS			
by	in	near	on	from	to

		ADJECTIVES			
big	black	blue	brown	china	coffee
desk	dining	filing	green	hard	laundry
living	orange	printer	red	rocking	small
soft	trash	white	work	yellow	

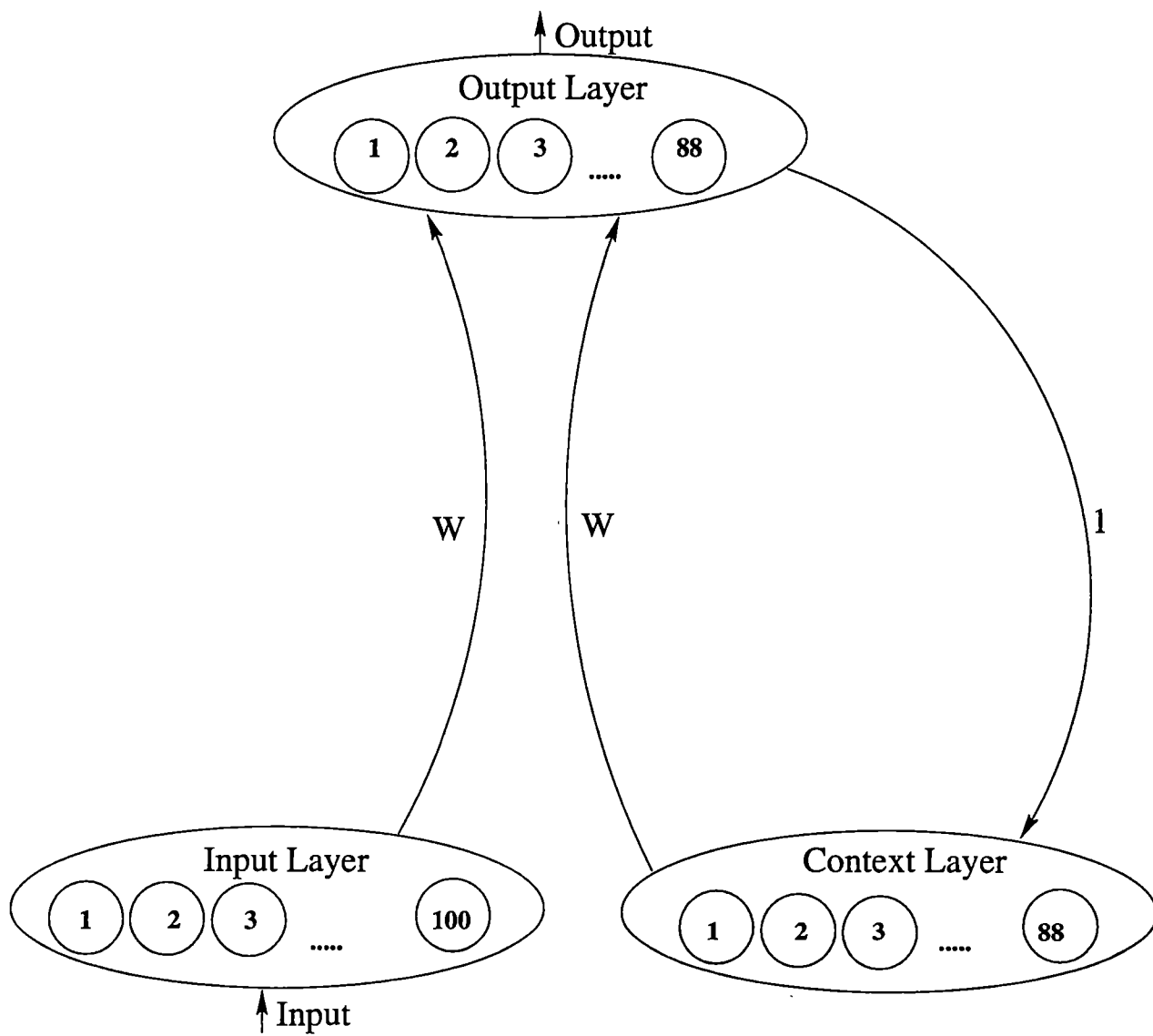
Appendix G

SRN with One Hidden Layer



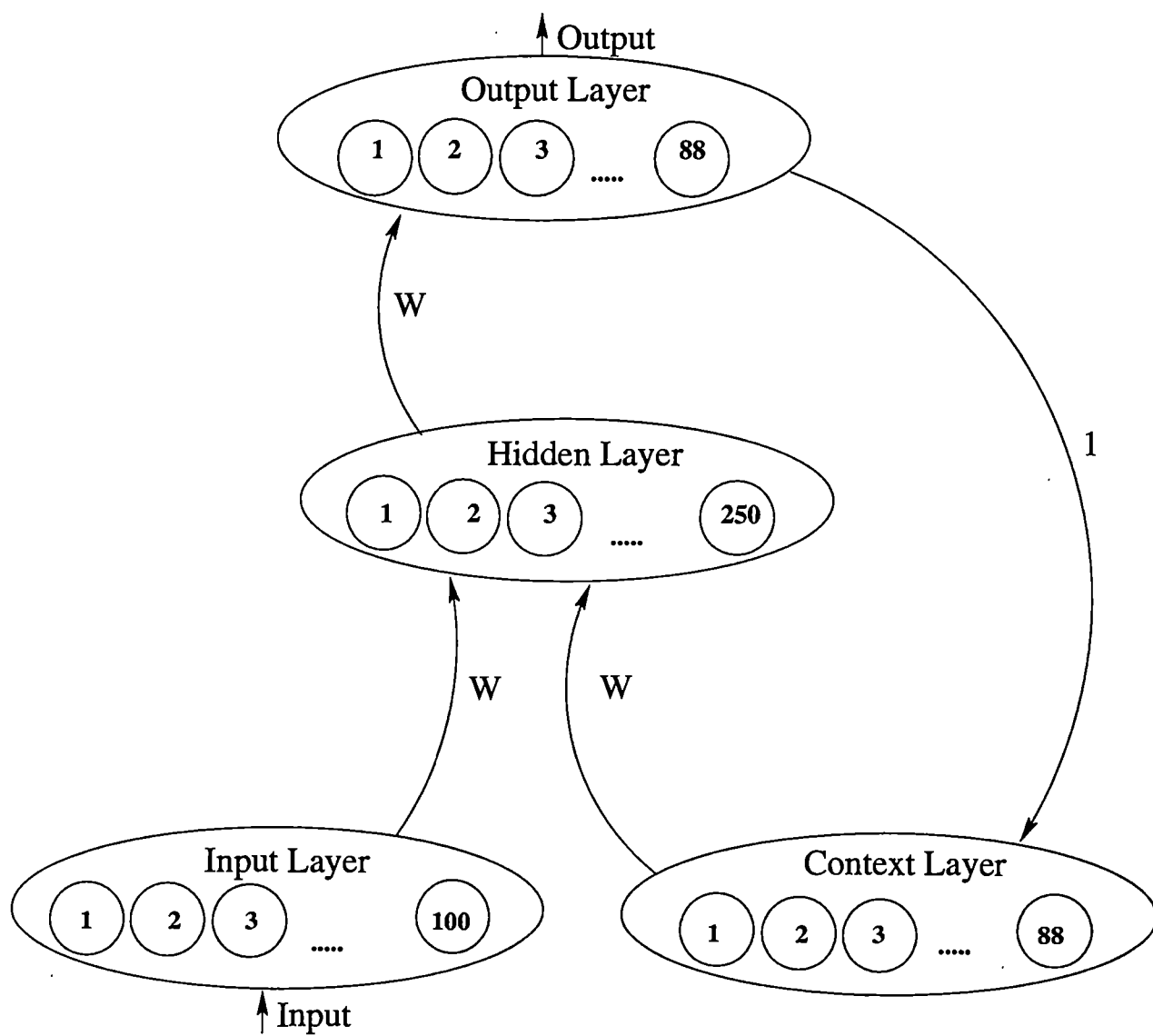
Appendix H

Jordan Network with No Hidden Layer



Appendix I

Jordan Network with One Hidden Layer



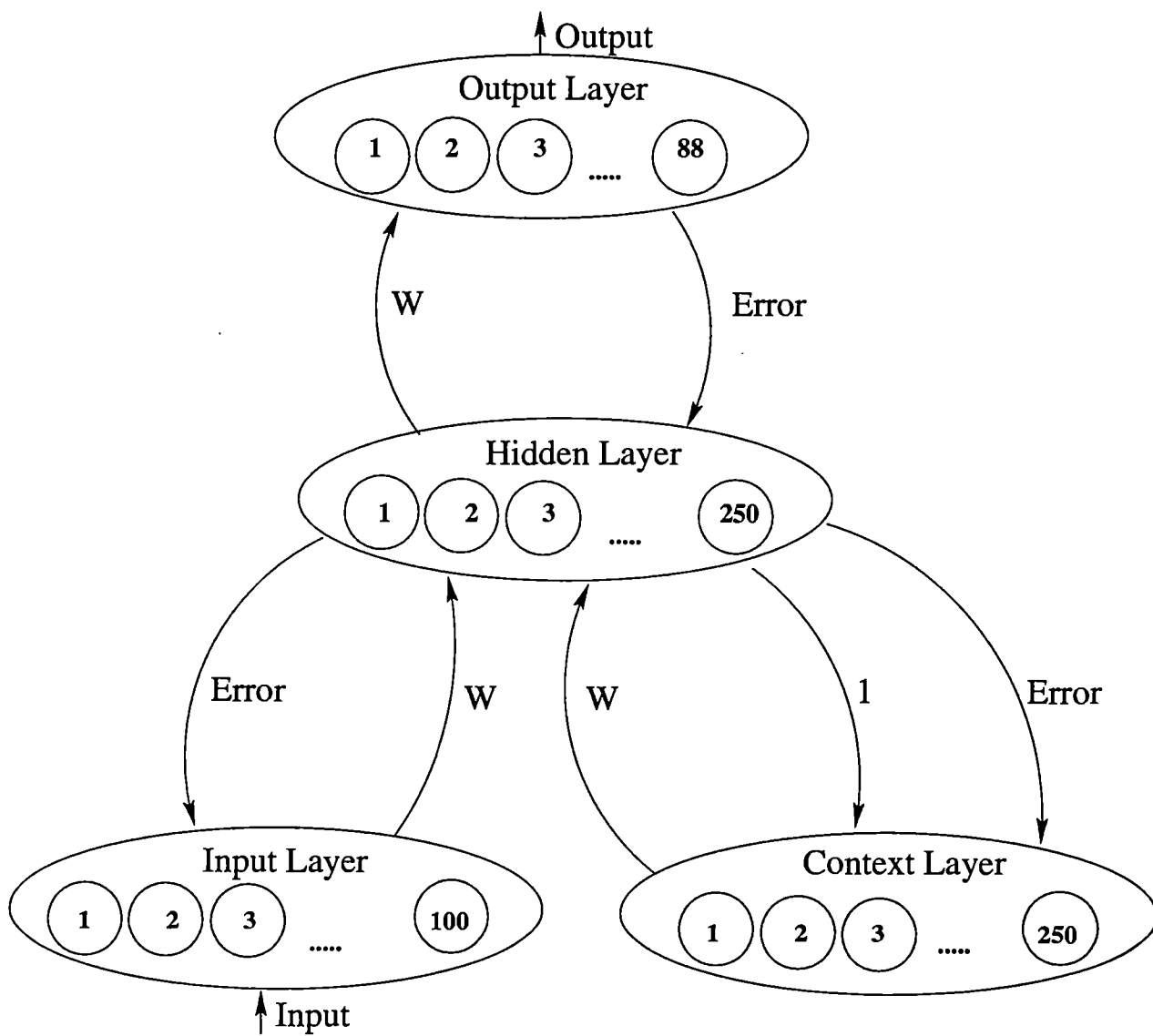
Appendix J

Tests on Neural Networks with Hebbian Learning

TEST	NN1	NN2	NN3	TEST CONDITION
A	Y	Y	Y	Bipolar Input Vectors
B	Y	Y	Y	Binary Input Vectors
C	Y	Y	Y	Normalized Input Vectors
