NTNU
Innovation and Creativity

A Connectionist Language Parser and Symbol Grounding

Experimental coupling of syntax and semantics through perceptual grounding

Sveinung Monsen

Master of Science in Informatics
Submission date: June 2006
Supervisor: Keith Downing, IDI
Co-supervisor: Sule Yildirim, Høyskolen i Hedmark

Norwegian University of Science and Technology
Department of Computer and Information Science

Thesis Description

0.1 The description

This is the original textual description of the thesis:

How are sentences represented in a neural network? Look into traditional and current approaches to natural language parsing and understanding, and focus on the connectionist strategy - using neural networks. Learn about how they perform and find areas in particular need of improvement. Investigate how the semantic component may be improved through perceptual grounding. Do experimental implementations of a connectionist parser, symbol grounding, and word-meaning relations, if possible.

Abstract

I hear and I forget. I see and I remember. I do and I understand.

- Confucius

The work in this thesis is about natural language processing and understanding, within the context of artificial intelligence research. What was attempted to achieve here was to investigate how meaning is contained in language, particularly with respect to how that information is encoded and how it can be decoded, or extracted. The aspects deemed most relevant for this quest was automated processing of the syntactic structure of sentences, and their semantic components. Artificial neural networks was chosen as the tool to perform the research with, and as such part of the goal became research on connectionist methods. A side-goal of interest was to look into the possibility of using insight into neural networks to gain deeper understanding of how the human brain processes information, particularly language. This area was not explicitly focussed on during the research.

The methodology selected for achieving the goals was to design and implement a framework for developing neural network models, and further to implement NLP and NLU[1] systems within this framework. The systems selected to explore and implement were: a parser for handling the syntactic structure and a symbol grounding system for dealing with the semantic component. A third system was also implemented for investigation into an evolutionary-based communication model on the development of a shared vocabulary between autonomous agents. All implementations were based on recent research and results by others.

[1]NLP: Natural Language Processing, NLU: Natural Language Understanding.

Preface

The best argument against democracy is a five-minute conversation with the average voter.

`- Winston Churchill`

This thesis was written during the fall of 2005 and the spring of 2006, at the Norwegian university of NTNU, department of IDI, at the faculty of IME. It is the result of investigations into the use of artificial neural networks in research on language processing and understanding. During the two-year program for the master's-degree, a variety of artificial intelligence related subjects were taken to lay the groundwork for writing a thesis on language and AI. The more relevant subjects taken includes: *"Machine Learning & Case Based Reasoning", "Knowledge Representation", "Advanced AI Programming", "Computational Linguistics", "Subsymbolic Artificial Intelligence", and a Neuroscience course.*

By covering an as wide range of subjects as possible, I hoped to gain sufficiently diverse knowledge to avoid suffering from the scientific version of tunnel-vision. That was the idea anyhow. All code for the implementations was written in the Java[2] programming language.

For help and guidance during my work on the thesis I had two supervisors, Keith Downing at *NTNU*, and Sule Yildirim, at *Høyskolen i Hedmark.* Sule was my main supervisor.

[2]Java SE, version 1.5.0 or higher required.

Acknowledgements

I love deadlines. I like the whooshing sound they make as they fly by.

`- Douglas Adams`

I would like to thank all the people, and things, that have made my life more endurable during the last two years - and who made it possible to complete this thesis. Just barely touching into the post-deadline zone.

I specifically want to thank my main supervisor, Sule Yildirim, for being exceptionally supportive and helpful throughout my work on this thesis. To all others concerned, and you know who you are, your contributions was and is greatly appreciated.

Sveinung F. Monsen,
Trondheim, June 2006

Introduction

In the beginning the Universe was created. This has made a lot of people very angry and has been widely regarded as a bad move.

- Douglas Adams

There are too many languages in the world. Most of them unpleasantly difficult to learn, and they are anything but compatible, translation-wise. It is almost as if they were made to be so diverse and incompatible, on purpose. If one were to take certain parts of the Holy Bible literally, God is the one to blame. This accusation is not so far fetched, or outrageous, as it may seem. In fact, in *Genesis* chapter 11, verse 1-9, it is stated that the lord confused the tongue of man and spread them all over the world. But, in however way this is interpreted, the undeniable fact remains that such an event is reflected by the current state of the world. Whether willfully enforced, or simply the result of a natural process, is irrelevant. This is what we've got, now we have to deal with it.

It is interesting to contemplate on possible effects of the *confusion of languages*, assuming it happened at an early point in the history of man. Apart from being an obvious communication barrier, it also facilitates segmentation of groups of people into isolated populations.[3] A useful and important aspect of this process is that it helps to ensure diversity in the development of new ideas, ranging from ways to understand nature to the formation of distinct structures of society. The point being made here is that within homogenous groups ideas tend to spread quickly and to be adopted through imitation. There, focus tends to fall on the *refinement* of ideas, whereas the existence of several distinct, geographically or communication-wise, isolated

[3]The separation itself is likely to further increase the formation of new languages, forming a positive feedback-loop.

groups enables the generation of *new* ideas and approaches.

Another important aspect of the diversity of languages, made relevant by increasing globalization, is the need for continual translation between them, coupled with the fact that manual translation is a non-trivial, time-consuming and expensive process. High quality, automated machine-translation would be an excellent solution to this, if someone could create such a system. That turned out to be a very difficult task. But, translation is not the only important element here. In recent years we've had an enormous production of information[4], and most of it has been made available digitally and represented in the form of human language.

So, how can we make use of all this stored knowledge, without having to search and read for hours, days or even years? A software agent can process huge amounts of data in an instant, but has no understanding of it whatsoever. If only the agent were able to treat the stream of bits-and-bytes as information-carrying packets of words and sentences, a huge breakthrough in information processing would be realized.

A small step towards the ultimate goal has been investigated and tested in this thesis. The first aspect of sentences that needs to be handled in some way is their syntactic structure, and for this a parsing system has been implemented. The other main aspect treated in my work is that of meaning and the meaning-content contained in sentences. Syntactic parsing is difficult accomplish, correctly extracting the semantic components is a whole lot worse. It is not even clear exactly how to define meaning and how it gets bound to words. The chosen approach to deal with this challenging and interesting problem is that of *semantic information processing* - by designing and implementing a perceptual symbol grounding system.

[4]Information - or data, it all depends upon the eye of the beholder.

Table of Contents

List of Figures

List of Tables

Chapter 1

Neural Networkology 101

It is a mistake to think you can solve any major problems just with potatoes.

`- Douglas Adams`

1.1 Introduction

The reader of this thesis is assumed to have general knowledge of the field of AI, and to have some basic familiarity of neural networks. Still, there are quite many technical terms and one cannot expect everyone to be familiar with all of them. In addition, some concepts are referred to by several different terms, (in the spirit of *a beloved child has many names*) as well as the fact that many authors tend to use slightly synonymous terms interchangeably, liberally. Thus, this contributes to more fuzzy and unclear definitions in the vocabulary of neural networks.

In this regard an introductory chapter on central terms and concepts within connectionism seems in order. Not to give exact definitions of all essential terms, but first and foremost to give the reader a general introduction to the relevant terminology. In fact, the descriptions given here are not even real definitions, in a strict sense. They are simply meant to show how these terms are commonly used, **and** what I, the author, mean when these words are used in this text. Hence, few citations and references are given.

1.2 Definitions

Connectionism has come to mean much the same as information processing using neural networks. This is especially true for the computer science community in AI. Originally, this was a much broader term, where neural networks were a branch within connectionism. Other communities, like cognitive science in biology, psychology and philosophy, still tend to use connectionism in the original, broader sense. [15]

A node, also called a unit, is the fundamental unit of a neural network, the equivalent of a biological neuron. It contains the connections to other nodes as well as the transfer function.

A link is the connection between neurons and each such link is associated with a weight. It's biological equivalent is a combination of the axon and the synapse, where the weight serves as the function of synapse excitability (and implicitly also as a threshold function).

The transfer function, also called the nodefunction, is responsible for integrating all incoming signals into one value that can be propagated to the next node, normalized to a number in the range [0.0, 1.0]. One of the most commonly used functions is the sigmoid function; 1 / (1 + exp(-x)). It is continuous and approximates 1.0 at x = -36.0 and 0.0 at x = 36.0. This is the transfer function used in my implementation.

An activation is the value that a node produces when its input is run through the transfer function. When we look at such activations from all or a group of nodes, we get what is called the activation pattern. It is usually taken from all nodes within a specific layer of neurons; the hidden-layer or output-layer. (Input-layer nodes obviously also produce activations, but are rarely considered for analysis).

Propagation is the process of transferring signals from node to node through weighted connections, the links.

The backpropagation algorithm, also called error backpropagation, is the core of any artificial neural net's ability to learn. Backpropagation is a deterministic algorithm that enables the net to learn by making small adjustments to all weighted connections. The size of an adjustment for a particular link is determined by the error of a neuron's activation, and that error value is initially computed as the difference of an output node's activation from the desired activation. As the name implies, the algorithm progresses backwards from the output-layer until the input-layer is reached. All consecutive error values are indirectly computed from the initial error by taking the derivative of the error at the previous layer, (n+1, as we are going backwards, and where n is the current layer), modulated by the strength of the connection over which a particular erroneous signal was received. It is a supervised learning algorithm [24].

Inputdata refers to what is presented to a network. Due to the nature of most transfer functions, any input data given to a neural network must be normalized to values in the range $\pm$ [0.0, 1.0].

The input-layer is the first layer of neurons in a network, and this is where input is presented from an external source. E.g. the environment, another network, etc.

The hidden-layer is the layer of nodes that is between the input-layer and the output-layer. Many simple, experimental neural networks only have one hidden layer, although there is no problem in having several hidden layers. In cases of multiple hidden-layers the term 'the hidden layer' will refer to the last hidden-layer, the one next to the output-layer, when speaking of the activation pattern. The activation pattern of this layer is of most interest when analyzing the 'how' and 'what' of a network's learning process, in terms of finding what kind of structure a network has identified in a given set of input data. Also, the activations in this layer shows the network in its most sub-symbolic state: no single neuron here represents a specific part of a concept, nor are the activations explicitly shaped in any direction during the learning process, but only through the errors computed at the output-layer. Thus, the pattern here is referred to as the network's internal representation of a (learned) concept.

The output-layer is where a network's performance is evaluated during training, and obviously also the last step in a series of transformations performed on the input-data. In most training schemes a network is supposed to learn not only to distinguish between input of distinct classes, but also to identify each such class as something specific. That is, each pattern that a net has learned to recognize must then be represented explicitly by a certain configuration of activation values on the output units. This way the output-layer activations act as a bridge from a net's sub-symbolic internal representation (the hidden-layer) to a much more symbolic (explicit) representation on its outer layer. Note though that this observation only holds if a network is being used in this common way, where such specific training targets are used by the backpropagation algorithm during the training process.

One-in-n bit representation. This is an encoding scheme using the one-in-n bit representation does exactly what it says. It generates a bit-pattern of n bits, where one, and only one, bit is turned "on". (Meaning 1, or 1.0). This kind of representation is often used when the original data is some kind of id number or class. It is also sometimes referred to as an *orthogonal bit-vector.*

1.3 Three basic models

Plain Feedforward - FNN. The Feedforward Neural Network is the simplest of the three models and all the others are based on this construction. As the name implies, propagation of values is strictly forward; there are no feedback-loops or any kind of redirection of propagated values. The minimum number of layers for any standard network is three, that is, one hidden-layer in addition the the input- and output-layer.

Simple Recurrent Network - SRN. The SRN [11] is an extension of the feedforward model where the output of all nodes in the (last) hidden-layer are fed back to context nodes in the input-layer. This works in a way such that each time an input pattern is given to the net and propagated forward, the values stored in the context units are propagated along with the current input just as if it were part of the new input. It is important to note that the values in the context units are those from the previous timestep, t-1, and as such makes up the network's internal representation of the previous input. What is accomplished by this then, is that the net can handle serial input. This is very useful when one needs to process input in real-time, and not all the input-data is immediately available. A typical example of such data would be sentences in natural language, where each word is presented one at a time, and there is a temporal aspect (word order) to consider.

A problem with the SRN concerning its very limited capacity for longer-term memory has been pointed out by several researchers. Mayberry III & Miikulainen have suggested an improvement regarding its usage in the technical report *SAARDSRN: A Neural Network Shift-Reduce Parser* [16].

Recursive Autoassociative Memory - RAAM. The RAAM is another interesting network model, attributed to Jordan Pollack [23]. The purpose of the RAAM is to enable neural networks to learn and represent structure, a problem that really caused concern in the early days of connectionism. (A typical field of research where representation of hierarchial structures is essential, is natural language processing, concerning syntax). The characteristic topology of this model is the sizing of its layers, particularly the input- and output-layer, which need to have the same number of neurons. The reason for this design derives from the way this network is trained, namely to reproduce on its output units the contents of its input units. Furthermore, the input- and output-layers are divided into two or more virtual partitions, thus enabling a training scheme to be set up that allows the net to learn and represent nested structure.

The whole process of training a RAAM to encode structure, and then recursively decode a pattern to extract all its subparts, is a complicated task. The topology of the RAAM itself is no more complex than the other models. Rather, the complexity stems from the way it is used. A detailed description of this process is beyond the scope of this brief overview, so a short explanation will have to suffice:

Imagine a task where the net is required to learn the full structure of a binary tree, that is, both the hierarchical relationships among the nodes and the content of each node, and represent this as one, fixed-width pattern. The layers are divided into `n` partitions, where $\texttt{n} = 2$ because we are working with a **binary** tree-structure. Training starts by presenting the input-layer with the content of two leaf nodes and autoassociating this composite pattern. Next, the activation pattern of the hidden layer are now taken to represent the root-node for this subtree. Training proceeds by encoding successively larger and larger subtrees into single patterns, in a bottom-up fashion.

The RAAM has a rather limited but difficult to precisely define storage capacity, and suggestions for improving its design and usage have been made. Levy and Pollack give a treatment on the topic in their technical report Infinite RAAM: A Principled Connectionist Substrate for Cognitive Modeling [17].

Chapter 2

Background and Motivation

I have come to the conclusion that my subjective account of my motivation is largely mythical on almost all occasions. I don't know why I do things.

- J.B.S. Haldane

2.1 Introduction

From here on *natural language processing and understanding* will be simplified by dropping the *natural* part, or fully abbreviated to NLP and NLU, respectively. It is also assumed that the *processing* part refers to automated, computational processes in the software/hardware paradigm of computer science, and their potential analogues in the neural networks of the human brain when analyzing sentences syntactically and semantically.

Why language processing and understanding? And why choose the connectionism paradigm which is well known for its computationally heavy training procedures? What speaks in favor of using this sub-symbolic methodological approach as opposed to the traditional one operating on the symbolic level, where one can directly utilize rules, task-specific algorithms, etc? Surely, there are many valid arguments both for and against both approaches, but the intention here is not to argue for either one of them as being better. Instead, I will try to give the rationale behind my decisions by looking at various kinds of motivating factors. These factors include advantages and disadvantages of each branch, with respect to NLP, as well as more strategic ones regarding similarities to neuroscience and fields of personal interest.

As for the initial question, why choose the language processing branch in AI, it will be treated in the next section, 2.2. Finally, a more subjective discussion of language itself is given in section 2.5, the purpose of which is to explain and justify the need for perceptual grounding in NLP and NLU.

2.2 Computational Language Processing

2.2.1 Introduction

Almost any number of positive effects can be pointed out as good reasons for wanting to work towards an ideal NLP/NLU system with human-like capabilities. Although utopian as a short-term goal, research in that direction should not be discouraged. To give a full treatment of possible interesting and useful applications of NLP/NLU research would require a paper in its own right, and would only serve to confuse the matter at hand: which is to point out the underlying motivations for choosing language processing and their justifications. With that in mind, only a few high-level aspects are considered. Common to all is the economic incentive in a global market hungry for new and better solutions to automated language processing systems, but it will not be elaborated upon any further.

2.2.2 Computer games - virtual environments

During the profound increase in the availability of PC's and the accompanying growth of the computer game industry in the early 80s, a lot of games were still largely text-based. This author spent a liberal amount of time playing them, with a particular interest in specimens of the adventure & role-playing category. The relevancy, finally, is that while the use of language in games had the potential to increase the gaming experience, the complexity level of NLP implementations was extremely low. And this has remained so. The dynamics of interaction with the virtual environment and the entities therein has been vastly improved, but only in regard to visual quality and physics simulations.

The bottom line is that any kind of virtual environment could greatly benefit from an advanced language processing module, not just computer video games. Whether you are talking to an AI-god to glean information about the immediate surroundings or communicating with a so-called intelligent agent, it would be nice to actually get a relevant and informative reply. This is currently not yet realized in any commercial package, and can thus be seen as a justification for further research into NLP in the language - AI domain.

2.2.3 Knowledge acquisition

One doesn't need a whole lot of imagination to realize the huge potential that lies in coupling of AI with decent language skills. The speed with which a software agent can process data would elevate the concept of speed-reading to whole new level, and if the agent could *understand* the text as well. Well, the implications should be obvious. *Understanding* is used here in the sense that recipient gains knowledge, in an orderly and logical fashion, derived from the processed text. Much in the way humans are believed to accomplish this feat.

