LEARNING THE THEMATIC ROLES OF WORDS IN SENTENCES VIA CONNECTIONIST NETWORKS THAT SATISFY STRONG SYSTEMATICITY

by

Parastoo Geranmayeh

B.Sc., University of Tehran, 2010

Thesis submitted in partial fulfillment
of the requirements for the degree of

Master of Science

in the
School of Computing Science
Faculty of Applied Sciences

SIMON FRASER UNIVERSITY
Fall 2013

APPROVAL

Name: Parastoo Geranmayeh

Degree: Master of Science

Title of Thesis: Learning the Thematic Roles of Words in Sentences via Connectionist Networks that Satisfy Strong Systematicity

Examining Committee:

Dr. Mark S. Drew
Professor
Chair

Dr. Robert F. Hadley
Senior Supervisor
Professor

Dr. Fred Popowich
Supervisor
Professor

Dr. John Alderete
SFU Examiner
Associate Professor
Department of Linguistics

Date Approved: 12 September 2013

Partial Copyright Licence

Abstract

This thesis presents two connectionist models, which can learn the thematic roles of words in sentences by receiving aspects of real world situations to which the sentences are referring, and exhibit strong systematicity without prior syntactic knowledge. The models are intended towards cognitively and biologically plausible connectionist models. Current models could be parts of the larger network to represent the meaning of a whole sentence. The first model, closest, of the two models, to being purely connectionist, attains an acceptable result (98.31% of the roles correctly identified). The second one, not purely connectionist, achieves a perfect result. It could be argued that humans learn the thematic roles, as an emergent property of learning the relationship between the words/sentences and the real world situations. However, it is not claimed that the models are the human learning mechanism for language acquisition.

Keywords: Computational Cognitive Science; Connectionism; Systematicity; Thematic Roles; Neural Networks; Semantic Role Labeling

To Pouran Bakhtiari and Ismael Mohagheghian.

"Language is wine upon the lips."

— Virgina Woolf

Acknowledgments

First and foremost, I would like to sincerely thank my senior supervisor, Dr. Robert Hadley, for a variety of reasons, including his assistance and guidance through writing this thesis, his encouragements, his knowledge, his instructive feedbacks and comments on my writings, and his patience with my mistakes. I consider myself lucky to have had the opportunity to be supervised by him.

I appreciate the comments and suggestions offered by Dr. Fred Popowich, I am also indebted to him for the initial counselling on how to change my academic path to cognitive science. I would also like to thank Dr. John Alderete, for his comments on this thesis.

Most of the writing process of this thesis took place in the central branch of the Vancouver Public Library, therefore, I should thank the library staffs, and specially the architects who created this attractive building, though, I do not think they would ever read this acknowledgement.

Then, I should express my gratitude to my friend and roommate, Caroline, for her patience, especially when I had to leave the mess of papers in the living room.

When I first arrived in Vancouver, two people made me feel at home. I wish to thank Fereshteh and Mostafa Kiarostami for their always beautiful, warm and welcoming home.

Last but not the least, I would like to thank my family and friends. I wish to thank Hanieh Khalilian, Golnoosh Saeedi, Amir Bakhtiari, Amir Aavani, Omid Kiarostami, and so many others who helped me as a friend.

I wish to thank my parents, for they had to devote a considerable time and energy of their youth for me. I wish to thank my grandparents, for their endless love and selfless hearts. The time, I believe, I had to spend to pay back their love, was spent faraway, writing this thesis.

Contents

List of Figures

Chapter 1

Introduction

Although semantic role labeling has been widely studied, the learning of such roles has rarely involved connectionism, and has not addressed the requirements of systematicity. This thesis presents research that is primarily intended as steps towards building cognitively and biologically plausible models that might be similar to parts of the language learning mechanisms in the human brain. Therefore, connectionism and systematicity, as two of the major concepts of cognitive science in language learning, figure prominently in this thesis.

1.1 Semantic/Thematic roles

Roughly speaking, the semantic role of an entity expressed in a sentence is a role played by that entity in the sentence. The semantic role labeling of entities in a sentence leads to the answer of questions such as, "Who did what to whom?" (Palmer et al. [2010]).

A list of semantic roles can be defined to different extents, they can be coarse-grained or fine-grained. In Jurafsky and Martin [2000], thematic roles are defined as a "a single finite list of roles", or more precisely they are a coarse-grained list of semantic roles. Therefore, according to Jurafsky and Martin [2000], they are a subclass of semantic roles.

The proposed models of this thesis deal with thematic roles. This is mainly because employing more fine-grained lists of roles would require a wider range of vocabulary and sentence structures, which is beyond the scope of this thesis. The list of the thematic roles used in the model designed in this research is *agent, patient, instrument, source, recipient,*

coagent, beneficiary, and *location*. More discussion is presented in 4.2.

Two other more fine-grained subclasses of semantic roles are defined in 2.1, and 2.2.

1.2 Connectionism and Systematicity

It is believed that we have certain intrinsic cognitive capabilities to learn a language in a *systematic* way. As an example, one who can understand the sentence "Mary loves John", will necessarily understand "John loves Mary" as well.

Fodor and Pylyshyn [1988] in their paper "Connectionism and cognitive architecture: A critical analysis" stated that human language has syntax and semantics, and there is a *systematic* relationship between the elements of a language. There, they introduced the term *systematicity* to refer to this property of language, and questioned the capacity of the connectionist models for exhibiting systematicity in a language. However, they did not give any precise definition for systematicity.

Later, Hadley [1994] in his paper "Systematicity in connectionist language learning", modified the concept and gave a more precise definition for the term. He defined three degrees of systematicity:

- Weak Systematicity
- Quasi-Systematicity
- Strong Systematicity

Weak Systematicity

A system can perform weak systematicity if it can, during its testing phase, at least successfully process sentences with the following properties:

1. All the words in all the test sentences are previously used while training the system.

2. Each test sentence is grammatically isomorphic to at least a sentence in the training set.
3. Every word in every tested syntactic position was previously used in that syntactic position during the training phase.

Since in some researchers' opinion, the definition of weak systematicity is too weak to be conceived as a type of systematicity, the term can be considered more as a technical term than a degree of systematicity.

Quasi-Systematicity

A system can display quasi-systematicity if, during its testing phase, it can at least

1. display weak systematicity,

   and can successfully process sentences containing embedded clauses such that,
2. all the words in all the sentences are previously used while training the system,
3. both the containing sentence and the embedded sentence are grammatically isomorphic to at least some sentences in the training corpus,
4. and every word in a syntactic position in any embedded sentence is previously used in the same syntactic position in a simple sentence during the training phase.

Strong Systematicity

A system can display strong systematicity if, during its testing phase, it can

1. display weak systematicity,

   and can successfully process simple sentences and sentences containing embedded clauses such that,
2. all the words in all the sentences are previously used while training the system,
3. and there are words in syntactic positions such that those words were not previously seen in that syntactic position during the training phase, in both simple sentences and sentences containing embedded clauses.[1]

[1] As an example, for a system to achieve strong systematicity, if it can process the sentence *"Men sell cookies to girls"*, and the word *boys* has never appeared as the role *recipient* of any verb, but has been previously encountered in other roles, the system must be able to process the sentence *"Men sell cookies to boys"*, as well. Otherwise, the third condition is not met.

Since these two watershed papers, satisfying systematicity, while learning a language, has been an issue in the realm of connectionism. As an example, Hadley and Hayward [1997] in their paper "Strong semantic systematicity from Hebbian connectionist learning", presented a neural network that could successfully exhibit strong semantic systematicity. Their network could deal with rather complex sentences, with embedded clauses. Nevertheless, it could not handle sentences with passive voice.

Later, Hadley and Cardei [1999] in their paper "Language acquisition from sparse input without error feedback", presented an evolved version of the previous network. Their network also exhibited strong semantic systematicity. It could deal with both active and passive sentences, besides more complex sentences with embedded clauses. However, in both these models, certain general aspects of innate knowledge of syntax in the human brain was assumed. Therefore, there are doubts that these models, with minimal prior knowledge of syntax wired into them, are purely connectionist models. It is controversial, however, to what degree a purely connectionist model should be free of such innate knowledge.

1.3 The research task, challenges and contributions

We do not learn a language by memorizing a phrase book, neither do we learn it by passively listening to a radio. Watching, or rather, sensing the world is a requirement when first learning a language. Therefore, a proper model for language learning must be involved not only in being exposed to the sentences, but also to the aspects of the real world situation the sentences are referring, as well.

In this research, connectionist models are designed and implemented to learn the thematic roles of sentence entities by receiving certain information about aspects of the real world situation to which the sentence is referring. More details on the task to be carried out by the models can be found in 4.1. The experiments are designed so that the models are also tested for strong systematicity. The system exhibits strong systematicity, which is a significant achievement. To the best of my knowledge, no connectionist model has so far been designed and implemented that could satisfy strong systematicity while simultaneously learning to assign a substantial range of roles to words. More information on the concept

of systematicity can be found in 3.7.

Also, the system learns the thematic roles without any previous knowledge of the language or syntactic parsing of the sentences. To the author's knowledge, no semantic role labeling system has ever been designed without assuming syntactic knowledge previously learned.

This thesis presents a designed model that can be a part of a larger model for representing sentence meaning as well. After sufficient training, the current model recognizes the thematic role of each word in a given sentence. A certain part of the model holds information on the thematic role of each word. The thematic roles of words can be passed to the other parts of the larger network to represent the meaning of a whole sentence using Holographic Reduced Representations. This larger model, more conclusions, challenges and contributions are discussed in chapter 5.

1.4 Holographic Reduced Representation

Holographic Reduced Representations (HRR) are distributed vector representations for representing compositional structures. By having the capability to represent complex structures such as trees, or other recursive structures, HRR can be employed to represent linguistic as well as logical structures (Plate [1995]).

In both the fields of logic and language, to represent a complex structure using HRR, role vectors are circularly convolved with their corresponding filler vectors and then added to the other role/filler combinations. One possible model for representing a sentence using HRR is to have the semantic/thematic role vector of each entity convolved with its corresponding entity vector. Then, the convolved role/filler vectors must be added to each other to represent the entire sentence. (See formula 1.1, where $*$ represents the circular convolution operation.)

$$sentence\ representation\ = \sum_i word_i * role_i \tag{1.1}$$

Finally, it must be mentioned that a major motivation of this thesis research is to allow a system to learn to discover which roles are relevant to the words in a given sentence so that the roles will be available for use in creating the given sentence's HRRs.

1.5 Thesis organization

The rest of the thesis is organized as follows. Chapter 2 covers a literature review on the semantic role labeling systems. These systems include the FrameNet, and the PropBank project, besides the systems that are based on these two projects, and also an unsupervised system. All the mentioned systems adopt the approaches and pursue the goals that are common in the field of Natural Language Processing. Besides these systems, a connectionist approach is reviewed as well.

In the first half of Chapter 3, basic concepts in connectionism are explained, including neural networks, the backpropagation of errors, Hebbian learning, competitive learning, and auto-encoder neural networks. The second half of this chapter first reviews Elman's famous network (1990) and his experiments with the network. Next, comes a discussion on systematicity. Finally, the chapter ends with a survey on a Hebbian-competitive network. This network together with the Elman's network were the main models that inspired this research.

Chapter 4 presents the primary original contributions of this thesis. First, the task for which the models are designed, is defined. Next, the details of the designed experiments, the designed models, and the results of the experiments are presented.

Finally, after giving a summary of the previous chapter and drawing some conclusions, chapter 5 ends by introducing possible future directions to extend this research.

Chapter 2

Semantic Role Labeling

This chapter reviews systems that are designed for Semantic Role Labeling mostly in the field of Natural Language Processing. Briefly speaking, the purpose of a Semantic Role Labeling System is to assign semantic roles to the words of a sentence. These systems are used in Information Extraction, Automatic Summarization, Machine Translation, and Automatic Question and Answering systems.

Recently, given the availability of semantically annotated corpora, there has been a considerable improvement in the Semantic Role Labeling Systems that are based on statistical learning. In sections 2.1 and 2.2 the two most important annotated corpora, FrameNet and PropBank, are introduced. Then, in section 2.3, Semantic Role Labeling Systems that employ FrameNet and PropBank as their training corpora are reviewed. These system are all designed for purposes that root in the field of Natural Language Processing.

However, there has also been at least some research in the field of cognitive science to design Semantic Role Labeling systems that are cognitively motivated. One of these connectionist systems is reviewed in section 2.4.

All the Semantic Role Labeling systems that are referred to so far use supervised learning algorithms. In section 2.5 a system that employs unsupervised learning is also reviewed.

Finally, it worth mentioning that all the reviewed systems process previously syntactically parsed sentences. This means that for any sentence, a large amount of prior syntactic

processing is done before the system tries to semantically labels its words, whereas the designed system in this thesis *assumes no syntactic knowledge and needs no prior syntactic parsing of a sentence.*

2.1 FrameNet

The FrameNet project [Baker, Fillmore, and Lowe, 1998, Johnson, Fillmore, Wood, Ruppenhofer, Urban, Petruck, and Baker, 2001, Ruppenhofer, Ellsworth, Petruck, Johnson, and Scheffczyk, 2010] is a semantic role labeling project that provides a lexical resource of semantic role labels to be used in data annotation. It had been both beneficial to the fields of linguistics and natural language processing. This project is one of the projects that provides training data for semantic role labeling algorithms, and the first major computational project that provides a source of predicates with their corresponding roles.

The FrameNet project is based on the theory of Frame Semantics [Fillmore, 1982, 1985, Fillmore, Johnson, and Petruck, 2003]. The Frame Semantics theory basically says that, to understand the meaning of a word, you need to know the essential knowledge that relates to that word. For instance, to understand the meaning of the word *sell*, one needs to know the concepts of *money, seller, buyer, etc.*

Roles in the FrameNet are specific to a frame (aka semantic frame). A frame is " a script-like conceptual structure that describes a particular type of situation, object, or event along with its participants and props" [Ruppenhofer et al., 2010]. Therefore, according to the Frame Semantics theory, the meaning of words can be understood based on the semantic frame they belong to.

As an instance, the word *sell* belongs to the *Commercial Transaction* frame and it is defined as follows. [1]

"These are words that describe basic commercial transactions involving a Buyer and a

[1] All the examples in this section are directly taken from the official website of FrameNet project.

Seller who exchange Money and Goods. The individual words vary in the frame element realization patterns. For example, the typical patterns for the verbs buy and sell are: BUYER buys GOODS from the SELLER for MONEY. SELLER sells GOODS to the BUYER for MONEY."

Semantic roles that are defined according to this frame definition are: *Buyer, Seller, Goods, Money, Means, Rate, and Unit.* These verb-specific and also frame-specific fine grained semantic roles are referred to as Frame Elements (FE).

For example, in the sentence: "Robin SOLD a car to Abbey for $5000.", *Robin* is the *Seller*, *a car* is the *Goods*, *to Abbey* is the *Buyer* and *for $5000* is the *Money*.

FEs are grouped in two categories: core and non-cores. Core FEs are the ones that are conceptually necessary for the frame. In the previously mentioned frame, the FEs *Buyer, Seller, Goods, and Money* are the core ones and *Means, Rate, and Unit* are the non-core ones. As in this example, the non-core roles may not appear as parts of the frame definition.

A frame can be defined across different verbs with close semantics. Actually, verbs are grouped together only according to their similarity in semantics, regardless of how different their syntactic behaviours are. For example, the verb *buy* lies into the same category as *sell* does. See the following example: *"Abby bought a car from Robin for $5,000."*

The methodology of the FrameNet is as follows. First, the frames are defined. As a frame is defined, so are the lexical items that invoke that frame. For example, the verb *sell* is one of the lexical items that invokes the *Commercial Transaction* frame. The word that invokes a specific frame is referred to as the Lexical Unit (LU). In most cases, verbs are the frame-evoking word of each FE. When the LUs for each frame are identified, the appropriate roles (FEs) of the frame are defined. At the end, example sentences in the corpus are found and annotated according to the frame.

It is worth noting that no parse tree is used in FrameNet, but annotaters mark the FEs in the text. The FrameNet has 10,000 lexical units and more than 1,000 semantic frames.

2.2 PropBank

Another significant semantic role labeling project is PropBank or Propositional Bank [Palmer, Gildea, and Kingsbury, 2005]. In this project, semantic roles are assigned to the nodes of the Penn Treebank[2]. The main goal behind developing PropBank was to provide an annotated training corpus as the training corpus for supervised machine learning and statistical systems that support automatic semantic role labeling systems. It has been the primary resource for research in this field and has been beneficial to statistical analysis in linguistics, as well.

PropBank is based on the studies of Levin [1993]. In her studies, she claims that the syntactic behaviours of verbs have strong correlation with their semantic behaviours and one can distinguish verbs into semantic classes by examining the syntactic behaviours of them. Therefore, PropBank is mostly based on the syntactic behaviours of verbs, in contrast to the FrameNet that is basically based on the semantics of verbs.

Role labels in PropBank are more coarse grained than the ones in the FrameNet. They do not have meaningful names, are not based on any theoretical standing and are verb-specific. Roles or arguments are mostly named using numbers: *Arg0, Arg1, etc.* . $Arg0$ and $Arg1$ usually corresponds to prototypical *agent* and prototypical *patient*, respectively. Other roles are not necessarily consistent across verbs.

As an example in the sentence [3]:"Chuck bought a car from Jerry for $1000.", *Chuck* is $Arg0$, *a car* is $Arg1$, *from Jerry* is $Arg2$ and *for $1000* is $Arg3$. As another example, in the sentence "Jerry sold a car to Chuck for $1000.", *Jerry* is $Arg0$, *a car* is $Arg1$, *to Chuck* is $Arg2$ and *for $1000* is $Arg3$.

For each verb, the labels of roles/arguments are specified in a framefile. A framefile consists of the set of arguments/roles for each verb and their meaning. The set of arguments for each verb is referred to as a frameset. Different senses of each verb have different framesets. On the other hand, syntactic alternations of a verb, among which the meaning

[2]The Penn Treebank project provides syntactic parse trees for a given text.

[3]Examples are taken from Palmer et al. [2005].

does not change, are kept together in a single frameset.

Compared to the FrameNet, there is less emphasis on the semantics of verbs and the annotation in PropBank remains close to the syntactic level. Unlike FrameNet, there is no metaphorical usage of roles [Ellsworth, Erk, Kingsbury, and Padó, 2004], but PropBank annotates every clause in the Penn TreeBank no matter how complicated the clauses are. In contrast to FrameNet, the sentences are initially syntactically parsed and PropBank actually adds a semantic layer to the syntax tree of every sentence. Finally, since in PropBank the role names are mostly not consistent across verbs, no inferences or generalization can be made based on the role labels.

2.3 Automatic Semantic Role Labeling Systems based on FrameNet and Propbank

In 2002, Gildea and Jurafsky designed a semantic role labeling system based on the FrameNet project. The main goal was to design a domain-independent, robust semantic role labeling system. An algorithm was designed and tested for identifying semantic roles filled by constituents in a sentence. They defined a list of syntactic features for this task. Their method used statistical techniques, including probabilistic parsing and statistical classification.

The algorithm was as follows. First, sentences were parsed using an automatic parser of Collins [1997]. The parser was previously trained on examples of Penn TreeBank. Then, the predefined syntactic features were extracted from the parser. Finally, the probabilities for semantic roles were estimated from the features and the best candidate was picked as the predicted role.

The list of features with a brief description is as follows:

Phrase Type

The phrase type *(phr)* of a constituent is its syntactic category. Different semantic

roles can be correlated to certain types of phrases. The phrase type of a constituent is extracted from the label of the immediate parent of that constituent, in case of unary production, the higher node's label is assumed.

Governing Category

The governing category *(gov)* feature only points to the NP's of a sentence and has two types that can be extracted from the parse tree of a sentence. The governing category of an NP is either S (for subject) or VP (for direct object(s)). As agents mostly appear as the subject of a sentence, this feature also helps to make a better guess about the role of the constituent.

Parse Tree Path

The parse tree path *(path)* feature of a constituent is the path from the target word (verb) to that constituent in the parse tree. The feature can help the semantic role prediction by indicating the syntactic relation of the constituent to the sentence.

Position

Position *(pos)* of a constituent simply indicates if the constituent is before or after the predicate defining the semantic frame. It helps to distinguish subject and objects in case there is an error in the parser.

Voice

The voice *(voice)* of a predicate is either active or passive. Subjects of active voiced predicates normally have the same semantic role as the direct object(s) of predicates with passive voice.

Head Word

Parsers assign each constituent a head word *(head)* using Collins [1999] rules. According to Gildea and Jurafsky [2002], the head word of a constituent is strongly correlated to its semantic type.

When the features of a constituent are extracted, the classifier is ready to get statistically trained. The probability of a given role for a target word is as follows:

$$P(role \mid phr,\ gov,\ path,\ pos,\ voice,\ head, target\ word) = \frac{\#(phr,\ gov,\ path,\ pos,\ voice,\ head, target\ word,\ role)}{\#(phr,\ gov,\ path,\ pos,\ voice,\ head, target\ word)} \tag{2.1}$$

Intuitively speaking, if only the constituents with the same features in the training set are considered, the probability would equal to the number of times the given role was assigned to any constituents over the number of all the constituents.

On the other hand, since the number of features compared to the size and the variety of the training set is large, many of the feature patterns may be hardly covered. Therefore, a simplification is made and the features are assumed to be independent. Then, by using the Bayes rule, the formula 2.1 is simplified, and the counting for every feature is done independently of the other features.

The sentences in the FrameNet were divided into 10 equal-sized groups, nine of which were used for training and the last for testing the system. The system could identify 80.4% of the roles correctly. Then, in another experiment, the set of roles was simplified into a set of thematic roles of size 18. The result of this experiment was slightly better, 82.0%.

As one may remember from previous sections, the main goal behind developing PropBank was to provide an annotated training corpus for automatic semantic role labeling. The same algorithm was implemented on the PropBank training corpus in Palmer et al. [2005]. Unlike the experiment in Gildea and Jurafsky [2002], the training and testing corpora were

parsed by hand, therefore the parsing process was error free.

