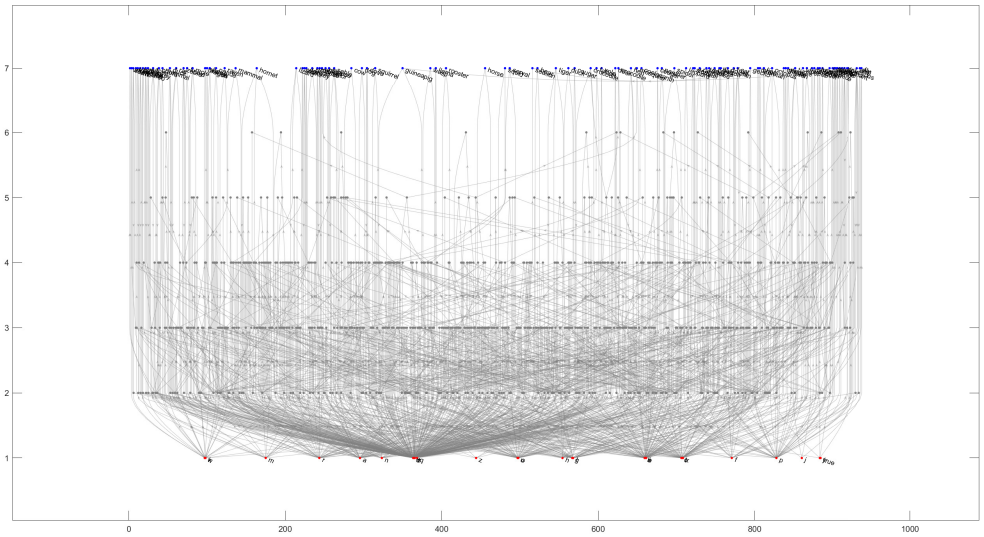
CHALMERS
UNIVERSITY OF TECHNOLOGY

UNIVERSITY OF GOTHENBURG



Simple Language Learning in Artificial General Intelligence

Master's thesis in Algorithms, Languages and Logic
and Complex Adaptive Systems

LOUISE JOHANNESSON
MARTIN NILSSON

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2018

Master's thesis 2018

Simple Language Learning in Artificial General Intelligence

LOUISE JOHANNESSON
MARTIN NILSSON

Department of Computer Science and Engineering
Chalmers University of Technology
University of Gothenburg
Gothenburg, Sweden 2018

Simple Language Learning in Artificial General Intelligence
LOUISE JOHANNESSON
MARTIN NILSSON

Supervisor: Claes Strannegård, Department of Computer Science and Engineering
Examiner: Christos Dimitrakakis, Department of Computer Science and Engineering

Master's Thesis 2018
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: The Animats dynamic graph after learning 100 words, visualised in Matlab.

Typeset in LaTeX
Gothenburg, Sweden 2018

Simple Language Learning in Artificial General Intelligence
LOUISE JOHANNESSON
MARTIN NILSSON
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg

Abstract

The development of Artificial Intelligence (AI) has made it possible for computers to solve more and more of the tasks that humans can perform. One area of interest within AI research is Artificial General Intelligence (AGI), which aims to develop AI systems capable of solving multiple varying tasks. The Generic Animat research project is one biologically inspired approach to AGI. Our thesis aims to explore Natural Language Processing within the Animat model, by mimicking the way young children start to learn their first language.

During the course of the thesis project we have developed two slightly different prototypes (referred to as the "Temporal Animat" and the "Chunking Animat"), both of which can learn to recognise and produce words that they are exposed to in a noise-free environment. Furthermore, functionality for word-to-word association and multimodal association has been implemented, thereby allowing the prototypes to associate between words, based on their occurences in some input text, and between words and other types of sensor input representing non-language concepts in the Animat's environment.

The prototypes were evaluated on small input texts with a limited number of words. The result of these evaluations suggests that both versions of the Animat are capable of achieving high degrees of accuracy when their word-to-word associations are compared to those of vector space models. The evaluation of multimodal associations has been done only for the Temporal Animat. The results indicate that the Temporal Animat makes mostly accurate associations between words and other input, although more tests on larger data sets are needed to determine how well it works on a larger scale. Due to the similarity in implementation between the two prototypes, the Chunking Animat is expected to be equally successful at this task.

Keywords: AI, natural language, NLU, artificial general intelligence, Animat, language acquisition, first language learning, artificial animal

Acknowledgements

We would like to thank our supervisor Claes Strannegård for his continued support and guidance during the entire course of the project. Without him, this project would not have been possible.