D	Y	Y	Y	Target presented once per word
E	Y	Y	Y	Target presented once per command
F	Y	Y	Y	Target presented once per trial
G	Y	Y	Y	Binary target vectors
H	Y	Y	Y	Bipolar target vectors
I	Y	Y	Y	Weight matrix initialized to all zeros
J	Y	Y	Y	Weight matrix initialized to biased values
K	Y	Y	Y	Weight matrix initialized to small random values
L	Y	Y	Y	Activations initialized to all zeros
M	Y	Y	Y	Activations initialized to midrange value
N	Y	Y	Y	Update weight matrix once per word
O	Y	Y	Y	Update weight matrix once per command
P	Y	Y	Y	Update weight matrix once per trial
Q	Y	Y	Y	Change learning rate by hand
R	Y	Y	Y	Change learning rate automatically
S	Y	Y	Y	Scaled weight matrix vlaues
T	Y	Y	Y	Nonscaled weight matrix values
U	Y	N	Y	Binary output range for hidden units
V	Y	N	Y	Bipolar output range for hidden units
W	Y	N	Y	Tests on hidden layer size
X	Y	Y	Y	Momentum term added to weight matrix update
Y	Y	Y	Y	No momentum term added to weight matrix update
Z	Y	Y	Y	Hyperbolic tangent function added to weight matrix update
AA	Y	Y	Y	Set weight matrix values close to zero to zero
AB	Y	Y	Y	Suppress weak correlations
AC	Y	Y	Y	Bias term added to activation function
AD	Y	Y	Y	Binary sigmoid added to activation function
AE	Y	Y	Y	Decay term added to activation function