The result on the PropBank data was that the system could successfully identify the correct role in 82.0% of the cases. Since the number of training examples for each predicate in FrameNet was slightly more than the similar ones in PropBank, parts of predicates were removed so that the results can be comparable. After this change, the experiment could achieve the accuracy of 82.8%.

In later researches on both methods, the list of features was enhanced, and the learning techniques were improved, and better accuracies were achieved. Here, we limit this discussion on these two watershed researches.

2.4 A connectionist approach to Semantic Role Labeling

Apart from some semantic role labeling systems which take machine learning approaches and try to identify semantic roles with acceptable accuracy, there are other systems that take a cognitive science approach and try to implement a semantic role labeling method that is biologically and cognitively plausible. Among these systems, a system called HTRP (Hybrid Thematic Role Processor) is reviewed here [Rosa and Françozo, 1999].

HTRP is a combination of neural networks. Given a sentence, HTRP verifies if the sentence is semantically sound, and if so, outputs the thematic grid of the sentence. The system can handle a predefined list of ten thematic roles. Each sentence can have at most three thematic roles besides the verb. Both semantically correct and incorrect sentences are generated by a sentence generator to be used as the training and the testing corpus of the system.

The main idea behind HTRP involves using meaningful feature vectors to represent words. Each word, either noun or verb is represented as a binary feature vector of size 20. To clarify how this helps the system to identify the thematic grids, take two sentences:

The stone broke the window.

The man broke the window with a stone.

The thematic grid of the first sentence is [Agent, Patient], while the thematic grid of the second one is [Cause, Patient, Instrument]. The two words, *stone* and *man* differ in thematic roles due to their difference in meaning. This difference in the meaning is reflected in the difference in their features. For example, the animate feature is active and the inanimate feature is inactive for the feature vector of the word *man*, while the inanimate feature is active, and the animate feature is inactive for the feature vector of the word *stone*.

While the feature vector of each given noun remains the same, the feature vector of some verbs may vary from sentence to sentence. Take the two above sentences. In the first one, the verb *break* has the feature 'no control of the action' activated, and in the second one, has the feature 'control of the action' activted. When any of these two sentences is presented to the system as an input, it is not yet known which feature vector of the verb *break* must be used, therefore, the average of the two vectors would be provided to the system.

The system contains three identical main neural networks, each for identifying one of the three arguments of the verb. Each neural network contains 11 identical independent subnetworks depicted in figure 2.1, one for the purpose of error detection, and 10 for the 10 thematic roles. Therefore, there are 33 subnetworks in total. Each of the subnetworks consists of three layer: the input layer, the hidden layer, and the output layer. The input layer has 40 nodes, 20 for inputting the verb and 20 for inputting the noun. The hidden layer has only 2 nodes, one for the verb and one for the noun. Finally, the output layer has only one node. In the error detector subnetwork, the activation of this node indicates that the sentence is semantically incorrect. In the thematic role subnetworks, the activation of the output node indicates that the inputted noun has the corresponding thematic role.

Two different versions of HTRP have been employed and tested: one with the random initialization of weights of the networks, and the other with the initially known values for weights. The networks were trained using backpropagation learning algorithm [Rumelhart, Hinton, and Williams, 1986]. It is claimed that after 3,000 epochs of training, the system can identify whether a sentence is semantically correct and if so can build the thematic grid with high certainty. The two different versions show unnoticeable differences in terms of the

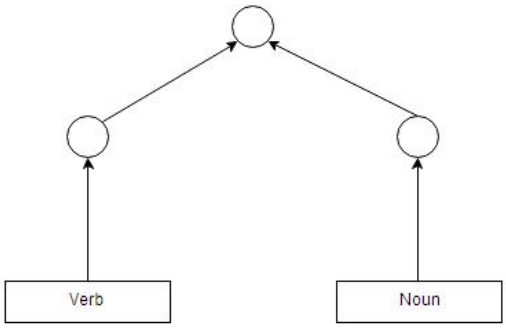

Figure 2.1: The architecture of a subnetwork for one thematic role.

final weights of the trained networks.

The design of the neural networks is highly contrived to get the exact results rather than being a plausible one. Also, since the system needs one neural network for each of the thematic roles, it cannot handle sentences with more than three roles. Moreover, how each sentence is segmented into a verb and nouns, is not explained in the paper. This requires the learner to be already competent about the sentence syntax. One possible explanation could be that sentences are previously parsed. But, this assumption is not fully realistic, since the learner does not yet have knowledge of the semantic role assignments. How a cognitively plausible model can be designed to parse the sentences, the authors do not consider.

2.5 An unsupervised system based on the VerbNet project

All the systems that are so far discussed use a supervised learning method. For supervised learning, a training corpus of labeled data must be provided. This can sometimes be expensive. The final system that will be reviewed here employs an unsupervised learning method [Swier and Stevenson, 2004].

The system is based on the VerbNet [Dang, Kipper, Palmer, and Rosenzweig, 1998, Kipper, Dang, and Palmer, 2000, Schuler, 2005] project. VerbNet is a set of verb classes based

on the works of Levin [1993]. Each class of verbs contains a set of syntactic frame patterns, and also the assigned semantic roles for every slot in that syntactic pattern. Therefore, the classification of verbs is based on the shared role patterns. See the following example of a VerbNet entry[4].

admire:

Frames:

Experiencer V Cause
Experiencer V Cause Prep(in) Oblique
Experiencer V Oblique Prep(for) Cause

Verbs in this class: [admire, adore, appreciate, cherish, enjoy,...]

There are 24 semantic roles defined in the VerbNet project. Besides the syntactic frame patterns and the semantic role assignments, semantic restrictions are also defined. Semantic restrictions are used to constrain the type of words that can be assigned to a semantic role. Unlike frame patterns, semantic restrictions are regarded as preferences rather than hard rules.

As one can see, the VerbNet project is very similar to the FrameNet and the PropBank project. But, unlike FrameNet and PropBank, it has no labeled corpus. Therefore, it is mostly used for research purposes rather than in practically applied semantic role labeling systems.

In the system that is designed based on VerbNet, for each given word in a previously parsed sentence, the system picks the best matching role. The decision for picking a given role for a target word is based on the probability of a role given the verb, the syntactic slot, and the given word. Mathematically speaking, it is based on

[4]This example is taken from Swier and Stevenson [2004].

$$P(role \mid verb,\ syntactic\ role,\ target\ word). \tag{2.2}$$

For each given unannotated sentence, the following procedure assigns roles to the words of the sentence. First, the roles that are unambiguous according to the verb's lexicon in VerbNet are assigned. Then, the following step is repeated iteratively, until there is no unannotated word left. Considering the newly assigned roles, the probability that a given role is assigned to a given unannotated word is recomputed. If the probability of that candidate role, and its difference with the probability of other candidate roles meet a certain threshold, the role is assigned to its corresponding word. When a new set of roles are assigned, the threshold values relaxes one more level. Finally, please note that, the statistics corresponding to each decision were based on the class of the target verb and the target word rather than by the verb and the word itself.

For the testing phase, five random example sentences of each target verb were taken from a text. Next, their argument slots were labeled by human annotators. Then, the result was compared with the system's result. The method could identify 90% of the roles correctly.

Chapter 3

Relevant Fundamentals of Connectionism

This chapter cover concepts, research, and experiments that are immediately used in the connectionist system designed in this thesis. First, in section 3.1 the concept of neural networks, which is the core concept of this thesis, is introduced. Then, a couple of learning algorithms that are later employed are briefly explained. In section 3.2, backpropagation of errors is explained. This very widely used learning algorithm is later employed in the network designed by Elman, that is discussed in section 3.6. Then, Hebbian learning and competitive learning algorithms are introduced in sections 3.3 and 3.4, respectively. These two learning algorithms are later used in the network discussed in section 3.8.

After an introduction to the basic concepts, three designed neural networks are discussed. First, in section 3.5, the auto-encoder neural network is briefly introduced. This neural network will be later used in the design of the main network. Then, section 3.6 reviews Elman's simple recurrent network. The network designed by Elman, and his experiments, had a great impact in connectionist research, especially in the realm of language learning. One of his experiments is discussed in section 3.6.1.

The concept of systematicity, and Hadley's three levels of systematicity are explained in section 3.7. After that, a Hebbian-competitive network is introduced in section 3.8. This

network, which was partially inspired by Elman's simple recurrent network, is another influential network designed in the realm of language learning. Its more detailed architecture, the designed experiment, the information flow in the network, the analysis of the result and a quick comparison to the results of Elman's network come in the following subsections. The network designed in this thesis is very much inspired by the Hebbian-competitive network.

3.1 Neural networks

Neural networks, which are also known as connectionist networks, artificial neural networks, or parallel distributed processing networks are biologically inspired, and some forms of them are biologically plausible models for information processing. In many cases, they are basically very similar to *functions* in the sense of mapping an input to an output.

Their design is based on the idea of the interaction of large number of connected simple elements. The simple elements, whose biological counterparts are known as neurons, are referred to as nodes. Each node in a neural network is connected to some other nodes through links. Each node receives an input signal and sends out an output signal. In some models, an activation function maps the input signal to the output signal. The value of the output signal of a node is called the activation of that node.

The input to each node comes either from the input to the network, or from the activation of other nodes in the network which are connected to that node. Also, the activation of each node is either part of the output of the network or the input to other nodes of the network to which the node is connected. As mentioned earlier, the nodes are connected through directed links. These links carry the activation of one node to another. In most cases, links are weighted and their weights impact the value of activations they are carrying. Mathematically speaking, if nodes $n_1, \ldots, n_k$ are respectively connected through links $w_1, \ldots, w_k$ to node n, and they have activation values $a_1, \ldots, a_k$, respectively, then the input signal to node n would be the following equation, where b is the bias value of node n:

$$b + \sum_i (w_i \times a_i) \tag{3.1}$$

To elaborate the concept, a typical neural network known as a three layered feedforward network will be explained here (See figure 3.1). The first layer of nodes in the network is the input layer. The nodes in the input layer are fully connected[1] to the second layer, and the nodes in the second layer are fully connected to the third layer. Except for these links, there are no other links in the network.

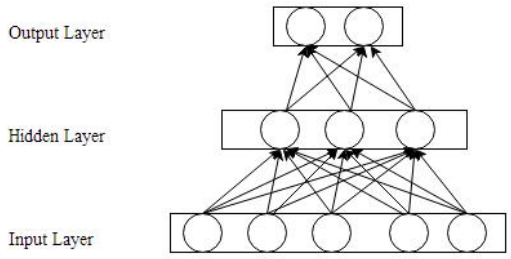

Figure 3.1: An example of three layered feedforward network. The input layer, the hidden layer, and the output layer each have five, two and three nodes. The Input layer is fully connected to the hidden layer, and the hidden layer is fully connected to the output layer.

For a typical three-layer neural network, the following will occur. Imagine that at time t, nodes in the input layer are activated as the result of an external stimulus. The activation of this layer will then move forward to the next layer through the weighted links between the two layers, according to formula 3.1. Then, the activation of nodes in the second layer (aka hidden layer) is calculated according to the second layer's activation function, assuming all the nodes in a layer have the same activation function. Next, the nodes in the second layer will send out their activation through the weighted links that connect them to the nodes in the output layer. Finally, the nodes in the output layer will calculate their activation by inputting their received activation to their activation function, and the output is ready.

[1]If nodes in one layer are fully connected to nodes in another layer, it means that all the nodes in the first layer are connected to all the nodes in the other layer.

This way, an input vector to the network is mapped to an output vector.

One interesting property of neural networks is learning. Learning is the process in which changes occur in the structure of the network so that the network can perform a specific task. Most of the time, the changes occur in the weight of the links. There are various learning algorithms, some of which are explained in later sections. Each neural network designed for each task is usually exposed to a training corpus as its input. The phase in which learning takes place in a network by using a training corpus is referred to as the training phase. A network is usually exposed to the same training corpus for more than once, during its training phase. Each pass through the training corpus is referred to as an epoch. Then after training, the network is tested usually using another set of inputs called the testing corpus. This phase, where normally no learning takes place, is referred to as the testing phase. The network is supposed to ouput the desired output, in this phase. For more reading on neural networks, see Rumelhart and McClelland [1986b].

3.2 Backpropagation of errors

Backpropagation of errors, or simply the backpropagation algorithm, is a supervised learning method which was first used in the realm of neural networks in 1986, in the paper "Learning representations by back-propagating errors" by Rumelhart, Hinton, and Williams [1986]. A supervised learning is a learning method in which the desired output to each input is known in advance during the training phase, and is used as a teacher to the network, so that the network learns to modify itself to produce the desired output. In other words, each input is labeled with the desired output for that input.

In the backpropagation algorithm, the error, which is the difference between the outputted vector and the teacher vector, is propagated in a backward direction of the activation flow in the network, and the link weights are modified accordingly. Consider the three layered feedforward network explained in the previous section. For each input, as the activation flows to the output layer, the output error is first calculated by comparing the output vector with the teacher vector, and the links between the output layer and the hidden layer are

modified accordingly. Then the indirect information about the error is propagated backwards from the hidden layer and the links from the input layer to the hidden layer are modified accordingly.

The backpropagation of error is in fact a very slow process, for this reason besides other reasons, it is often alledged to be cognitively implausible.

3.3 Hebbian learning

Hebbian learning was first discussed in 1949, by Hebb [1949], in the book "The organization of behaviour: a neuropsychological theory". In a nutshell, the Hebbian learning occurs when,

"Cells that fire together, wire together."

Which means that neurons/nodes that are active at the same time *"will tend to become 'associated', so that activity in one facilitates activity in the other."* Hebb [1949]

There are numerous variations, but in most implementations, this means that the weight of the link between two nodes that have activation value of the same sign (positive or negative) will increase, whereas the weight of the links between two nodes that have opposite activation value will decrease.

If the Hebbian learning takes place between two layers of a network, after sufficient training, certain nodes of one layer would be associated with certain nodes of the other layer. More precisely, the links between those pairs of nodes of the two layers, that were often associated during the training, would have a greater weight, than the links between those pair of nodes that have been rarely associated. This way, the network learns the associations between pairs of nodes in the two layers of the network. Therefore, weights of the links actually carry information on the pattern of training data.

An example equation for weight change would be the following equation, where w_{ij} is the weight of the links between node i and node j, r is the learning rate, a_i is the activation

of node i, and a_j is the activation of node j:

$$\Delta w_{ij} = \ r \times a_i \times a_j. \tag{3.2}$$

3.4 Competitive learning

Competitive learning is when nodes in a layer, referred to as a competitive layer, compete to win for a particular pattern of input activation. Competitive learning is an unsupervised form of learning, meaning that unlike supervised learning, inputs are not labeled with the desired output. It is mostly used to detect features or classify patterns of its input layer. One type of competitive learning is briefly explained in the following paragraph.

At first, all the nodes in the layer are identical, except for a randomly distributed parameter that is assigned to each. The nodes are connected together through inhibitory links. When two nodes are connected through inhibitory links, the increase in the activation of one, will decrease the activation of the other one. So, when the activation of the input level reaches the competitive layer, first the nodes compete until the activation of all the nodes stabilizes, then one node, the winner, is assigned the maximum activation level and the rest are assigned the minimum one. Then, the links between the input and the competitive layer are trained, such that each winner will win for a particular input or a particular pattern in the input (See figure 3.2). More reading can be found in Rumelhart and Zipser [1985].

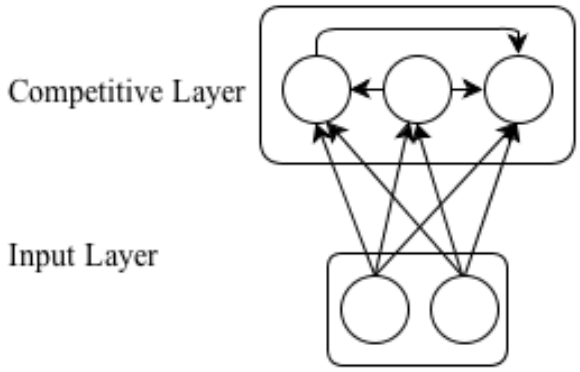

Figure 3.2: A simple network with an input and a competitive layer.

3.5 Auto-encoder neural network

An auto-encoder neural network is a neural network used to map the activation pattern of one layer of nodes to that of another layer of a smaller size, such that there is a one-to-one mapping between the two layers. Imagine we have a layer of nodes, such that the number of nodes needed to represent all the possible activation vectors which can arise in that layer in a specific network is less than the size of the layer. In this case, obviously, the layer could be replaced with another layer of a smaller size, and each possible activation vector with another one of a smaller size. This compression can be done using an auto-encoder neural network.

An auto-encoder neural network is a three-layered neural network (see figure 3.3). It consists of an input, an output and a hidden layer. It is called auto-encoder, because the network is trained to produce the same output as the input. Therefore, the input and the output layers have the same size, and the hidden layer's size is less than theirs.

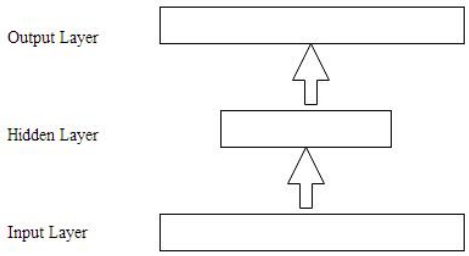

Figure 3.3: The architecture of an auto-encoder neural network.

The network is trained using the backpropagation learning algorithm. If after a number of training epochs the error in producing the correct output is small enough, then the output vector would be identical to the input vector with a small error. Besides, there is a functional relationship between the input and the hidden layer, and also the hidden and the output layer of the network. But, the input and output are almost identical after training, and therefore there is a one-to-one mapping between the hidden layer and the input layer. Thus, the hidden layer could be regarded as the compressed form of the input layer. In this

way the auto-encoder's hidden layer could be employed to represent the compressed form of the input layer. For more reading on auto-encoders, see Rumelhart and McClelland [1986a].

3.6 Elman's network

Many language behaviours express themselves in temporal sequences and thus the concept of time is involved in the field of linguistics. Then, if a system is to deal with language, it must also be concerned with the concept of time. One of the systems that is developed to implicitly deal with this concept is due to Elman [1990].

In 1990, Jeffrey L. Elman, in his paper "Finding Structure in Time", introduced a connectionist network which dealt with the concept of time. The architecture and the flow of information in the network are as follows. The network consists of four layers: input, hidden, context and output layer. The input layer is fully connected through trainable links to the hidden layer, and the hidden layer is fully connected through trainable links to the output layer. There is a loop between the hidden layer and the context layer. One set of links consists of one-to-one links that copy the contents of hidden layer to the context layer, and another set of links consists of fully trainable links through which the activation of the context layer flows to the hidden layer. See figure 3.4.

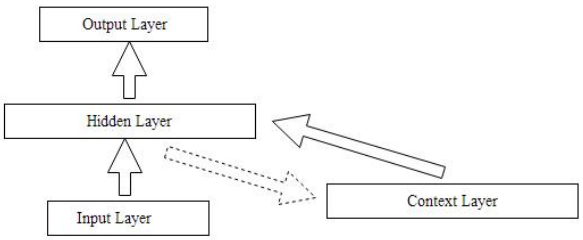

Figure 3.4: The architecture of the network designed by Elman.

The input vectors are presented to the network sequentially; this is done by the use of a clock. Imagine that at time t an input vector is presented to the input layer of the network. The activation of this input flows to the hidden layer.

When it comes to the hidden layer, it is not only the contents of the input layer that feed into it, but the contents of the context layer feed into it as well. The context layer acts as a memory for the network. Whenever the resulting contents of the hidden layer are ready, they will be copied into the context layer. Therefore, the two layers have equal sizes.

The contents of the hidden layer are not only copied into the context layer, but also propagate to the output layer. When the activation of the hidden layer reaches the output layer, it is compared with an input teacher and all the trainable links of the network get trained through backpropagation algorithm. At time $t + 1$ the above procedure is repeated with the next input vector.

As in the feedforward networks, the hidden layer develops an internal representation of the input layer which can usually enable the correct output to be generated. Here, not only the external input provides the input to the hidden layer, but also the context layer which is acting as a memory of the previous states of the network. Therefore, the hidden layer can be interpreted as an approximate encoding of the sequential input, and not only the current input. The degree of approximation depends upon the degree of success of network's training. This characteristic of the SRN will be studied in the next section.

3.6.1 Elman's experiment

In his paper "Finding Structure in Time", Elman reports several experiments implemented with the SRN, among which the most relevant to this thesis is explained here. In this experiment, the task to be carried out by the SRN is to predict the next word of a sentence given previous words of that sentence. For this purpose, 29 words are used, each of which is represented to the network by a vector of size 31. Each vector is a series of 0's with a single bit flipped on. This means that all pairs of vectors are orthogonal and the vector representation of words is not in the form of a distributed representation.

A sentence generator is employed to make short sentences of size 2 or 3 out of a predetermined word set. This sentence generator uses syntactic and semantic restriction to make 10,000 meaningful sentences. These sentences altogether have 27,534 words. To make the

training set, vectors of the words of all the sentences are concatenated in sequence with no breaks between successive sentences. The vectors in the training set are then fed to the network one after another.

The architecture of the SRN and the information processing are identical to what is already explained in the previous section. The input and output layer each have a size equal to the size of each word vector which is 31, and the size of the hidden layer and context layer is 150 units each. The SRN experiences six complete passes through the training corpus.

During the testing phase, Elman first measures the error by comparing the outputted vector with the next word's vector and the result was found to be acceptable, subject to some qualification. Since the prediction task is non-deterministic, measuring the performance of the network is not straightforward. So, Elman defines a likelihood vector for each input vector. The likelihood vectors were of size 31 in which each bit is a fractional number that represents the probability of the occurrence of the word corresponding to that bit, given the current word and the previous words of current sentence. This probability is measured using the statistics of the training set. Using this method results in the observation that the network is outputting very acceptable vectors and is able to extract information from the text.

The question which arises here is whether the network has only memorized the sequence of words or is able to extract classes of words and learn how they are composed together to make sentences. To answer this question, Elman studies the internal representations that are developed by the network, which are the activation pattern produced on the units of the hidden layer for each word as it appears in its context.