Approaches like data-mining would become much more efficient and attractive. Being able to easily and accurately communicate with the mining agent, and to actually get the requested results, is another good candidate for invoking interest in computational language processing and understanding. Human - agent communication, automated knowledge acquisition, advanced reasoning, etc. are all accomplished through the use of natural language as the source of information. as well as the medium to convey the knowledge.

2.2.4 Automated machine translation - MT

There are quite many languages in use today, although a select few have tended to dominate the scene, like English, German, French and Spanish. However, none are about to completely subsume the others anytime soon, and thus there exists a very real need for translation between a significant number of languages. Consider then, the fact that the translation process itself is very time and resource consuming, especially when two languages are profoundly different. A large international body, like the EU, is a good example of a multilingual environment. In such an organization, the production of one document immediately necessitates a lot of extra work in the form of translation into the native tongue of all interested parties. The workload of these manual translation processes are further increased by a characteristic property of bureaucracy: excessive text-production.

Although some machine translation systems have been developed that perform fairly well, like Systran [27], they are far from perfect. Currently, they have to be developed specifically for a chosen language-pair, are quite expensive and take a long time to develop. The steep price of good MT systems thus effectively prevents access for the general public and smaller enterprises. Also, because only large or otherwise significant languages are being implemented in these serious MT systems, only the privileged few can enjoy the fruits of this work.

While the economic aspect may not be a real issue for a huge body like the EU, there are other considerations that makes manual translation necessary. One is the continual addition of new languages to this institution, and so the developers cannot provide all the required combinations of language-pairs to translate between. Another important reason is the need for high accuracy in the translations, resulting from the requirement that important content in documents be exactly and correctly translated.

2.3 Traditional approaches - GOFAI

2.3.1 Introduction

In the previous section (2.2) we looked at some central motivations for choosing to work with NLP and NLU, and why that interest could be justified. However, the treatment was on a level independent of any particular methodology, and as such without subjective bias towards a specific approach.

The remainder of this chapter will be more devoted to specific approaches, which can be narrowed down to roughly two groups: GOFAI and connectionism, where the GOFAI acronym stands for *Good Old-Fashioned AI*, and is meant to encompass all traditional methods. Two such approaches are briefly discussed in this section, with emphasis on advantages and disadvantages. Finally, in section 2.4, the paradigm of connectionism will be considered, and the reasons for choosing this platform to do research on computational language processing are put forward.

2.3.2 Rule-based parsing

The Early-algorithm [9], by Jay Early, is a typical, well known algorithm for parsing sentences using predefined grammar rules. This particular instance of rule-based parsing is constructed to use context-free grammar [14], or CFG, as a way of simplifying some aspects of a grammar. The reason for bringing attention to CFG is to point out that a complete modeling of language by a nice and clean set of rules was found troublesome from the very beginning. Because of these difficulties, ways to reduce the complexity were sought. As a result, CFG was developed. The fact that such a problematic aspect of rules was identified and accepted by *proponents* of this approach, not its *adversaries*, strongly suggests the seriousness of the problem. Thus, one can hardly be accused of unduly negative bias for seeking out a radically different methodology to language processing research.

As the name of this group implies, rule-based parsing relies on a set of rules to perform its syntactic analysis and, in some cases, semantic interpretations. The rules are explicit and usually kept in a rule-base. This method is efficient, easy to implement and such systems perform well, in principle. As long as the grammar rules are few and simple and few exceptions need to be handled, the rule-based strategy seems like an excellent choice.

The problems become painfully clear when one realizes the large number of rules needed to account for all syntactic structures occurring in the human language. In addition, natural language is notoriously rich in rule-exceptions, which requires the addition of new rules to an already very large rule-base. The mere size of it makes it very difficult to manage, concerning backlash between new and old rules, as well as the order of application of simultaneously matching rules.

Another problem with rules is the rules themselves, at least concerning the way they are commonly applied and their format. That is not to say rules are inherently bad, but to suggest there are domains in which they are less suitable to perform. The mismatch between strict and inflexible rules[1] and the relatively chaotic and imprecise domain of natural language strongly suggests

[1]Referring to rules as traditionally used, where they are usually defined to handle very specific events or circumstances, and cannot easily handle approximate equality or similarity.

another approach to NLP and NLU.

2.3.3 Probabilistic parsing

After the strictly rule-based approach to parsing turned out to be less than ideal, some factions in the AI community felt the need for a new approach, with properties sufficiently distinct from that of explicit rules. Probabilistic parsing [18] was born, and it is safe to say that it has profoundly different characteristics.

This scheme is centered around the mathematical model of probability calculus, utilizing both conditional and unconditional modes of computation. Of most interest and impact, are those grounded in conditional probability, as they incorporate the *context* factor. A popular method for achieving certain forms of parse-relevant context is N-GRAMS [18], and is commonly used in part-of-speech prediction, sense disambiguation [18], for finding likely syntactic bindings, etc. Bayesian [1] models are popular here, and particularly the *naive Bayes* [2] conditional probability model has been known to be widely used.

Though the approach is relatively new, and has only recently gained much attention, current consensus is that it is a promising new branch. No explicit rules are used, all that is needed is a lot of training data in the form of text-corpora to build tables of context dependent probabilities. In this regard, probabilistic parsing is very distinct from other traditional approaches, and bears, arguably, surprisingly much resemblance to connectionism[2].

So, what is wrong with a probability-based approach to NLP? Actually, as far as performance measures regarding speed of execution and correctness are considered, nothing. That is, it is by no means perfect, but not lacking with respect to other contemporary models, like that of connectionism. Rather, the issue lies with the second part of the NLP - NLU pair, namely natural language *understanding*. Arguably, NLU has more to do with semantics than syntax, morphology, pre- and suffixes, etc. Apparently, the semantic components have been badly neglected in language processing, and to say that this is much due to the very difficult-to-define nature of meaning can hardly be

[2]Note: personal opinion of author, me.

called jumping to conclusions.

Now, while probabilistic methods may be better suited to deal with the fuzzy concept of meaning than the old style rule-based approach, no serious effort to incorporate deep semantics have surfaced.[3] A final comment on this will be an old saying that goes like this; when old methods fail, try something new. In spirit of that statement, the world of neural networks are a tempting new candidate. Being both new and adequately distinct from the other methods it is unlikely to get stuck in the same kinds of problems.

2.4 Why Connectionism?

2.4.1 Introduction

Having argued mostly for *why not* use the purely symbolic GOFAI, this section will look at appealing properties of connectionism. Now that the tenet of objective argumentation is clearly neglected, it seems prudent to explicitly acknowledge the fact. The justification is that this chapter (Chap. 2) is not meant to be scientifically objective, no approach is to be proven *the best*. Motivations are necessarily biased by personal preference. How else could there be many different approaches to a single problem, assuming all groups have access to essentially the same information? That is, if there actually were *one* objectively *correct* approach to NLP & NLU, then all or most researchers would had chosen that approach[4].

2.4.2 Parallel Distributed Processing

The concept of parallel distributed processing, or PDP, is closely linked to distributed representations, discussed in section 2.4.3, and has two specific properties relevant to the current discussion. Only the aspect of parallel processing are considered here.

- Speed of execution. Although most experimental, software-based implementations of neural networks run on serial- mode computer hardware, the processes in the network can be considered to be of parallel

[3]The truth of the statement is obviously limited by the current time of writing, May 2006, and what was actually discovered in my search for such enterprises)

[4]Assuming, of course, that most researchers are serious and dedicated to a scientific approach, and that they act in a rational way.

nature for all other purposes. So, a (physical) implementation on parallel hardware would be extremely fast, and as an extra benefit, the hardware would be relatively simple.

- There is no need for an extra control system for deciding in which order the neurons should fire, as everyone fires simultaneously, one layer at a time. Also, there is no need for conflict resolution between several matching rules, which was previously seen to be a complicating factor in section 2.3.2.

2.4.3 Distributed Representation

Every distributed representation is a pattern of activity across all units in a neural network. To be sure, representations are composed of the activities of the individual units, but none of these units represent any symbol. The representations are sub-symbolic in the sense that analysis into their components leaves the symbolic level behind.

A cluster of units represents a concept, where a cluster usually consists of a subset of the net's units. Exact identification of which units code for a given concept is not possible, in the sense that all units contribute to the activation pattern, but some units contribute more than others, in the sense of strength of the activation. In any case, this type of representation has two interesting properties.

- Every learned pattern, more or less, overlaps each other, and this necessarily causes similarities to be enhanced and differences to be smoothed out. One essential part of learning is generalization, and its core requirement is this very effect: finding the general structure in received stimuli.

- Considering a single learned concept, the distributed representation makes a network robust to damage, whether on the connections or neurons themselves. Since almost every unit contributes to the processing of an input, the removal or change of a few activations does not significantly alter the whole activation pattern. Thus, in case of localized damage or otherwise distorted input-data, the net will still perform admirably well. The overall favorable effect explained here is also referred to as *graceful degradation* [28], a property not commonly attributed to traditional approaches. (Although the recent probabilistic methods do exhibit this behavior to some degree).

2.4.4 Neuroscience and the Brain

The brain amounts to a gigantic neural network processor, and the problem of psychology is transformed into questions about which operations account for the different aspects of human cognition. The sub-symbolic nature of distributed representation provides a novel way to conceive information processing in the brain.

Thus, the final part on motivations for choosing to work within the realm of connectionism relates to the field of neuroscience and our brain. The fact that artificial neural networks are modeled after biological networks makes it plausible to assume that semantics and language understanding are possible to accomplish in connectionism. Moreover, AI research on neural networks may also contribute to the field of biological neuroscience, as attempted by O'Reilly and Munakata in their book "Understanding the Mind by Simulating the Brain" [21]. Paul Churchland is another researcher and author who has invested much time and effort into the boundary between artificial neural networks in AI and neuroscience. Two of his publications directly related to the current topic are the books *Matter and Consciousness* [6] and *The Engine of Reason, the Seat of the Soul: A Philosophical Journey into the Brain* [7].

2.5 A clinical perspective on Language

2.5.1 Introduction

Language, spoken or written, is notoriously difficult to analyze automatically. Purely in terms of syntactic structures, there are at least some identifiable parts of it. But, even at this basic level of analysis, there are severe problems. Word sense ambiguity, for instance, causes a host of problems when trying to interpret a given word as having one of many possible meanings. Even so, systems have been developed that perform decently on grammar handling and parsing of syntactic structures, if mostly in some or other restricted domain.

On the semantic level, though, there is nothing but trouble. A likely suspect is *meaning*, a word that itself defies the very notion of being precisely defined and operationalized. Where does the meaning, associated with a particular word, come from? Independent of how meaning is defined, words in our language do have a content, and it is common to all people within the language group[5]. In the following sections a perspective on the properties of language and its usage are given. A schematic view of a logical separation of language and the external environment of an agent[6] are also proposed.

2.5.2 The external world

The purpose of this section is to define the ***something else*** that is distinct and separate from the singularity of an agent's I-consciousness. Without that concept, there would be nothing to understand, describe or communicate and therefore nothing for a language to operate on. Thus, language neither would nor could ever arise.

Characterized by properties as continuous, dynamic, analog, complex and only partially observable, the external world refers to any environment external to a language-capable entity. There is structure and systematicity in this reality, but only implicitly. A more complete definition also includes events and processes in virtual environments internal to the agent, as long as it is observable by the its conscious awareness. (or other method of introspection). The reasons for what has been pointed out in this section, and the implications for the meaning-concept, are humbly suggested in section 2.5.4.

2.5.3 The internal map

Necessarily incomplete, and with reduced overall detail levels of what is represented, the internal map refers to all and any kind knowledge of all that is known about the environment in which an agent is situated. Examples of things known can be the existence of certain objects, properties, proper-

[5]This is not always the case, as some people seem totally out of sync with the rest of society. Frequently occurring miscommunications support this, but in most cases meanings are common to all.

[6]Agent is used in the broadest sense, including autonomous software and hardware entities, as well as humans.

ties of objects, events, relations, effects of events, *etc, ad infinitum*. For all purposes of the current discussion, the internal map can be equated to the brain of an agent. Although systematicity and structure can be said to be implicit to the external world, they are explicitly represented here as a form of knowledge.

2.5.4 Language and symbol grounding

This section is about language as a tool and how a truly useful semantic component can be realized and bound to words. Also considered here is how the language system can be elevated to enable the representation of actual knowledge in linguistic form, and even the formation of new knowledge.

So what is language? Basically, it is a system consisting of a set of symbols and a set of rules specifying the allowed usage of the symbols. This system is used to label and describe objects and concepts, among numerous other things, in an external word as defined in section 2.5.2. Recall now, from that section, the characteristic properties assigned to it: *continuous, dynamic, analog, complex and only partially observable*. Imagine that the language system can be assigned a wholly different set of properties using words like *limited, discrete, digital, simple, fully observable* and *static*.

An analogy might help bring some clarity as to what is being said here. Let language be a physical net with relatively large fixed- size masks (a typical fishing net), where the grid size indicates the (low) resolution. In this respect, the resolution represents the limit of the level of details that can be addressed and referenced in a continuous world. The fixed grid-size represents the initial inflexibility of the language.

The idea of all this is to portray language as initially simple and without any concept of meaning unless it becomes linked to some external world. However, by using it to label perceived objects on a simple naming basis, the groundwork for expanding language to support complex representations like prepositions and verbs are laid down. Now, in terms of the analogy, the process of expanding the capabilities of language can be visualized as successive operations of stacking the net onto itself and rotating it. Thereby achieving a higher resolution without actually changing the grid-size of each individual net. Hence, its ability to capture and represent the finer details of the underlying substrate, the external world, is increased by doing so.

Thus, language is boosted simply by using it, and the new constructs are grounded through this process. Incidentally, this is much like the principle utilized in computer programming languages: the power and expressiveness increases as a function of how much they are used. C++ got its `++` postfix for that reason, alluding to the effect of this postfix-notation of the increment operator: the value of such an expression increases *after* it has been used.

The main point is that a transition to more advanced levels of language use seems impossible without some form of symbol grounding. Perceptual Symbol Grounding may be an adequate tool to achieve it.

Chapter 3

The Goal

The tragedy of life doesn't lie in not reaching your goal. The tragedy lies in having no goal to reach.

- Benajamin Mays

3.1 Introduction

The goal of all research logically implies a quest for insight and new knowledge, in some chosen domain. From the part on "Background and Motivation", chapter 2, it is hopefully clear that the more abstract and overall goal here is to investigate computational language processing, particularly within the paradigm of connectionism. It is perhaps also in place to restate the all-encompassing area of research in which all the work is being done - Artificial Intelligence and Learning.

The goal of the research in my thesis originally started out vaguely as *how would a sentence in natural language be represented in a neural network?* For example, the sentence "*The flower on the table is red*" has both syntactic structure and semantic content, neither of which can be identified by considering each word in isolation. From this starting point there were two obvious areas to pursue: Parsing systems for working on the syntactic structures, and some type of semantic processing approach to get to the meaning of a sentence. In the following sections a more orderly and specific formulation of the goal and subgoals is attempted.

3.2 The Goal and its Subgoals

The intention is to achieve the main goal by fulfilling the tasks specified in the subgoals, and can be formulated as follows:
To gain more insight into the area of computational language processing, and try to find and suggest ways to improve current systems.

3.2.1 Gain insight into neural networks

This part of the goal relates to the usage and inner workings of neural networks. I had practically no knowledge whatsoever of the computational framework or models at the beginning of the two-year master program, so a study of neural networks was both necessary and useful. In principle, independent of any particular application. It can be more compactly formulated like this: *Gain insight into how neural networks work and learn the basic models*[1].

3.2.2 Neural networks applied to language

This concerns two specific aspects of language that any dedicated approach should handle, the syntactic structure of sentences and the temporal order in which words are accessed. Representation of structure, like that of a hierarchical syntactic parse-tree, was long held as a major problem for neural networks. Descent handling of serially input data was also a known problem in connectionism. The subgoal then becomes: *Empirically investigate the capabilities of neural networks to handle structured data and temporal data.*

3.2.3 NLP - realizing a parsing system

Almost an extension of subgoal 3.2.2, this goal is more specific, aiming for an implementation as physical result. In effect: *Investigate, implement and test a connectionist parser to deal with the syntactic structure of sentences, and look for ways to improve performance in terms of capabilities.*

[1]The FNN, SRN and RAAM.

3.2.4 NLU - realizing a symbol-grounding system

Another, both interesting and important, part of the NLP & NLU branch is dedicated to the semantic component of language. The category of systems referred to here are those attempting some form of symbol-grounding, and that is what is aimed at in this subgoal. It can be more formally stated as follows: *Investigate, implement and test a perceptual grounding system, and see how meaning might be extracted from sentences and coupled to language processing.*

3.2.5 Spinoffs - Neuroscience and biological models

The phrase "Understanding the mind by simulating the brain", from the book thus titled by O'Reilly & Munakata [21] comes to mind as an appropriate description of this subgoal. Although it has been included here in that respect, it should be characterized more as *an interesting possibility to learn more about how the brain works*, particularly with respect to how we process and understand language. No explicit research on the subject was done during the work on this thesis, but some insight might still be gleaned from working with artificial neural networks. That is, assuming that the analogies to the brain are adequately accurate, a view advanced by several researchers, including Paul Churchland in his book *The Engine of Reason, the Seat of the Soul* [7].

3.3 Related research

3.3.1 Introduction

This section will serve mostly to summarize and explicate what is already mentioned in the chapter on ”Background and Motivation”, (Chap. 2), and chapters in the results part, chapters 5, 6 and 7.