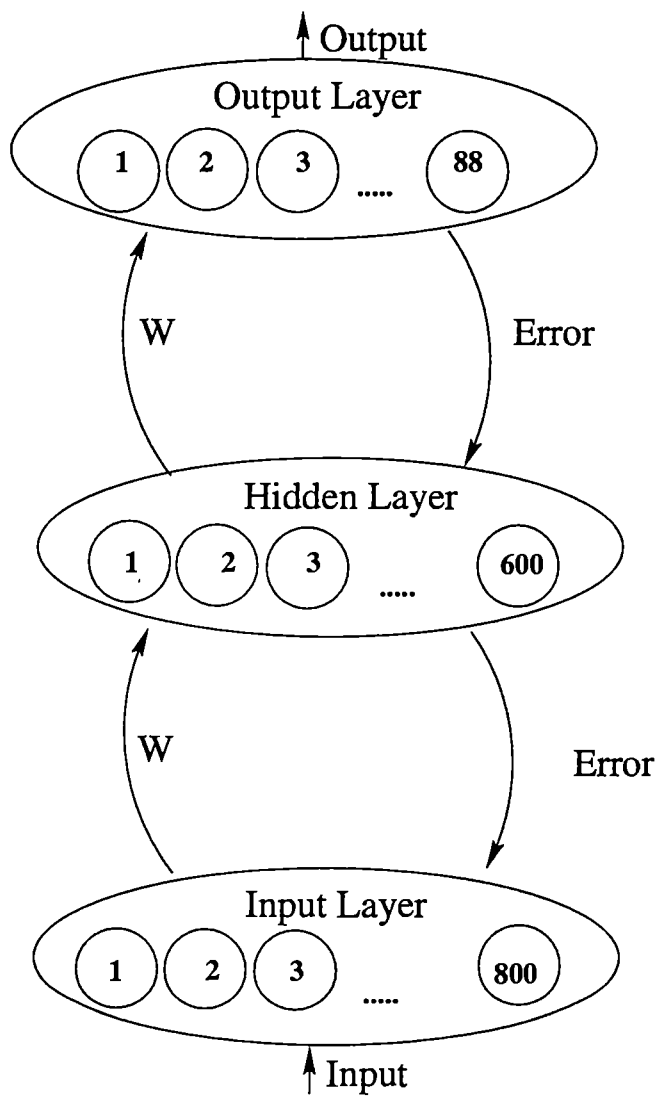
Appendix K

Simple Recurrent Network with Back Propagation Learning



Appendix L

Nonrecurrent Neural Network with Back Propagation Learning



Appendix M

SRN with Hebbian Learning

M.1 Euclidean Distance for Question Four

TRIALS	AVERAGE	DIAGONAL AVERAGE	NONDIAGONAL AVERAGE
100	10.106	0	10.1158
200	9.1272	0	9.136
300	7.7667	0	7.7742
400	6.2544	0	6.2604
500	5.043	0	5.0479
600	3.9376	0	3.9414
700	3.1163	0	3.1193
800	2.431	0	2.4334
900	2.2205	0	2.2227
1000	2.0011	0	2.0031

TRIALS	AVERAGE	DIAGONAL AVERAGE	NONDIAGONAL AVERAGE
100	41.8441	93.5154	41.7941
200	69.1875	111.4797	69.1476
300	103.3038	134.2125	103.2738
400	135.4843	156.0621	135.4667
500	158.3203	172.3418	158.3075
600	174.2166	183.9779	174.2068
700	184.7063	190.8994	184.6974
800	188.9453	194.7085	188.9413
900	189.5657	195.8763	189.5615
1000	190.4398	196.7095	190.4375

M.2 Covariance Matrix for Question Four

Appendix N

Other Results

N.1 SRN with Hebbian Learning - Learning rate vs. Error rate

Learning Rate	Error Rate	Trials
0.00001	1829.6483	1000
0.0001	1871.3375	1000
0.001	846715	1000
0.01	846715	1000
0.1	846715	1000

Hidden Layer Size	Error Rate	Trials
100	812734	1000
150	813653	1000
200	1829.6483	1000
250	847238	1000
300	847223	1000

N.2 SRN with Hebbian Learning - Hidden layer size vs. Error rate

Learning Rate	Error Rate	Trials
0.00001	32.3417	1000
0.0001	25.6418	1000
0.001	23.7506	1000
0.01	21.8184	1000
0.1	20.7133	1000

N.3 SRN with Back Propagation Learning - Learning rate vs. Error rate