For each of the 29 words, the internal representations for that word in all the contexts are averaged and then the averages were subject to a hierarchical clustering analysis. The words are primarily categorized in two groups: verbs and nouns. In smaller details, they are categorized according to their contextual word order roles. The contextual word order role of each word is derived from that word's context.

3.7 Systematicity

As discussed in 1.2, human language has syntax and semantics, and there is a systematic relationship between elements of a language. Fodor and Pylyshyn [1988], in their paper "Connectionism and cognitive architecture: A critical analysis", introduced the term systematicity to refer to this property of language. They claimed that systematicity is not only in the external representation of a language, but also in its internal representation in the brain, and therefore in human thoughts. They believed that connectionism cannot explain this property of language, unless it is used to *implement* traditional linguistic methods.

Later, Hadley [1994] in his paper "Systematicity in connectionist language learning", modifies the concept of systematicity. He defines systematicity in three levels (See section 1.2). Citing experiments previously done in psycholinguistics, he argues that human language exhibits at least strong systematicity. Therefore, it is desirable for a cognitive system to exhibit this level of systematicity.

Hadley's definition also provides a clear way to test a system for systematicity. Since there is no word in any novel position while the SRN is tested in the experiment designed by Elman, Hadley claims that the network was not shown to exhibit strong systematicity.

3.8 A Hebbian-competitive network

As discussed in section 3.6, the SRN designed by Elman [1990] employs local representation of words in which each word is represented by flipping on a bit. To broaden the range of syntactic patterns that a connectionist network can learn and to increase the degree of systematicity it can achieve, the network designed in the paper "Syntactic systematicity arising from semantic prediction in a Hebbian-competitive network", by Hadley, Rotaru-Varga, Arnold, and Cardei [2001], employs distributed representation of words. Here, the distributed representation of a word is a vector in which every bit corresponds to a semantic feature. A set of semantic features were previously defined. The set of features were not chosen to be entirely realistic, but were selected to be examples that help to provide "proof of concept". Consequently, the task defined for the network will be to predict the typical

semantic features of the next word, rather than predicting the next word itself.

The overall architecture of the network is as follows. The network has four layers: the input layer, the first hidden layer (HL1), the second hidden layer (HL2) and the output layer. Like the SRN, a clock determines the presentation of the input to the network. Also like the SRN, the input and output layers are of equal sizes, and their size is equal to the size of the word vectors.

The first hidden layer, HL1, consists of three distinct regions: A, B and C. The input layer is fully connected to the region A through trainable links. Region A is a winner-take-all (WTA) competitive cluster of size 20. Regions B and C act as memory copies of region A. Therefore, region A is connected to region B through one-to-one mapping links, and so is region B to region C. At each time step, the contents of region B is copied to region C, and so is the contents of region A to region B. See figure 3.5.

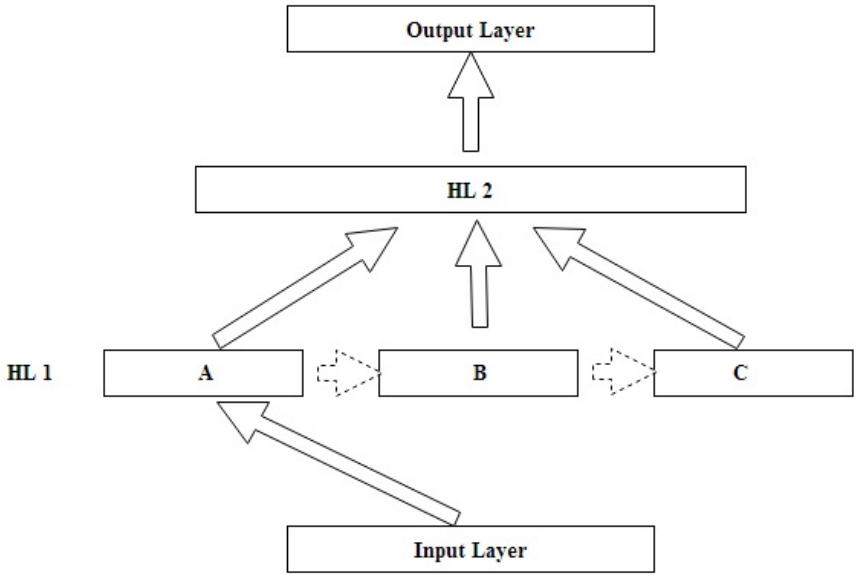

Figure 3.5: The architecture of the Hebbian-competitive network.

All three regions of HL1 are fully connected to HL2 through trainable links. HL2 is also a WTA competitive cluster of size 400. This layer is also fully connected to the output layer

through trainable links.

All in all, the information flow is as follows. The previous contents of region B is already copied into C, and the previous contents of region A into B. The activation from the input layer flows to region A. A competition occurs in region A and a single node is selected as the winner. Then, when the activation of nodes in regions A, B and C is stabilized, the activation flows from HL1 to HL2. There, another competition occurs in HL2 and a single node is selected as the winner.

During the training phase, the output layer is already activated with the semantic vector of the next word and learning takes place on the links from HL2 to the output layer. During the testing phase, the activation of HL2 flows into the output layer to produce an output vector. The details will come in later sections.

3.8.1 Experiment on the Hebbian-competitive network

In the experiment which is designed to test the network, there are 27 words: 12 nouns, 11 verbs, also the words 'that', 'from', 'with', and '.'. The period is used as a break between sentences. Each distinct word is represented to the network by a corresponding semantic feature vector of size 51.

3000 sentences are made out of these words as the training set, and 3000 sentences as the testing set. The sentences are of the following forms: they are either simple sentences of size 3, having the form (Noun Verb Noun), or they have a prepositional phrase including either of the words 'with' or 'from', or they have one or two relative clauses.

During the training phase, two third of nouns are limited to a single position in the sentences, so that the network can later be tested for systematicity. This restriction is subsequently relaxed during the testing phase.

3.8.2 Information flow and the analysis of the results

Input vectors, which are the semantic features of the words of a sentence, are presented to the network sequentially, in the same order as their corresponding words appear in the sentences.

The activation of the input flows to the nodes in region A. Please remember that before any change in the activation of nodes occurs in region A, its previous activation pattern is copied into region B, as is the activation pattern of region B into region C. Then, the activation of the nodes in region A is updated according to the following formula, where $Received_j$ is the weighted sum of activation received in node j, Max_j is the maximum-so-far value of $Received_j$, and c is a constant. This formula is also used in HL2. In region A, $c = 0.9$ and in HL2, $c = 0.1$.

$$V_j = c * Received_j + \{(1 - c) * (Received_j/Max_j))\}. \tag{3.3}$$

The competition starts when V_j is computed for every node in the layer. The winner of the competition is the node with the utmost value of V_j. The winner will then take the value of $+1$, and the activation of all other nodes will fall to zero.

For region A of HL1, repeated instances of such competition eventually results in grouping together words which have close semantic feature vectors. In other words, the activation of words of close semantic features will cause the same node to win.

For modifying weights leading to the winning node of the competitive layer, as in Von der Malsburg [1973], the von der Malsburg's algorithm, which is a Hebbian-inspired competitive learning algorithm, is employed, and uses weight modification equation 3.4, where j indexes the nodes in the competitive layer, i and k both index the nodes in the layer below the competitive layer, a_i is the activation of node i in the lower layer, and finally w_{ij} is the weight of the link from node i to node j, and η_0 is the learning rate. In region A, $\eta_0 = 0.1$ and in HL2, $\eta_0 = 0.001$

$$\Delta\omega_{ij} = \eta_0(a_i / \sum_k a_k - \omega_{ij}). \tag{3.4}$$

Now, the activation of nodes in all three regions of HL1 flows into HL2. There, another competition similar to the one in HL1 takes place. Here, the parameters c and η_0 are set so that the result of the competition would be a one-to-one mapping from activation patterns in HL1 to a winner in HL2.

When the activation in HL2 has settled down, learning takes place in links from HL1 to HL2. The learning algorithm is the same as the one for the links from input to region A in HL1.

At this moment during the *training phase*, the next word's vector is already in the output layer. Links going from HL2 to the output layer are then modified using a simple Hebbian learning method in which each link that connects the active node in HL2 to an active node in the output layer is incremented by a constant value that is equal to 2.5×10^{-5}.

During the *testing phase*, the activation flows from HL2 to the output layer. During the testing phase, as each word in the word list appears in different sentences, all the output vectors corresponding to that word are averaged. For each word, the average of correct output vectors is then compared with the average vector resulting from the testing phase. The comparison is made by calculating the cosine of the two vectors. The result of this comparison is that the network can make an appropriate prediction of the semantic features of the next word of the sentence. This appropriate prediction is made whether the word is in a novel position in the sentence or not.

Briefly, by employing distributed representations rather than the local representation used by Elman, and also due to the architecture of the network, the network was able to group together words of semantically similar types. By the use of this grouping and other aspects of the architecture, it could achieve strong systematicity.

On the other hand, by using Hebbian learning rather than backpropagation learning used in SRN, the network can perform more than 30 times faster than the SRN in Elman's

paper. Comparing the number of training sentences and the number of passes used for training will illuminate this fact.

3.9 The dual-path model

All the already reviewed systems are either aiming for semantic role labeling (chapter 2), or are immediately used in the model designed in this thesis (chapter 3). However, the dual-path model (Chang et al. [2006]) is of interest to this thesis for the similarity of its design to the system proposed in this thesis.

Chang, Dell, and Bock, in their paper "Becoming Syntactic", introduce a system which is designed to resolve some debates in psycholinguistics that are beyond the scope of this thesis. The task to be carried out by their model is to output the next word in a sentence using the previous word in the sentence. According to previous researches (Altmann and Kamide [1999], Federmeier and Kutas [1999], Kamide et al. [2003], Wicha et al. [2003]), the authors claim that word prediction occurs during listening, and the process of predicting the next word affects the process of language acquisition.

Word prediction is not the only task to be carried out by the dual-path system. The model also employs the word prediction skill, together with another component that contains the representation of an intended message, to produce a sentence. In fact, the sequence of outputted words by the system is interpreted as the produced sentence.

The designed model, which is a variant of the dual-path model presented in Chang [2002], consists of two main components (See figure 3.6):

- The sequencing system
- The meaning system

The design of the network is roughly explained in the following paragraphs.

The sequencing system

The sequencing system is basically an SRN. As discussed in section 3.6.1, SRN can

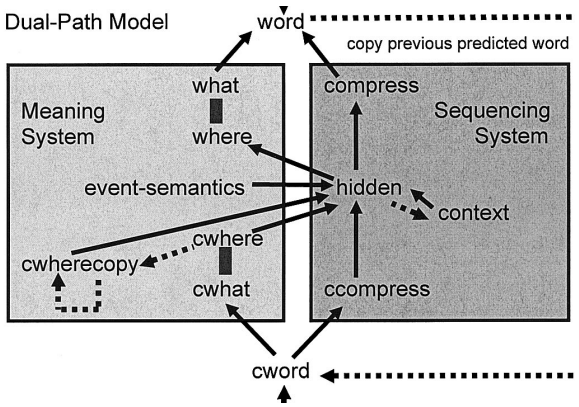

Figure 3.6: The architecture of the dual-path model (Chang et al. [2006]). The right component is the sequencing system, and the left one is the meaning system.

be employed to predict words. The challenge with the SRN is that unlike people, it is not able to use words in "novel ways". The authors do not use the the term systematicity, and by *novel ways* they mean words in novel roles, or words in previously unencountered part of speeches [2], etc.

The words are presented to the system in the same local representation fashion as the Elman [1990]'s experiment. Therefore, to overcome the SRN's challenge in generalization, the modified version of the SRN is employed. This version was initially introduced in Elman [1991]. As one can see in figure 3.6, two layers are added to the SRN's architecture: one between the input layer and the hidden layer, and the other between the hidden layer and the output layer. Since these layers compress the input/output layers [3], they are referred to as compress layers. In the experiments presented in Elman [1991], scrutinization of the contents of these two layers for different words, after sufficient training, revealed that the content of these layers encode important syntactic knowledge such as major syntactic category distinctions or verb class information. Therefore, the modified version of the SRN can help to generalize over words of the same category.

[2]The authors use the word *Google* as an example, claiming that one who has never heard this word as a verb, can understand a sentence in which *Google* appears as a verb, and guess that it means *searching in Google*.

[3]They have less number of nodes than the input/output layers.

The other minor divergence from the Elman's network (Elman [1990]) is that the input to the system is not the current word, but the normalized sum of the current word and the predicted word for the previous word. This minor alteration would help the system to *produce* a sentence, when no current word is available as the input.

The meaning system

The meaning system contains the intended *message* of the sentence. It consists of two parts: the where-what links, and the event-semantics.

The event-semantics contains information on the number of arguments in a sentence, together with their relative prominence, which is basically their order in the sentence. During sentence production they are set from an external environment and remain for the entire sentence.

The where-what link is the temporary binding between the thematic role (where), and the concept (what) of the current/next word. The binding is implemented by temporarily increasing the weight between them. The increase in the weight of the links is also set from an external environment.

When the system is supposed to produce a sentence for an intended message, the intended message is represented to the system via the event-semantics for the whole sentence, and also the where-what links for each word.

The training process of the model is by exposing it to sentences (sequence of words), and also their meanings (intended messages). Except the where-what links, and the links from hidden layer to the context layer, all the other links are trained by backpropagation of error. The model was first exposed to a training set of 60,000 sentence-message pairs. Then it is was tested by being exposed to a set of 2,000 messages for which it was supposed to produce a sentence. The two sets had less than 1% overlap. For 89.1% of the messages, the system could produce a grammatically correct sentence (not necessarily as intended). For

82% of the messages, the intended sentence was produced.

The designed model can only deal with sentences with a single clause, and it is unable to handle sentences with embedded clauses. Also, it is not purely connectionist. The meaning system is a symbolistic system. On the other hand, the structure of the meaning system contains syntactic information of the sentence to be produced.

Chapter 4

Solution Methods

This chapter covers the models designed and implemented for identifying the thematic roles of words in a given sentence, in a way that the proposed model can satisty strong systematicity. First, the task for which the proposed connectionist models are designed and implemented is defined in section 4.1. Then, in section 4.2, the designed experiments by which the models are studied are described.

There are two approaches taken to deal with the defined task. The first approach comes in two versions: the initial version and the modified version. The architecture of the initial version of the designed model, the flow of information in it, the results of the experiment, and the analysis of the results come in sections 4.3.1 to 4.3.6. Next comes the design of the modified version and the results of the experiments and their analysis in section 4.4. Likewise, the second approach is then explained in section 4.5. Finally, the chapter ends with giving the numerical and algorithmic details of different parts of the proposed model in section 4.6.

4.1 Task overview

The goal is to design a connectionist model that can learn to discover and later recognize the thematic roles of words in sentences. The model is intended to comprise a substantial step in the direction of the cognitive plausibility. For example, no exhaustive training is allowed. It must exhibit strong systematicity, as well. Also, the model is supposed to perform

the task for a considerable range of syntactic sentences, including active and passive voices, sentences containing relative clauses, and prepositional phrases among others.

The design of the model is based on the presumption that we do not learn the thematic roles of words in sentences by merely listening to sentences from radio. While learning a language, we are typically exposed to the *situations* to which words and sentences are referring. Arguably, we frequently learn to infer the thematic role of each word by observing the situation that word is referring to in the real world. Then, we learn how to associate the structure of the sentence together with the position of the word in the sentence with a thematic role.

To perform the defined tasks, two neural networks are designed based on the presumption already explained. How the networks are trained and tested is explained in the following paragraphs. What immediately follows holds for both networks.

During the training phase, not only the words of a sentence, but also encoding of aspects of their situation in the real world are presented to the input layer of the network. More exactly, what is meant by *word* is the approximate encoding of *semantic features* of that word. Henceforth, *word* is sometimes the shorthand for "the semantic features belonging to a word".

The situation of an input word is what the listener perceives about the current input word in the real world. For example, if the sentence presented to the listener is: *"Women eat cookies."* , and the listener is also observing that *"Women eat cookies."* , and the current word is *women* and the listener can see that *women* are the doers or agents of the action *eat*, then an aspect of the situation presented to the input layer of the network would be conceptualized, in some manner, as *agent*. Therefore, the input presented to the network in the situation layer could also be interpreted as the thematic role of the current input word in the current sentence. One can assume that the two inputs, the word and its situation in the world, are two parts of a single input presented to two different parts of the network.

During the testing phase, it is assumed that the listener can only hear the sentence, and *cannot see the situation.* Thus, the situation of the current word is not presented to the

network, and the network is supposed to output the thematic aspect of the situation corresponding to the current word. This could be equivalent to the listener's ability in imagining the situation corresponding to the words of a sentence. In fact, the ability of the listener in comprehending a sentence requires him/her to be able to understand the thematic role of each word in a sentence, or in other words, the thematic aspects of the real world situation to which the words are referring.

4.2 The designed experiment

A training and a testing corpus, each containing 3,000 sentences, are designed to train and then test the network. There is also a set of roles assigned to each word of each sentence in both corpora. For training the system the training set and the corresponding training roles are inputted to the network. During the testing phase, only the testing corpus is available to the network. The roles assigned by the trained network to each word of the testing set are employed to verify the outputs produced by the network.

The sentences of the training and testing corpus are generated using a set of 33 words. Each word is presented to the network as a binary feature vector of size 69. Thirty three elements of each vector correspond to a semantic feature. Nouns and verbs are represented using these 33 elements. Thirty six elements correspond to more complicated word features. Prepositions, auxilary verb *are*, and the period are represented using these 36 elements. Therefore, the semantic interpretation of these 36 elements is not straightforward.

The features were not chosen to be entirely realistic, but rather were selected to be examples that help to provide "proof of concept". The selection of features was initially based on the the features selected in Hadley et al. [2001]. Later modifications were needed, and more features were introduced for representing the words not included in the initial word list. It would be later discussed in coming sections that the exact semantic feature vector representation of each word would only affect its semantic classification, and each word is treated according to its semantic class, rather than its semantic feature vector. Therefore, minor changes in each word's semantic feature representation may not influence the whole system.

Features assigned to nouns are as follows: *visible-object, tangible-object, occupies-space, has-shape, can-eat, can-breath, inanimate-object, human, animal, child, female-human, mews, can-fly, location, outdoor, can-play-in, part-of-body, kitchen-utensil, can-eat-with, edible, metal, form-of-money*, and features assigned to verbs are *action, emotive, feeling-nice, involves-light, involves-receiving, involves-delivering, involves-money, has-rules*. Somewhat more arbitrary features are assigned to pronoun *that*, prepositions, words *by* and *are*, and the period. See Appendix A for the list of words and their feature assignments.

Eight distinct thematic roles for noun phrases are used : *agent, patient, location, instrument, coagent, source, recipient, and beneficiary.* A role named *action* is also used relative to the verb of each sentence. In the partially simplified experimental setup, in each sentence, there are also words to which no situation in the real world corresponds. For example, in the sentence *"Women eat cookies in parks"* the word *in* refers to no thematic role. This word, together with the word *parks*, constructs a noun phrase which associates to the thematic role *location.* For the sake of simplicity, a role called *nothing* is assumed to correspond to the word *in* and the thematic role *location* is only associated to the word *parks.* Like words, the roles are also presented to the network as binary vectors, each of size 40. See Appendix B for the list of thematic roles and their vector representations.

The training and testing sentences are generated according to the same grammar rules. According to the verb of each sentence, extra restrictions are set on words when they are taking roles, so that sentences would be meaningful. For instance, the agent of the verb *love* can only be an *animate* noun. Sentences could be as short as two words, or as long as seven, be active or passive, be simple or include an embedded clause. The set of grammar rules is designed so that the models can be tested for strong systematicity, and can cover a wide variety of rules. The list of grammar rules are as follows.

S $\rightarrow$ N V | N V in N. | N V N. | N are V by N. | N V N in N. | N are V by N in N. | N V N PP. | N are V by N PP. | N RC V N. | N V N RC.

RC $\rightarrow$ that N V | that V N

N → women | boys | girls | men | cats | mice | birds | houses | parks | streets | spoons | coins | cookies | hands

V → love | hate | eat | shine | buy | borrow | get | give | sell | lend

PP → Prep N

Prep → with | to | from | for

The full list of grammar rules with the roles assigned to each word, an example for each and the frequency of each type of sentence are further explained in Appendix C.

To test the network for *strong systematicity* , for each role, some words are excluded from taking that role in the training set. This restriction is relaxed during the testing phase. The list of words excluded from taking a certain role during the training, but taking that role during the testing phase is presented in Appendix D.

The training and testing corpora are generated in MATLAB[1]. The network and the designed experiments are also implemented in MATLAB. All the programming codes are included in Appendix E and F.

4.3 First approach

4.3.1 The architecture of the network

The network consists of the following components:

- An input layer through which the current input word is presented to the network,
- A winner-take-all (WTA) competitive cluster layer referred to as the *semantic classifier layer* (SC layer). This layer is similar to region A in HL1 in the Hebbian-competitive network of Hadley et al. [2001]. See section 3.8.

[1]The training and testing corpora are generated using function *trainingSentenceGenerator*.

- A simple recurrent network (SRN) similar to the network in Elman [1990]. See section 3.6.
- A *memory layer*. Links to this layer do not go through any learning process. They only copy the contents of another layer to this layer.
- A three-layered feed-forward backpropagation network (BPN),
- Another WTA competitive cluster layer referred to as the *role layer*,
- And finally, an I/O layer where aspects of the situation corresponding to the input word are inputted/outputted. This layer is referred to as the *situation layer*.

These components are connected together as depicted in figure 4.1. As one can see, one component might be the input or output of another component. How the components are connected is explained in the following paragraphs. The functional overview of each component is explained in sections 4.3.3, 4.3.4.

The input layer is fully connected through trainable links to the semantic classifier layer (this layer is referred to as the input layer in 4.1). The semantic classifier layer is the input to the SRN, thus it provides target values for the output of the SRN as well. The hidden layer of the SRN is connected through links to the memory layer. These links copy the contents of the hidden layer of SRN to the memory layer.

The hidden layer of the SRN together with the memory layer are the input layer of the BPN. These two layers —the hidden layer of the SRN together with the memory layer —are fully connected through trainable links to the hidden layer of the BPN. The hidden layer of the BPN is fully connected through trainable links to the output of the BPN. The output of the BPN is the role layer.