3.3.2 General overview of relevant research

There is a lot of related and relevant research on language, both past and present. When considering such research independent of methodology and the implementational level, ideas and results from several branches apply. These include the theoretical-philosophical work by Noam Chomsky, for example his generative grammar and the Chomsky hierarchy[5, 4], computational linguistics in the traditional approach[18] on the symbolic level, as well as general research on language processing using the connectionist approach.

While ideas and research from the diverse sources just mentioned are relevant to my work in varying degrees, it is first and foremost research on the sub-symbolic level that is being used as a source of strategies and models. In fact, my implementational work is very much based on three distinct and relatively recent systems, covering the three main areas of interest: a syntax parser[8], a perceptual grounding system[3] and a communication evolution model[29, 30]. Since most these are rather thoroughly described in each respective chapter, namely chapters 5, 6 and 7, only a listing of them is included here.

3.3.3 Related research within Connectionism

Parsing systems

- Bart Selman’s masters thesis *Rule-Based Processing in a Connectionist System for Natural Language Understanding* [26].
- The top-down parsing algorithm called the *Early-parser* [9] for use with context-free-grammar [14].
- Jeffrey Elman’s simple recurrent network [11] and his work on the design as described in his article *Finding Structure in time* [11].

- Work by Noel & Amanda Sharkey in their paper *A Modular Design For Connectionist Parsing* [8].
- Jordan Pollack's work to enable representation of structure in neural networks, in the paper *Recursive Distributed Representations* [23] where the RAAM model was designed.

Symbol-grounding and Neuroscience

- Work by Stevan Harnad on symbol grounding. He has done a lot of research into this topic, and his paper *Symbol Grounding is an Empirical Problem: Neural Nets are just a Candidate Component* [12] was used as part of the theoretical background.
- Research by Angelo Cangelosi into perceptual symbol grounding, particularly the article *Approaches to Grounding Symbols in Perceptual and Sensorimotor Categories* [3].
- Research on symbol grounding by Vogt and Ziemke in *The physical symbol grounding problem* [22].
- Bredeche, Zhongzhi and Zucker in the technical report *Perceptual Learning and Abstraction in Machine Learning* [19].
- Coradeschi and Saffiotti in their work described in the article *Anchoring symbolic object descriptions to sensor data* [25].
- Sergei Nirenburg and Victor Raskin in their recent book *Ontological Semantics* [20], where they give an in-depth treatment of semantics.
- The book *Semantic Information Processing* [10], compiled and edited by Marvin Minsky. Although old, it contains a collection of articles on the topic of meaning as well as various approaches to semantic information processing.
- The book *The Engine of Reason, the Seat of the Soul* where neural networks are explored specifically as a tool for explaining our biological brains, including cognition, emotions and consciousness[7].
- *The book Connectionist Models in Cognitive Psychology.* This is actually a collection of up-to-date contributions of renowned researchers, in

fields of AI, neuroscience, language development, cognitive psychology, etc. [13].

Chapter 4

Methodology of Research

By three methods we may learn wisdom: First, by reflection, which is noblest; Second, by imitation, which is easiest; and third by experience, which is the bitterest.

- Confucius

4.1 The Methodological Approach

The methods of research used in this thesis are a combination of the model- and experimental based, resulting in a hybrid approach. A general framework for experimenting with various implementations was designed in the modeling phase. In particular, three systems was implemented within the paradigm of connectionism.

The implementation of the three experimental systems was done in part to test empirically the abilities of neural networks in the area of NLP, and in part to gain deeper understanding of how language is processed and handled in the human brain.

4.2 Result - Framework and Models

A general framework for testing neural networks was implemented. As part of this framework a set of modules was created for handling three distinct modes of operation.

- The FNN model, for handling otherwise plain pattern transformation and recognition.
- The SRN model, for handling data having a crucial serial and/or temporal aspect.
- The RAAM model, for handling data with hierarchial structure

The choice of what systems to implement was strategically guided by the wish to cover distinct, but related areas within computational language research. Specifically, the parsing system by Sharkey [8] and the perceptual grounding system by Cangelosi [3] were selected partly because of the interesting possibilities that may be achieved by coupling NLP (parser) with NLU (perceptual grounding).

The third experimental system that was implemented simulates the development of word-meaning bindings through co-evolution [29, 30]. Here, a simple model for communication evolution was designed to test the theory proposed by Wang.

Approach and Results

Hell, there are no rules here - we're trying to accomplish something.

- Thomas Edison

Chapter 5

A Connectionist Parser

Imitation is the sincerest of flattery.

- Charles Caleb Colton

5.1 Introduction

This chapter will try to give a detailed description of a modular parsing system, based on the work by Noel and Amanda Sharkey [8]. The main purpose for this re-implementation is threefold:

1. to further familiarize myself with important types of neural networks by hands-on training especially the SRN and RAAM.
2. to learn how several distinct networks can be combined to work together
3. to see to what extent the grammatical structure of sentences in natural language can be identified through a sub-symbolic approach like this when no syntactic information is given.

5.2 Dataflow

The parsing system is composed of three connectionist modules: A Simple Recurrent Network, known as the SRN, a Recursive Autoassociative Memory or RAAM for short, and a plain feedforward network (FNN) for mapping representations between the two. The black-box schematic in figure 5.1, should help give a preliminary overview of the dataflow in the system.

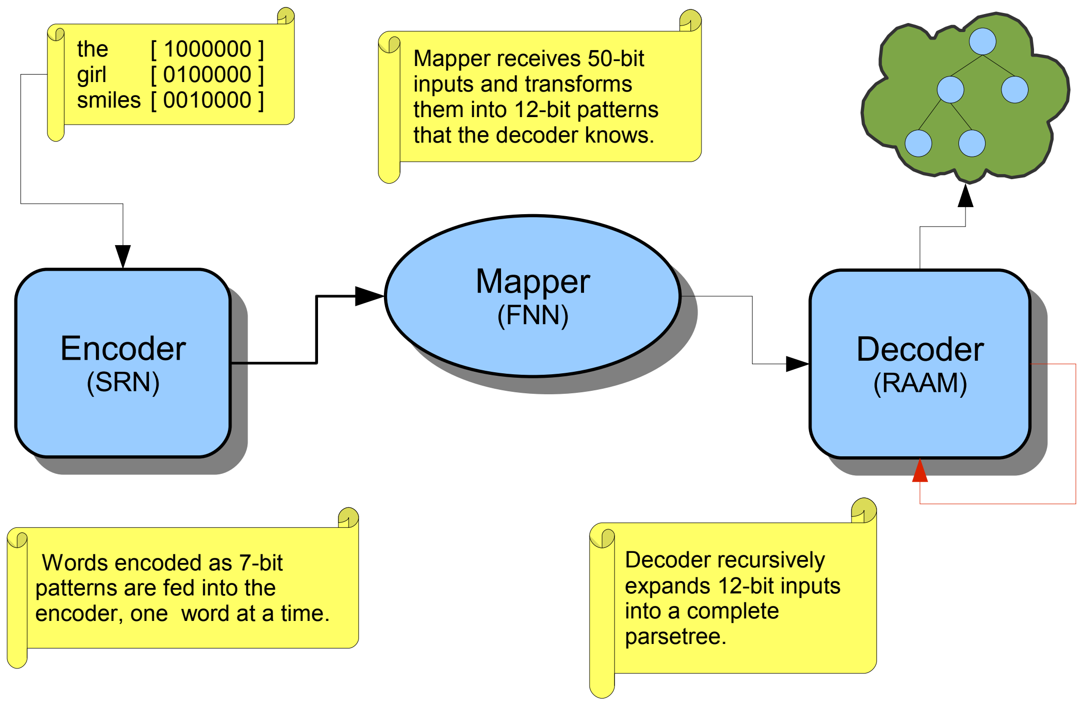

the [1000000]
girl [0100000]
smiles [0010000]
Mapper receives 50-bit inputs and transforms them into 12-bit patterns that the decoder knows.
Encoder (SRN)
Mapper (FNN)
Decoder (RAAM)
Words encoded as 7-bit patterns are fed into the encoder, one word at a time.
Decoder recursively expands 12-bit inputs into a complete parsetree.

Figure 5.1: Dataflow of the parsing system

In figure 5.1, the SRN frontends the system and serves as the entry point of external input, here in the form of words. An SRN used recurrent connections to form compressed representations [11]. A sentence is presented to the SRN one word at a time and in order. After the presentation of the last word in a sentence, a compressed representation of the entire sentence is generated. The RAAM is the backend of the system and serves to decode compressed representations into their constituents, that is, part-of-speech (POS) elements. A compressed version of an input-sentence like [the girl smiles] is what we get when that input has been processed by both the SRN and the FNN, now transformed into a 12-bit pattern. It contains information about both the syntactic structure and which POS elements the sentence is made up of. An intermediate component, the Mapper, is located between the encoder and decoder. Its function is to map sentence-representations from encoder into patterns the decoder has been trained to recognize. These patterns are the parse-trees made up of POS elements.

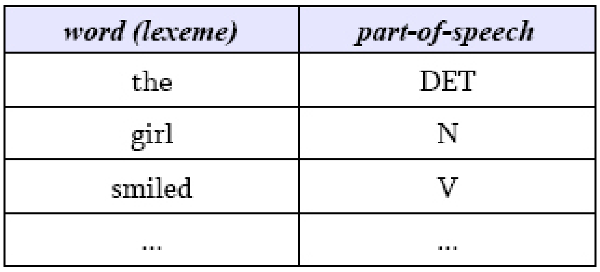
Each module first had to be separately trained to enable each module to learn its own functionality, and also to enable all three modules to work together as an integrated system for parsing. The SRN was trained to recognize grammatical structure in a set of natural language sentences. The RAAM was trained to enable decoding a compressed representation of a sentence into its POS constituents, along with structural information (syntactic structure). The FNN was trained to map between the two modules.

5.3 The Dataset

5.3.1 Introduction

A parsing system firstly should know and represent the grammatical structures that it is expected to parse the natural language sentences into. It also should have a way of representing the POS types that occur in these grammatical structures.

On the other hand, to test the actual performance of the parsing system, a set of natural language sentences are presented to the parser. These sentences are constructed according to the grammatical structures that the parser knows. A dictionary which is kept separately holds the mapping from the words in the natural language sentences into their POS. The grammatical structures and the POS knowledge of the parser system as well as the NL sentences and the dictionary are named as the dataset of the parsing system.



word (lexeme)	*part-of-speech*
the	DET
girl	N
smiled	V
...	...

Table 5.1: Dictionary structure

5.3.2 Dataset content

The dataset for the system is:

1. A dictionary of 70 words, consisting of 31 nouns, 26 verbs, 2 auxiliary verbs 6 prepositions, 5 adverbs, and 1 relative pronoun. Words in dictionary are tagged with their respective POS class. The structure of the dictionary is given in the table 5.1.

2. A representation of POS types, which are determiner (DET), noun (N), verb (V), axillary verb (AUX), preposition (P), adverb (ADV) and relative pronoun (RELPN).

3. The 8 syntactic structures the system should learn to recognize and parse, which is also the training-data for the RAAM. The eight structures are given in table 5.2.

- 1. (DET N) (RELPN V) (ADV V (DET N))
- 2. (DET N) (RELPN V ADV) (V (DET N))
- 3. (DET N) (V (DET N) (RELPN (DET N) (V (DET N))))
- 4. (DET N (V (DET N (RELPN (DET N) V)) (P (DET N))))
- 5. (DET N) (V (DET N)) (ADV (DET N) (V (DET N)))
- 6. (DET N (V (DET N (AUX V))))
- 7. (DET N (V (DET N (P DET N))))
- 8. (DET N (V (DET N) (P DET N)))

Table 5.2: The 8 syntactic structures

4. A set of generated natural language sentences, each of which has the format of one of the predefined 8 structures. These sentences are the training and the test-data for the SRN, the encoder.

5.3.3 The inputdata

The input-data consisted of sentences generated from the eight structures given above. This was done by substituting a POS element by a random word from the dictionary having that particular POS category. One example sentence from each of the eight categories are given in table 5.3.

On the other hand, sometimes, the intention of the speaker makes the difference in how a sentence should be parsed. That is, although any given two sentences might have exactly the same POS sequence, they might have two different syntactic structures.

1. (the boy) (who shouted) (greedily took (the sausages))
2. (the boy) (who laughed loudly) (ate (the cake))
3. (the boy) (warned (the woman) (that (the dog) (disliked (the delinquent))))
4. (the hiker) (warned (the woman) (that (the boy) disliked)) (about (the boat))))
5. (the policeman) (caught (the thief)) (before (the teacher) (rang (the boy)))
6. (the mountain-rescuer (remembered (the woman) (had been warned))))
7. (the policeman (chased (the delinquent (with the limp))))
8. (the policeman (hit (the delinquent) (with the stick)))

Table 5.3: Examples of input-sentences

For example the parser should be able to capture intentions of the speaker in the following two sentences and parse them differently.

1. (the boy) (who shouted) (greedily took (the sausages))
2. (the boy) (who laughed loudly) (ate (the cake))

In the above sentences the POS structures are exactly the same, whereas their two different intended syntactic structures are given through braces. The intentions are different because the adverb greedily bind to namely the verb took in the first sentence, but the adverb loudly binds to its preceding verb laughed in the second sentence. The same thing applies to the pairs of 3 and 4, 5 and 6, and 7 and 8 in the given list of natural language sentences above. It is these eight hand-coded structures that serves as the blueprint for what kind of parse-tree a sentence should form. To see this more clearly, look at figure 5.2, where the parse-trees for sentence 1 and 2 are given.

5.3.4 The encoding scheme

The input to the system is sentences, composed of a variable number of words. As we have 70 words in the dictionary, we need to chose a way of representing each word uniquely as inputs, and preferably using the same number of bits for each word. One way is to use the one-in-x bit scheme, resulting in a 70-bit pattern having exactly 1 bit set with that ON-bit designating which word number the pattern represents. This approach was used in the work by Sharkey [8]. A potential disadvantage for this scheme is that is requires a lot of input units, and all units except one are zero.

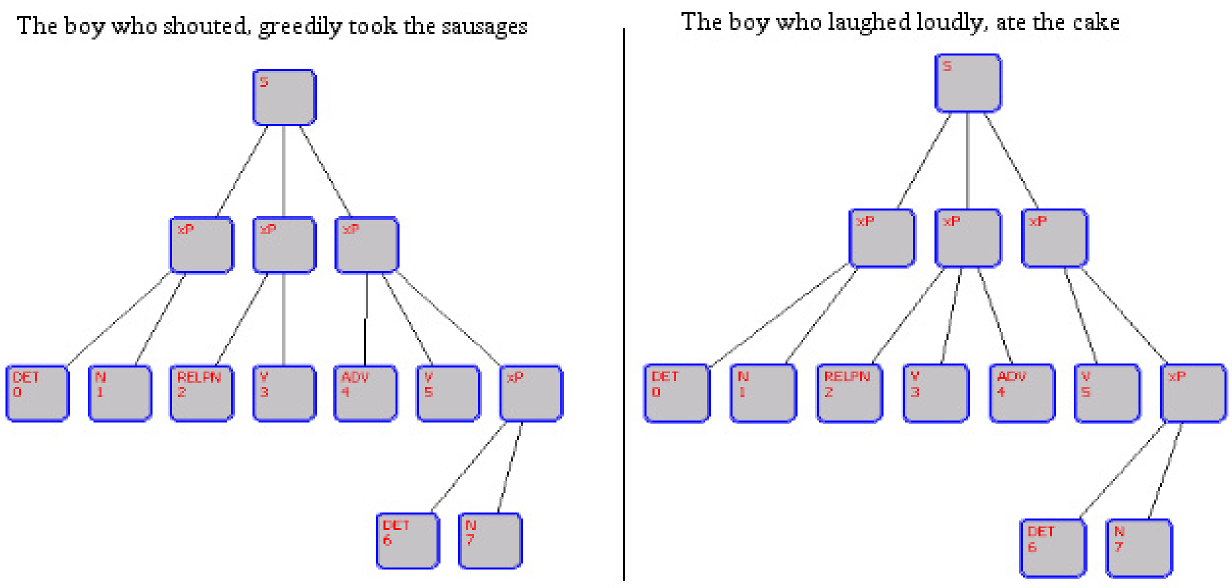

The boy who shouted, greedily took the sausages
S
xP
xP
xP
DET 0
N 1
RELPN 2
V 3
ADV 4
V 5
xP
DET 6
N 7
The boy who laughed loudly, ate the cake
S
xP
xP
xP
DET 0
N 1
RELPN 2
V 3
ADV 4
V 5
xP
DET 6
N 7

Figure 5.2: Example parse-trees

Another way to encode the same information (word number) into patterns of equal length is to simply use the binary representation of the word number. E.g. word #7 in the dictionary would be represented as 111. This is the approach chosen in my re-implementation. The number of input units required is then derived from the number of bits required to represent the size of the dictionary in this case 70 and can be represented by 7 bits. (scales up to handle 128 different words).

This allows for a significant reduction of input units required for representation. However, this reduction of the number of input units can cause problems. That is, more work will be demanded from fewer number of weights in the network, which is a result of fewer number of input units.