Hidden Layer Size	Error Rate	Trials
100	23.3789	1000
150	22.3838	1000
200	21.8185	1000
250	20.9296	1000
300	20.3114	1000
350	19.4908	1000
400	19.2844	1000
500	18.5072	1000

N.4 SRN with Back Propagation Learning - Hidden layer

size vs. Error rate

Learning Rate	Error Rate	Trials
0.00001	3.5882	1000
0.0001	2.1392	1000
0.001	1.6213	1000
0.01	7.4997	1000
0.1	8.4023	1000

N.5 Nonrecurrent Net with Back Propagation Learning -

Learning rate vs. Error rate

Hidden Layer Size	Error Rate	Trials
300	1.7926	1000
400	1.8412	1000
500	1.7240	1000
600	3.5882	1000
700	1.9866	1000
800	2.1306	1000

N.6 Nonrecurrent Net with Back Propagation Learning -

Hidden layer size vs. Error rate

Appendix O

Robot Control System Data

O.1 House 1

ROOM	=	living room	
sofa	loveseat	easy chair	coffee table
television	ashtray	lamp	telephone
curtains	walls	ceiling	fireplace
ROOM	=	dining	
dining table	table chair	hutch	china cabinet
china	lamp	curtains	walls
ROOM	=	porch	
rocking chair	rocking chair	grill	swing
ROOM	=	office	
desk	desk chair	work table	printer
printer	printer paper	computer	book
telephone	walls	curtains	ceiling
files	filing cabinet		
ROOM	=	bedroom	
bed	nightstand	clothes	ceiling
walls	curtains	hair brush	pillow
dresser	chest		
ROOM	=	kitchen	
refrigerator	stove	sink	ceiling
curtains	pantry	soda	dishes
coffee cup	clock	telephone	microwave
washer	dryer		
ROOM	=	bathroom	
sink	bathtub	toothpaste	toothbrush
scale	toilet	ceiling	walls
curtains	mirror		