Two different sets of trainable links connect the role layer and the situation layer. One set goes from role layer to the situation layer, and the other one goes from the situation layer to the role layer.

For numerical details on each layer see section 4.6 .

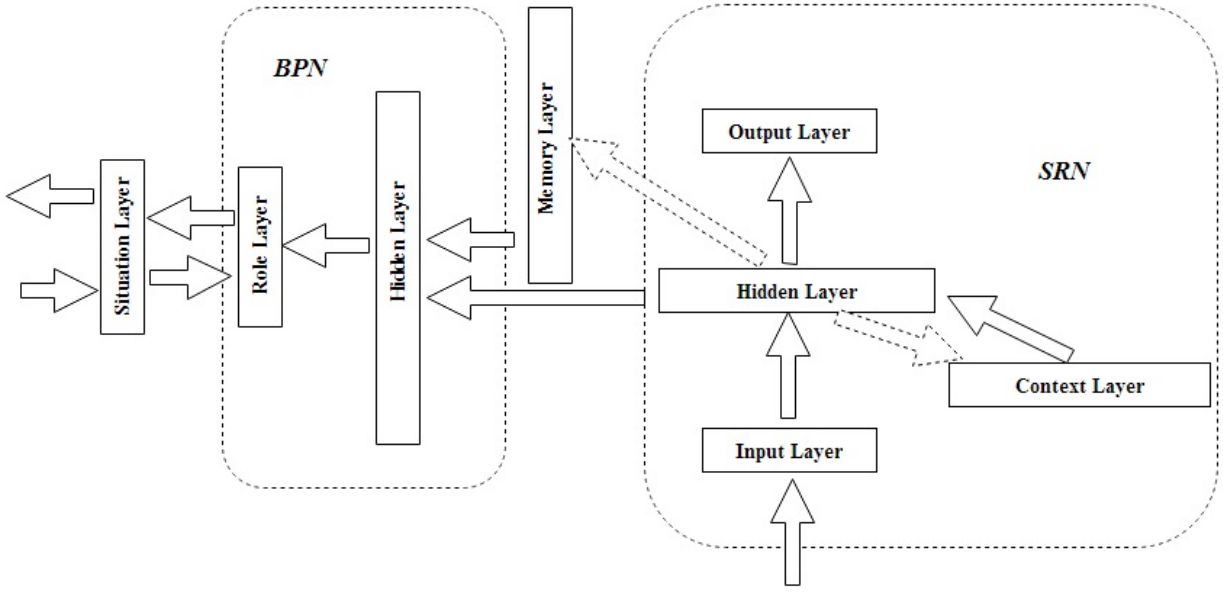

Figure 4.1: The architecure of the designed network and how the components are connected together. BPN stands for the backpropagation network, and SRN stands for the simple recurrent network. Please note that the layer indicated as the input layer of SRN is also the SC layer; and the layer indicated as the role layer, is also the output layer of the BPN.

4.3.2 The flow of information

During the training phase, each sentence is processed by the network twice, or in other words, in two passes. In the first pass, the network goes through the words of the sentence to extract the semantic-syntactic structure of the sentence. The network then keeps the structure of the whole sentence in the memory layer. The memory layer retains that while the second pass is in progress.

In the second pass, the network goes through the words of the sentence again. As each word of the sentence is being processed, the network will have information on the semantic-syntactic structure of the current word together with words that precedes the current word.

Also note that, during the training phase, the *external world* situation of the current word is also available to the network. Therefore, the network needs to learn the functional relationship between the semantic-syntactic structure of the whole sentence, together with the syntactic structure of the part of the sentence up to and including the current word, and the situation of the current word. In other words, the network will learn function f, where f is defined as follows,

$$\begin{aligned} &f((\textit{semantic \& syntactic structure of sentence}, \\ &\textit{semantic \& syntactic position of the word and and its preceding words})) \\ &= \textit{role layer}. \end{aligned} \tag{4.1}$$

This functional relationship will be further elaborated in later sections.

Before moving to the details, it is important to note that when we refer to semantic-syntactic information, we are *not* referring to traditional semantic-syntactic analysis, such as a parse tree, but are referring to *implicit* structural information, such as the kind that Elman [1990] discovered as a result of neural network training. Obviously, no semantic-syntactic knowledge is assumed by the designed network.

4.3.3 First pass

The flow of information in the first pass is depicted in 4.2. The current word of the sentence is now in the input layer. The activation of the input flows to the SC layer (SC layer is the input of the SRN in figure 4.1). The SC layer's parameters are set so that as a result of competitive training, words will be clustered into: *animate nouns, inanimate nouns, verbs*. Also each preposition, pronoun *that*, and *'.'* are clustered differently into single-member clusters. The cluster to which a word belongs is considered as the semantic type of that word. This categorization is the result of feature vectors of the input word, competitive training of links entering the SC layer, together with the number of nodes in that layer.

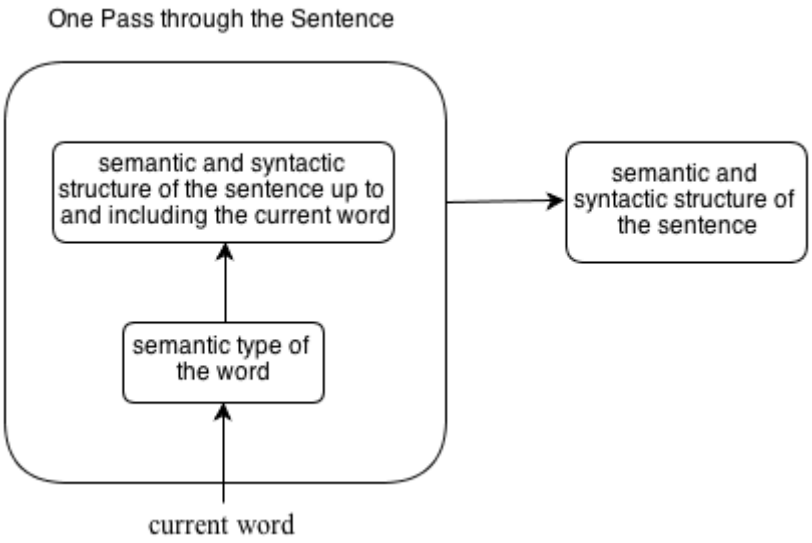

Figure 4.2: The flow of information in the first pass.

When the training pass in the SC layer ends, the content of this layer is copied into the input layer of the SRN. Then, the next word of the sentence is presented to the input layer of the main network. The competition in the SC layer goes on again and the content of the SC layer is copied into the output layer of the SRN. When the input and the output of SRN are ready, backpropagation learning takes place in the SRN as described in Elman [1990].

The flow of information in the SRN is identical to the one described in Elman [1990]. The input of the network is the semantic type of the current word, and the SRN's task

is to predict the semantic type of the next word in the sentence. On the other hand, the semantic type of one word is not enough to reliably predict the semantic type of the next word. Therefore, when the learning takes place in the SRN, the network learns to keep the semantic types and sequential order of all previous words in its context/hidden layer. When the next word is '.', the sentences' first pass is completed and the content of the hidden layer of the SRN is ready to be copied into the memory layer. In both passes, just after the last word of the sentence is processed, the context layer's content is cleared to make the SRN ready for the next pass/next sentence. This is done, because, at this point, the content of the context layer of SRN contains information on the already processed sentence, and will interfere with the information of the next sentence.

The content of the memory layer now contains information on the semantic-syntactic structure of the sentence. By this, we mean that if the contents of the memory layer for different sentences are compared, the ones with more similar semantic-syntactic structure will be closer in Euclidean metrics. As an example, if clustered, the ones for active sentences will be clustered together and differently from the ones for passive sentences.

4.3.4 Second pass

The flow of information in the second pass is depicted in 4.3. In the second pass, again the SC layer and the SRN are used as described in the earlier section. The content of the memory layer is ready from the first pass. When the content of the context layer of SRN is ready as well, the activation of these layers propagate into the hidden layer of BPN and from there to the output layer of the BPN which is the *role layer*.

At this moment during the training phase, the *situation layer* is also provided as an input to the network. The activation of the nodes in the *situation layer* propagates to the *role layer* and then the WTA competition takes place in the *role layer*. The single winner node in this layer corresponds to the thematic role of the current word, once the *role layer* is well-trained. When the competition is completed and the winner is selected, the output of the BPN is ready and the backpropagation learning takes place in the BPN. The learning takes place first between the *role layer* and the hidden layer of BPN, and then between the hidden layer and the input layer of BPN which consists of the context layer of SRN and

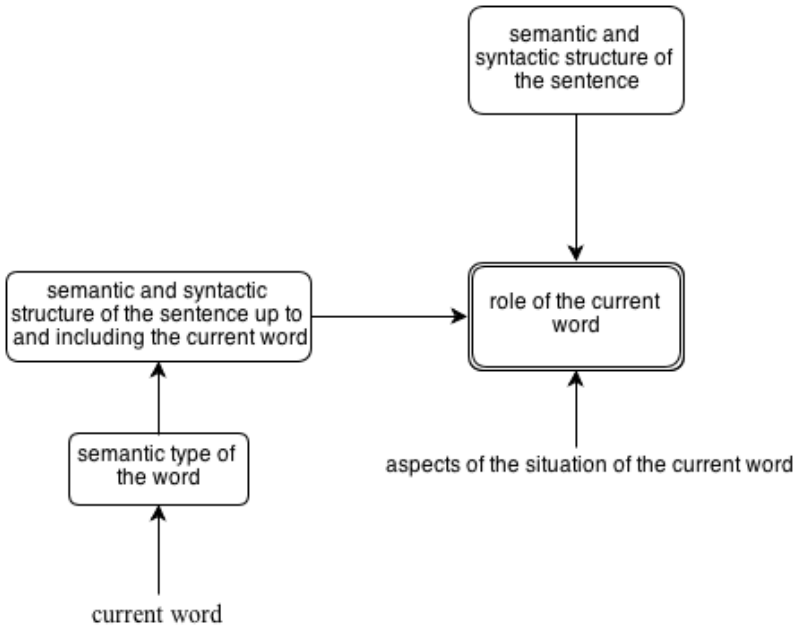

Figure 4.3: The flow of information in the second pass.

the memory layer.

When the competition in the *role layer* ends, a simple Hebbian learning also takes place between the *role layer* and the *situation layer*. This learning should not be confused with the one that takes place in the opposite direction from the *situation layer* to the *role layer*. The purpose of this learning will be more clear in the explanation of the testing phase.

At this level, the word's processing is finished, and the network moves to the next word of the sentence or the first word of the next sentence.

In summary the function of each component of the network is as follows.

- The SC layer classifies the words into semantic classes according to their semantic features.
- The SRN learns the semantic-syntactic structure of the sentence up to and including the current word in the input.
- The memory layer stores a distributed encoding of the semantic-syntactic structure of

the whole sentence for the second pass.

- The BPN learns the association between the semantic-syntactic structure of the whole sentence together with the semantic-syntactic structure of the sentence up to and including the current word and the thematic role of the current word.

- During the training phase, the role layer classifies the provided aspects of the situations of the current word into thematic role classes. During the testing phase, when the nodes in the layer are activated, they feed into the situation layer to output the identified situation of the current word.

4.3.5 The testing phase

For testing the system, it is assumed that the system is only *hearing* or *reading* the sentence, but does not know the external situation. Testing the system is to see whether it is able to understand the sentence and "imagine" the aspects of situation corresponding to each word.

Therefore, during the testing phase, nothing is presented to the *situation layer*. When the activation reaches the hidden layer of BPN, it moves forward to the *role layer* and from there to the *situation layer*. When the nodes in the *situation layer* are activated, the network is actually outputting aspects of the real world situation that corresponds to the role being played by the current word. In other words, the listener is imagining an aspect of situation of the current heard/read word of the sentence in the real world.

4.3.6 Results of the experiment and the analysis of the results

The network was trained[2] and then tested[3] using the corpora described in section 4.2. First, the input layer, the SC layer and the SRN were isolated and trained for 25 number of epochs through the training set. This might be justified by assuming different learning rates for different regions of the brain. Then, the whole network was trained for 270 more epochs.

[2]Function *train* implements the training process of the network.

[3]The network is tested using function *test*.

Finally, the network was tested using the testing corpus.

Altogether around 1008 words' roles and situations among 15,596 were not outputted correctly. This means that around 6.26% of roles and situation vectors were not outputted correctly. Since none of the two errors were in the same sentence, around 33.6% of sentences had a mistake in them. Several experiments were run and they had almost the same result. Also, using more epochs of training was of no help for decreasing the errors. Due to the small number of the thematic roles, the testing results may not be acceptable.

To see where the problem comes from, the context layer of SRN for all the positions in all the words were clustered. The result of clustering was acceptable for all the sentences, whether erroneous ones or not; meaning that words and their preceding words that had identical semantic types were grouped together and differently from other words.

More scrutinizing on the experiments led to the idea that the problem may have arisen due to the large size of the BPN's hidden layer. The BPN is too big, and if its hidden layer were to be directly decreased in size, it would not function due to its large input layer. Therefore, in the modified version a method is used to decrease the size of the input of the BPN and thus its hidden layer. Details on the numerical parameters could be found in section 4.6.

4.4 The enhanced version

One solution to decrease both the size of the input layer and also the hidden layer of the BPN is to find a way to decrease the size of the hidden layer of SRN and thus the memory layer. But according to the experiments, the size of the hidden layer of the SRN cannot be decreased, or the SRN will not function well. On the other hand, one could think of compressing the content of this layer into a smaller layer. An auto-encoder (AE) is employed for this purpose. The architecture and the information flow of AE was previously explained in section 3.5.

By employing an AE to compress the hidden layer of SRN, the architecture of the previously designed network will be modified as follows. The hidden layer of SRN will function

as the input layer of AE, which in turn propagates activation to the hidden layer of AE. The hidden layer of AE is connected to its output layer, as previously described in section 3.5. Also, the hidden layer of AE is connected to the memory layer, and together with the memory layer, it will function as the input layer of BPN. Therefore, the links from the hidden layer of SRN to the memory layer and to the hidden layer of BPN will all disappear. See figure 4.4.

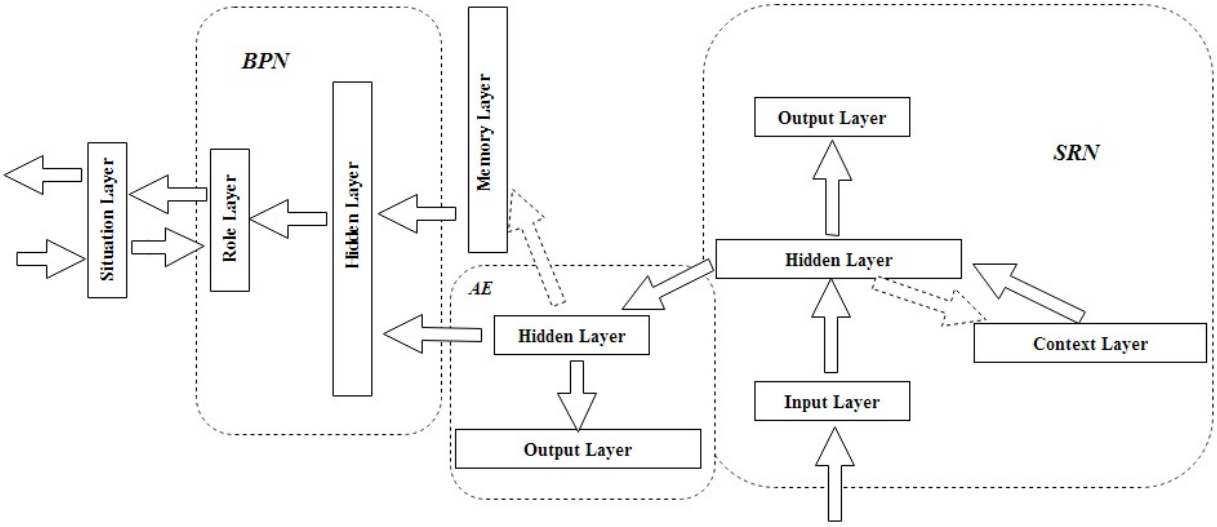

Figure 4.4: The architecture of the modified designed network and how the components are connected together when AE is included.

The information flow in this modified version is as follows. At the end of the first pass, instead of copying the content of the hidden layer of SRN to the memory layer, the content of the hidden layer of SRN is inputted to the AE and after training the AE sufficiently, the content of the hidden layer of AE is then copied into the memory layer. This is done, because the hidden layer of AE after training, will contain a compressed equivalent of the content of the SRN's hidden layer.

In the second pass, instead of presenting the hidden layer of the SRN as the input to the BPN, the hidden layer of SRN is presented as the input to AE and as the information flows to the hidden layer of AE, the content of the hidden layer of AE together with the memory layer will provide the input to the BPN.

4.4.1 Results of the experiment and the analysis of the results

The enhanced network was trained and then tested using the corpora described in section 4.2. As in the initial version, first the input layer, the SC layer and the SRN were isolated and trained for 25 epochs. Then, the training continued with that part of the network attached to AE for 175 epochs. Finally, all the network was under training for 300 more epochs.

The testing phase revealed that among the 15,596 word/role pairs, 263 predicted roles were not correct. This means that 1.69% of the outputted role/situations were not correct. These incorrect outputs were always for the last word of the sentence, and since there were 3,000 sentences in total, 8.77% of sentences had an incorrect output for their last word. For the rest of the sentences, which is 91.23% of the whole testing corpus, the roles and situations for each of the words in each sentence were identified correctly.

The pattern of errors were as follows. The *recipient role* was sometimes identified as the *source role*, and the *source role* was identified as the *recipient role*. 23% of all *recipient roles* were mistakenly identified as the *source role* and 64.67% of the *source roles* were mistakenly identified as the *recipient role*. All the other roles/situations were identified correctly. This means that all the *agent, patient, location, instrument, coagent, and beneficiary roles*, as well as 77% of all the *recipient roles* and 35.33% of all the *source roles* were identified correctly.

As the layers of the network were scrutinized, it was found out that a possible explanation for the incorrect identifications is that the content of the hidden layer of SRN for the confused roles are very similar. The syntactic and semantic structure of the troublesome sentences are identical except for the preposition. Therefore, the similarity in their content layer of the SRN is inevitable.[4] *Fortunately, the erroneous role assignment will vanish upon a further refinement which is described presently.*

[4]For example, for the two sentences *Men sell cookies to boys*, and *Women buy spoons to girls* the semantic-syntactic structure would be very similar.

The network was trained and tested for couple of times and around the same number of errors happened each time. The errors always involved the last word of the sentence, but the incorrectly identified roles varied from one experiment to another.

One may mistakenly assume that the better performance of this version of the network compared to the initial version is due to the greater number of epochs employed in this version. But, using more epochs did not improve the performance of the initial version. Besides, the greater number of epochs was used to train the AE, and not the BPN which is the troublesome part of the network. Moreover, it is worth noting that the sizes of the layers of BPN in the modified version were dramatically decreased, and this means that the training time for each epoch was also dramatically decreased.

Finally, please note that according to the experiments, the network's ability in identifying the right role was independent of the word itself, meaning that it would output the same percent of errors for words in novel positions as the ones in the already encountered positions. This means that the network exhibits strong systematicity.

4.5 Second approach

Since, the human brain could function better than the already described systems, further refinements have been adopted to improve the accuracy of the system just described, and to make the training phase shorter. Please note that the following approach is more of a theoretical value to support the idea behind the system design. The new system may be a too simplified version of the human learning, and may or may not be biologically plausible. The newly designed network is explained presently.

As discussed in section 4.3.2, there is a functional relationship between the semantic-syntactic structure of the whole sentence, together with the semantic-syntactic structure of the part of the sentence up to and including the current word, and the situation/role of the current word. The function was previously defined in (4.1).

In both of the preceding approaches, the input layer of the BPN was either the hidden layer of the SRN or its compressed form, together with the memory layer, which reflected either the hidden layer of the SRN or its compressed form for the whole sentence. The output layer of the BPN was also the role layer. This means that the purpose of BPN was to learn the function f.

4.5.1 The architecture of the refined network

In the following refinement, instead of employing a backpropagation network to learn this function, a *lookup table*, henceforth referred to as LUT, is used to memorize this function, and the BPN is eliminated. Since the LUT does not necessarily need data compression, the AE is also removed. Therefore, the architecture of the overall network is substantially simplified.

The input data value of each entry in the LUT, which will correspond to the input of the function f, will be the content of the hidden layer of the SRN, and also the content of the memory layer. The output value of the entry in LUT will be the current content of the role layer.

As explained in previous sections, the role vector is ready when the situation layer's activation flows to the role layer and the competition in the role layer is finished. Therefore, the situation layer and the role layer remained unchanged.

All in all, the neural network will consist of the input layer of the network, the SRN, the LUT, the role layer and the situation layer. The architecture of the links between these layers will be identical to the first approach.

4.5.2 The flow of information

As in the previous approaches, the training corpus designed in section 4.2 is employed to train and test the network. The SRN is isolated and trained for 30 epochs, and then in a single epoch through the training corpus, the LUT is created.

In the LUT creation process, for every word in each given training sentence, the information flows as in the first approach in two passes, and the vector *(hidden layer of SRN, memory layer)* is constructed when its two parts are ready in the network. This vector is actually the concatenation of its two parts. Then, a new entry for every newly encountered *(hidden layer of SRN, memory layer)* vector is created. This new entry also saves the output of the function as its output field in the entry, which is the corresponding role vector.

The process of adding a new entry for the *(hidden layer of SRN, memory layer)* vector for each word is as follows [5]. For each word in each sentence, the LUT is linearly searched, and the *(hidden layer of SRN, memory layer)* vector of the word is compared with each input entry thus far stored in the LUT. The comparison is made by computing the normalized dot product of the two vectors. To check if two vectors are close enough, the result of the normalized dot product is compared with a certain threshold. This threshold, which is 0.999, is set experimentally.

If the result of the comparison for any of the entries indicates that the two vectors are close enough, it means that the *(hidden layer of SRN, memory layer)* vector is almost identical to one of the entries in the LUT. Therefore, the search is terminated and no new entry is added to the LUT. If none of the entries in the LUT is close enough to the *(hidden layer of SRN, memory layer)* vector, then this vector together with its corresponding output field is added to the end of the LUT. Finally, as the result of going through the training corpus for one time, around 6,500 entries were saved in the LUT.