We would also like to thank our examiner Christos Dimitrakakis for encouraging us to, early on in the project, find good state of the art models to compare with and properly define how to evaluate our results in the project plan.

Finally, we would like to thank our families and friends for their encouragement and support.

Louise Johannesson and Martin Nilsson, Gothenburg, May 2018

Contents

List of Figures

List of Tables

1

Introduction

Artificial intelligence (AI) and machine learning algorithms have become capable of solving more and more tasks during recent years, and the algorithms used have become increasingly general. One area where machine learning has made large progress is natural language processing (NLP). However, the most commonly used algorithms for NLP are designed specifically for language, and not made to be applied to other problems. Our aim is to use general learning rules to create an AI system capable of learning simple cases of natural language.

1.1 Background

During recent years artificial intelligence (AI) has been introduced in many diverse fields, ranging from traffic, where self-driving cars are being developed, to medicine, where algorithms can be used to screen for medical conditions [1]. While many types of AI have been developed with the intention of solving some specific task, another aim is to develop what might be referred to as artificial general intelligence (AGI). The idea behind AGI is that it should be able to function in many different environments, much in the same way that a human can adapt to many diverse situations. To function within different fields and to be able to handle diverse situations and tasks, a machine with general intelligence would need to have many different skills, or a way of developing (learning) many different skills. One particular type of skill that would be useful in many situations is knowing a language, which is the focus of this master thesis.

Natural Language Processing is an interesting subject for many reasons. Having a language is of value because it allows communication. For many of the tasks that an AI system might come to perform in the future, it is necessary for it to be able to communicate with humans. If an AGI system could learn the natural language used by humans it could be used for tasks that require interaction with people without any particular technical expertise. If an AGI system had good enough language skills, people would not have to adjust their way of expressing their intentions and demands into code, special commands or sequences of pressing buttons to communicate them

to the AGI system. Humans could simply talk to machines in much the same way that they talk to each other.

To humans, language is more than a tool for communicating with others, it also impacts the way we think and perceive the world [2]. From this perspective, it is interesting to see how language skills affect the development of AI. Since the diversity and change of human language cannot easily be captured by predefined rules and dictionaries, it would be good if a computer AI could learn it through interaction. To avoid ad-hoc solutions, it would be preferable for an AGI system if its language skills could be learned using the same logic and rules as it uses when learning other skills.

1.2 Aim

The aim of this master thesis is to explore how and to what extent a model for general artificial intelligence can be used for the task of first language learning. The model that we have used in this thesis is the model developed in the Generic Animat research project [3]. The Generic Animat research project aims to develop artificial animals called Animats. The Animat is considered intelligent if it can learn to perform behaviours that allow it to fulfil its needs and stay alive. Thus, the Animat approach is a biologically inspired approach to model learning and intelligence. In accordance with the biological approach, our idea is for the Animat to be able to learn language in a way that is inspired by the way that human children learn their first language.

While some AI models can associate words to other words, the idea here is to be more general. Associations could be made multimodal, in that words could also be associated to other forms of sensory input. For example the word "cold" could be associated not only with words such as "snow", "ice" and "winter" but with the detection of a low value in the sensors that monitor the environment's temperature.

The overarching research question of this master thesis project can be formulated as:

> "How can an artificial animal be designed (in a non ad-hoc fashion) so that it can learn simple cases of language?"

The purpose of this thesis is to get an indication of whether or not this biologically inspired approach to learning can successfully be used for language in the Animat model, and if not, to shed light on what problems remain.

1.3 Problem Formulation

As stated in the previous section, the aim of this project was to evaluate how first language learning could be done in the Generic Animat research project. For the Animat to be considered to have learnt a language, two goals should be met. The Animat should be able to:

1. associate words to words

2. associate words to other types of sensor input

For the first of the goals to be considered fulfilled, the associations made by the Animat should be similar to the associations made by vector space models (see section 2.2), as described by Clark in [4], when tested with the same input. For the second goal to be considered fulfilled, the Animat should be able to associate between some specific words and other types of sensor input that have been present together with those words. In order for the goals to be reached, several intermediate steps needed to be implemented first. The steps were to make the Animat capable of:

1. mapping motor nodes to letters

2. recognising words

3. producing words

As stated in the previous section, the idea was for the learning rules to be general and biologically inspired. As such, the Animat needed to make use of the same learning rules that had been developed in the Animat project for other tasks. This ensured that the all the learning rules employed by the Animat could be used for learning other tasks than language. No assumptions have been made about the grammatical structure of the language, so that the approach should work equally well for all languages.