In turn, this may impair the ability of the net to learn, if the complexity of a task is too high relative to the number of weights. An analogy may help clarify this. Imagine having 5000 glass beads to use for representing a complex shape, and assume that this number is adequate for creating a good approximation of that shape. Now, trying to accomplish the same task using only 100 beads will most certainly result in a very crude approximation, not showing much of the details of the original shape. Another effect of significantly reducing the number of input units is specific to the SRN. By having 7 fresh inputs and 50 feedback units from the hidden-layer as additional inputs, the ratio of new number of inputs to the old number of inputs becomes rather small, and the inputs tend to be dominated by the feedback.

The encoding of POS elements is done using the one-in-x-bits scheme though, and this applies to the RAAM module, both in the form of inputs and outputs. As an example, POS encoding into 12-bit patterns looks like this:

determiner = 100000000000 noun = 010000000000

5.4 The Encoder - SRN

5.4.1 Introduction

This model was developed by Jeffrey Elman [11] to test the ability of neural networks to handle temporally and serially ordered input data.

5.4.2 Topology

The topology for this recurrent net has the usual layer characteristics; three layers comprised of one input layer, one hidden layer and one output layer; 57:50:70. The number of input units is 57 and is composed of 50 feedback units from the hidden-layer, and 7 for the number of bits required to represent a word in the chosen encoding scheme.

As hinted at already, the hidden layer has 50 units, exactly as used by Sharkey [8]. The size of this layer is guided by the complexity and type of task at hand, and indirectly by number of units in the other layers. The number 50 does not correspond directly to any external factor, as the dimensions of both the input- and output layers do, but is chosen to achieve a sufficient capacity of the net. The output layer has 70 units, 1 for each word in dictionary. The reason for this will become clear when the learning task for the SRN is described.

In the case of this system, using only 7 inputs (along with the 50 feedback units) did not seem to prevent the net from learning its task. In order to avoid having the feedback totally dominating the input, I used a decay factor for the context units, which is 0.4, and it worked well with the current dataset and learning task, symbolized in the figure 5.3, by the ellipse at the lower left.

5.4.3 The Learning Task

In the training process the net was presented with one word a time, represented by a 7 bit pattern. Also included with each input cycle was the hidden layer representation of input from the previous cycle; the so-called activation pattern which was multiplied by the aforementioned decay factor of 0.4. For each such input, the training target is to predict which words may follow the

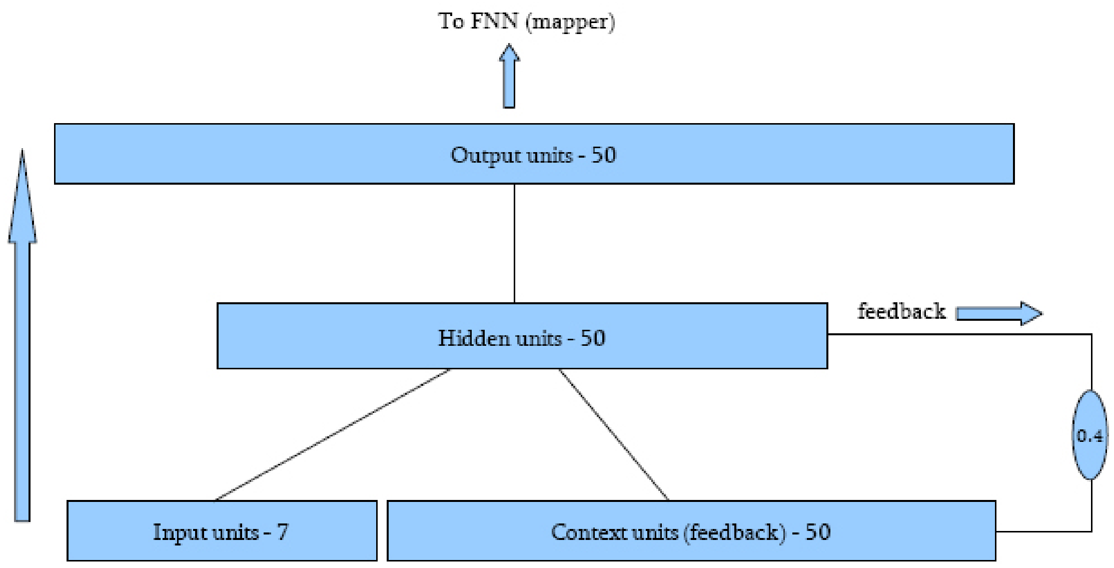


Figure 5.3: The ENCODER (SRN)

current input sequence. This is achieved by letting the output nodes represent a word number, such that output unit 1 corresponds to word number 1 in the dictionary, and so on. Thus, for any given input-word all words being a legal successor should have all their corresponding outputs turned ON, and giving an output ranging from approximately [0.7, 1.0]. An example using the first of the syntactic structures from section 5.3.2 should make this clear.

(DET N)	(RELPN V)	(ADV V	(DET N))
(the boy)	(who shouted)	(greedily took	(the sausages))

If the input sentence being processed is [the boy who shouted greedily took the sausages], and the current input-sequence is [the boy who], we see that the following POS is a verb. Thus, the legal successors for who are all verbs in the dictionary.

After the last word of a sentence has been given, we have a compressed representation of the entire sentence. This will eventually be the output given to the next module, the mapper, after training has been successfully accomplished. Note that this representation is the hidden layer activations. The activations of the output layer at the end of a sentence only represent the grammatically legal successors for the last word in that sentence, and that should obviously not be used as a representation of the sentence.

5.5 The Decoder - RAAM

5.5.1 Introduction

One important first note is that, unlike the SRN which works with real words as input, the RAAM [23] works at a higher level of abstraction: With POS elements. Thus, leaf-nodes in a parse tree does not represent words.

5.5.2 Topology

As with the encoder, this net also has three layers. What is special about RAAM architecture is the **nk : k : nk** dimension of the layers. In this implementation the n is 3 and the k is 12, resulting in a **36:12:36** network.

While n defines the number of partitions that the input and output layers will have, it can also be seen as a compression ratio, such that 36 bits are represented by 12 bits in the hidden layer. The function of RAAM, which is an autoassociator, can actually be described as training the net to reproduce the input at the output layer. The reproduction of the inputs at the output layer explains why the input and output layers must be of same size. In effect the RAAM acts much as a lossy compression algorithm, like jpeg image-compression. The RAAM is actually both an encoder and a decoder, where the first layer and the hidden layer makes up the encoder, while the hidden layer and the output layer comprises the decoder.

The example inputs shown in figure 5.4 is the typical case where the input-pattern is a concatenation of two POS-elements, a determiner and a noun. Note that in this situation partition 3 is left unused, and that this happens more often than not while encoding a parse-tree. Relevant issues are discussed in section 5.5.4.

5.5.3 The Learning Task

As briefly mentioned, the basic training target for the RAAM module is an accurate as possible replica of the input. The input data is the 8 syntactic structures given in the introduction, where structure can be represented in a hierarchical structure of a parse-tree and each leaf-node is a 12 bit pattern

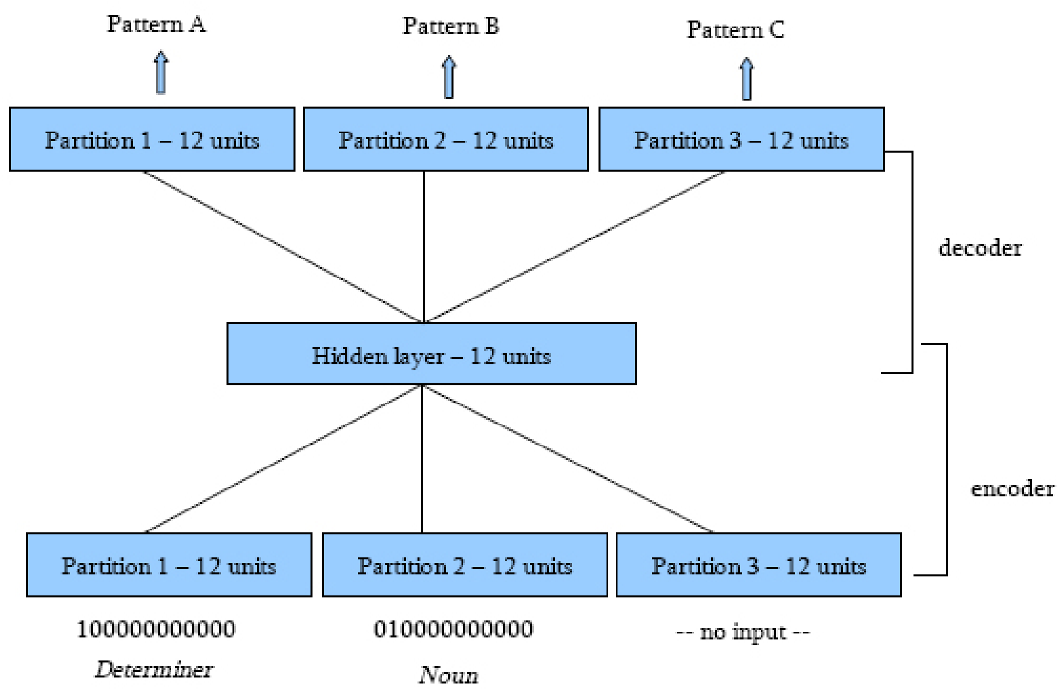


Figure 5.4: The decoder (RAAM)
Note that this example-input leaves the last partition unused.

representing a POS element. The structure itself is realized using objects in the employed programming language, Java. Such an object based realization makes the encoding / training process much easier. However, there is more to the encoding process than merely a few separate autoassociations. Firstly, to achieve the global training goal we must end up with only one pattern for representing the entire tree. Secondly, we must be able to reconstruct that tree by using only that pattern. In order to do this, we need to recursively encode (by autoassociation) successively larger subtrees ultimately reaching the stage where we end up with one pattern, which is the root- pattern as symbolized by the node at the top labeled S. Training starts with patterns at the bottom of the tree, and recursively progresses upwards as patterns for a parent-node is acquired.

To clarify this, see nodes 6 and 7 in figure 5.5. They are both leaf-nodes, each of which is represented by a 12 bit pattern: a determiner and a noun. They form the two children of a subtree. At the very beginning of the training process only leaf-nodes have a defined pattern, and consequently, repre-

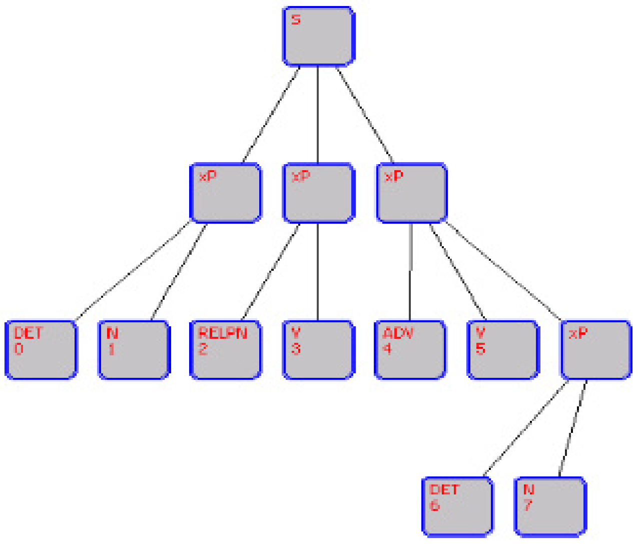


Figure 5.5: Example parse-tree

sentations for parent nodes must be generated during training, where these patterns are acquired by autoassociating their children. In this example, the DET(6) and N(7) are given as one concatenated input to the RAAM, and the hidden layer activation is stored as the pattern for their parent, which is the xP node immediately above them. This is repeated for all 5 subtrees in this example.

Autoassociation of patterns after only one iteration cannot be achieved, since quite many weight updates are necessary. A particular difficulty occurring in this learning task is that some of the training targets become moving targets, that is, they change as training proceeds. This does not affect leaf-nodes as they have explicitly predefined patterns that stem from the POS they represent. The parent nodes, on the other hand, continuously change as patterns at lower levels in the tree slowly approach autoassociation. Training a network with several different inputs that cause opposing demands on weight changes is common to all networks. However, in the case of the RAAM we also have interdependency between intermediate inputs and outputs during the encoding process. This results naturally from the fact that we are working with hierarchically structured input and training targets. There are two easy ways of reducing this problem somewhat, both of which translates into two distinct implementational details:

1. By using a very low learning rate, 0.15 to 0.001, which helps prevent weight updates for the different patterns to completely ruin each other's progress.

2. By observing that patterns (nodes) higher in the tree receives their training targets from all connected nodes below them. Therefore, it should be helpful to pre-structure the net by autoassociating all nodes with known, constant patterns; the leaf nodes. Thus we pre-train on patterns in subtrees in which all children are leaf-nodes. In figure 5 this amounts to nodes 0 and 1, 2 and 3 and 6 and 7. The exact amount of training to do for this kind of pre-structuring should be adapted to the specific dataset at hand, but for this system, a success rate of approximately 80%-90% of adequate autoassociation was used.

The end result of training the RAAM on the 8 syntactic structures gives us a set of eight 12-bit patterns representing them. It is these patterns that will be the training targets for the mapper module.

5.5.4 Valency Issues

The key concept in this section, variable valency, means that the number of children a node can have is not a predefined constant, but rather varies from 0 to some chosen maximum. The word valency can be equated with the number of children of a given node. In the parsing system implemented here the valency ranges from 0 to a maximum of 3. An example of the opposite, constant valency, is found in binary trees, where all nodes has either exactly two children, or none at all.

As mentioned briefly at the end of section 5.5.2, most subtrees in the set of 8 syntactic structures only have two children. Since our RAAM has three partitions, the last twelve units are rarely used. The fact that some input partitions do not receive input during the encoding process has consequences for weight updates, and also for the design of the decoding cycle. First we will look at this from the training side of things.

If we ignore unused partitions, in which the units consequently have zero input, a value of 0.5[1] will be propagated over weighted connections from these units, and significantly affect the input of all units in the hidden layer and onward. To avoid such noise I modified the input-routine to simply not propagate values units in unused partitions. In order to have any real effect though, a slight modification had to be done to the backpropagation algorithm also. This change consisted of making the error of output units in unused partitions zero, thus effectively preventing useless (and disruptive) weight updates.

The implication of a variable valency for the decoding process should be obvious: how many children does a particular node or pattern have? It can be either 0, 1, 2 or 3. Also, due to the modifications done to the encoding algorithm, another complication arises. The effect of ignoring unused partitions during training is that we cannot know whether a partition on the output layer actually represents a learned non-leaf pattern, or simply is undefined activations. This does not apply to leaf-nodes as they have a known, characteristic pattern of one 1.0 and the rest are 0.0. A workaround for this is to use the already generated parse trees to guide the decoding process. This will be explained in section 5.5.5.

5.5.5 Decoding

For the decoding process, we do not need the input-layer and its associated weights, but rather only the hidden- and output-layer appropriately called the decoder part. For testing purposes, we use as input those eight 12-bit patterns that the training process yielded. Note that for actual testing of the system as a whole, the input will come from the mapper module, and those patterns will be close approximations of the ones generated during the RAAM training.

Now, about the decoding procedure: Input data for the decoding part are not presented to the input units, but rather directly to the hidden layer. It is also important to remember that the input comes from the activations at the hidden units, and therefore it should not be run through the transfer function again. Rather, the input is propagated to the output-layer through an alternate method, an identity function that does not change the input.

[1]This is a result of the transfer function used, the standard sigmoid function $\frac{1}{(1+(exp(-input)))}$, which will output the value 0.5 when the input is 0.0, or *nothing.*

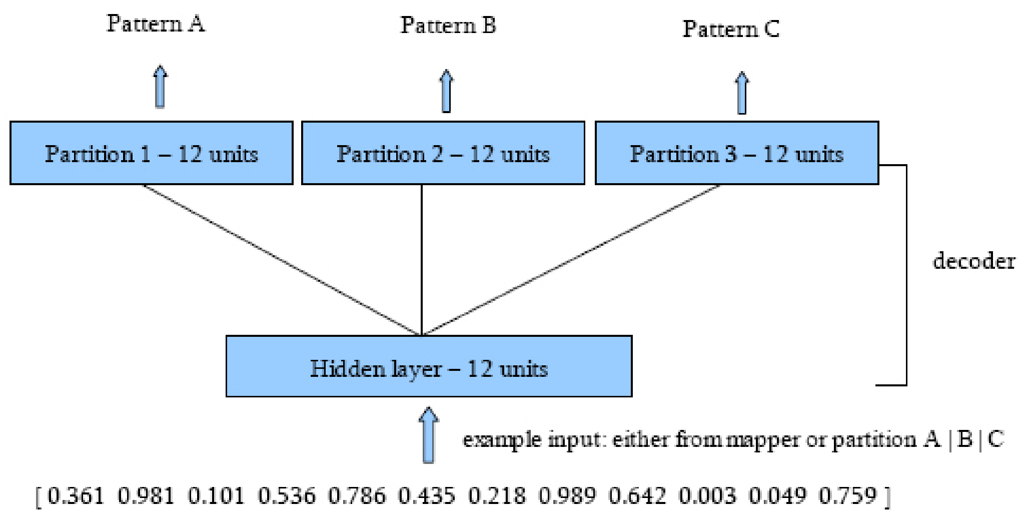


Figure 5.6: RAAM, decoder part

(Obviously the weighted connections are still applied).