In the *testing phase*, for each word in a sentence, again the information flows in the network in two passes. When the *(hidden layer of SRN, memory layer)* vector has been produced for a word, the LUT is consulted. The process is as follows [6]. The LUT is searched linearly and the entry with the greatest normalized dot product of its input field with the *(hidden layer of SRN, memory layer)* vector of that word is found, and its output field is returned as the role vector of the current word in the testing corpus. As the role vector is extracted, the role layer is updated, and the activation flows from there to the situation layer.

[5] Adding a new entry vector to the LUT is implemented using function *findSentenceWordType*

[6] Function *findClosestInTable* searches the LUT for the closest match of the input vector.

The reasoning behind this process is as follows. The word in the training corpus corresponding to the selected entry of the LUT, supposedly had the most similar or even identical semantic-syntactic structure of the sentence, together with semantic-syntactic position of the word and its preceding words, to the current word in the testing corpus. Therefore, its corresponding output entry, which was the role vector of the word in the training corpus, is output as the role vector of the current word in the testing phase.

4.5.3 The results of the experiment

During the testing phase, all the words' role/situation in all sentences were identified correctly. So, the accuracy of this method is 100%, which is a notable advance. Moreover, the network *exhibits strong systematicity*, which is one of the major goals of this thesis.

The result of the refined network demonstrates that the overall logic of using the competitive learning together with the SRN and memory layer, was entirely correct. The previous difficulties arose from the vast search space created by the current application of the back-propagation algorithm, given the size of the input vectors within the BPN.

4.6 Numerical and algorithmic details of the designed networks

In the following subsections the numerical and algorithmic details of each part of the designed neural network in all the versions is elaborated.

4.6.1 The Input Layer

The input layer is the layer through which the feature vector of each word in a sentence is presentend to the network. Therefore, the size of this layer equals to the size of the feature vector size of the words, which is 69.

4.6.2 The two competitive layers: the Semantic Classifier Layer and the Role Layer

There are two WTA competitive cluster layers in the network: the SC layer and the Role layer. The structure of these two layers and the links leading to them are identical in all the designed versions [7]. These layers are similar to the region A of HL1 in Hadley et al. [2001], which is previously described in section 3.8.

The two competitive layers have similar structures. They both have 30 nodes. The parameter c, which is used in formula (3.3) to select the winning node, is set to 0.5 in the SC layer and 0.6 in the Role layer.

Initially, links leading to these two competitive layers take a random weight value between 0 and +1, such that the sum of weights leading to each node in the layer equals to one.[8] During the training phase, the weights of the links leading to these two layers are modified using formula (3.4). The learning rate or the η_0 parameter in this formula is set to 0.5 in the SC layer, and 0.1 in the role layer. These values are selected experimentally.

4.6.3 The Simple Recurrent Network

The SRN employed as a subnetwork in all these neural networks is identical to the SRN introduced in Elman [1990]. This network is previously discussed in section 3.6. The input, the hidden/context, and the output layer have 30, 200 and 30 number of nodes, respectively.[9]

Initially, the links between the layers of the SRN take a random value between -0.1 and +0.1. Then the backpropagation learning algorithm is used for training the network. The backpropagation learning parameter is initially set to 0.1 and gradually drops to 0.001 after a certain number of epochs. Finally, please note that a normal Sigmoid function is used as the threshold function of the backpropagation algorithm.

---

[7] The training algorithm for the links leading to all competitive layers is implemented in the function *trainCL*.

[8] All the links and also nodes are initialized using function *initialize*.

[9] The training algorithm for the links in SRN is implemented in the function *trainSRN*.

4.6.4 The Auto-encoder Network

The AE that is only used in the enhanced version of the first approach has three layers: the input layer, the hidden layer and the output layer. The design of this network is previously described in section 3.5. The input layer of the AE is the hidden layer of the SRN, so the input layer has 200 nodes. The hidden layer of the AE has 35 nodes. The output layer, which is identical to the input layer, also has 200 nodes. These three layers, with the links between them, comprise the AE.[10]

Initially, the links between the layers of AE take a random value between -0.1 and +0.1.Then, the backpropagation learning algorithm with learning parameter set to 0.1 is employed to train the network. This parameter does not change during the training phase. It is selected experimentally. The normal Sigmoid function is also used as the threshold function.

4.6.5 The Memory Layer

In the very first approach, the memory layer has 200 nodes. The links leading to this layer only copy the content of the hidden layer. Obviously, no training or modification takes place on these links. In the modified version, the links copy the content of the hidden layer of AE. Therefore, in this version the memory layer has 35 nodes.

4.6.6 The Backpropagation Network

The BPN has three layers. The input layer, the hidden layer and the output layer. The input layer of the BPN consists of the concatenation of the memory layer and the hidden layer of the SRN, or its compressed form. These two layers are considered as a single layer. In the initial version, with no AE included, the input layer would have 400 nodes and the hidden layer has 1,000 nodes. These sizes are selected experimentally, and they are the ones

[10]Links in the AE are trained using function *trainAE*.

for which the network apparently performs best. [11]

In the modified version, with AE also included, the input layer's size would be 75 and the hidden layer's size is 60. The output layer, which is the role layer, has 30 nodes in both versions. These three layers, with the links between them, construct the BPN.

Initially, the links between the layers of BPN take a random value between -0.1 and +0.1.The backpropagation learning parameter is initially set to 0.1 and gradually drops to 0.001 after a certain number of epochs. As in the SRN and AE, the normal Sigmoid function is used as the threshold function.

4.6.7 The Situation Layer

During the training phase, the situation layer acts as one of the input layers, and during the testing phase, it acts as the output layer of the network. This layer has 40 nodes, which is the size of a situation vector. There are links going from this layer to the role layer, and also links from the role layer to this layer[12]. The training and modification of links leading to the role layer has been previously discussed.

Intially, the weight of the links leading from the role layer to the situation layer are all set to 0. For training these links, a very simple Hebbian algorithm is used. In this algorithm, for each word, the weights of the links coming from the winner node of the role layer to the active nodes of the situation layer are incremented by a constant size. The constant size is set to 0.025.

[11]The training algorithm for the links in BPN is implemented in the function *trainBL*, where BL stands for back lines.

[12]The function *trainOutputLayer* trains the links, which go from the role layer to the situation layer.

Chapter 5

Conclusion and Future Directions

5.1 Summary and Conclusion

In the previous chapter, connectionist models that can learn, and later recognize the thematic roles of words in sentences were presented. The range of sentence types covered a wide variety of forms, including rather long sentences with active or passive voice, sentences with more than one verb, containing relative clauses, and sentences containing prepositional phrases, among others.

To deal with the task of identifying roles in sentences, two connectionist approaches were taken. In both approaches, the task was performed by assuming that the learner can perceive aspects of the real world situation corresponding to each given sentence. Meaningful feature vectors were employed to present the words of the sentence and also the aspects of the real world situation for each of the words presented to the network. Using distributed representation and meaningful feature vectors was one significant property of the presented connectionist models. This method helps the network to generalize across words, and achieve strong systematicity.

Both approaches employed competitive layers, together with an SRN. The task of one of the competitive layers was to categorize the words according to their semantics. This categorization later leads to generalization among words with close semantics. The role of the SRN was to derive the syntactic-semantic pattern underlying each sentence or a part of the sentence. The two approaches shared another competitive layer as well. The layer

categorized aspects of the world situation into thematic roles.

The difference between the two approaches concerned how each learned the relationship between the thematic role of a given word in a given sentence with the information available in other parts of the network. To learn the foregoing relationship, the first approach employed a backpropagation network, whereas the second one used a lookup table.

The models were trained using a set of meaningful sentences, and then tested using a different set of sentences. The first approach attained an acceptable result in identifying the thematic roles, whereas the second one achieved a perfect result.

The training and testing corpora were designed in a way that the models could be tested for strong systematicity. No exhaustive training was used, because the network was trained using an impoverished training set. During the training, for each thematic role some nouns never appeared in that thematic role. This restriction was later relaxed during the testing phase. The connectionist models could manage to exhibit strong systematicity, defined by Hadley [1994]. The model was able to perform this without any previous syntactic knowledge or syntactic parsing of a sentence. From these results, one can conclude that it is possible to design and train a neural network in a fashion such that it can exhibit strong systematicity in a manner that differs from what was proven by Hadley and Hayward [1997], and Hadley et al. [2001]. This ability of the network arises not only from the design of the network, but also from the data it receives as the input, which encode aspects of the semantics of the words of the sentences and of the real world situation.

As discussed in section 1.3, some proposed connectionist models in the field of language learning need the thematic role of words of a sentence to be known in advance. In particular, one possible model for representing a sentence meaning is to use HRR. The role layer in the presented connectionist model could be presented to these models in order to provide the role vectors.

Finally, it could be argued that humans learn the thematic roles of words in a sentence, as an emergent property of learning the relationship between the words/sentences and the

real world situations. However, please note that, we do not claim that the presented connectionist model is the human learning mechanism for language acquisition. Still, we believe it represents steps towards a cognitively and biologically plausible connectionist model, similar to aspects of human language learning mechanisms.

5.2 Future Directions

The current networks have only been exposed to syntactically and semantically correct sentences. Future experiments can be done to verify if the models can distinguish between correct and incorrect sentences and still identify the roles correctly. One approach would be to create a training and a testing corpus which contain some semantically and syntactically incorrect sentences. Some sentences with semantic anomaly could refer to no acceptable world situation, and other sentences with syntactic anomaly could refer to the world situation which the corrected sentence would have referred to, if possible. Minor modifications in the design of the networks might be needed to deal with these cases.

The training and the testing set could be extended in other fashions as well. The vocabulary could be easily expanded. As long as the SC layer can categorize the words in acceptable classes, the network and the number of training epochs would possibly need no modification to deal with the extended vocabulary. Other experiments could also be designed to verify if the network can deal with more types of sentences. New thematic roles and new sentence structures could be introduced to the training and testing corpora. Including negated sentences could be considered as well.

Another future direction would be to modify the situation layer in a way that renders it closer to the real world situations to which a human learner is exposed. For this purpose, the situation layer could be altered in the following way. Instead of presenting one situation for each word of a sentence, the whole situation corresponding the whole sentence could be presented as the input for all the words in that sentence. Then, the network would also need to learn to distinguish between parts of the situation layer. It would also need to learn the association between the input word and a part of the situation layer and ignore the rest of that layer. The process of learning this association, would require learning the

statistical correlation between the input words and parts of the situation layer. This could be accomplished by adapting the method presented in Hadley and Hayward [1997].

One drawback of the current models concerns the SRN. To learn different sentence structures, the SRN needs a relatively large hidden layer of size 200. On the other hand, due to the sentence structures, the information vectors stored in the SRN hidden layer for different sentences are rather close. These two characteristics make the further learning between this layer and the role layer rather difficult. One future enhancement would be to replace the SRN with another sub-system. The new sub-system must be able to encode the sentence structures in vectors, so that the vectors can be later passed to other parts of the whole network.

In our model, the SRN was used to encode complete and incomplete sentence types. One possible alternative system for the SRN could be a model that employs the idea of holographic reduced representation, Plate [1995]. As mentioned earlier in section 1.4, HRR can be used to store and represent data with complex compositional structures. Due to the type of data that is to be stored, the following approach might be used to substitute for the SRN.

In the current model, when the activation of the nodes in the SC layer are settled, one node is selected as the winner. Therefore, the content of this layer would be a vector, where the activation of all remaining nodes is zero, but the activation of the winner node is +1. The identity of the winner node can be used as the seed to generate another random vector.

Suppose sc_i represents the random vector generated by the winner's identity number within the SC layer for word i in a given sentence. To encode and store the sentence structure up to and including the word i in the given sentence, a neural network must be designed and implemented that computes and stores results using the following recursive equation, Plate [1995]:

$$sc_1 + (sc_1 * sc_2) + \ldots + (sc_1 * sc_2 * \ldots * sc_i), \tag{5.1}$$

where $*$ represents the circular convolution operation. This HRR approach is introduced in Plate [1995], as a representational method that can encode the order of a sequence of

items.

The encoding that results from using this method would need to be stored in a specific layer of the newly designed network. The represented data in the newly designed network would flow to the rest of the network, as did the hidden layer of the SRN. Consequently, the rest of the network would need no modification.

Finally, it must be remembered that the designed and implemented models, as well as the sketch of the suggested models for future work, are not claimed to be the human learning mechanism for language acquisition. They are intended as steps towards cognitively and biologically plausible connectionist models that might be similar to parts of the language learning mechanisms in the human brain.

Bibliography

Gerry Altmann and Yuki Kamide. Incremental interpretation at verbs: Restricting the domain of subsequent reference. *Cognition*, 73(3):247–264, 1999.

C. F. Baker, C.J. Fillmore, and J.B. Lowe. The berkeley framenet project. In *Proceedings of the 17th international conference on Computational linguistics - Volume 1*, COLING '98, pages 86–90. Association for Computational Linguistics, 1998.

Franklin Chang. Symbolically speaking: A connectionist model of sentence production. *Cognitive Science*, 26(5):609–651, 2002.

Franklin Chang, Gary S Dell, and Kathryn Bock. Becoming syntactic. *Psychological review*, 113(2):234, 2006.

Michael Collins. Three generative, lexicalised models for statistical parsing. In *Proceedings of the eighth conference on European chapter of the Association for Computational Linguistics*, pages 16–23. Association for Computational Linguistics, 1997.

Michael Collins. *Head-Driven Statistical Models for Natural Language Parsing*. PhD thesis, University of Pennsylvania, 1999.

Hoa Trang Dang, Karin Kipper, Martha Palmer, and Joseph Rosenzweig. Investigating regular sense extensions based on intersective levin classes. In *Proceedings of the 17th international conference on Computational linguistics-Volume 1*, pages 293–299. Association for Computational Linguistics, 1998.

M. Ellsworth, K. Erk, P. Kingsbury, and S. Padó. Propbank, salsa, and framenet: How design determines product. In *Proceedings of the LREC 2004 Workshop on Building Lexical Resources from Semantically Annotated Corpora, Lisbon*, 2004.

J. L. Elman. Finding structure in time. *Cognitive Science*, 14:179–212, 1990.

J. L. Elman. Distributed representations, simple recurrent networks, and grammatical structure. *Machine learning*, 7(2-3):195–225, 1991.

Kara D Federmeier and Marta Kutas. A rose by any other name: Long-term memory structure and sentence processing. *Journal of memory and Language*, 41(4):469–495, 1999.

C. Fillmore. Frame semantics. *Linguistics in the morning calm*, pages 111–137, 1982.

C. Fillmore. Frames and the semantics of understanding. *Quaderni di Semantica*, 6(2): 222–254, 1985.

C. J. Fillmore, C. R. Johnson, and M.R.L. Petruck. Background to framenet. *International Journal of Lexicography*, 16(3):235–250, 2003.

J.A. Fodor and Z.W. Pylyshyn. Connectionism and cognitive architecture: A critical analysis. *Cognition*, 28(1):3–71, 1988.

D. Gildea and D. Jurafsky. Automatic labeling of semantic roles. *Computational Linguistics*, 28(3):245–288, 2002.

R.F. Hadley. Systematicity in connectionist language learning. *Mind & Language*, 9(3): 247–272, 1994.

R.F. Hadley, A. Rotaru-Varga, D.V. Arnold, and V.C. Cardei. Syntactic systematicity arising from semantic predictions in a hebbian-competitive network. *Connection Science*, pages 73–94, 2001.

Robert F Hadley and Vlad C Cardei. Language acquisition from sparse input without error feedback. *Neural Networks*, 12(2):217–235, 1999.

Robert F Hadley and Michael B Hayward. Strong semantic systematicity from hebbian connectionist learning. *Minds and Machines*, 7(1):1–37, 1997.

D.O. Hebb. *The organization of behavior; a neuropsychological theory.* Wiley, 1949.

C.R. Johnson, C.J. Fillmore, E.J. Wood, J. Ruppenhofer, M. Urban, M.R.L. Petruck, and C.F. Baker. The framenet project: Tools for lexicon building. 2001.

Daniel Jurafsky and James H. Martin. *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition.* Prentice Hall PTR, 1st edition, 2000. ISBN 0130950696.

Yuki Kamide, Gerry Altmann, and Sarah L Haywood. The time-course of prediction in incremental sentence processing: Evidence from anticipatory eye movements. *Journal of Memory and Language*, 49(1):133–156, 2003.

Karin Kipper, Hoa Trang Dang, and Martha Palmer. Class-based construction of a verb lexicon. In *Proceedings of the National Conference on Artificial Intelligence*, pages 691–696. Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press; 1999, 2000.

Beth Levin. *English verb classes and alternations : a preliminary investigation.* 1993.

Martha Palmer, Daniel Gildea, and Paul Kingsbury. The proposition bank: An annotated corpus of semantic roles. *Computational Linguistics*, 31(1):71–105, 2005.

Martha Palmer, Daniel Gildea, and Nianwen Xue. *Semantic Role Labeling*. Morgan & Claypool Publishers, 2010.

Tony A Plate. Holographic reduced representations. *Neural networks, IEEE transactions on*, 6(3):623–641, 1995.

Joao Luis Garcia Rosa and Edson Françozo. Hybrid thematic role processor: Symbolic linguistic relations revised by connectionist learning. In *Proceedings of IJCAI'99–Sixteenth International Joint Conference on Artificial Intelligence*, volume 2, pages 852–857, 1999.

D.E. Rumelhart and J.L. McClelland. *Learning internal representations by error propagation*, chapter 8. MIT Press, 1986a.

D.E. Rumelhart and J.L. McClelland. *Parallel distributed processing: explorations in the microstructure of cognition. Volume 1. Foundations*. MIT Press, 1986b.

D.E. Rumelhart and D. Zipser. Feature discovery by competitive learning. *Cognitive science*, 9(1):75–112, 1985.

D.E. Rumelhart, G.E. Hinton, and R.J. Williams. Learning representations by back-propagating errors. *Nature*, 323(6088):533–536, 1986.

J. Ruppenhofer, M. Ellsworth, M.R.L. Petruck, C.R. Johnson, and J. Scheffczyk. Framenet ii: Extended theory and practice. *framenet.icsi.berkeley.edu*, 2010. URL `https://framenet2.icsi.berkeley.edu/docs/r1.5/book.pdf`.

Karin Kipper Schuler. Verbnet: A broad-coverage, comprehensive verb lexicon. 2005.

Robert Swier and Suzanne Stevenson. Unsupervised semantic role labelling. In *Proceedings of EMNLP*, volume 95, page 102, 2004.

C. Von der Malsburg. Self-organization of orientation sensitive cells in the striate cortex. *Kybernetik*, 14(2):85–100, 1973.

Nicole YY Wicha, Eva M Moreno, and Marta Kutas. Expecting gender: An event related brain potential study on the role of grammatical gender in comprehending a line drawing within a written sentence in spanish. *Cortex*, 39(3):483–508, 2003.