ROOM	=	living room	
sofa	loveseat	television	coffee table
end table	end table	lamp	telephone
curtains	walls	ceiling	
ROOM	=	kitchen	
refrigerator	sink	stove	dishes
microwave	ceiling	walls	
ROOM	=	living room	
sofa	easy chair	easy chair	coffee table
fireplace	mirror	lamp	walls
ceiling	curtains		
ROOM	=	bedroom	
bed	chest	lamp	nightstand
mirror	curtains	ceiling	walls
ROOM	=	bedroom	
bed	hutch	rocking chair	lamp
chest	clothes	curtains	ceiling
walls			
ROOM	=	bedroom	
bed	nightstand	nightstand	chest
mirror	hair brush	clothes	
ROOM	=	bathroom	
bathtub	sink	toilet	toothpaste
towel	walls	ceiling	washcloth
ROOM	=	porch	
rocking chair	rocking chair	grill	

O.2 House 2

ROOM	=	bedroom	
bed	lamp	book	chest
clothes	nightstand		
ROOM	=	living room	
sofa	end table	book	bookshelf
coffee table	easy chair	stereo	cds
ashtray			
ROOM	=	kitchen	
refrigerator	stove	sink	kitchen cabinet
soda			
ROOM	=	bathroom	
sink	toilet	bathtub	toothpaste
towel	washcloth	toothbrush	hair brush
mirror	soap	shower	
ROOM	=	bedroom	
bed	chest	dresser	clothes
mirror	clothes hanger	laundry basket	

O.3 House 3

ROOM	=	kitchen	
refrigerator	stove	sink	dining table
table chair	dishwasher	kitchen cabinet	soda
dishes	clock	coffee cup	microwave
ROOM	=	dining	
china cabinet	dining table	table chair	picture
hutch			
ROOM	=	living room	
sofa	end table	lamp	easy chair
fireplace	stereo	curtains	walls
ceiling	hutch		
ROOM	=	living room	
sofa	end table	lamp	book
bookshelf	television	easy chair	loveseat
fireplace	clock	curtains	walls
ceiling			
ROOM	=	porch	
rocking chair	swing	grill	
ROOM	=	bathroom	
sink	toilet	bathtub	toothpaste
towel	washcloth	mirror	soap
hair dryer			
ROOM	=	bedroom	
lamp	chest	dresser	bed
mirror	pillow	curtains	walls
ceiling			

O.4 House 4

O.5 House 5

ROOM	=	living room	
sofa	end table	lamp	television
coffee table	loveseat	curtains	walls
ceiling			
ROOM	=	dining	
china cabinet	dining table	table chair	stereo
clock	picture	curtains	walls
ceiling	hutch		
ROOM	=	office	
desk chair	desk	book	bookshelf
computer	filing cabinet	diskettes	printer
printer paper	files	telephone	curtains
walls	ceiling	trashcan	stapler
ROOM	=	bathroom	
sink	toilet	bathtub	toothpaste
towel	mirror	soap	scale
walls	ceiling	shower	
ROOM	=	bedroom	
bed	lamp	picture	pillow
curtains	walls	ceiling	nightstand
ROOM	=	bedroom	
bed	lamp	chest	dresser
hair brush	mirror	curtains	walls
ceiling	nightstand		
ROOM	=	bathroom	
sink	toilet	toothpaste	towel
soap	walls	ceiling	shower
ROOM	=	living room	
sofa	end table	lamp	television
coffee table	magazine	telephone	clock
picture	ashtray	pillow	curtains
walls	ceiling		
ROOM	=	kitchen	
refrigerator	stove	sink	pantry
dishwasher	kitchen cabinet	dishes	curtains
walls	ceiling	oven	microwave