The output units represent whatever constituents were contained in the given input pattern, and this is where the problem of the unknown valency enters the picture. As the chosen encoding scheme makes it impossible to distinguish between an empty partition and one containing a composite representation, (as all non-leaf nodes do), a solution to the unknown valency had to be found. It was solved by using the parse-trees for the syntactic structures as guide during decoding. But, this begs the question of which tree?.

Actually, any input given to the RAAM for decoding is recursively decomposed using all of these 8 parse-trees, one at a time. This way we know which partitions to decode further, and consequently when to stop decoding. During each of these eight decoding attempts, an error value is computed by comparing the resulting pattern of each decoded leaf-node with the corresponding leaf-nodes in the parse-tree we are attempting to decode it as. The idea here is to find/guess the correct parse for an input by selecting the parse-tree that yielded the lowest error.

An obvious implication of this is that the decoder will always identify an input as one of the 8 structures it has been trained to recognize, no matter what the input looks like. This makes the parsing system less flexible with respect to which syntactic structures it can identify. However, the purpose of the system was not to enable partial recognition of unknown syntactic variations, and hence, this is an acceptable limitation. The number of structures recognized can easily be extended by adding new ones to the dataset, along with the required scaling up of the networks.

Recall though, that as the decoding process is recursive, either pattern A, B or C is fed back into the hidden layer, depending on which pattern we want to decode further. Also, to further clarify the problem of the unknown valency, look at the example input. This is what a hidden layer representation may look like, but also very similar to what would appear in an empty, or unused partition. Imagine now, that after giving the example input as specified in the figure, patterns A, B, C takes on the following values:

A: [0.007 0.062 0.009 0.943 0.061 0.035 0.018 0.089 0.042 0.013 0.049 0.056]
B: [0.921 0.081 0.001 0.506 0.086 0.035 0.018 0.013 0.004 0.003 0.029 0.099]
C: [0.263 0.462 0.309 0.233 0.761 0.735 0.518 0.189 0.342 0.013 0.349 0.456]

By knowing what properly encoded leaf-node patterns look like (all zero's except one), patterns A and B are confidently identified as POS elements and should not be decoded any further! This is not the case with pattern C, which may either be the undefined output for an empty partition (a subtree with only two children) or an actual representation of another subtree. Attempting to decode patterns containing no encoded structure leads to undefined outputs, and a potentially neverending recursion. In any case it pretty much ruins the chance for a successful decoding of a parse-tree.

5.6 The Mapper - FNN

5.6.1 Introduction

The function of the mapper may not seem very important or interesting, compared to the other modules, but without it the other two modules would not be able to communicate. So, even though its function is to simply map, or convert, one pattern into another, that function is an essential one. Also keep in mind that there is a lot of information is packed into both the input-data and the output-data associated with this mapper.

5.6.2 Topology

This is the most straightforward module in the system. Being a plain feed-forward network with three layers, the topology is simple: **50:35:12**. The number of inputs is determined by the size of the hidden layer in the SRN module from which it receives its input. The 35 hidden units is chosen as an appropriate compression factor that is sufficient to finally map the 50-bit input into 12 bit patterns that closely approximates those generated by the RAAM. Thus, the output-layer has 12 units, and is determined by the size of the hidden-layer of the RAAM, which is where the output of the mapper is routed.

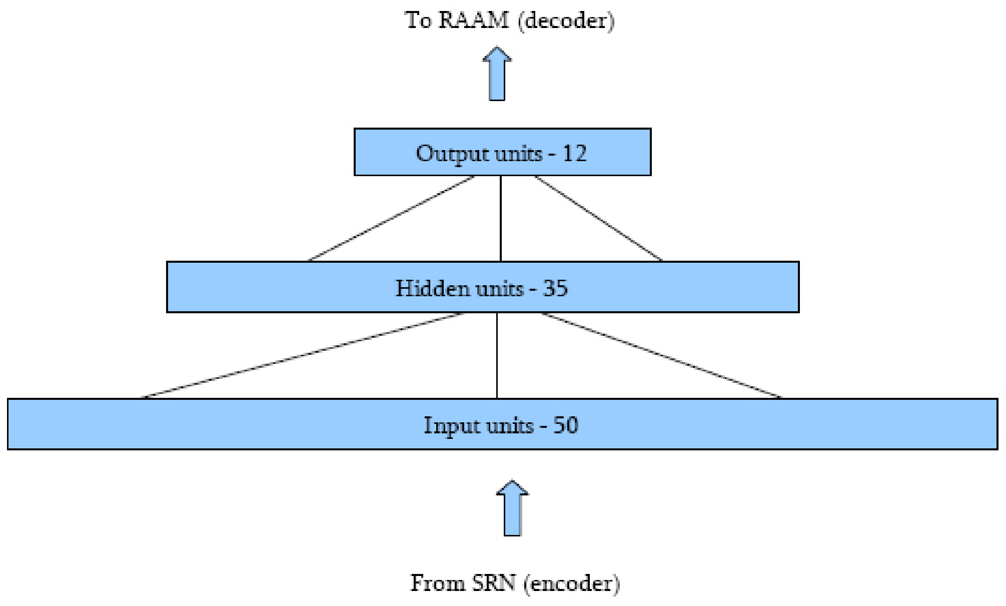


Figure 5.7: The Mapper network (FNN)

5.6.3 Training

The training of the mapper is dependent on results from both the encoder and decoder modules, and therefore it can only be trained after the others have finished. The input given to the mapper is the patterns from the hidden layer of the SRN, which contains certain rules that have been extracted from the input data by teaching the SRN to predict legal successors (in terms of words) given the current sequence of POS elements. These 50-bit inputs must to be transformed into 12-bit approximations of the patterns the decoder has learned to use as representations for the parse-trees and these patterns are the training targets.

Chapter 6

Perceptual Symbol Grounding

All our knowledge has its origin in our perceptions.

`- Leonardo da Vinci`

6.1 Introduction

Earlier, we have argued for the need for a stronger semantic component in the analysis and interpretation of natural language. Semantic information processing [10, 20] can help improve both the syntactic parsing process as well as the interpretation of the intended meaning. There is much diverse literature on semantics and several researchers have approached this subject. Various levels of ambition and numerous methods have been attempted. The model described in this chapter is based on work done by Angelo Cangelosi, as described in a recent paper *Approaches to Grounding Symbols in Perceptual and Sensorimotor Categories* [3].

The level of ambition for my implementation was set relatively low compared to the framework discussed in the paper by Cangelosi. One reason for this was the purpose of writing an experimental system on perceptual grounding, which can be summarized as follows: To test the theory empirically, and to gain more insight and practical experience in the field of perceptual symbol grounding.

6.2 The Training Parameters

An adaptive learning rate and momentum was implemented for this system, and therefore these parameters will not be specified in the sections concerning the training process. It will have to suffice to state here that an initial learning rate of 0.2 was used, and the momentum was set to start at 0.65.

6.3 The Scene

The scene, or field-of-view (fov), is the place in which everything visual input is derived from. It is a very simple construct that can represent the field of view that is covered by the eyes (or an eye). This is from whence the system receives its visual input that are to be analyzed, recognized, and "understood", the process by which a word can be perceptually grounded. The percept is visual.

The scene first and foremost has two properties. One is its size as width and height. In the test runs, the scene was set to be 192 pixels wide and 192 pixels high. That particular number was derived from the decision to allow room for 6 shapes in both directions, and each shape being fixed at 32x32 pixels. 32 * 6 = 192. Another property that can be attributed to scene is the spatial granularity, or intrinsic resolution, which determines the number of positions that exist in the fov-world. This, then directly determines the number of positions a shape can be located at, and thereby also the number of patterns that the location-network can be trained on. A granularity of 32 was chosen, not coincidentally exactly the size of a shape, and this greatly simplifies the process of finding shapes. Other, more implicit properties of the scene are the fact that it is purely two-dimensional and black-and-white. Another practical effect of the coarse spatial granularity was that the job of the scene-analyzer could be greatly simplified, and will be explained in section 6.11.

6.4 The Virtual Data-routing Algorithm

A problem that arises from having only *one* word-to-concept mapper, section 6.10, and *two* concept networks, was decided to be ignorable. It just happens to be a problem of the routing of activation-data from the hidden-layer of the word-to-concept mapper to either the shape net or the location net, and is not relevant to the symbol grounding experiment. This current state of affairs appeared after a revision of the design of the symbol grounding system. So, for the purposes of this system, the routing problem will be considered solved by a virtual routing algorithm.

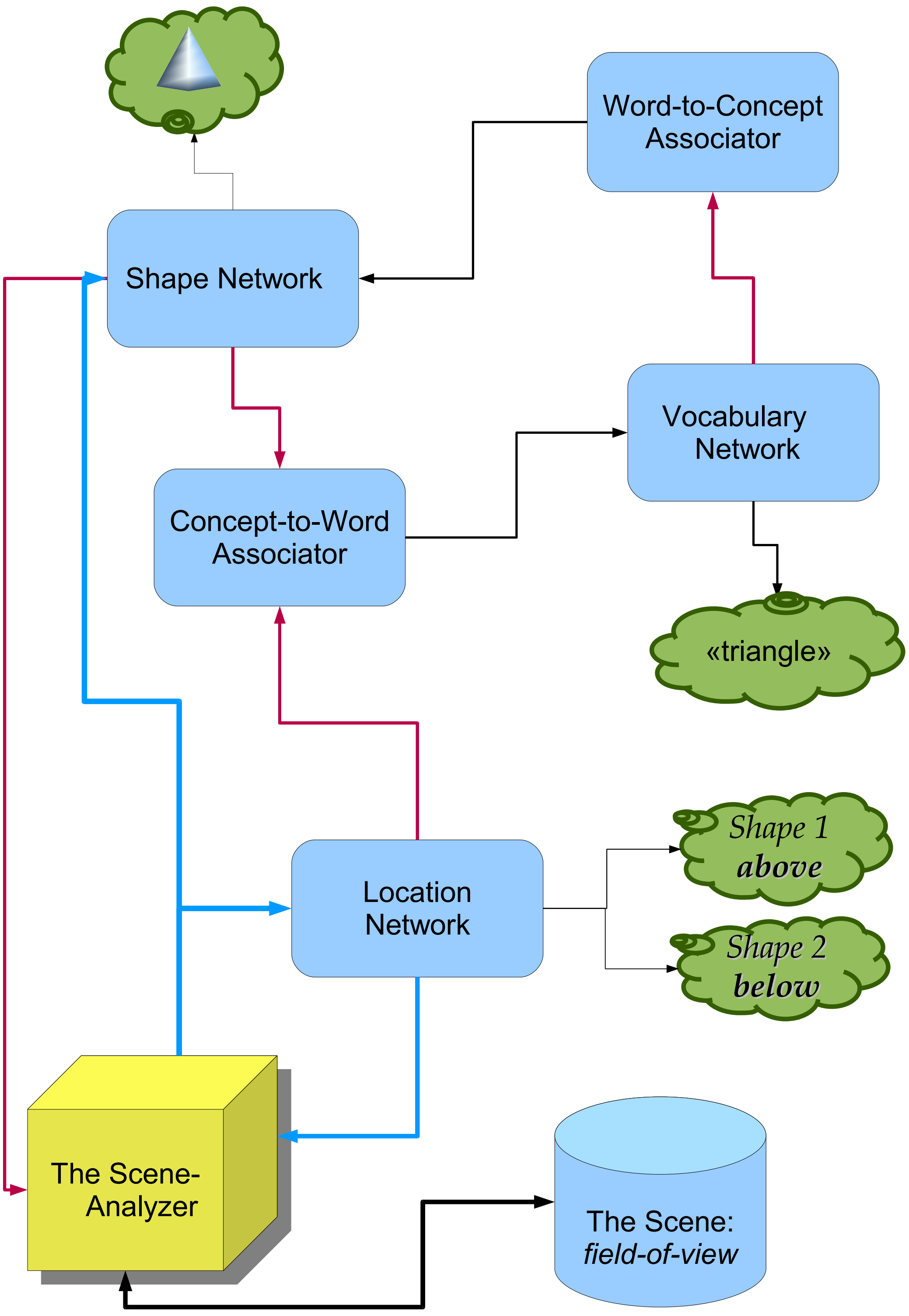


Figure 6.1: Dataflow diagram of the Symbol Grounding System

6.5 Basic Dataflow and the Components

To give an initial overview of the system and its components, the basic dataflow of the system is shown in figure 6.1. All the main components are also listed here, briefly describing their function.

The system is comprised of five neural networks and one hybrid module, each serving a specific role. The word recognizer, section 6.6, provides the functionality of a dictionary, i.e. the system's vocabulary. The shape recognizer, section 6.7, provides vision processing and grounding of visual perceptions. The location recognizer, section 6.8, provides a capability of learning certain location concepts[1]. The concept-to-word mapper, section 6.9, serves the function of associating a 'concept' with a word. The word-to-concept mapper, section 6.10, serves the function of associating a word with a 'concept'. Finally, the scene-analyzer, section 6.11, provides a rudimentary functionality for identification and extraction of individual shapes from a visual input.

6.6 Word recognizer

6.6.1 Dataset

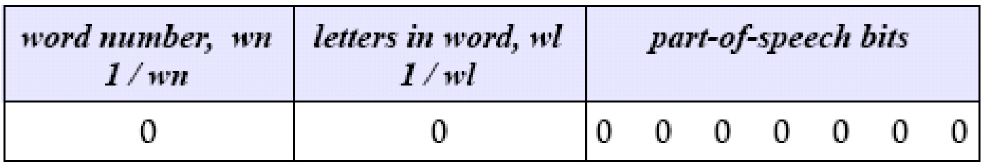
Words, or labels, for any object or concept are what makes up the dataset for this module. In the current setup, this comprises the names of four geometric shapes and five prepositions. The words are: *triangle, rectangle, hexagon, circle, above, below, left, right, on.* This resulted in a small dataset of only 9 patterns.

6.6.2 Encoding of input-data

Each word is encoded into a 9-bit pattern. The first two bits represent its unique id number and the number of letters in the word. The remaining 7 bits contain part-of-speech information. See table 6.1 for a detailed view of pattern structure.

This encoding scheme ensures a fixed-width pattern for all words, which is useful when the receiving network has an FNN topology. The choice to

[1]The location concepts in question are the binary and relative position-relations commonly known as prepositions.



word number, wn *1 / wn*	*letters in word, wl* *1 / wl*	*part-of-speech bits*						
0	0	0	0	0	0	0	0	0

Table 6.1: Structure of the bit-pattern of an encoded word

include some word-specific information in the encoding was made with respect to an important principle regarding transformation of data: as much as possible of any potentially relevant properties of the original data should be retained through a conversion. However, for the limited, experimental purpose of words in this system, the encoding could just as well have been using a plain numbering scheme.

6.6.3 Topology

The strategy chosen for presenting input is the parallel, all-at-once type, and consequently the choice of network model becomes the FNN. From the size of the input-patterns, the input-layer is set to have 9 input nodes. The size of the hidden-layer was set to 14, and this was decided in part empirically, in part by commonsense. The size of the output-layer was directly determined by number of words in the vocabulary, 9. Thus, resulting in a 9:14:9 FNN network.

6.6.4 Training

The training process for this network is very simple, and not particularly interesting. In fact, one could say the training is mostly for show, as all that is required of the dictionary network is to uniquely identify each word-pattern. Given the small number of words in the dictionary, each input-pattern is likely to give rise to sufficiently distinct output-patterns with little or no training at all.

However, a specific training target was chosen, and consequently a full training process was required. As already stated, the network was trained to uniquely identify each word, and a one-in-9 bit representation was selected as the target pattern for the output layer. That is, each shape was to be identified as a shape number, where the first would produce `1 0 0 0 0 0 0 0 0`, the second `0 1 0 0 0 0 0 0 0` etc., as activations at the output-layer.

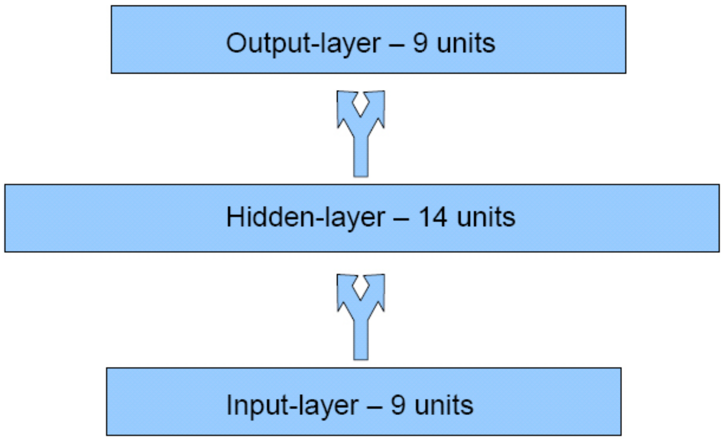


Figure 6.2: The Word-recognizer network

6.7 Shape recognizer

6.7.1 Dataset

The dataset for the shape recognizer consists of bitmap images of the four shapes: a triangle, a rectangle, a hexagon and a circle. To avoid any unnecessary complications, the shapes are purely two-dimensional, as is the scene. The dimensions of each image are 32x32 pixels, totalling 1024 pixels. To reduce the number of elements in a shape-pattern, an algorithm for extraction of shape-data only was implemented to pre-process the visual input. In addition to removing the ”background” pixels, all pixels inside the shapes was removed, leaving only the outline, or circumference, of each shape. This significantly reduced the number of pixels in each shape representation. Thus, the dataset was comprised of four shapes, resulting in 4 patterns.