Appendix A

Assignment of features to words

number of words: 33;

number of features: 69;

	visible-object	tangible-object	occupies-space	has-shape	can-eat	can-breath	inanimate-object	human	animal	child	female-human	news	can-fly	location	outdoor	can-play-in	part-of-body	kitchen-utensil	can-eat-with	edible	metal	form-of-money	action	emotive	feeling-nice	involves-light	involves-receiving	involves-delivering	involves-money	has-rules	action	action	action
women	1	1	1	1	1	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
boys	1	1	1	1	1	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
girls	1	1	1	1	1	0	1	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
men	1	1	1	1	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
cats	1	1	1	1	1	0	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
mice	1	1	1	1	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
birds	1	1	1	1	1	0	0	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
houses	1	1	1	1	0	0	1	0	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
parks	1	1	1	1	0	0	1	0	0	0	0	0	0	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
streets	1	1	1	1	0	0	1	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
spoons	1	1	1	1	0	0	1	0	0	0	0	0	0	0	1	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
coins	1	1	1	1	0	0	1	0	0	0	0	0	0	0	0	0	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
cookies	1	1	1	1	0	0	1	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
hands	1	1	1	1	0	0	1	0	0	0	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
love	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	0	0	0	0	0	1	1	1
hate	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	1	1	1	1
eat	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	0	0	0	1	1	1	1
shine	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1	0	0	0	0	1	1	1	1
buy	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1	0	1	1	1	1	1	1	1
borrow	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1	0	0	1	1	1	1	1	1
get	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1	0	0	1	1	1	1	1	1
give	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1	0	0	1	1	1	1	1	1
sell	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	1	1	1	1	1	1	1	1
lend	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	1	0	1	1	1	1	1	1
with	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1
in	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1
to	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1
from	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1
for	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1
that	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
are	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
by	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
.	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Appendix B

Thematic Role Vectors

agent	1111000000000000000000000000000000000000
patient	0000111100000000000000000000000000000000
action	0000000011110000000000000000000000000000
nothing	0000000000001111000000000000000000000000
location	0000000000000000111100000000000000000000
instrument	0000000000000000000011110000000000000000
coagent	0000000000000000000000001111000000000000
beneficiary	0000000000000000000000000000111100000000
source	0000000000000000000000000000000011110000
recipient	0000000000000000000000000000000000001111

Appendix C

Grammar Rules for Training/Testing Set

number of training sentences: 3000; number of testing sentences: 3000;

Rule	Thematic Role Association	Example	Percent
N V.	[agent action].	Coins shine.	5
N V in N.	[agent action nothing location]	Women eat in streets.	5
N V N.	[agent action patient.]	Girls love women.	3
N are V by N.	[patient nothing action nothing agent]	Girls are loved by women.	2
N V N in N.	[agent action patient nothing location.]	Girls love boys in parks.	3
N are V by N in N.	[patient nothing action nothing agent nothing location.]	Girls are loved by boys in parks.	2
N V N with N(inanimate).	[agent action patient nothing instrument.]	Men eat cookies with spoons.	6
N are V by N with N(inanimate).	[patient nothing action nothing agent nothing instrument.]	Cookies are eaten by men with spoons.	4
N V N with N(animate).	[agent action patient nothing coagent.]	Girls eat cookies with boys.	6
N are V by N with N(animate).	[patient nothing action nothing agent nothing coagent.]	Birds are bought by men with women.	4
N V N from N.	[agent action patient nothing source.]	Men get birds from women.	6
N are V by N from N.	[patient nothing action nothing agent nothing source.]	Birds are bought by women from men.	4
N V N for N.	[agent action patient nothing beneficiary.]	Men borrow spoons for women.	6
N are V by N for N.	[patient nothing action nothing agent nothing beneficiary.]	Spoons are borrowed by men for women.	4
N V N to N.	[agent action patient nothing recipient.]	Men sell tables to boys.	6
N are V by N to N.	[patient nothing action nothing agent nothing recipient.]	Tables are sold by men to boys.	4
N that N V V N.	[agent patient agent action action patient.]	Girls that women love eat cookies.	7.5
N that V N V N.	[agent agent action patient action patient.]	Girls that love women eat cookies.	7.5
N V N that N V.	[agent action patient patient agent action.]	Girls love women that eat cookies.	7.5
N V N that V N.	[agent action patient agent action patient.]	Girls eat cookies that women love.	7.5

Appendix D

Words and Roles

Role	Excluded Words during Training	Included Words during Training
Agent/Co-agent	streets, boys, cats	men , women , girls , birds , mice , parks , houses , spoons , coins
Patient	women, birds, houses, spoons	men , girls , boys , mice , cats , parks , streets , spoons , cookies , hands , coins
Location	parks	houses , streets
Instrument	coins	spoons , hands
Source	houses, mice	men , boys , girls , cats , birds , parks , streets , women
Beneficiary	girls, birds	men , boys , mice , cats , women
Recipient	women, cats	men , boys , girls , birds , mice

Appendix E

The Programming Code for the First Approach

Matlab Code

```
function [net] = main()
%this is the main function of the program

clear all

%tStartMain keeps the start time of the running program
% for log
tStartMain = tic;

net.numInputs = 2; %num of inputs
net.numLayers = 9; %num of non-i/o layers
net.numOfEpochs = 500;%total number of epochs
net.numOfInitEpochs = 25;%num of epochs for training SRN
net.numOfMidEpochs = 200;%num of epochs for training SRN + AE (not BPN)

% Semantic Classifier Layer
net.layer{1}.c = 0.5;
net.layer{1}.learningRate = 0.5;
net.layer{1}.size = 30;
net.layer{1}.firstTime = 1;%is initially set to 1, and then to 0

%SRN
% Hidden Layer of SRN
net.layer{2}.size = 200;
net.layer{2}.alpha = 0.1;%alpha = learning rate
```

```
26
27 % Context Layer of SRN
28 net.layer{3}.size = net.layer{2}.size;
29 net.layer{3}.alpha = 0.1;
30
31 %SRN Output
32 net.layer{4}.size = net.layer{1}.size;
33 net.layer{4}.alpha = 0.1;
34
35 %BPN
36 %Hidden Left (of BPN)
37 net.layer{6}.size = 60;
38 net.layer{6}.alpha = net.layer{2}.alpha;
39
40 % Role Layer - output of BPN
41 net.layer{7}.size = 30;
42 net.layer{7}.c = 0.6;
43 net.layer{7}.learningRate = 0.1;
44 net.layer{7}.firstTime = 1;
45
46 %AE
47 % AE hidden Layer
48 net.layer{8}.size = 35;
49 net.layer{8}.alpha = 0.01;
50
51 %AE output layer
52 net.layer{9}.size = net.layer{2}.size;
53 net.layer{9}.alpha = net.layer{8}.alpha;
54
55 %Memory Layer
56 net.layer{5}.size = net.layer{8}.size;
57 net.layer{5}.alpha = 0.1;
58
59
60 %prepare the training/testing corpora (vectors)
61 [ trainingVector, roleVectors, testVector, roleTestVector ] = prepareInputOutput();
62 display('Input Output Ready.');
63
64 % set the size of the input layer of  the network according to the input
65 % files
66 net.input{1}.size = size(trainingVector, 3);
67 net.input{2}.size = size(roleVectors, 3);
68 %set the hebbian learning rate
69 net.input{2}.increment = 0.025;
70
71 % initialize the network
72 net = initialize(net);
```

```
73  display('Network Initialized');
74  %train the network
75  net = train(net, trainingVector, roleVectors);
76  display('Network Trained');
77
78
79  %%Clustering the contents of layers for debugging
80
81  %clustering according to the sentence type in memory layer
82  groups = linkage(net.contextLayerSentence);
83  net.clustersSent  = cluster(groups, 50);
84
85  %clustering according to words of each sentence (context layer of SRN)
86  groups = linkage(net.contextLayerWords);
87  clustersWord  = cluster(groups, 100);
88  net.clustersWord = makeItReadable(clustersWord, net.aTrain);
89
90  %clustering the hidden layer of BPN (corresponds to roles)
91  groups = linkage(net.trainSix);
92  net.clusterRole = cluster(groups, 10);
93
94
95  %test the network
96  net = test(net, testVector);
97  display('Network Tested');
98
99  %Verify  if the outputs of the testing phase is correct
100 net = justify(net, roleTestVector, testVector);
101 display('Results Verified.');
102
103 %to keep the running time of the program
104 %for log
105 tElapsed = toc(tStartMain);
106 minutes = tElapsed / 60
107
108 beep
109
110 end
111 %%%
112 function [ trainingVector, roleVectors, testVector, roleTestVector ] =
        prepareInputOutput()
113 %PREPAREINPUTOUTPUT prepares the vector input of the network
114 %This function first reads the word/role vectors and then reads the
115 %train/test corpora and assigns a vector to each word/role in the corpora.
116
117 % extract the role/word vectors
118 % word.txt contains the vocabulary list and vectors
```

```
119 wordList = readTrainingWords('words.txt');
120 % role.txt contains roles list and vector
121 roleList = readTrainingWords('roles.txt');
122
123 % prepare training sentence vectors
124 % trainSet.txt contains training sentences
125 trainSet = readSentences('trainSent.txt');
126 % parse sentences into vectors
127 trainingVector = parseSentences(trainSet, wordList);
128
129 % prepare training role vectors
130 % trainRole.txt contains roles corresponding to sentences in the training corpus
131 roleSet = readSentences('trainRole.txt');
132 % parse sentences into vectors
133 roleVectors = parseSentences(roleSet, roleList);
134
135 % prepare testing sentence vectors
136 % testSent.txt contains testing sentences
137 testSet = readSentences('testSent.txt');
138 % parse sentences into vectors
139 testVector = parseSentences(testSet, wordList);
140
141 % prepare testing role vectors
142 % testRole.txt contains roles corresponding to sentences in the testing corpus
143 roleTest = readSentences('testRole.txt');
144 % parse sentences into vectors
145 roleTestVector = parseSentences(roleTest, roleList);
146
147 end
148 %%%
149 function words = readTrainingWords( fileName )
150 %READTRAINING reads the word/role list and their vectors
151 %   outputs a struct containing word/role and type and vector
152 % Note: type is not used in this program
153
154
155 fid = fopen(fileName);
156
157 %read first line
158 line = fgets(fid);
159 numOfLines = 1;
160 while ischar(line)% while the input file is not yet finished
161         % extract word, then type, then vector from the line
162     [word, line] = strtok(line);
163     [type, line] = strtok(line);
164         % cast string-vector to numerical vector
165     vector = str2num(line);
```

```
166         % construct the output struct
167     words{numOfLines} = struct('word', word, 'type', type, 'vector', vector);
168 
169         %read next line
170     numOfLines = numOfLines + 1;
171     line = fgets(fid);
172 end
173 
174 
175 fclose(fid); % close the file
176 
177 end
178 
179 end
180 %%%
181 function sentences = readSentences( fileName )
182 %READSENTENCES reads and ouputs sentences from a file (fileName)
183 %   each line is a sentence
184 
185 fid = fopen(fileName);%open the file
186 
187 % read first line
188 numOfLines = 1;
189 line = fgetl(fid);
190 while ischar(line)% while the input file is not yet finished
191         % this line is the current sentence
192     sentences{numOfLines} = line;
193         %read next line
194     numOfLines = numOfLines + 1;
195     line = fgetl(fid);
196 end
197 
198 fclose(fid); % close the file
199 
200 end
201 %%%
202 function [ words ] = parseSentences( sentences, wordList)
203 %PARSESENTENCES reads *sentences* and by the use of *wordList*
204 % parses them into *words*.
205 
206 %sizeS is the number of sentences
207 [dummy sizeS] = size(sentences);
208 
209 for sentenceIndex= 1:sizeS %for all sentences
210         % rest is first set to the whole sentence (sentence(sentenceIndex))
211         % and then the rest of the sentence
212     rest = sentences(sentenceIndex);
```

```
213     thisWord = rest;
214         %index is the word index in current sentence
215     index = 1;
216     while ~strcmp('.', thisWord)% parse each sentence, until period is encountered
217                 % thisWord is the current word of the sentence
218         [thisWord rest] = strtok(rest);
219                 % find and set the corresponding vector (to thisWord in sentence(i))
220         words(sentenceIndex, index, :) = findWordsVector(thisWord, wordList);
221         index = index + 1;
222     end
223         %next sentence index
224 end
225
226 end
227
228 function vector = findWordsVector(thisWord, wordList)
229 % findWordsVector looks for thisWord in wordList
230 % and output its corresponding vector.
231
232 %sizeW is the number of words in wordList
233 [dummy sizeW] = size(wordList);
234 for i = 1:sizeW %for all the words in the wordList
235         % if thisWord is the current word of the list
236         % vector (output) will be the corresponding vector of thisWord
237         %else continue the search through the list
238     if (strcmp(thisWord, wordList{i}.word))
239         vector = wordList{i}.vector;
240         return
241     end
242 end
243 %output thisWord if it is not in the list (ERROR)
244 thisWord
245 display('parseSentence: word not found.');
246 end
247 %%%
248 function net = initialize( net )
249 %INITIALIZE initializes the weights of the links
250 % and the initialization values of activation of nodes in layers.
251
252 % initialize weights from input-word to Semantic Classifier layer.
253 net.layer{1}.IW = initializeIWofCompetitiveLayers(net.input{1}.size, net.layer{1}.
        size);
254
255 % initialize weights of SRN
256 net.layer{2}.IW = initializeIWofBackPropLinks(net.layer{1}.size, net.layer{2}.size);
257 net.layer{3}.IW = initializeIWofBackPropLinks(net.layer{2}.size, net.layer{3}.size);
258 net.layer{4}.IW = initializeIWofBackPropLinks(net.layer{3}.size, net.layer{4}.size);
```

```
259
260 % initialize weights of BPN (The same values as in SRN)
261 net.layer{5}.IW = initializeIWofBackPropLinks(net.layer{8}.size + net.layer{5}.size,
        net.layer{6}.size);
262 net.layer{6}.IW = initializeIWofBackPropLinks(net.layer{6}.size, net.layer{7}.size);
263
264 %initialize weights from input-role to Role Layer
265 net.layer{7}.IW = initializeIWofCompetitiveLayers(net.input{2}.size, net.layer{7}.
        size);
266 %initialize weights from ROLE layer to input-role (Hebbian Links thus 0)
267 net.input{2}.IW = zeros(net.layer{7}.size, net.input{2}.size);
268
269 % initialize weights of AE
270 net.layer{8}.IW = initializeIWofBackPropLinks(net.layer{2}.size, net.layer{8}.size);
271 net.layer{9}.IW = initializeIWofBackPropLinks(net.layer{8}.size, net.layer{9}.size);
272
273
274 %initialize Bias Value
275 net.layer{2}.bias = initializeIWofBackPropLinks(1, net.layer{2}.size);
276 net.layer{4}.bias = initializeIWofBackPropLinks(1, net.layer{4}.size);
277 net.layer{6}.bias = initializeIWofBackPropLinks(1, net.layer{6}.size);
278 net.layer{7}.bias = initializeIWofBackPropLinks(1, net.layer{7}.size);
279 net.layer{8}.bias = initializeIWofBackPropLinks(1, net.layer{8}.size);
280 net.layer{9}.bias = initializeIWofBackPropLinks(1, net.layer{9}.size);
281
282
283 %Initialize all the nodes/max-up-to-now-node-values to zero.
284 net.layer{1}.next = zeros(1, net.layer{1}.size);
285 for i = 1:net.numLayers
286     net.layer{i}.nodes = zeros(1, net.layer{i}.size);
287     net.layer{i}.max = zeros(1, net.layer{i}.size);
288 end
289
290 end
291 %%%
292 function IW = initializeIWofCompetitiveLayers(bottom, up)
293 %initializeIWofCompetitiveLayers is used for initialization of links to competetive
        layers
294 %   The initialization is such that the sum of the weights leading to each
295 %   node in the layer = 1, so that it's fair.
296 IW = unifrnd(0, 1, [bottom, up]);
297
298 for i = 1:up
299    IW(:, i) = IW(:, i) ./ sum(IW(:, i));
300 end
301
302 end
```

```
303 %%%
304 function IW = initializeIWofBackPropLinks(bottom, up)
305 %initializeIWofBackPropLinks initializes the weights that are trained
306 % using backpropagation algorithm.
307 %   each link take a random value between -0.1 and +0.1
308
309 IW = unifrnd(-0.1, 0.1, [bottom, up]);
310
311 end
312 %%%
313
314 function [ net ] = train( net, sentences, roles)
315 %TRAIN takes the network, and the network's inputs and trains the links of the
        network.
316 % For #ofEpochs do
317 % For each sentence
318 % initialize context layer to 0. Nothing in the memory.
319 % For each word in the sentence
320 % Train the SRN
321 % Keep the result in the memory
322 % Again go through the words of the sentence
323 % train the BPN.
324
325 %not for train-for log
326 index = 1; % this is used for keeping the context layer index to cluster if later
327 indexInner = 1;
328
329 numOfSentences = size(sentences, 1);
330 % -1: The period at the end of the sentence is not counted.
331 numOfWords = size(sentences(1, :, :), 2) - 1;
332
333 %not for train-for log
334 net.ifThisWord = zeros(numOfSentences, numOfWords);
335
336 %for num of epochs
337 for i = 1:net.numOfEpochs
338
339     net.i = i;
340     %set the learning rate by the iteration number, alpha:0.1->0.001
341     if (i == net.numOfMidEpochs + 20)
342         net.layer{2}.alpha = 0.05;
343         net.layer{3}.alpha = 0.05;
344         net.layer{4}.alpha = 0.05;
345         net.layer{6}.alpha = 0.05;
346         net.layer{5}.alpha = 0.05;
347     end
348     if (i == net.numOfMidEpochs + 40)
```

```
349         net.layer{2}.alpha = 0.01;
350         net.layer{3}.alpha = 0.01;
351         net.layer{4}.alpha = 0.01;
352         net.layer{6}.alpha = 0.01;
353         net.layer{5}.alpha = 0.01;
354     end
355     if (i == net.numOfMidEpochs + 60)
356         net.layer{2}.alpha = 0.005;
357         net.layer{3}.alpha = 0.005;
358         net.layer{4}.alpha = 0.005;
359         net.layer{6}.alpha = 0.005;
360         net.layer{5}.alpha = 0.005;
361     end
362     if (i == net.numOfMidEpochs + 90)
363         net.layer{2}.alpha = 0.001;
364         net.layer{3}.alpha = 0.001;
365         net.layer{4}.alpha = 0.001;
366         net.layer{6}.alpha = 0.001;
367         net.layer{5}.alpha = 0.001;
368     end
369     %for time log
370     tStartTrain = tic;
371     for sentenceIndex = 1:numOfSentences
372         %context/hidden layer - clean memory
373         net.layer{3}.nodes = zeros(1, net.layer{3}.size);
374         net.layer{2}.nodes = zeros(1, net.layer{2}.size);
375
376                 %no of words in each sentence is one less than its size (no period)
377         numOfWords = size(sentences(sentenceIndex, :, :), 2) - 1;
378
379         %FIRST PASS: SRN training
380         for wordIndex = 1:numOfWords
381             %tempNext keeps the next word'
382             tempNext(:, :) = sentences(sentenceIndex, wordIndex + 1, :);
383             nextWord = tempNext';
384             %if the next word is not empty = if this word is not .
385             if (~any(nextWord))
386                 break;
387             end
388             % temp keeps current word
389             temp(:, :) = sentences(sentenceIndex, wordIndex, :);
390             word = temp';
391             %tempNodes keeps the contents of layer 4 for the previous word.
392             %For speeding up the computation only. Semantic of current word.
393             tempNodes = net.layer{1}.next;
394             %First compute the Semantic of the next word - layer 4 (saved as
395             %layer{1}.next).
```

```
396             net.activeLayer = 1;
397             net = trainCL(net, nextWord);
398             net.layer{1}.next = net.layer{1}.nodes;
399
400             %If the first word of the sentence then tempNodes is not use.
401             %Find the Semantic of the current word.
402             if (wordIndex == 1)
403                 net = trainCL(net, word);