O.6 House 6

ROOM	=	kitchen	
refrigerator	stove	sink	dining table
table chair	dishwasher	kitchen cabinet	soda
dishes	scale	clock	coffee cup
ashtray	curtains	walls	ceiling
oven	microwave		
ROOM	=	dining	
china cabinet	dining table	table chair	china
picture	curtains	walls	ceiling
ROOM	=	living room	
sofa	end table	lamp	television
easy chair	telephone	curtains	walls
ceiling	piano		
ROOM	=	office	
desk chair	desk	lamp	bookshelf
work table	telephone	walls	ceiling
ROOM	=	bathroom	
sink	toilet	bathtub	toothpaste
towel	washcloth	mirror	soap
walls	ceiling	shower	
ROOM	=	bedroom	
sofa	bed	lamp	book
chest	dresser	clothes	mirror
pillow	curtains	walls	ceiling
ROOM	=	bedroom	
bed	chest	clothes	hair brush
mirror	picture	curtains	walls
ceiling	laundry basket		
ROOM	=	bedroom	
bed	chest	dresser	clothes
pillow	curtains	walls	ceiling
nightstand			
ROOM	=	bathroom	
sink	toilet	toothpaste	towel
washcloth	soap	magazine	scale
walls	ceiling		

ROOM	=	living room	
sofa	end table	lamp	bookshelf
television	coffee table	loveseat	stereo
picture	curtains	walls	ceiling
ROOM	=	kitchen	
refrigerator	stove	sink	china cabinet
dining table	table chair	dishwasher	kitchen cabinet
soda	dishes	clock	coffee cup
curtains	walls	ceiling	trashcan
washer	dryer	oven	microwave
ROOM	=	bathroom	
sink	toilet	bathtub	toothpaste
towel	washcloth	mirror	soap
walls	ceiling	shower	
ROOM	=	bedroom	
bed	lamp	chest	rocking chair
pillow	curtains	walls	ceiling
ROOM	=	bedroom	
bed	lamp	chest	dresser
stereo	clothes	mirror	pillow
curtains	walls	ceiling	nightstand
ROOM	=	bathroom	
sink	toilet	toothpaste	towel
washcloth	toothbrush	hair brush	mirror
soap	scale	walls	ceiling
shower			

O.7 House 7

ROOM	=	living room	
sofa	lamp	television	coffee table
curtains	walls	ceiling	
ROOM	=	dining	
dining table	table chair	walls	ceiling
table chair	table chair	table chair	
ROOM	=	kitchen	
refrigerator	stove	sink	dishwasher
kitchen cabinet	dishes	walls	ceiling
ROOM	=	bedroom	
bed	lamp	chest	clothes
clothes hanger	curtains	walls	ceiling
nightstand			
ROOM	=	bathroom	
sink	toilet	bathtub	toothpaste
towel	toothbrush	mirror	soap
walls	ceiling		

O.8 House 8

O.9 House 9

ROOM	=	living room	
sofa	end table	lamp	television
coffee table	dining table	table chair	loveseat
stereo	clock	picture	curtains
walls	ceiling	hutch	
ROOM	=	office	
desk chair	desk	book	bookshelf
computer	filing cabinet	cds	diskettes
printer	printer paper	files	telephone
picture	curtains	walls	ceiling
trashcan			
ROOM	=	bathroom	
sink	toilet	bathtub	toothpaste
towel	toothbrush	hair brush	mirror
soap	scale	walls	ceiling
trashcan	shower		
ROOM	=	bedroom	
bed	lamp	work table	picture
curtains	walls	ceiling	nightstand
ROOM	=	bathroom	
sink	toilet	toothpaste	towel
toothbrush	mirror	soap	walls
ceiling	trashcan	laundry basket	washer
dryer	shower		
ROOM	=	bedroom	
bed	chest	dresser	clothes
hair brush	mirror	picture	clothes hanger
pillow	curtains	walls	ceiling
nightstand	hair dryer	stool	
ROOM	=	porch	
work table	table chair	swing	grill