6.7.2 Encoding of inputdata

On average, the shape bitmaps contain 105 pixels after the pre-processing described in the previous section. However, using RGB color values as training data is not a good idea for learning to recognize a shape. Instead, relative (x,y)-coordinates were used, the upper left coordinate in a shape being (0,0).

A fixed-width representation for shape-patterns was chosen, and therefore the width had to be set according to the maximum number of pixels occurring in all the shapes, 120. Thus, because each coordinate has two components, the patterns became 240 bits wide. (Obviously, each x and y value was normalized to values between 0.0 and 1.0).

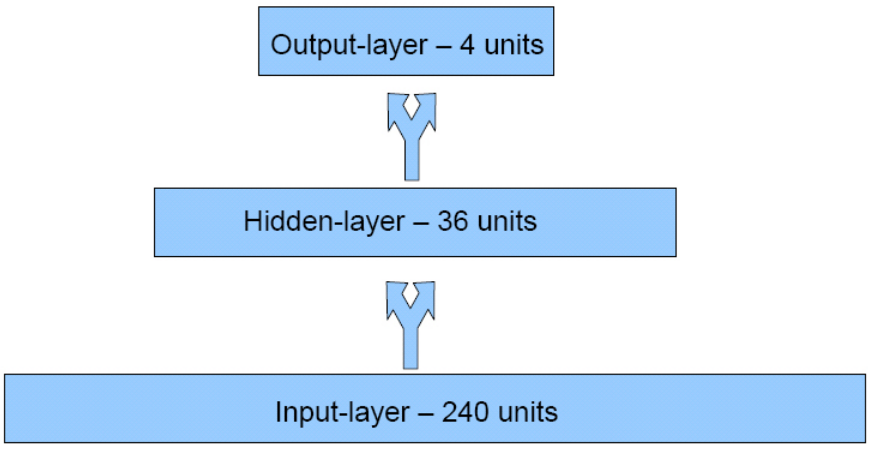


Figure 6.3: The Shape-recognizer network

6.7.3 Topology

A plain FNN model was used here too, which is why a fixed-width input-pattern was required. The input-layer therefore had 240 nodes. The size of the hidden-layer was set relatively high, to 36 nodes, because of the unusually large number of elements in the input-patterns. Finally, the output-layer had 4 nodes, one for each of the shapes. In other words, a 240:36:4 topology.

6.7.4 Training

Initially, the intention was to train the network to "count" the number of edges in a shape, by using a training target that would somehow represent the number of edges in a given shape. This quickly turned out to be quite tricky, for a number of reasons. First, what kind of training-target could be used to make the hidden-layer actually represent a count of the number of edges? Clearly, there is no obvious way to accomplish this directly from the current input, using only one net[2]. So, for the sake of simplicity and original purpose of the system, the original, more ambitious training scheme was discarded.

The training of the shape-net was therefore done in the simplest possible way, namely to train the net to identify each shape-pattern as a *shape number*. Again, the one-in-n bit representation was applied when forming the training targets.

[2]One cannot say definitively that such a representation may not form, but it's most likely not possible to deterministically produce it directly by means of a single training target and only one net.

6.8 Location recognizer

6.8.1 Dataset

The rather dynamic dataset for the location net consisted of coordinate-pairs of two shapes. The location concept used was that of prepositions, and they have two specific properties regarding the spatial relation they represent. One, they describe a binary relation. Two, the positional relation is relative. That is, the position of one object is described in terms of the other. Both of these aspects affects the topology and the training process, as we will see.

The exact number of patterns in the dataset is dependent on a number of things. An obvious factor being how many distinct positions a shape can have. This is further dependent on the size of the shapes, the size of the scene, and finally its granularity. The two latter parameters was set to 192x192 and 32, respectively. This computes to 6 possible positions along each of the two dimensions, the x and y axis, by the division 192 / 32. All of the possible location-patterns compute to 36, from 6x6. However, as we are dealing with relative positions between two objects, **each** can have 36 positions independent of the other, yielding a total number of 1296 permutations[3]. See section 6.3 for more details.

Although they are part of the training-targets, the preposition concepts should be included here as part of the dataset, because of the central role they have. Similar to the word-recognizer net, each of the prepositions[4] {above, below, left, right, on} was represented by a unique number, using the standard one-in-n bit encoding.

6.8.2 Encoding of inputdata

The encoding of the inputdata is straightforward. Two (x,y) pairs are merged directly into a 4-bit pattern, where the first two bits represent the location of the first shape, whereas the two next represent the location of the other. No surprises there.

6.8.3 Topology

Yet another FNN. The input-layer has 4 nodes, the hidden-layer needed to be enlarged to compensate for the small input-layer, and so 36 nodes was used here. Another important *practical* reason for having exactly 36 hidden units is because it should be of equal size as the shape-net's hidden-layer, and will be explained in section 6.9. The output-layer got 10 nodes, 5 for each of the two shape-locations

[3]This number is not corrected by removing duplicate or otherwise redundant positions.

[4]Note that "preposition" here refers to the concept of a relative position, not the word.

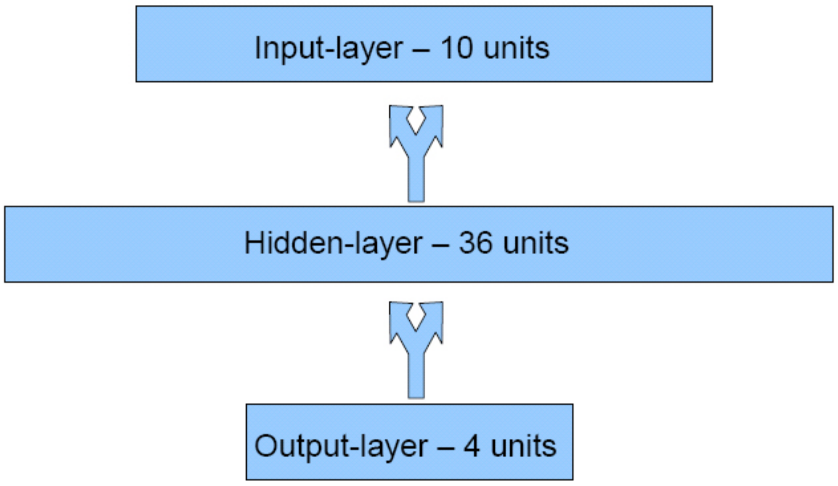


Figure 6.4: The Location-recognizer network

in question. This means that the first five output units represent the position of the first object relative to the second object, and the five last output units represent the position of second object relative to the first. In compact topology representation, 4:36:10.

6.8.4 Training

The training of the location net consisted of making it transform an input-pattern, (two coordinates), into the two corresponding location concepts (prepositions). As an example, consider the following situation: We have two shapes with locations (x=50,y=10) and (x=50,y=90), respectively. In such a case, one can clearly see that the first shape is *above* the second shape, and consequently the second shape is *below* the first. The net is supposed to identify both relations, thus producing on the output-layer a pattern representing these two related preposition concepts.

6.9 Concept $\rightsquigarrow$ Word associator

6.9.1 Introduction

This network has the important function of associating a concept to a name, a word in some language. In other words, bridging the gap between an intrinsically meaningless word and an internal concept, thus accomplishing some form or level of symbol grounding. That is, assuming the associator net is connected to some other network so that it can receive activations as they occur.

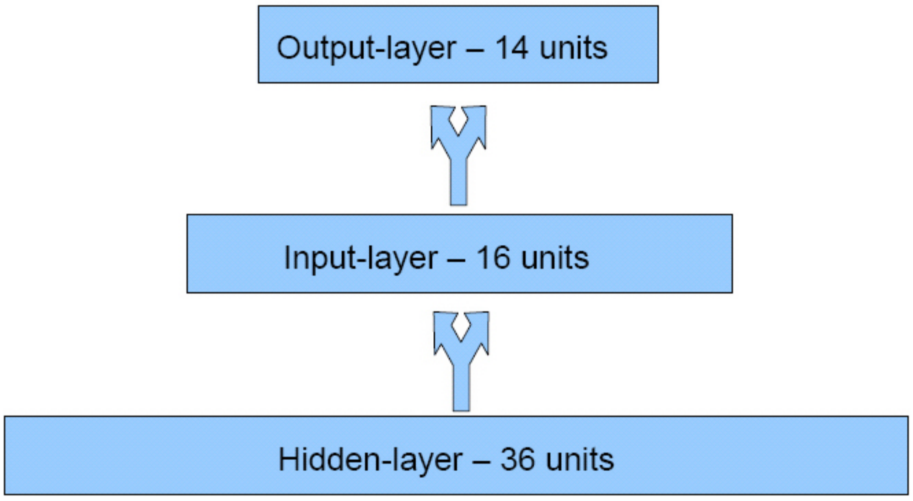


Figure 6.5: The Concept-to-Word associator network

6.9.2 Dataset

The dataset for this associator network was the hidden-layer representations for each of the learned concepts in the system, and these are the prepositional locations (5 pcs), and the shapes (4 pcs). These are all 36-bit patterns, a convenience resulting from having equally sized hidden-layers in the location-net and the shape-net. By having all concepts represented by patterns of the same size, only one associator network was required, instead of one for every such concept network. The total number of patterns in this dataset was 9, the sum of the prepositions and the shapes.

6.9.3 Topology

The topology of this, yet another, FNN network is as follows. The input-layer has 36 units, for reasons mentioned in the previous section, 6.9.2. The hidden-layer was given 16 units. Finally, the output-layer got its size from the hidden-layer of the word-recognizer net (section 6.6), which was 14 units. In the short format, 36:16:14.

The dependency between the size of its output-layer and the hidden-layer of the word-recognizer net is this: The output activations of the concept-to-word associator are sent directly into the hidden-layer of the word-recognizer net. Thus, causing the vocabulary network (language center) to produce the word associated with this sensory stimuli. A plausible, if not entirely correct, model of how the brain might accomplish the mapping between a recognized percept and the associated word.

6.9.4 Training

As already mentioned in the introduction, (section 6.9.1), the training consisted of learning to map activation patterns[5] originating from a concept-network, into activation patterns closely resembling those of the word-recognizer net. More specific, a hidden-layer activation from either the shape-net or the location-net is to be transformed into the hidden-layer activation that would have been produced by the word-recognizer if presented with the corresponding word.

To help clarify matters, consider the following example. The word-net, if presented to a known word, say '**triangle**', produces its internal representation of that word in the hidden-layer. Let's call this pattern `PAT-dst`. The shape-net, when presented to a known **triangle-shape**, produces some triangle-specific activation pattern in its hidden-layer. Let's call this pattern `PAT-src`. The learning task for the associator net is to transform `PAT-src` into `PAT-dst`, sufficiently similar to make the word-net believe it has been presented to the real word.

6.10 Word $\rightsquigarrow$ Concept associator

6.10.1 Introduction

This has the corresponding "opposite" function of the concept-to-word associator detailed in section 6.9. To avoid tedious repetition, descriptions here will be limited to the specific differences. The overall general functionality and dataflow are the same, resulting in a network module that gets input as activation patterns from the hidden-layer of the vocabulary net, and sends its output to the hidden-layer of the corresponding[6] concept network.

6.10.2 Dataset

The dataset for this associator network was the hidden-layer representations for each of the learned words in the vocabulary and these are 14-bit patterns. The total number of patterns in this dataset was 9, the number of words in the vocabulary, and they are the names of the prepositions and the shapes.

6.10.3 Topology

The topology of the network was determined in the same way as the concept-to-word network, and the resulting layer-dimensions thus became: 14:16:36.

[5]The activation patterns here refers to those of the hidden-layer, selected due to their special status as internal representations.

[6]As determined by the virtual routing algorithm.

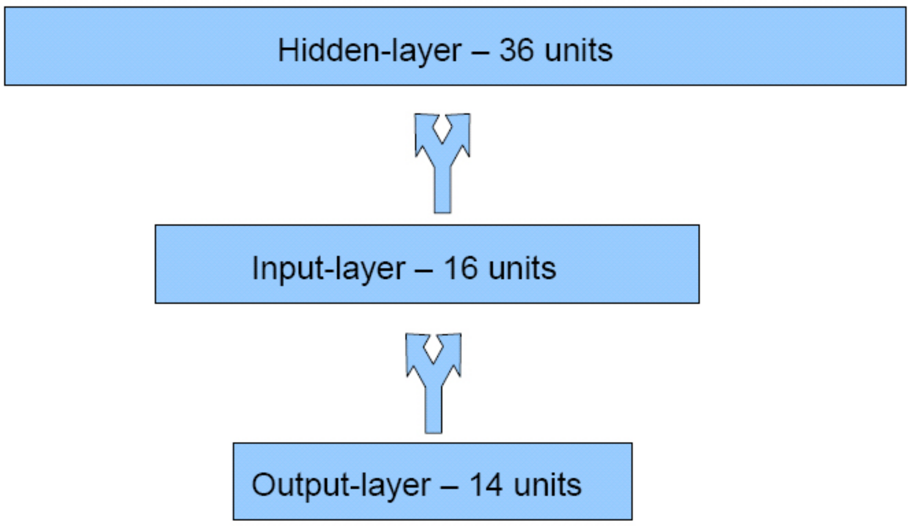


Figure 6.6: The Word-to-Concept associator network

6.10.4 Training

The training is performed as described in section 6.9.4, although with a different dataset.

6.11 Scene analyzer

6.11.1 Introduction

The scene-analyzer is a hybrid module containing algorithms and control-code on the normal symbolic level, in addition to actively using the shape-recognizer net (6.7), and the location-recognizer net (6.8). The purpose of this component is to decompose and *understand* what is in the scene (6.3), also called the field-of-view. In the current implementation this basically boils down to two things: to find and extract any shape that may be in the scene, and to interpret their relative positioning in terms of prepositions.

In addition, the interpreted scene is to be described using words in natural language, but that is actually a whole new area within the realm of computational language processing, namely speech and text production. Obviously, designing a such a system could not be done within the existing time-frame of this project, and it would also be beside the point to do so. The end result is that the text production system implemented is little more than a few simple rules for putting together nouns and prepositions.

6.11.2 Dataset

The dataset for the scene-analyzer is a combination of two kinds of data. On the symbolic level, explicit (x,y)-coordinates which describes both the location of a shape, as well as the shape itself. On the sub-symbolic level, activation patterns from the concept networks (6.7, 6.8), when they are fed the shape and location data.

Visual data is retrieved from the scene by an extraction algorithm, in portions of 32x32 pixel bitmaps. Each of these image samples taken from the scene is sent to the shape-recognizer network for processing, where the activations of both the hidden-layer and the output-layer is retrieved and analyzed. Also, when retrieving each bitmap-sample from the scene, the location of that viewport into the scene is included - as an (x,y)-coordinate.

6.11.3 Analyzing the scene

Ideally, one would send the entire scene into an integrated system of network modules, and somehow retrieve every shape that actually was there, handling both variable shape-sizes and unrestricted positioning. However, that is neither a trivial task, nor is it immediately clear how that should be done. For the experimental purpose of the perceptual grounding system, a simpler approach was taken. See figure 6.7 for a visualization of the process.

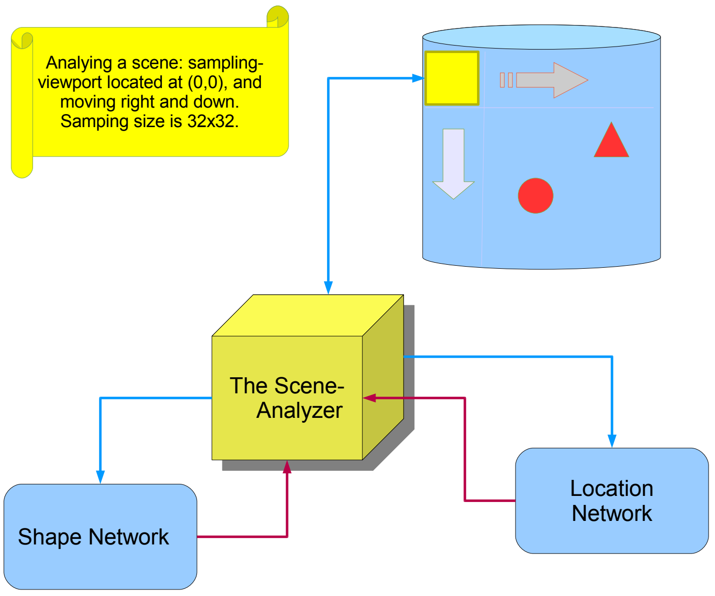

Analying a scene: sampling-viewport located at (0,0), and moving right and down. Samping size is 32x32.
The Scene-Analyzer
Shape Network
Location Network

Figure 6.7: Analyzing the scene: the sampling process

Basically, the process of interpreting the scene consists of retrieving visual and positional data through a viewport into it, and sending this data to the shape-recognizer net and the location-recognizer net, respectively. The results, in the form of activation patterns from both nets, are then brought back and analyzed to figure out the most likely interpretation. It is in particular the activations from the shape-net that may need further analysis, in case the shape under scrutiny only slightly resembles one (or more) of the known shapes. In practice, this extra level of analysis was dropped because of time constraints. Also, since only known shapes were used, this did not pose a problem.

From what positions in the scene were the samples taken, and what was the size of those samples? The sampling size, as stated in section 6.11.2, was set to 32x32, and this ensures that complete shapes **can** be retrieved. The locations which are sampled also has to be done at same boundaries at which shapes are located to ensure that **only** complete shapes are retrieved.