404             else
405                 net.layer{1}.nodes = tempNodes;
406             end
407
408             %train the SRN
409             net.activeLayer = 2;
410             net = transferSRN(net);
411             net = backpropSRN(net);
412
413
414             if (i > net.numOfInitEpochs)
415                 %train autoencoder
416                 net = transferAE(net);
417                 net = backpropAE(net);
418
419             end
420                         %LOG - error in AE
421             net.errorAE(sentenceIndex, wordIndex) = sqrt(sum((net.layer{9}.nodes -
                    net.layer{2}.nodes) .^ 2));
422
423             %LOG - error in SRN
424             net.errorSRNRight(sentenceIndex, wordIndex) = sqrt(sum((net.layer{4}.
                    nodes - net.layer{1}.next) .^ 2));
425
426                         %LOG - semantic classifier, context layer contents
427             net.ifThisWord(sentenceIndex, wordIndex) = 1;
428             if (i == net.numOfEpochs)
429                 net.aTrain(sentenceIndex, wordIndex) = find(net.layer{1}.nodes);
430                 net.contextLayerWords(index, :) = net.layer{3}.nodes;
431                 index = index + 1;
432             end
433         end
434             %if it's time to start training BPN
435         if (i < net.numOfMidEpochs)
436             continue;
437         end
438
439
440         %FOR LOG: keep the hidden layer
```

```
441         net.contextLayerSentence(sentenceIndex, :) = net.layer{8}.nodes;
442         %Keep the HL of SRN in the MEMORY layer
443         net.layer{5}.nodes = net.layer{8}.nodes;
444
445                 %context/hidden layer - clean memory
446         net.layer{3}.nodes = zeros(1, net.layer{3}.size);
447         net.layer{2}.nodes = zeros(1, net.layer{2}.size);
448
449         %%%%%%%%%%%
450         %SECOND PASS
451
452         %for each word in the sentence
453         for wordIndex = 1:numOfWords
454             %tempNext keeps the next word'
455             tempNext(:, :) = sentences(sentenceIndex, wordIndex + 1, :);
456             nextWord = tempNext';
457             %if the next word is not empty = if this word is not .
458             if (~any(nextWord))
459                 break;
460             end
461             % temp keeps current word
462             temp(:, :) = sentences(sentenceIndex, wordIndex, :);
463             word = temp';
464
465             %tempNodes keeps the contents of layer 4 for the previous word.
466             %For speeding up the computation only. Semantic of current word.
467             tempNodes = net.layer{1}.next;
468             %First compute the Semantic of the next word - layer 4 (saved as
469             %layer{1}.next).
470             net.activeLayer = 1;
471             net = trainCL(net, nextWord);
472             net.layer{1}.next = net.layer{1}.nodes;
473             %If the first word of the sentence then tempNodes is not use.
474             %Find the Semantic of the current word.
475             if (wordIndex == 1)
476                 net = trainCL(net, word);
477             else
478                 net.layer{1}.nodes = tempNodes;
479             end
480             %activation propagates through SRN and AE (no training)
481             net.activeLayer = 2;
482             net = transferSRN(net);
483             net = transferAE(net);
484             %role is input{2}, the situation vector from file
485             tempR(:, :) = roles(sentenceIndex, wordIndex, :);
486             role = tempR';
487
```

```
488             %activation propagetes to output of BPN, and a single node is
489             %selected in the WTA competition in Layer 7 (role layer)
490             net.activeLayer = 7;
491             net = trainCL(net, [role]);
492
493             %BPN trained
494             net = transferBL(net);%
495             net = backpropBL(net);%
496
497             %The links from layer{7}(role layer) to input/output layer
498             %trained-Simple Hebbian
499             net.activeLayer = 7;
500             net = trainOutputLayer(net, [role]);
501             %LOG
502             net.errorSRNLeft(sentenceIndex, wordIndex) = sqrt(sum((net.layer{7}.nodes
                    - net.layer{7}.fake) .^ 2));
503             % LOG
504             if (i == net.numOfEpochs)
505                 net.trainSeven(sentenceIndex, wordIndex) = find(net.layer{7}.nodes);
506                 net.trainSix(indexInner, :) = net.layer{6}.nodes;
507                 indexInner = indexInner + 1;
508             end
509         end
510     end
511     %LOG the accuracy
512     x = sum(net.errorSRNRight);
513     s = sum(net.ifThisWord);
514     y = x ./ s;
515     net.averageErrorRight(i, :) = y;
516
517     if (i > net.numOfInitEpochs)
518
519         xx = sum(net.errorAE);
520         %     size(x)
521         yy = xx ./ s
522     end
523     %LOG the accuracy
524
525     net.averageErrorRight(i, :) = y;
526         %LOG the accuracy
527
528     if (i > net.numOfMidEpochs)
529         z = sum(net.errorSRNLeft);
530         t = z ./ s
531         net.averageErrorLeft(i, :) = t;
532
533         %   max(net.errorSRNRight)
```

```
534             max(net.errorSRNLeft)
535         end
536             % LOG the time
537         tElapsed = toc(tStartTrain)
538         min = tElapsed / 60
539
540     end
541
542     end
543     %%%
544
545     function [ net ] = trainCL( net, input)
546     %trainCL trains the competetive layer
547     %   input is the input layer to the network
548
549     layerNo = net.activeLayer;
550     net = transferCL(net, input); %calculate the active layers' activation value
551     winner = find(net.layer{layerNo}.nodes, 1); % winner is the winner node in the active
            layer (for speeding up the learning phase)
552     % train the links leading to the winner node.
553     net.layer{layerNo}.IW(:, winner) = vonDerMalsburg(net, input) + net.layer{layerNo}.IW
            (:, winner);
554
555     end
556
557
558     function net = transferCL(net, input)
559     %transferCL transfer function for competetive layers
560     % input is the input layer to the active layer, the activation flows to
561     % the active layer through this function.
562
563     layerNumber = net.activeLayer;
564     %recieved is the activation received in the layer.
565     received = input * net.layer{layerNumber}.IW;
566     % if the first time learning (no max still available.)
567     if ( ~ net.layer{layerNumber}.firstTime)
568         %transfer function
569         activation = (net.layer{layerNumber}.c * received) + ...
570             ((1 - net.layer{layerNumber}.c) * (received./(net.layer{layerNumber}.max)));
571         temp = compet(activation'); %compet : 0 if not the winner. ow 1.
572     else
573         net.layer{layerNumber}.firstTime = 0;
574         temp = compet(received');
575     end
576     % nodes <- temp
577     net.layer{layerNumber}.nodes = zeros(1, net.layer{layerNumber}.size);
578     net.layer{layerNumber}.nodes(1, find(temp, 1)) = 1;
```

```
579 % update the max
580 net.layer{layerNumber}.max = max([received;net.layer{layerNumber}.max]);
581
582 end
583 %%%
584 function dw = vonDerMalsburg(net, input)
585 %VONDERMALSBURG is used for training the links.
586 %   input is the layer below the active layer. dw is the change in the
587 %   links between the input layer and the active layer.
588
589 layerNo = net.activeLayer;
590 % winner is found to speed up the learning.
591 winner = find(net.layer{layerNo}.nodes, 1);
592 % the change is only made in the links leading to the winner node.
593 dw =  net.layer{layerNo}.learningRate .* ...
594     ( (input ./ sum(input))' - net.layer{layerNo}.IW(:, winner));
595
596 end
597 %%%
598 function  net  = transferSRN( net )
599 %TRANSFERSRN is the feed forward phase of the SRN
600
601 % hidden layer <- logsig( context layer & input(POS)layer * their weights & bias )
602 % to speed up the calculation of layer 2
603 winner = find(net.layer{1}.nodes, 1);
604 net.layer{2}.nodes = logsig(net.layer{3}.nodes * net.layer{3}.IW + net.layer{2}.IW(
        winner, :) + net.layer{2}.bias);
605
606 %copy hidden layer (2) into context layer (3)
607 net.layer{3}.nodes = net.layer{2}.nodes;
608
609 %output layer (4) <- logsig( hidden layer * their weights & bias )
610 net.layer{4}.nodes = logsig(net.layer{2}.nodes * net.layer{4}.IW + net.layer{4}.bias)
        ; % output layer <- hidden layer & bias
611 end
612 %%%
613 function  net  = backpropSRN( net )
614 %BACKPROPSRN performs the backpropagation algorithm on the SRN
615
616 % delta{out} = teacher - output
617 % other delta's = delta_upper * W in the middle
618 % W += alpha * input * [output * (1 - output) * delta_output]
619
620 % to speed up the computation
621 % delta <- output layer .* (1 - output layer) .* (teacher - output layer)
622 delta = (net.layer{4}.nodes .* (1 - net.layer{4}.nodes)) .* (net.layer{1}.next - net.
        layer{4}.nodes);
```

```
623 % delta hidden layer <- (delta * links to outputlayer) .* hidden layer .* (1 - hidden
        layer)
624 deltaHidden = ((net.layer{4}.IW) * (delta)')' .*  net.layer{2}.nodes .* (1 - net.
        layer{2}.nodes);
625
626 %modify the links/biases according to delta of each layer
627 net.layer{2}.IW = net.layer{2}.IW + net.layer{2}.alpha *  net.layer{1}.nodes' * (
        deltaHidden );
628 net.layer{3}.IW = net.layer{3}.IW + net.layer{3}.alpha *  net.layer{3}.nodes' * (
        deltaHidden );
629 net.layer{4}.IW = net.layer{4}.IW + net.layer{4}.alpha *  net.layer{2}.nodes' * (
        delta );
630
631 net.layer{2}.bias = net.layer{2}.bias + net.layer{2}.alpha * ( deltaHidden );
632 net.layer{4}.bias = net.layer{4}.bias + net.layer{4}.alpha * ( delta );
633
634 end
635 %%%
636 function  net  = transferAE( net )
637 %transferAE transfers activation through the AE
638 %input:layer{2}, hidden:layer{8}, output:layer{9}
639 % each layer <- lower layer * weights + bias
640 net.layer{8}.nodes = logsig(net.layer{2}.nodes * net.layer{8}.IW + net.layer{8}.bias)
        ;
641 net.layer{9}.nodes = logsig(net.layer{8}.nodes * net.layer{9}.IW + net.layer{9}.bias)
        ;
642
643 end
644 %%%
645 function [ net ] = backpropAE( net )
646 %backpropAE does the backpropagation learning process though AE
647 %input:layer{2}, hidden:layer{8}, output:layer{9}
648
649 % delta{out} = teacher - output
650 % other delta's = delta_upper * W in the middle
651 % W += alpha * input * [output * (1 - output) * delta_output]
652
653 % to speed up the computation
654 % delta <- output layer .* (1 - output layer) .* (teacher - output layer)
655 delta = (net.layer{9}.nodes .* (1 - net.layer{9}.nodes)) .* (net.layer{2}.nodes - net
        .layer{9}.nodes);
656 % delta hidden layer <- (delta * links to outputlayer) .* hidden layer .* (1 - hidden
        layer)
657 deltaHidden = ((net.layer{9}.IW) * (delta)')' .*  net.layer{8}.nodes .* (1 - net.
        layer{8}.nodes);
658
659 %modify the links/biases according to delta of each layer
```

```
660 net.layer{8}.IW = net.layer{8}.IW + net.layer{8}.alpha *  net.layer{2}.nodes' * (
        deltaHidden);
661 net.layer{9}.IW = net.layer{9}.IW + net.layer{9}.alpha *  net.layer{8}.nodes' * (
        delta);
662
663 net.layer{8}.bias = net.layer{8}.bias + net.layer{8}.alpha * (deltaHidden);
664 net.layer{9}.bias = net.layer{9}.bias + net.layer{9}.alpha * (delta);
665
666 end
667 %%%
668
669 function [ net ] = transferBL( net )
670 %transferBL transfers the activation through the BPN (aka BL:backprp lines)
671
672 %For readability - these two make the input layer:[contextLayerNodes,memoryLayerNodes
        ]
673 contextLayerNodes = net.layer{8}.nodes;
674 memoryLayerNodes = net.layer{5}.nodes;
675 %hidden layer: layer{6}
676 %output layer : role layer : layer{7}
677
678 net.layer{6}.nodes = logsig([contextLayerNodes,memoryLayerNodes] *  net.layer{5}.IW +
         net.layer{6}.bias);
679 net.layer{7}.fake = logsig(net.layer{6}.nodes * net.layer{6}.IW + net.layer{7}.bias);
680
681 end
682 %%%
683 function [ net ] = backpropBL( net )
684 %does the backpropagation learning in the BL(aka BPN)
685 %For readability - these two make the input layer:[contextLayerNodes,memoryLayerNodes
        ]
686 % delta{out} = teacher - output
687 % other delta's = delta_upper * W in the middle
688 % W += alpha * input * [output * (1 - output) * delta_output]
689
690 contextLayerNodes = net.layer{8}.nodes;
691 memoryLayerNodes = net.layer{5}.nodes;
692 %hidden layer: layer{6}
693 %output layer : role layer : layer{7}
694
695 % to speed up the computation
696 % delta <- output layer .* (1 - output layer) .* (teacher - output layer)
697 delta = (net.layer{7}.fake .* (1 - net.layer{7}.fake)) .* (net.layer{7}.nodes - net.
        layer{7}.fake);
698 % delta hidden layer <- (delta * links to outputlayer) .* hidden layer .* (1 - hidden
         layer)
```

```
699 deltaHidden = ((net.layer{6}.IW) * (delta)')' .*  net.layer{6}.nodes .* (1 - net.
        layer{6}.nodes);
700
701 %modify the links/biases according to delta of each layer
702 net.layer{5}.IW = net.layer{5}.IW + net.layer{5}.alpha *  [contextLayerNodes,
        memoryLayerNodes]' * (deltaHidden);
703 net.layer{6}.IW = net.layer{6}.IW + net.layer{6}.alpha *  net.layer{6}.nodes' * (
        delta);
704
705 net.layer{6}.bias = net.layer{6}.bias + net.layer{6}.alpha * (deltaHidden);
706 net.layer{7}.bias = net.layer{7}.bias + net.layer{6}.alpha * (delta);
707
708 end
709
710 %%%%
711
712 function [ net ] = trainOutputLayer( net, input )
713 %trainOutputLayer trains the links between the situation layer (input{2})
714 % and the role layer (layerNo)
715 % The links are from role layer to the situation layer
716 % The training algorithm = simple Hebbian
717
718
719 layerNo = net.activeLayer;
720 %to speed up computation
721 winner = find(net.layer{layerNo}.nodes);
722
723 %increase the weights from the winner node to the active nodes in situation
724 % layer by the constant "increament"
725 net.input{2}.IW(winner, :) = net.input{2}.IW(winner, :) ...
726      + input .* (net.input{2}.increment);
727 end
728
729 %%%
730 function cosine = normalizedDotProduct(x, y)
731 % NormalizedDotProduct computes the value of the
732 % cosine between two  vectors. x.y = |x|*|y|*cos(xoy); where x.y means the
733 % dot product of the two vectors x and t, |x| and |y| mean the
734 % norm/magnitude of vectors x and y respectively, and xoy is the cosine
735 % between vectors x and y.
736 % According to this, cos(xoy) = x.y/(|x|*|y|).
737 % NDP ~ Correlation
738
739 normx = norm(x);%the function norm computes the norm/magnitude of a vector
740 normy = norm(y);
741
742 if (size(x, 2) ~= size(y, 2))
```

```
743     %if the two vectors are not of the same size/dimension;no dot product
744     %defined.
745     display('normalizedDotProduct: vector size dismatch');
746     cosine = 0;
747 elseif( normx == 0 || normy == 0 )
748     %if one of the vectors is origin, they are perpendicular
749     cosine = -100;
750     display('normalizedDotProduct: One of the vectors was O.');
751 else
752     cosine = (x*y') / (normx * normy);
753 end
754
755 end
756 %%%
757 function net = test(net, sentences)
758 % TEST tests the trainednetwork
759 % inputs are net and the test sentences
760 % All the outputs (situation layer for every word) is saved
761 % to be later verified
762
763
764 numOfSentences = size(sentences, 1);
765 %LOG index
766 index = 1;
767
768 %For every sentence in the test corpus
769 for sentenceIndex = 1:numOfSentences
770     %context/hidden layer - clean memory
771     net.layer{3}.nodes = zeros(1, net.layer{3}.size);
772     net.layer{2}.nodes = zeros(1, net.layer{2}.size);
773
774         %num of words is less than the size of each sentence (no period)
775     numOfWords = size(sentences(sentenceIndex, :, :), 2) - 1;
776         %%%FIRST PASS
777         %for each word in the sentence
778     for wordIndex = 1:numOfWords
779         temp(:, :) = sentences(sentenceIndex, wordIndex, :);
780                 %word is the current word of the sentence
781         word = temp';
782                 %tempNext keeps the next word'
783         tempNext(:, :) = sentences(sentenceIndex, wordIndex + 1, :);
784         nextWord = tempNext';
785         if (~any(nextWord))% if the sentence is processed
786             break;
787         end
788
789         net.activeLayer = 1;
```

```
790                 %find the semantic layer activation
791             net = transferCL(net, word);
792             %then the info flow and find the activation in layers of SRN
793             net.activeLayer = 2;
794             net = transferSRN(net);
795                     % and then AE (to compress)
796             net = transferAE(net);
797         end
798         %copy the hidden layer of AE (compressed form of hid of SRN)into layer M
799         net.layer{5}.nodes = net.layer{8}.nodes;
800             %LOG
801         net.TESTfive(sentenceIndex, :) = net.layer{5}.nodes;
802
803         %context/hidden layer - clean memory
804         net.layer{3}.nodes = zeros(1, net.layer{3}.size);
805         net.layer{2}.nodes = zeros(1, net.layer{2}.size);
806         %SECOND PASS
807         for wordIndex = 1:numOfWords
808                     temp(:, :) = sentences(sentenceIndex, wordIndex, :);
809                     %word is the current word of the sentence
810             word = temp';
811                     %tempNext keeps the next word'
812             tempNext(:, :) = sentences(sentenceIndex, wordIndex + 1, :);
813             nextWord = tempNext';
814             if (~any(nextWord))% if the sentence is processed
815                 break;
816             end
817
818             %find the semantic layer activation
819             net.activeLayer = 1;
820             net = transferCL(net, word);
821             %then the info flow and find the activation in layers of SRN
822             net.activeLayer = 2;
823             net = transferSRN(net);
824             %then the info flow and find the activation in layers of AE
825             net = transferAE(net);
826                     % propagate the info through BPN
827             net = transferBL(net);
828             %exactOut the exact values that flow into output - situation layer
829             net.exactOut(index, :) = net.layer{7}.fake * net.input{2}.IW;
830             index = index + 1;
831             %LOG
832             net.TESTa(sentenceIndex, wordIndex) = find(net.layer{1}.nodes);
833                     %this is when the activation is flowed to output after a WTA in R
                            layer.
834             net.TESToutput(sentenceIndex, wordIndex) =  find(net.layer{7}.fake == max(net
                    .layer{7}.fake));
```

```
835
836     end
837 end
838
839 end
840 %%%
841 function net = justify( net, roles, sentences )
842 %JUSTIFY justifies/verifies the results
843 %goes through the saved outputs for each word
844 %checks it with the roles in the testing corpus
845
846 numOfSentences = size(roles, 1);
847 index = 1;
848 %For every sentence
849 for sentenceIndex = 1:numOfSentences
850     numOfWords = size(roles(sentenceIndex, :, :), 2);
851         %For every word
852     for wordIndex = 1:numOfWords
853         temp(:, :) = roles(sentenceIndex, wordIndex, :);
854                 %role is the role of the current word
855         role = temp';
856         % if the sentence is not finished yet
857         if (~any(role))
858             break;
859         end
860                 %calculate the received value from layer 3 to situation layer
861         %compare them with roles vector
862         winner =  net.TESToutput(sentenceIndex, wordIndex);
863                 %IF AN ERROR OCCURS
864                  if (~any(net.input{2}.IW(winner, :)))
865             winner
866             display('**************');
867                 end
868                 %correct is the normalized dot product of the correct result and
                        output (after WTA)
869                 net.correct(sentenceIndex, wordIndex) = normalizedDotProduct(net.
                        input{2}.IW(winner, :) , role);
870                 %exactCorrect is the normalized dot product of the exactOutput  and
                        output (no WTA)
871                 net.exactCorrect(sentenceIndex, wordIndex) = normalizedDotProduct(net
                        .exactOut(index, :) , role);
872                 index = index + 1;
873     end
874 end
875 end
```