O.10 House 10

ROOM	=	porch	
rocking chair	swing	trashcan	grill
ROOM	=	office	
desk chair	desk	book	bookshelf
work table	computer	soda	diskettes
printer	printer paper	files	magazine
telephone	clock	coffee cup	curtains
walls	ceiling	trashcan	stapler
microwave			
ROOM	=	bedroom	
end table	bed	lamp	book
television	chest	dresser	coffee table
easy chair	fireplace	stereo	clothes
cds	hair brush	mirror	magazine
telephone	picture	ashtray	pillow
curtains	walls	ceiling	nightstand
hair dryer			
ROOM	=	kitchen	
china cabinet	pantry	dining table	stereo
china	cds	tapes	dishes
telephone	picture	curtains	walls
ceiling	hutch	stool	
ROOM	=	living room	
sofa	end table	lamp	book
bookshelf	television	coffee table	easy chair
fireplace	stereo	rocking chair	cds
tapes	diskettes	magazine	telephone
clock	coffee cup	picture	ashtray
curtains	walls	ceiling	
ROOM	=	kitchen	
refrigerator	stove	sink	pantry
dishwasher	kitchen cabinet	china	towel
washcloth	soap	telephone	clock
coffee cup	toaster	curtains	walls
ceiling	trashcan	stool	oven
microwave			
ROOM	=	bathroom	
sink	toilet	bathtub	toothpaste
towel	washcloth	toothbrush	hair brush
mirror	magazine	scale	picture
curtains	walls	ceiling	trashcan

ROOM	=	office	
desk chair	desk	lamp	book
bookshelf	computer	diskettes	printer
printer paper	files	magazine	telephone
clock	walls	ceiling	
ROOM	=	bedroom	
lamp	television	bed	stereo
cds	tapes	mirror	clock
pillow	curtains	walls	ceiling
nightstand			
ROOM	= dining room		
china cabinet	dining table	table chair	walls
ceiling	hutch		
ROOM	=	living room	
sofa	end table	lamp	television
loveseat	telephone	walls	ceiling
piano			
ROOM	=	kitchen	
refrigerator	stove	sink	pantry
coffee table	dishwasher	kitchen cabinet	towel
dishes	coffee cup	toaster	curtains
walls	ceiling	trashcan	oven
microwave			
ROOM	=	bathroom	
toilet	bathtub	washcloth	toothbrush
hair brush	mirror	soap	curtains
walls	ceiling	laundry basket	hair dryer
shower			

O.11 House 11

O.12 House 12

ROOM	=	bedroom	
lamp	television	bed	chest
dresser	clothes	hair brush	mirror
telephone	clock	picture	ashtray
clothes hanger	pillow	curtains	walls
trashcan	nightstand	laundry basket	hair dryer
ROOM	=	office	
desk	desk chair	lamp	book
bookshelf	work table	computer	filing cabinet
diskettes	printer	printer paper	files
magazine	telephone	clock	picture
ashtray	curtains	walls	trashcan
stapler			
ROOM	=	dining room	
china cabinet	dining table	table chair	china
telephone	clock	picture	ashtray
curtains	walls	hutch	piano
ROOM	=	living room	
sofa	end table	coffee table	easy chair
fireplace	rocking chair	cds	tapes
magazine	telephone	clock	picture
ashtray	curtains	walls	piano
ROOM	=	kitchen	
refrigerator	stove	sink	pantry
dishwasher	kitchen cabinet	soda	telephone
clock	coffee cup	ashtray	toaster
curtains	walls	trashcan	washer
dryer	oven	microwave	
ROOM	=	bathroom	
sink	toilet	bathtub	book
toothpaste	clothes	towel	washcloth
toothbrush	hair brush	mirror	soap
magazine	telephone	scale	clock
walls	trashcan	laundry basket	hair dryer
shower			
ROOM	=	porch	
ashtray	grill		

VITA

Karlyn Ammons Barilovits was born in Asheville, North Carolina on September 23, 1965. She grew up in Mars Hill, North Carolina and graduated from Madison High School in June, 1983. She entered the University of North Carolina at Asheville in August 1983 and received a Bachelor of Science degree in Computer Information Systems in May, 1987. After working as a programmer/consultant for Jet Propulsion Laboratory in Pasadena, California, she entered Clemson University in August, 1987. She received a Master of Science degree in Computer Science from Clemson University in December, 1989. She entered the University of Tennessee at Knoxville in January 1990. While pursuing a Doctorate of Philosophy degree in Computer Science, she taught at Tusculum College in Greeneville, Tennessee. In September, 1993, she married Paul Barilovits. She gave birth to Hannah Marie Barilovits in April, 1998. She received her doctoral degree in May, 1999. She currently resides in Spartanburg, South Carolina with her husband and daughter and has accepted an Assistant Professor position at Converse College beginning September, 1999.