How was this enforced? There are two constraints that must be in place for this to be possible. First, recall the granularity of the scene, section 6.3, which determines the number of possible positions an object can occupy. By having this factor set equal to the size of the shapes, the first of the two constraints was realized. Second, there is the sampling size, and as this was also set to the same dimension, 32x32 pixels, the remaining requirement was realized. These self-imposed restrictions greatly simplified the process of decomposing a scene to find its constituents, the shapes, and allowed for more focus on the grounding principle.

Chapter 7

Coevolution of Language

Tower of Babel: *"Come, let us go down and confuse their language so they will not understand each other." So the LORD scattered them from there over all the earth, and they stopped building the city. That is why it was called Babel, because there the LORD confused the language of the whole world.*

`- God, Genesis 11`

7.1 Introduction

In this chapter we describe the implementation of a very simple system for evolution of communication. It is based on a recent work by Wang, as presented in his paper *A Simple Evolutionary Communication Model: Theoretical Analysis and Computer Simulations* [29] and the more recent paper *Convergence Analysis for Collective Vocabulary* [30]. In these papers, he offers an explanation of how communication can evolve between two or more agents[1], in which words are grounded in some meaning. Moreover, the theory specifically considers a multi-agent environment, and as such tries to account for the converging process of word - meaning binding. That is, the communicating agents gradually approach some maximal degree of agreement on *which words means what.* As we will see, that maximal degree is not always 100%.

7.2 The theory

Apart from what was mentioned in the brief introduction to the theory, another aspect of the coevolution of communication was central to the experiment: What

[1]Recall, as stated in chapter 2, section 2.5, that *agent* includes both simulated and living entities.

relations exist between the sizes of the sets of words and meanings? In other words, how does level of successful communication vary as a function of changes to the number words versus the number of meanings. Also, at what word:meaning ratio is the convergence-process unable to reach any stable state, independent of how many epochs the simulation runs?

7.3 Relevancy

The inclusion of a communication evolution model might seem slightly at odds with the main focus of the thesis, which is a language parser and a perceptual symbol-grounding system. However, as computational language research is the overall area in which it is rooted, the relevancy should not be too hard to acknowledge.

7.4 The basic setup

The experimental setup suggested by Wang is very simple, and the system implemented during the work on this thesis has retained much of that simplicity. Basically, the model consists of an environment, agents, a fixed set of keywords and a fixed set of meanings. Each agent contains an encoder and a decoder network. All these components are described in more detail in the next sections, (7.4.1, 7.4.2, 7.4.3, 7.4.4, 7.4.5). Only two agents were used in the test-runs.

7.4.1 The environment

The environment is exceedingly simple, and serves only as a framework for the agents to *exist in*. It contains functionality for adding and removing agents, setting the size the vocabulary of keywords and setting the size of the set of meanings. The environment contains no other objects for the agents to interact with, but acts as the supervisor of the communication process and determines whether an interpretation (decoding) of a keyword is correct.

7.4.2 The codecs

The encoder and decoder are both simple neural networks with only two layers. An important implication of this is that there is no need for a complex learning algorithm like backpropagation [24]. Since there is a direct relationship between the activation of an output node and the input of an input node, errors are easily computed and the weighted connections can be identified as being either right or wrong. - The encoder is used for converting a chosen meaning into a keyword. The decoder is used for interpreting a received keyword into a meaning.

7.4.3 The agents

The agents are implemented as autonomously running processes, and try to initiate a communication whenever they are not busy trying to interpret a message from another agent. Each agent contains an encoder and a decoder.

7.4.4 The words

The words, called keywords in Wang's paper, are the vocabulary of the agents and are represented by a unique number.

7.4.5 The meanings

The meanings are also represented by a unique number in the set. They do not refer to anything in the environment.

7.4.6 Format of the input-data

The standard orthogonal encoding principle of the one-in-n bit scheme is used. Thus, given a vocabulary of 6 keywords, keyword number 1 would be represented as 0100000, number two as 001000, etc.

7.5 Communicating - running the simulation

The running of a simulation is configured by a series of user-controllable parameters. These are the number of epochs, the number of words and the number of meanings. The number of agents are fixed at two. One epoch consists of n communication attempts, where n is the number of meanings. Selection of meanings is randomized, but all meanings are selected each epoch, and only once.

When a simulation is started, the agents choose a meaning (a concept or whatever someone may want to communicate) at random. The representation of the chosen meaning, a 6-bit pattern, is sent through the agent's encoder to get a keyword to communicate. That keyword is now the symbolic representation of the chosen meaning. Note though, that this meaning $\leftrightsquigarrow$ word mapping is local to the agent. Another agent may very well have a different mapping. The communicating agents have to learn to converge on the same mappings, and this is accomplished by several successive attempts at communication.

Similar to the process of initiating a communication event is the reception and subsequent decoding of a keyword into a meaning. A received keyword is sent into the decoder to be interpreted as one of the globally available meanings. In case of a correct interpretation, the weights supporting that mapping are increased.

Whenever an interpretation is wrong, the responsible weights are decreased.

At the end of the simulation, the average and final level of successful communication is computed. This is showed as a percentage, and the entire evolutionary process is visualized by a graph. The y-axis gives the success rate, 0 - 100%, and the x-axis represents the time on a per-epoch basis.

7.6 Results

Recall that *m* is the number of meanings and *n* is the number of keywords. The results of the experiment were much as expected, and they supported the theory. Both the simulations done by Wang and those run in this re-implementation behaved similarly. Both of them shows a linear decrease in successful communications as the m:n rate is increased. Put simply, if two persons trying to communicate has a lot of things they want to convey [2], but a very limited vocabulary[3], communication becomes difficult or utterly futile. A central theorem in his paper is included here for linking the results to a mathematical description.

Theorem 1. *For an evolutionary communication model described above, suppose each message is played by the two agents periodically, then an effective communication system can emerge with its communication accuracy given by* $\min\{\frac{n}{m}, 1\}$*, if the number of messages, m, and the number of codewords, n, satisfy the inequality condition* $\frac{2n+m-3}{n} * (1 - \frac{1}{n})^{m-2} > 1$ *or the approximately equivalent linear inequality* $\lceil \boldsymbol{n > 0.87m - 0.61} \rceil$. *The condition is called emergence condition.*

Although the general results were as expected, one particular part stood out as interesting. Closely related to the emergence condition, this empirically derived result tells us that there is a specific ratio of *m:n* at which the convergence process suddenly grinds to halt. What was surprising was the steepness of the pivot point. One would intuitively expect the communication evolution process to deteriorate in a linear fashion as the *m:n* ratio increases. As expected, this is true, but only before the pivot point is reached. The approximate formula for this critical number of codewords relative to the number of meanings can be written simply as:

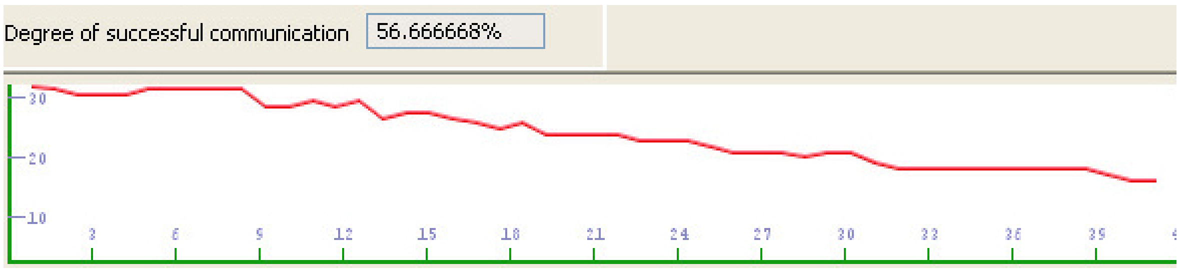
$n = \lceil 0.87m - 0.61 \rceil$
given that the number of messages is m.

[2]Communication restricted to words in a language, for the sake of the argument.

[3]A pointer to the importance of a strong semantic component in language processing: it is possible to construct an entire sentence from one basic word, and still be interpretable. The versatile word f*ck can assume nearly any case-role or part-of-speech, like in the sentence: verb adjective noun. (Expand the statement at your own risk)

To further illustrate the effect of varying the *m:n* ratio, a series of graph plots have been included. Each run of a simulation is done with two different number of epochs, to show the effect of that parameter on the evolution process. The values of the parameters used in each simulation are provided along with the corresponding figure.

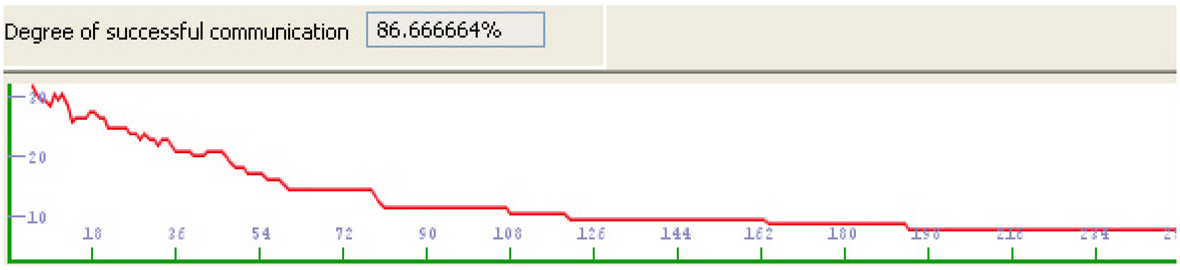
Graph of the communication evolution process, figure 7.1. The simulation was run for 50 epochs, and the final rate of success was 56.7%. Number of meanings, m, was 45. Number of keywords, n, was 45.



Degree of successful communication 56.666668%

Figure 7.1: Graph of the evolution - run 1A

Graph of the communication evolution process, figure 7.2. The simulation was run for 200 epochs, and the final rate of success was 86.7%. Number of meanings, m, was 45. Number of keywords, n, was 45.



Degree of successful communication 86.666664%

Figure 7.2: Graph of the evolution - run 1B

Graph of the communication evolution process, figure 7.3. The simulation was run for 200 epochs, and the final rate of success was 74.0%. Number of meanings, m, was 50. Number of keywords, n, was 45.

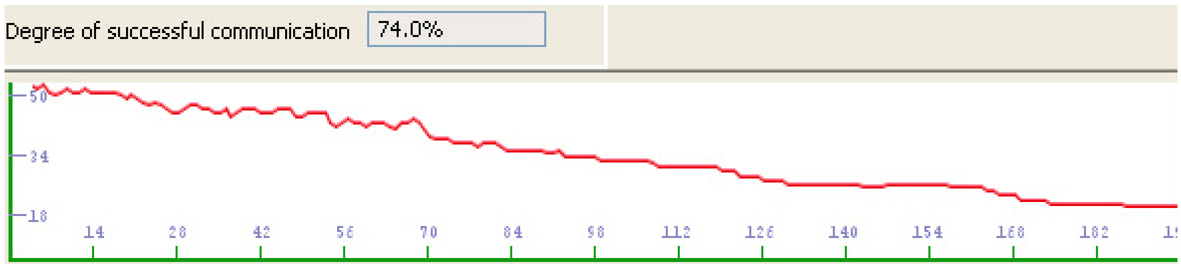


Figure 7.3: Graph of the evolution - run 2A

Graph of the communication evolution process, figure 7.4. The simulation was run for 400 epochs, and the final rate of success was 82.0%. Number of meanings, m, was 50. Number of keywords, n, was 45.

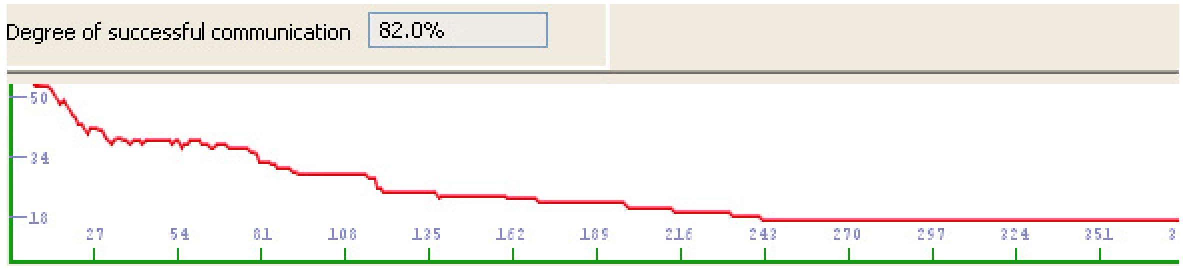


Figure 7.4: Graph of the evolution - run 2B

Graph of the communication evolution process, figure 7.5. The simulation was run for 200 epochs, and the final rate of success was 9.5%. Number of meanings, m, was 63. Number of keywords, n, was 45. At this m:n ratio, the futility of achieving a converging process, and finally a stable state, is evident. The graph continually oscillates between 0.0% and 9.5%.

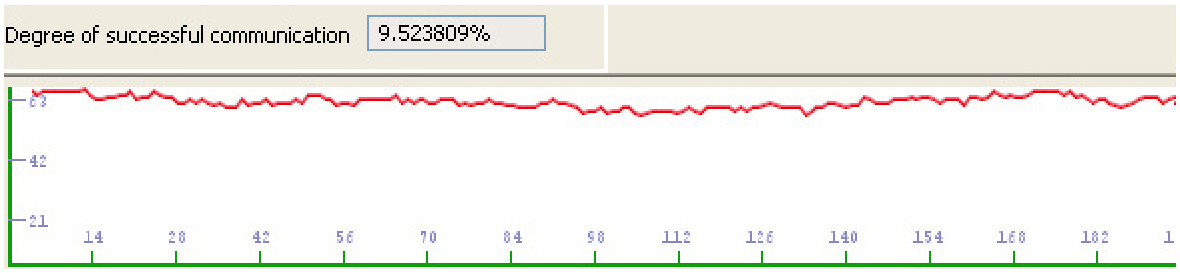

Degree of successful communication 9.523809%

Figure 7.5: Graph of the evolution - run 3A

Graph of the communication evolution process, figure 7.6. The simulation was run for 600 epochs, and the final rate of success was 6.4%. Number of meanings, m, was 63. Number of keywords, n, was 45.

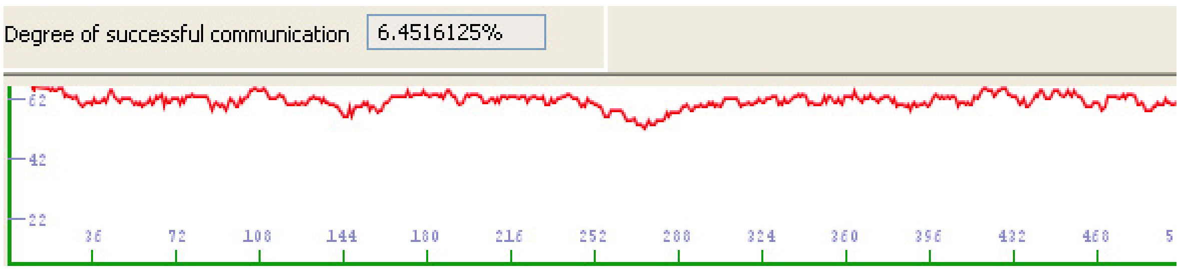

Degree of successful communication 6.4516125%

Figure 7.6: Graph of the evolution - run 3B

Since the part concerning the pivot point is not directly relevant to the reason for implementing the simulation system, no further analysis on this particular result was done.

Chapter 8

Evaluation of the Results

Not everything that can be counted counts, and not everything that counts can be counted.

- Albert Einstein

8.1 Introduction

In this chapter the results actually achieved will be compared to the goals originally set for this thesis. The degree of successful realization of each subgoal is briefly discussed, including what was failed to accomplish. Some of the subgoals have less tangible results, and in that respect difficult to prove by pointing to specific results. The goals falling into that category are subgoal 1 (8.2) and 5 (8.6) as they clearly comprises a subjective belief of gained insight and knowledge.

The order in which the individual goals are treated are the same as that used in the chapter describing them, chapter 3, but, needless to say, those representing the major part of the workload in this thesis, subgoal 3 and 4 (8.4, 8.5), should be considered the most important.

A bar-chart was created to show an estimated success rate of each subgoal, depicted in figure 8.1 at the next page. Because of the similarity between subgoal 1 and subgoal 2, they have been conveniently merged into one column, the green. Each bar in the figure are labeled to show what they represent, and their width are meant to signify their attributed importance. Note that the heights of each bar is meant only to illustrate a subjective feel of the completion factor for each goal, not an exact percentage.

Finally, at the end of the chapter, limitations of the two main systems, the parser and the perceptual grounding system, are summarized. As both systems are thoroughly described in their respective chapters, the reader is referred to the corresponding chapters for detailed explanations, if the need should arise.