Appendix F

The Programming Code for the Second Approach

Matlab Code

```
function [net] = main()
%this is the main function of the program

clear all

%tStartMain keeps the start time of the running program
% for log
tStartMain = tic;

net.numInputs = 2; %num of inputs
net.numLayers = 5;  %num of non-i/o layers
net.numOfEpochs = 40; %total number of epochs
net.thresholdDP = 0.999;%To check if two vectors are close enough, the result of
    their normalized
                                                %dot product is compared with this
                                                    threshold.
net.logDP = 10; % for logging purposes

% Semantic Classifier Layer
net.layer{1}.c = 0.5;
net.layer{1}.learningRate = 0.5;
net.layer{1}.size = 30;
net.layer{1}.firstTime = 1;%is initially set to 1, and then to 0

% Hidden Layer of SRN
```

```
net.layer{2}.size = 200;
net.layer{2}.alpha = 0.1;%alpha = learning rate

% Context Layer of SRN
net.layer{3}.size = net.layer{2}.size;
net.layer{3}.alpha = 0.1;

%SRN Output
net.layer{4}.size = net.layer{1}.size;
net.layer{4}.alpha = 0.1;


%Memory Layer
net.layer{5}.size = net.layer{2}.size;

% Role Layer
net.layer{6}.size = 30;
net.layer{6}.c = 0.6;
net.layer{6}.learningRate = 0.1;
net.layer{6}.firstTime = 1;

%prepare the training/testing corpora (vectors)
[ trainingVector, roleVectors, testVector, roleTestVector ] = prepareInputOutput();
display('Input Output Ready.');

% set the size of the input layer of  the network according to the input
% files
net.input{1}.size = size(trainingVector, 3);
net.input{2}.size = size(roleVectors, 3);
%set the hebbian learning rate
net.input{2}.increment = 0.025;

% initialize the network
net = initialize(net);
display('Network Initialized');

%train the network
net = train(net, trainingVector, roleVectors);
display('Network Trained');

%test the network
net = test(net, testVector);
display('Network Tested');

%Verify  if the outputs of the testing phase is correct
net = justify(net, roleTestVector, testVector);
display('Results Verified.');
```

```
71
72 %to keep the running time of the program
73 %for log
74 tElapsed = toc(tStartMain);
75 minutes = tElapsed / 60
76
77 beep
78
79 end
80 function [ trainingVector, roleVectors, testVector, roleTestVector ] =
       prepareInputOutput()
81 %PREPAREINPUTOUTPUT prepares the vector input of the network
82 %This function first reads the word/role vectors and then reads the
83 %train/test corpora and assigns a vector to each word/role in the corpora.
84
85 % extract the role/word vectors
86 % word.txt contains the vocabulary list and vectors
87 wordList = readTrainingWords('words.txt');
88 % role.txt contains roles list and vector
89 roleList = readTrainingWords('roles.txt');
90
91 % prepare training sentence vectors
92 % trainSet.txt contains training sentences
93 trainSet = readSentences('trainSent.txt');
94 % parse sentences into vectors
95 trainingVector = parseSentences(trainSet, wordList);
96
97 % prepare training role vectors
98 % trainRole.txt contains roles corresponding to sentences in the training corpus
99 roleSet = readSentences('trainRole.txt');
100 % parse sentences into vectors
101 roleVectors = parseSentences(roleSet, roleList);
102
103 % prepare testing sentence vectors
104 % testSent.txt contains testing sentences
105 testSet = readSentences('testSent.txt');
106 % parse sentences into vectors
107 testVector = parseSentences(testSet, wordList);
108
109 % prepare testing role vectors
110 % testRole.txt contains roles corresponding to sentences in the testing corpus
111 roleTest = readSentences('testRole.txt');
112 % parse sentences into vectors
113 roleTestVector = parseSentences(roleTest, roleList);
114
115 end
116 %%%
```

```
117 function words = readTrainingWords( fileName )
118 %READTRAINING reads the word/role list and their vectors
119 %   outputs a struct containing word/role and type and vector
120 % Note: type is not used in this program
121
122
123 fid = fopen(fileName);
124
125 %read first line
126 line = fgets(fid);
127 numOfLines = 1;
128 while ischar(line)% while the input file is not yet finished
129         % extract word, then type, then vector from the line
130     [word, line] = strtok(line);
131     [type, line] = strtok(line);
132         % cast string-vector to numerical vector
133     vector = str2num(line);
134         % construct the output struct
135     words{numOfLines} = struct('word', word, 'type', type, 'vector', vector);
136
137         %read next line
138     numOfLines = numOfLines + 1;
139     line = fgets(fid);
140 end
141
142
143 fclose(fid); % close the file
144
145 end
146
147 end
148 %%%
149 function sentences = readSentences( fileName )
150 %READSENTENCES reads and ouputs sentences from a file (fileName)
151 %   each line is a sentence
152
153 fid = fopen(fileName);%open the file
154
155 % read first line
156 numOfLines = 1;
157 line = fgetl(fid);
158 while ischar(line)% while the input file is not yet finished
159         % this line is the current sentence
160     sentences{numOfLines} = line;
161         %read next line
162     numOfLines = numOfLines + 1;
163     line = fgetl(fid);
```

```
164 end
165
166 fclose(fid); % close the file
167
168 end
169 %%%
170 function [ words ] = parseSentences( sentences, wordList)
171 %PARSESENTENCES reads *sentences* and by the use of *wordList*
172 % parses them into *words*.
173
174 %sizeS is the number of sentences
175 [dummy sizeS] = size(sentences);
176
177 for sentenceIndex= 1:sizeS %for all sentences
178         % rest is first set to the whole sentence (sentence(sentenceIndex))
179         % and then the rest of the sentence
180     rest = sentences(sentenceIndex);
181     thisWord = rest;
182         %index is the word index in current sentence
183     index = 1;
184     while ~strcmp('.', thisWord)% parse each sentence, until period is encountered
185                 % thisWord is the current word of the sentence
186         [thisWord rest] = strtok(rest);
187                 % find and set the corresponding vector (to thisWord in sentence(i))
188         words(sentenceIndex, index, :) = findWordsVector(thisWord, wordList);
189         index = index + 1;
190     end
191         %next sentence index
192 end
193
194 end
195
196 function vector = findWordsVector(thisWord, wordList)
197 % findWordsVector looks for thisWord in wordList
198 % and output its corresponding vector.
199
200 %sizeW is the number of words in wordList
201 [dummy sizeW] = size(wordList);
202 for i = 1:sizeW %for all the words in the wordList
203         % if thisWord is the current word of the list
204         % vector (output) will be the corresponding vector of thisWord
205         %else continue the search through the list
206     if (strcmp(thisWord, wordList{i}.word))
207         vector = wordList{i}.vector;
208         return
209     end
210 end
```

```
211 %output thisWord if it is not in the list (ERROR)
212 thisWord
213 display('parseSentence: word not found.');
214 end
215 %%%
216 function net = initialize( net )
217 %INITIALIZE initializes the weights of the links
218 % and the initialization values of activation of nodes in layers.
219
220 % initialize weights from input-word to Semantic Classifier layer.
221 net.layer{1}.IW = initializeIWofCompetitiveLayers(net.input{1}.size, net.layer{1}.
        size);
222
223 % initialize weights of SRN
224 net.layer{2}.IW = initializeIWofBackPropLinks(net.layer{1}.size, net.layer{2}.size);
225 net.layer{3}.IW = initializeIWofBackPropLinks(net.layer{2}.size, net.layer{3}.size);
226 net.layer{4}.IW = initializeIWofBackPropLinks(net.layer{3}.size, net.layer{4}.size);
227
228 %initialize weights from input-role to Role Layer
229 net.layer{5}.IW = initializeIWofCompetitiveLayers(net.input{2}.size, net.layer{5}.
        size);
230 %initialize weights from ROLE layer to input-role (Hebbian Links thus 0)
231 net.input{2}.IW = zeros(net.layer{5}.size, net.input{2}.size);
232
233 %initialize Bias Value
234 net.layer{2}.bias = initializeIWofBackPropLinks(1, net.layer{2}.size);
235 net.layer{4}.bias = initializeIWofBackPropLinks(1, net.layer{4}.size);
236
237 %a dummy input into LUT - creating the table
238 net.sentenceWordTable = zeros(1, 2 * net.layer{2}.size);
239
240 %Initialize all the nodes/max-up-to-now-node-values to zero.
241 net.layer{1}.next = zeros(1, net.layer{1}.size);
242 for i = 1:net.numLayers
243     net.layer{i}.nodes = zeros(1, net.layer{i}.size);
244     net.layer{i}.max = zeros(1, net.layer{i}.size);
245 end
246
247 end
248 %%%
249
250 function IW = initializeIWofHL(bottom, up)
251 %INITIALIZEIWOFHL is used for initialization of competetive layers
252 %   The initialization is such that the sum of the weights leading to each
253 %   node in the layer = 1, so that it's fair.
254 IW = unifrnd(0, 1, [bottom, up]);
255
```

```
256 for i = 1:up
257     IW(:, i) = IW(:, i) ./ sum(IW(:, i));
258 end
259
260 end
261 %%%
262 function IW = initializeIWofSRN(bottom, up)
263 %INITIALIZEIWOFSRN initializes the weights in the SRN
264 %   each link take a random value between -0.1 and +0.1
265
266 IW = unifrnd(-0.1, 0.1, [bottom, up]);
267
268 end
269 %%%
270
271 function [ net ] = train( net, sentences, roles)
272 %TRAIN takes the network, and the network's inputs and trains the links of the
        network.
273 % For #ofEpochs do
274 % For each sentence
275 % initialize context layer to 0. Nothing in the memory.
276 % For each word in the sentence
277 % Train the SRN
278 % Again go through the words of the sentence
279 % Make the LUT.
280
281 %not for train-for log
282 index = 1; % this is used for keeping the context layer index to cluster if later
283 indexInner = 1;
284
285 numOfSentences = size(sentences, 1);
286 % The period at the end of the sentence is not counted.
287 numOfWords = size(sentences(1, :, :), 2) - 1;
288
289 %not for train-for log
290 net.ifThisWord = zeros(numOfSentences, numOfWords);
291
292 %for num of epochs
293 for i = 1:net.numOfEpochs
294
295     for sentenceIndex = 1:numOfSentences
296         %context/hidden layer - clean memory
297         net.layer{3}.nodes = zeros(1, net.layer{3}.size);
298         net.layer{2}.nodes = zeros(1, net.layer{2}.size);
299
300                 %no of words in each sentence is one less than its size (no period)
301         numOfWords = size(sentences(sentenceIndex, :, :), 2) - 1;
```

```
302
303         %FIRST PASS: SRN training
304         for wordIndex = 1:numOfWords
305             %tempNext keeps the next word'
306             tempNext(:, :) = sentences(sentenceIndex, wordIndex + 1, :);
307             nextWord = tempNext';
308             %if the next word is not empty = if this word is not .
309             if (~any(nextWord))
310                 break;
311             end
312             % temp keeps current word
313             temp(:, :) = sentences(sentenceIndex, wordIndex, :);
314             word = temp';
315             %tempNodes keeps the contents of layer 4 for the previous word.
316             %For speeding up the computation only. semantic classifier of current
                    word.
317             tempNodes = net.layer{1}.next;
318             %First compute the semantic classifier of the next word - layer 4 (kept
                    as
319             %layer{1}.next).
320             net.activeLayer = 1;
321             net = trainCL(net, nextWord);
322             net.layer{1}.next = net.layer{1}.nodes;
323
324             %If the first word of the sentence then tempNodes is no use.
325             %Find the semantic classifier of the current word.
326             if (wordIndex == 1)
327                 net = trainCL(net, word);
328             else
329                 net.layer{1}.nodes = tempNodes;
330             end
331
332             %train the SRN if not the last epoch
333             net.activeLayer = 2;
334             net = transferSRN(net);
335             if (i < net.numOfEpochs)
336                 net = backpropSRN(net);
337             end
338
339             %role is input{2}, the situation vector from file
340             tempR(:, :) = roles(sentenceIndex, wordIndex, :);
341             role = tempR';
342             %activation propagetes to role layer, and a single node is
343             %selected in the WTA competition in Layer 6 (Role layer)
344             net.activeLayer = 6;
345             net = trainCL(net, [role]);
346
```

```
347                     %The links from layer{6}(role layer) to input/output layer
348             %trained-Simple Hebbian
349               net.activeLayer = 6;
350             net = trainOutputLayer(net, [role]);
351
352         end
353
354             % if SRN still needs training - keep training
355         if (i < net.numOfEpochs)
356             continue;
357         end
358
359         % Now layer 2 represents the sentence type. Search if any point is
360         % close enough to this point. If so find the index and save the words
                accordingly.
361         % Otherwise, save if this is a new one.
362
363         %Keep the HL of SRN in the MEMORY layer
364         net.layer{5}.nodes = net.layer{2}.nodes;
365
366         %context layer - clean memory
367         net.layer{3}.nodes = zeros(1, net.layer{3}.size);
368         net.layer{2}.nodes = zeros(1, net.layer{2}.size);
369
370         numOfWords = size(sentences(sentenceIndex, :, :), 2) - 1;
371
372         %Second PASS  the last epoch through the sentence
373         for wordIndex = 1:numOfWords
374             %tempNext keeps the next word'
375             tempNext(:, :) = sentences(sentenceIndex, wordIndex + 1, :);
376             nextWord = tempNext';
377             %if the next word is not empty = if this word is not .
378             if (~any(nextWord))
379                 break;
380             end
381             % temp keeps current word
382             temp(:, :) = sentences(sentenceIndex, wordIndex, :);
383             word = temp';
384             %tempNodes keeps the contents of layer 4 for the previous word.
385             %For speeding up the computation only. semantic classifier of current
                    word.
386             tempNodes = net.layer{1}.next;
387             %First compute the semantic classifier of the next word - layer 4 (kept
                    as
388             %layer{1}.next).
389             net.activeLayer = 1;
390             net = trainCL(net, nextWord);
```

```
391             net.layer{1}.next = net.layer{1}.nodes;
392
393             %If the first word of the sentence then tempNodes is no use.
394             %Find the semantic classifier of the current word.
395             if (wordIndex == 1)
396                 net = trainCL(net, word);
397             else
398                 net.layer{1}.nodes = tempNodes;
399             end
400
401             %go through the SRN
402             net.activeLayer = 2;
403             net = transferSRN(net);
404
405             % Now layer 2 represents the sub-sentence type up to this point. Search
                    if any point is
406             % close enough to this point. If so find the index and save the words
                    accordingly.
407             % Otherwise, save if this is a new one.
408             %find the role
409
410                         %role is input{2}, the situation vector from file
411             tempR(:, :) = roles(sentenceIndex, wordIndex, :);
412             role = tempR';
413             %activation propagetes to role layer, and a single node is
414             %selected in the WTA competition in Layer 6 (Role layer)
415             net.activeLayer = 6;
416             net = trainCL(net, [role]);
417                         % which node is on in role layer? = role type
418             roleType = find(net.layer{6}.nodes);
419                         %The links from layer{6}(role layer) to input/output layer
420             %trained-Simple Hebbian
421             net.activeLayer = 6;
422             net = trainOutputLayer(net, [role]);
423
424                         %net.sentenceWordType <- see which row of the table the (
                                hidden layer, memory layer) belong to
425             net = findSentenceWordType(net, roleType);
426                         %LOG
427             net.logSeven(sentenceIndex, wordIndex) = roleType;
428             net.logSentWords(sentenceIndex, wordIndex) = net.sentenceWordType;
429
430                         %fill the LU table, that row keeps the role type and the (
                                hidden layer, memory layer)
431             net.LUtable(1,  net.sentenceWordType) = roleType;
432
433         end
```

```
434
435     end
436
437 end
438
439 end
440 %%%%
441 function [ net ] = trainCL( net, input)
442 %trainCL trains the competetive layer
443 %   input is the input layer to the network
444
445 layerNo = net.activeLayer;
446 net = transferCL(net, input); %calculate the active layers' activation value
447 winner = find(net.layer{layerNo}.nodes, 1); % winner is the winner node in the active
        layer (for speeding up the learning phase)
448 % train the links leading to the winner node.
449 net.layer{layerNo}.IW(:, winner) = vonDerMalsburg(net, input) + net.layer{layerNo}.IW
        (:, winner);
450
451 end
452 %%%
453 function net = transferCL(net, input)
454 %transferCL transfer function for competetive layers
455 % input is the input layer to the active layer, the activation flows to
456 % the active layer through this function.
457
458 layerNumber = net.activeLayer;
459 %recieved is the activation received in the layer.
460 received = input * net.layer{layerNumber}.IW;
461 % if the first time learning (no max still available.)
462 if ( ~ net.layer{layerNumber}.firstTime)
463     %transfer function
464     activation = (net.layer{layerNumber}.c * received) + ...
465         ((1 - net.layer{layerNumber}.c) * (received./(net.layer{layerNumber}.max)));
466     temp = compet(activation'); %compet : 0 if not the winner. ow 1.
467 else
468     net.layer{layerNumber}.firstTime = 0;
469     temp = compet(received');
470 end
471 % nodes <- temp
472 net.layer{layerNumber}.nodes = zeros(1, net.layer{layerNumber}.size);
473 net.layer{layerNumber}.nodes(1, find(temp, 1)) = 1;
474 % update the max
475 net.layer{layerNumber}.max = max([received;net.layer{layerNumber}.max]);
476
477 end
478 %%%
```

```
479 function dw = vonDerMalsburg(net, input)
480 %VONDERMALSBURG is used for training the links.
481 %   input is the layer below the active layer. dw is the change in the
482 %   links between the input layer and the active layer.
483
484 layerNo = net.activeLayer;
485 % winner is found to speed up the learning.
486 winner = find(net.layer{layerNo}.nodes, 1);
487 % the change is only made in the links leading to the winner node.
488 dw =  net.layer{layerNo}.learningRate .* ...
489     ( (input ./ sum(input))' - net.layer{layerNo}.IW(:, winner));
490
491
492 end
493 %%%
494 function  net  = transferSRN( net )
495 %TRANSFERSRN is the feed forward phase of the SRN
496
497 % hidden layer <- logsig( context layer & input(POS)layer * their weights & bias )
498 % to speed up the calculation of layer 2
499 winner = find(net.layer{1}.nodes, 1);
500 net.layer{2}.nodes = logsig(net.layer{3}.nodes * net.layer{3}.IW + net.layer{2}.IW(
        winner, :) + net.layer{2}.bias);
501
502 %copy hidden layer (2) into context layer (3)
503 net.layer{3}.nodes = net.layer{2}.nodes;
504
505 %output layer (4) <- logsig( hidden layer * their weights & bias )
506 net.layer{4}.nodes = logsig(net.layer{2}.nodes * net.layer{4}.IW + net.layer{4}.bias)
        ; % output layer <- hidden layer & bias
507 end
508 %%%
509 function  net  = backpropSRN( net )
510 %BACKPROPSRN performs the backpropagation algorithm on the SRN
511
512 % delta{out} = teacher - output
513 % other delta's = delta_upper * W in the middle
514 % W += alpha * input * [output * (1 - output) * delta_output]
515
516 % to speed up the computation
517 % delta <- output layer .* (1 - output layer) .* (teacher - output layer)
518 delta = (net.layer{4}.nodes .* (1 - net.layer{4}.nodes)) .* (net.layer{1}.next - net.
        layer{4}.nodes);
519 % delta hidden layer <- (delta * links to outputlayer) .* hidden layer .* (1 - hidden
        layer)
520 deltaHidden = ((net.layer{4}.IW) * (delta)')' .*  net.layer{2}.nodes .* (1 - net.
        layer{2}.nodes);
```

```
521
522 %modify the links/biases according to delta of each layer
523 net.layer{2}.IW = net.layer{2}.IW + net.layer{2}.alpha *  net.layer{1}.nodes' * (
        deltaHidden );
524 net.layer{3}.IW = net.layer{3}.IW + net.layer{3}.alpha *  net.layer{3}.nodes' * (
        deltaHidden );
525 net.layer{4}.IW = net.layer{4}.IW + net.layer{4}.alpha *  net.layer{2}.nodes' * (
        delta );
526
527 net.layer{2}.bias = net.layer{2}.bias + net.layer{2}.alpha * ( deltaHidden );
528 net.layer{4}.bias = net.layer{4}.bias + net.layer{4}.alpha * ( delta );
529
530 end
531 %%%
532
533 function [ net ] = trainOutputLayer( net, input )
534 %trainOutputLayer trains the links between the situation layer (input{2})
535 % and the role layer (layerNo)
536 % The links are from role layer to the situation layer
537 % The training algorithm = simple Hebbian
538
539
540 layerNo = net.activeLayer;
541 %to speed up computation
542 winner = find(net.layer{layerNo}.nodes);
543
544 %increase the weights from the winner node to the active nodes in situation
545 % layer by the constant "increament"
546 net.input{2}.IW(winner, :) = net.input{2}.IW(winner, :) ...
547      + input .* (net.input{2}.increment);
548 end
549
550 %%%
551
552 function net = findSentenceWordType(net, roleType)
553 % finds the column of LUT to which (hidden layer, memory layer) belongs to
554 % if none creats new column for that.
555 % role layer is here for error detection
556 % net.sentenceWordType will be the column of the current (hidden layer, memory layer)
557
558 % Now (memory layer, hiddedn layer) represents the sentence type and words posiotion.
559 % Search if any point is  close enough to this point.
560 % If so find the index and save the words accordingly.
561 % Otherwise, save if this is a new one.
562
563 [tableSize dummy] = size(net.sentenceWordTable);
564 for i = 1:tableSize
```

```
565 %for every column in the table
566
567         % see how close are (memory layer, hiddedn layer) and the current item of the
                LUT
568     dotP = normalizedDotProduct([net.layer{5}.nodes net.layer{2}.nodes], net.
            sentenceWordTable(i, :));
569
570         % if the first item in the table
571     if ( dotP == -100)
572         %newEntry
573         net.sentenceWordTable(i, :) = [net.layer{5}.nodes net.layer{2}.nodes];
574         net.sentenceWordType = i;
575         return;
576     end
577     if( net.LUtable(1,  i) ~= roleType)
578         continue;
579     end
580     if ( dotP >= net.thresholdDP )%if they are close enough
581         net.sentenceWordType = i; % column found
582                 %LOGGING PURPOSES TO SET THRESHOLD
583         if (dotP < net.logDP)
584             net.logDP = dotP;
585         end
586         return;
587     end
588
589 end
590 % if not returned means new entry must be added at the end of the table
591 %display('one added')
592 net.sentenceWordTable(tableSize + 1, :) = [net.layer{5}.nodes net.layer{2}.nodes];
593 net.sentenceWordType = tableSize + 1;
594
595 end
596 %%%
597
598 function cosine = normalizedDotProduct(x, y)
599 % NormalizedDotProduct computes the value of the
600 % cosine between two  vectors. x.y = |x|*|y|*cos(xoy); where x.y means the
601 % dot product of the two vectors x and t, |x| and |y| mean the
602 % norm/magnitude of vectors x and y respectively, and xoy is the cosine
603 % between vectors x and y.
604 % According to this, cos(xoy) = x.y/(|x|*|y|).
605 % NDP ~ Correlation
606
607 normx = norm(x);%the function norm computes the norm/magnitude of a vector
608 normy = norm(y);
609
```

```
610 if (size(x, 2) ~= size(y, 2))
611     %if the two vectors are not of the same size/dimension;no dot product
612     %defined.
613     display('normalizedDotProduct: vector size dismatch');
614     cosine = 0;
615 elseif( normx == 0 || normy == 0 )
616     %if one of the vectors is origin, they are perpendicular
617     cosine = -100;
618     display('normalizedDotProduct: One of the vectors was 0.');
619 else
620     cosine = (x*y') / (normx * normy);
621 end
622
623 end
624 %%%
625 function net = test(net, sentences)
626 % TEST tests the trainednetwork
627 % inputs are net and the test sentences
628 % All the outputs (situation layer for every word) is saved
629 % to be later verified
630
631 numOfSentences = size(sentences, 1);
632 %LOG index
633 index = 1;
634
635 %For every sentence in the test corpus
636 for sentenceIndex = 1:numOfSentences
637     %context/hidden layer - clean memory
638     net.layer{3}.nodes = zeros(1, net.layer{3}.size);
639     net.layer{2}.nodes = zeros(1, net.layer{2}.size);
640
641         %num of words is less than the size of each sentence (no period)
642     numOfWords = size(sentences(sentenceIndex, :, :), 2) - 1;
643         %%FIRST PASS
644         %for each word in the sentence
645     for wordIndex = 1:numOfWords
646
647                 temp(:, :) = sentences(sentenceIndex, wordIndex, :);
648                 %word is the current word of the sentence
649         word = temp';
650                 %tempNext keeps the next word'
651         tempNext(:, :) = sentences(sentenceIndex, wordIndex + 1, :);
652         nextWord = tempNext';
653         %if the next word is not empty = if this word is not .
654         if (~any(nextWord))% if the sentence is processed
655             break;
656         end
```

```
657         %find the semantic layer activation
658         net.activeLayer = 1;
659         net = transferCL(net, word);
660         %then the info flow and find the activation in layers of SRN
661         net.activeLayer = 2;
662         net = transferSRN(net);
663     end
664         %copy context layer to the memory layer
665     net.layer{5}.nodes = net.layer{2}.nodes;
666     %context/hidden layer - clean memory
667     net.layer{3}.nodes = zeros(1, net.layer{3}.size);
668     net.layer{2}.nodes = zeros(1, net.layer{2}.size);
669
670     %SECOND PASS
671     for wordIndex = 1:numOfWords
672        temp(:, :) = sentences(sentenceIndex, wordIndex, :);
673                 %word is the current word of the sentence
674         word = temp';
675                 %tempNext keeps the next word'
676         tempNext(:, :) = sentences(sentenceIndex, wordIndex + 1, :);
677         nextWord = tempNext';
678         if (~any(nextWord))% if the sentence is processed
679             break;
680         end
681         %find the semantic layer activation
682         net.activeLayer = 1;
683         net = transferCL(net, word);
684         %then the info flow and find the activation in layers of SRN
685         net.activeLayer = 2;
686         net = transferSRN(net);
687         %find the match for the memory layer + context layer of SRN in the LU
688         sentenceWordType = findClosestInTable([net.layer{5}.nodes net.layer{2}.nodes
                ], net.sentenceWordTable);
689         %find the role, by the role
690         roleType = net.LUtable(1, sentenceWordType);
691         %this is going to be the role layer
692         net.layer{7}.nodes = zeros(1, net.layer{7}.size);
693         net.layer{7}.nodes(1, roleType) = 1;
694         %exactOut(index, :) : output -situation layer
695         net.exactOut(index, :) = net.layer{7}.nodes * net.input{2}.IW;
696                 %LOG
697         index = index + 1;
698                 %LOG
699         net.logTestSentType(sentenceIndex, wordIndex) = sentenceWordType;
700         net.TESTa(sentenceIndex, wordIndex) = find(net.layer{1}.nodes);
701         net.TESToutput(sentenceIndex, wordIndex) = roleType;
702
```

```
703     end
704 end
705
706 end
707 %%%
708 function index = findClosestInTable(input, table)
709 %finds the closest (greatest normalized dot product) item in the LU to the input
710 %to find the corresponding role
711 [sizeT dummy] = size(table);
712 min = -1;
713 %go through all the items in table
714 for i = 1:sizeT
715         %dotP is how close is this item to the input
716     dotP = normalizedDotProduct(input, table(i, :));
717         %if this item is not better than the previously found item move on
718     if (dotP < min)
719         continue;
720     end
721         % other wise keep this item's index and update min (distance)
722     min = dotP;
723     index = i;
724
725 end
726
727 end
728 %%%
729 function net = justify( net, roles, sentences )
730 %JUSTIFY justifies/verifies the results
731 %goes through the saved outputs for each word
732 %checks it with the roles in the testing corpus
733
734 numOfSentences = size(roles, 1);
735 index = 1;
736 %For every sentence
737 for sentenceIndex = 1:numOfSentences
738     numOfWords = size(roles(sentenceIndex, :, :), 2);
739         %For every word
740     for wordIndex = 1:numOfWords
741         temp(:, :) = roles(sentenceIndex, wordIndex, :);
742                 %role is the role of the current word
743         role = temp';
744         % if the sentence is not finished yet
745         if (~any(role))
746             break;
747         end
748
749             %calculate the received value from layer 3 to situation layer
```

```
750             %compare them with roles vector
751             winner =  net.TESToutput(sentenceIndex, wordIndex);
752                     %ERROR LOG
753             if (~any(net.input{2}.IW(winner, :)))
754                 winner
755                 display('**************');
756             end
757             net.correct(sentenceIndex, wordIndex) = normalizedDotProduct(net.input{2}.IW(
                    winner, :) , role);
758             net.exactCorrect(sentenceIndex, wordIndex) = normalizedDotProduct(net.
                    exactOut(index, :) , role);
759             index = index + 1;
760         end
761     end
762     end
```