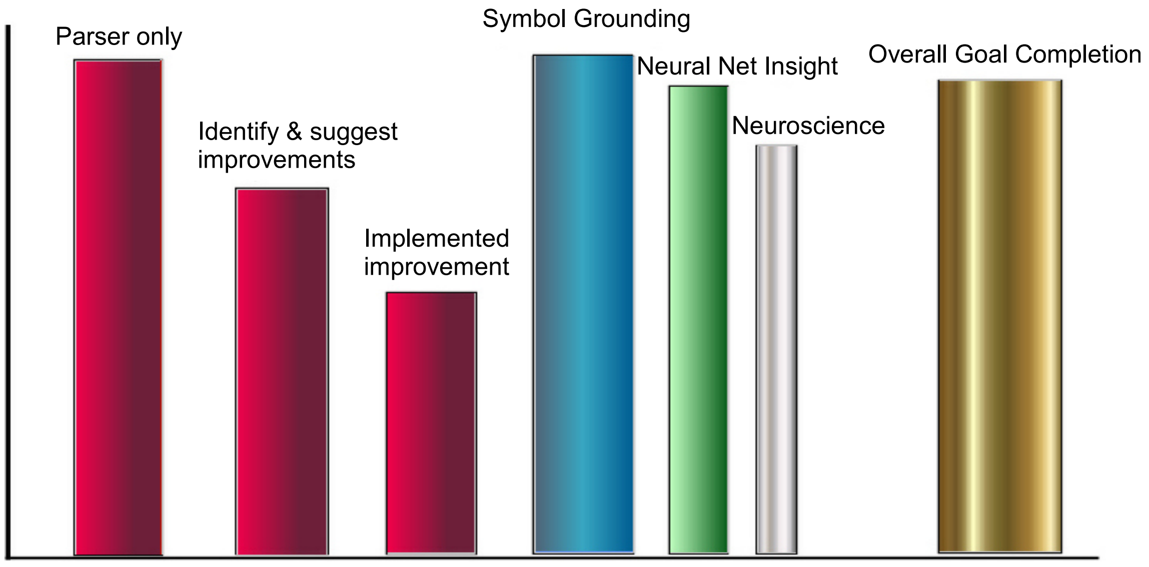


Figure 8.1: Illustrative levels of goal completion
Note that the parser is split into three bars, one for each aspect of this subgoal.

8.2 Subgoal 1: Learn about Neural Nets

Although a subjective evaluation, it seems that this part has been accomplished to a satisfying degree. Since no specific level of ambition was stated with regard to the desired skill-level, the insight gained has to be evaluated using some other scale of measurement. The most direct, and perhaps least subjective, method is to look at the use of neural networks in the implementations: For the most part, the systems are implemented using neural networks and the basic models[1], apparently successfully as they all perform according to their specified tasks.

8.3 Subgoal 2: Neural Nets and Language

While being closely related to the previous subgoal, in terms of content, what was intended here was to successfully implement and test two critical aspects of language that connectionism must be able to handle: to represent syntactic *structure* and to handle temporal input of data. As part of the work on the connectionist parser, (chapter 5), both of these aspects were encountered and handled through the use of an SRN [11] and a RAAM [23]. There should be no doubt, therefore, that this modest subgoal was accomplished.

[1]FNN, SRN, RAAM.

8.4 Subgoal 3: Realizing a Parser

Being one of the more complex parts in the thesis, as well as central to the focus of research, the fulfilling of this part is considered to be of great importance. Because the result of this subgoal is a concrete implementation with specific and measurable capabilities, a more objective evaluation of it is made possible.

In terms of being largely based on a modular design by Sharkey & Sharkey [8], some comparison with that system can help establish the success of my implementation. Though obvious, the fact that it actually works according to specifications, shows that the basic part of the subgoal is achieved. Each module did its job; the SRN [11] encoder learned to identify and extract syntactic structure from sentences, the mapper successfully transformed representations from the encoder to a format recognized by the decoder, and the RAAM [23] decoder had learned to successfully recursively decode the syntactic structures.

To be more specific about performance when saying *successfully*, the Sharkey parser [8] achieved an average of 75.6% correctly identified syntactic parse-trees on both training and test data. My parser got very similar results (75% - 80%) when using the same decoding procedure as Sharkey, but, as explained in chapter 5, section 5.5.5, this increased to 95% - 100% after the change to the decoding procedure. But, that increase in performance came at the cost of an overall decreased ability of the parser to generalize to unseen input that has an unknown syntactic structure[2].

Even though a problem was identified, and a change was implemented to solve it, the result became a parser that was improved in one area and weakened in another. Also, the initial plan was to do an exact implementation of the Sharkey parsing system [8], then design and implement a new system, much less bound to the structural design of their parser. Insufficient time and unexpected implementational difficulties effectively prevented this from happening. Instead, a more theoretical improvement was suggested, namely that of integrating the parser with a symbol grounding system. Whether the idea is novel can be debated, but it was at least not seen explicitly mentioned in literature at the time. Thus, the part of the goal regarding identification of performance bottlenecks and suggesting and implementing improvements was not completely achieved. However, the concept of a perceptually grounded parser can, arguably, elevate the degree of success on this part from *less-than-perfect* to *acceptable*.

[2]Note that the Sharkey parser [8] was not tested on unseen syntactic structures, only unseen input-sentences, in terms of different combinations of lexical items. The syntactic structure was always one of the eight that it was trained to recognize.

8.5 Subgoal 4: Realizing Symbol Grounding

This is undoubtedly the other large and important subgoal of the work done as part of the thesis. From the goal-specification it is tempting to simply declare this part for 100% completed and move on to the final subgoal. However, a few lines must be used to support the successfulness: Independent of how advanced a symbol grounding system is, as long as it is able to ground a concept to a word, the principle of symbol grounding is realized. In this case, through visual perceptions of shapes and prepositional location-concepts.

As an additional note of some relevance to subject of symbol grounding systems, it should be said that to get a really useful coupling between such systems and parsers, a third component is much needed. That component is a language production system, and its purpose is to bridge the linguistic gap between the grounding system and the parser by taking the individually grounded words and combining them into full natural language sentences that can be presented to the parsing system.[3]

8.6 Subgoal 5: Analogies to Neuroscience

Of all the subgoals this is the one with the least tangible and visible results. Any new insight into how the brain might process language and achieve its many advanced cognitive functions is very much internal to myself, and difficult to make explicit. What can be said though, is that I have a personal belief that much of the artificial neural network processes and models are adequately close to the truth about how the brain and its networks operates, a view supported by researcher and philosopher Paul Churchland in his book *The Engine of Reason, the Seat of the Soul* [7], among many others. Although not one of the real goals in this thesis, its implications are important. However, trying to exactly derive the level of accomplishment is neither reasonable nor useful. Suffice to say that the literature read, and the neuroscience course taken, as part of the masters degree is likely to have given me some new insight and understanding.

8.7 Limitations of the Systems

8.7.1 The Parsing System

In terms of limitations, there are two aspects of the parser that should be mentioned. The first concerns what kind of novelty of the input-sentences to the parser is handled. The second relates to how syntactic structures are handled.

[3]None of this was part of any goal and should obviously not affect the goal-result relation of subgoal 4.

Novelty of input. Because of the way the encoder part of the system was trained, it was able to handle novel input in terms of new combinations of lexical items. Or, put another way, as long as the individual words in a given sentence are contained in the vocabulary that the system was trained on, any word may be swapped with another word of the same part-of-speech type. Exactly what kind of structure is identified by the encoder is unclear, although presumably related to syntactic regularities. The point, however, is that because of this, it is not trivial to predict the system-behavior in case of unknown words in the input. Also, no such type of input was tested on the parser, neither in my work, nor was anything related to this mentioned in the Sharkey-paper *A Modular Design For Connectionist Parsing* [8]. Hence, no results in this regard are available for discussion on capabilities and limitations.

Syntactic structures. There are eight syntactic structures [p.32, section 5.3.2] that the system is trained to recognize and decode. The original decoding process, as used by Sharkey [8], performed its decoding purely by interpreting the data (recursively expanding the patterns), and this had two particular consequences:

1. A potential ability[4] to handle unseen syntactic structures.
2. A lower[5] rate of successful decoding because there was no external guide to constrain the decoding process.

The parsing system in this thesis, by guiding the decoding process to interpret **any** syntactic structure as one of the eight, usually got all the parse-trees right, but at the cost of not being able to partially identify any other structure.

[4]In theory, this represents the feature of graceful degradation, but, from what can be found in Sharkey's paper [8], it was apparently not tested.

[5]Lower, as relative to the results achieved by the decoding procedure used in my implementation.

8.7.2 The Perceptual Grounding System

The perceptual grounding system was designed to achieve a simple form of symbol grounding, and that was accomplished, as stated in section 8.5. Because the implementation is relatively simple, a set of assumptions and restrictions had to be imposed on the input. As for the grounding-capability itself, two concepts was learned; *relative location* and *shapes*, all through visual percepts.

Relative location. The concept of relative location was successfully tested on five prepositions, above, below, left, right and beside, and consequently limited to those spatial relations.

The Shapes. With respect to the identification of shapes and finding them within a scene, as well as the scene-analyzing process, a series of assumptions and restrictions was made. This is presented below in a list-format for increased readability.

1. The set of recognized shapes is limited to a triangle, a rectangle, a hexagon and a circle.
2. The dimension of the shapes is restricted to 32x32 pixels.
3. Recognition of rotated shapes is not supported.
4. The shapes must be single-colored. White is default.
5. The background of the scene must be single-colored. Black is default.
6. The location of the shapes is restricted to fall on specific boundaries. That is, the grid-size of the scene is 32x32, and thus a shape must be located at some multiple of 32, starting at (0,0).
7. The scene-analyzer is restricted to retrieve only 32x32-sized samples from the scene, and only from the same 32x32 boundaries as the shapes were restricted to.

Chapter 9

Summary and Conclusion

If we knew what it was we were doing, it would not be called research, would it?

`- Albert Einstein`

9.1 Summary

An important, but perhaps somewhat understated fact, is that all work done in the thesis was done within the context of the AI paradigm. Having emphasized that, focus can be returned to the more concrete details of what was done and how the goals were attempted solved. The global motivation of the work performed here is that of research on language and computational language processing and understanding.

As a starting point, a sentence, *The flower on the table is red*, was contemplated upon with respect to how its inherent syntactic structure and semantic components could be represented in a neural network. For that reason, as well as the profound similarity between artificial neural networks and the networks in the brain, connectionism was chosen as the tool to perform the research both **on** and **with**. A framework for creating, training and testing neural networks was designed. Three distinct systems were selected to investigate specifically, and these were: a parsing system [8], a perceptual grounding system [3] and a communication evolution model [29, 30]. A recent report or paper on each of these was selected, and used as the basis for implementing the experimental systems.

The parser implemented consists of three modules, each providing a particular functionality. The part receiving the input-sentences is called the encoder, and consists of an SRN [11]. The part handling recursive decoding of the syntactic structures is made up of a RAAM [23]. The *middleware* component between the input and output modules enabled the encoder and the decoder to communicate. Functionally, the parser worked well and was successfully implemented, but because the implementation was more complicated than initially expected, the realization of any truly novel and significant improvements were limited. Grounding the parser by coupling it to a symbol grounding system was suggested.

The perceptual symbol grounding system implemented is comprised of five neural networks, a scene component and a scene-analyzer module. There were two concept networks, one for learning to recognize shapes and another for learning relative positions. One network served as the system's vocabulary, and final two networks provided the functionality of association: a concept-to-word associator, and a word-to-concept associator. The scene component contained the visual input, along with implicit positional information. Interpretation of the scene was performed by the scene-analyzer.

The final area investigated was that of a communication evolution model. It involved experimentally testing how and if a shared vocabulary would develop, in an environment of two autonomous agents. The implemented system contained two agents in a simple environment, and a common set of keywords $\boldsymbol{n}$ and meanings $\boldsymbol{m}$. Simulations were run to test how and if a common set of meaning-keyword bindings would occur. Results obtained were similar to those obtained by Wang, and supported the existence of a pivot-point at a specific **n:m** ratio.

9.2 Conclusion

Normally, or at least in many cases, one is able to finally state in the conclusion whether some specific theory or model actually worked out in practice. The diverse work done in this thesis, combined with the relatively wide focus of the research, makes the formulation of such a short and concise conclusion difficult. However, certain goals were specified and three systems were successfully implemented a result. The conclusion, as I see it, can be stated as follows, comprising the remainder of the current section:

Most of the goals set for my thesis were achieved to a satisfactory degree, although the part on the parsing system failed to accomplish some of its subgoals. A parser, a perceptual symbol grounding system and a model of the evolution of a shared vocabulary was implemented. The core of the connectionism paradigm, neural networks, was investigated and used extensively. Insight and hands-on experience was gained. Also, connectionism was empirically found to be both functional and to have interesting properties useful for NLP and NLU[1] -related tasks. Research

[1]Abbreviation repeated here for convenience: NLP = Natural Language Processing, NLU = Natural Language Understanding.

on computational language processing and AI can reasonably be said to constitute an interesting and promising field of research, and to be worthy of both past, current and continued efforts.

9.3 Further Research

Further research on language processing is recommended, and particularly with respect to new ways design a significantly improved parsing system, and more advanced perceptual grounding systems.

Bibliography

[1] T. Bayes. *Bayesian Methods.* http://en.wikipedia.org/wiki/Thomas_Bayes, 1764.

[2] T. Bayes. *Naive Bayes Classifier.* http://en.wikipedia.org/wiki/Naive_Bayes_classifier, 1968.

[3] A. Cangelosi. *Approaches to Grounding Symbols in Perceptual and Sensorimotor Categories.* Preprint, 2005: Cognitive Science, Elsevier. Adaptive behaviour & Cognition Research Group, School of Computing, Communiction and Electronics. University of Plymouth, UK.

[4] N. Chomsky. *Chomsky Hierarchy.* http://en.wikipedia.org/wiki/Chomsky, 1900.

[5] N. Chomsky. *Generative Grammar.* http://en.wikipedia.org/wiki/Chomsky, 1900.

[6] P. Churchland. *Matter and Consciousness.* MIT. Press, 1988.

[7] Paul. Churchland. *The Engine of Reason, the Seat of the Soul: A Philosophical Journey into the Brain.* MIT. Press, 1995.

[8] Noel E. and Amanda J. C. Sharkey. *A Modular Design For Connectionist Parsing.* Center for Connection Science, Department of Computer Science, University of Exeter, U.K., 1992.

[9] J. Early. An ecient context-free parsing algorithm. *Comm. ACM*, 13-2:94–102, 1970.

[10] M. Minsky (Editor). *Semantic Information Processing.* MIT Press, 1968.

[11] J. L. Elman. Finding structure in time. *Cognitive Science*, 14:179–211, 1990.

[12] S. Harnad. *Symbol Grounding is an Empirical Problem: Neural Nets are just a Candidate Component.* Proceedings of the Fifteenth Annual Meeting of the Cognitive Science Society. NJ: Erlbaum, 1993.

[13] G. Houghton, D. Schanks, J. Kruscke J. Bullinaria, R. O'Reilly, E. C. Leek, G. Dell, and M. Zorzi. *Connectionist Models in Cognitive Psychology.* Psychology Press, 2005.

[14] i. Wikipedia. *Context free frammar.* http://en.wikipedia.org/wiki/Context-free_grammar, 1950.

[15] ii. Wikipedia. *Connectionism.* http://en.wikipedia.org/wiki/connectionism, 1950.

[16] Marshall R. Mayberry III and Risto Miikulainen. *SAARDSRN: A Neural Network Shift-Reduce Parser.* Techical Report AI98-275, 1990.

[17] S. Levy J. B. Pollack. *Infinite RAAM: A Principled Connectionist Substrate for Cognitive Modeling.* Dynamical and Evolutionary Machine Organization Volen Center for Complex Systems, Brandeis University, Waltham, MA 02454, USA, 2001.

[18] D. Jurafsky and J. H. Martin. *SPEECH and LANGUAGE PROCESSING: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition.* Prentice Hall, 2000.

[19] J-D. Zucker N. Bredeche, S. Zhongzhi. *Perceptual Learning and Abstraction in Machine Learning.* Institute of Computing Technology - Chinese Academy

of Sciences - Beijing, China Laboratoire dInformatique Medicale et de Bio-informatique - Universite Paris-Nord - Paris, France, 2003.

[20] S. Nirenburg and Victor Raskin. *Ontological Semantics.* MIT Press, 2004.

[21] R. C. O'Reilly and Munakata. *Computational Explorations in Cognitive Neuroscience: Understanding the mind by simulating the brain.* MIT Press, 2000.

[22] Tom Ziemke(Ed.) P. Vogt. The physical symbol grounding problem. *Cognitive Systems Research, Elsevier*, 3:429–457, 2002.

[23] J. B. Pollack. *Recursive Distributed Representations.* Laboratory for AI Research & Computer & Information Science Department, Ohio State University, 1990.

[24] D. Rumelhart. *The backpropagation algorithm.* http://en.wikipedia.org/wiki/backpropagation, 1986.

[25] A. Saffiotti S. Coradeschi. Anchoring symbolic object descriptions to sensor data. *Linkping Electronic Articles in Computer and information science*, 4 no.9:N/A, 1994.

[26] B. Selman. Rule-based processing in a connectionist system for natural language understanding. Master's thesis, University of Toronto, Departement of Computer Science, Canada, 1985.

[27] P. Toma. *Systran: A Machine Translation system.* http://en.wikipedia.org/wiki/Systran, 1968.

[28] vi. Wikipedia. *Graceful degradation.* http://en.wikipedia.org/wiki/Graceful_degradation, 1900.

[29] J. Wang. *A Simple Communication Evolution Model: Theoretical Analysis and Computer Simulations.* 2005. preliminary tecnical report.

[30] J. Wang, L. Gasser, and J. Houk. *Convergence Analysis for Collective Vocabulary.* 2006. Proceedings of the Fifth International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2006). Hakodate, Japan.