1.4 Limitations

To narrow the scope of the project, we focused on developing basic functionality and left further explorations to other projects. Since the focus of the Animat project is on developing the Animat's "brain", all language input and output was represented as text rather than as sound, allowing us to focus on the learning process rather than on complicated sensors. The plan was for the Animat to learn a language which it was exposed to, much like a human child would learn its first language. English was chosen for testing for simplicity. However, the algorithm was not made specifically

for English, and therefore any human language could be used. The end goal was to have a language consisting of single words, therefore the grammatical structure of natural language was not considered in our thesis.

1.5 Related Work

Natural language processing (NLP) is a wide field of research, where many different approaches have been explored. In this section, we will first provide a brief introduction to the area in general, and then proceed to present some research within NLP which have employed ideas inspired by biological processes.

1.5.1 Natural Language Processing

A lot of work has been done in the field of NLP. The goal and type of methodology varies greatly between different approaches to NLP. Two major methodologies are to use predefined grammatical rules or to use statistics [5]. There have also been attempts to create AI systems that learn language in some kind of biologically inspired way.

As mentioned above, one approach to NLP is to manually define rules for the grammar of the language. The structure of sentences and the relation between words in them can be modelled by using context-free grammars, formal languages and finite automata [6]. Often meanings of phrases are represented in set theoretic models where the key intuition is that the world is full of objects that have relations to each other [4].

One of the statistical approaches commonly used in natural language processing is the use of vector space models [4]. This approach uses large amounts of text, to find statistical relations between words. Other approaches that use this methodology have also attempted to design programs capable of inferring grammars from examples [7]. For example, unsupervised machine learning can be used to learn word classes, that is, to classify words as belonging to different lexical categories [6]. The benefit of grammar induction is that an AI system that has learnt a grammar could use the approaches mentioned above without the need for manual definitions of the grammar.

Another area of interest is how to model sequences, as the words in natural language can be viewed as sequences of letters, and sentences as sequences of words. One way of modelling sequences in general is to use Markov models. In a Markov chain the probability of the current state depends on the previous state:

$$Pr(X_{k+1} = x | X_k = x_k, \ldots, X_1 = x_1) = Pr(X_{k+1} = x | X_k = x_k)$$

In a n-order Markov chain, the state depends on the last n states:

$$Pr(X_k = x_k | X_{k-1} = x_{k-1}, \ldots, X_1 = x_1)$$

$$= Pr(X_k = x_k | X_{k-1} = x_{k-1}, \ldots, X_{k-n} = x_{k-n}) \text{ for } k > n$$

Markov chains can be relevant for NLP, since in language the probability of a word or letter occuring tends to depend on previous words or letters [8].

An extension to ordinary Markov chains is Variable Order Markov Models (sometimes referred to as "context trees" [9]), where the number of past states used for estimating the probability of the next event can vary [10]. This can be a preferable model to use in some situations, since in some cases, the length of the context which is relevant for predicting the next state can vary.

1.5.1.1 Biologically inspired approaches to NLP

Another approach to NLP has been to draw inspiration from nature and biology, and to imitate the way that humans learn to communicate. One such biologically inspired approach to language learning has been used by Franchi et al [2]. Their aim was for the program to be able to learn to imitate input sound, through a process similar to human children's "babbling". When babbling, children produce and imitate sounds and thereby learn how to pronounce them. Franchi et al conclude that this approach can successfully be used by their program for the same purpose.

Another biologically inspired project is LESA (Linguistically Enabled Synthetic Agent) [11]. The experiments in the LESA project use humanoid robots and attempt to imitate the way human children learn to use words and phrases and what they mean through babbling and through interaction with a human, thereby grounding language in social and physical interaction and in perception. The idea is to let the robot interact with a human who talks about the shared environment and the objects in it, like a carer would talk to a human child about toys and other objects in the environment.

The LESA approach is in line with some of the ideas discussed by Cangelosi in [12], which argues that the words first acquired by a language-speaking agent should be directly grounded in action and perception. The words in the initial lexicon would represent sensorimotor properties of the world and be concrete and imaginable. Later, the lexicon could be expanded to include words without direct grounding, which would instead be defined using other words.

2

Theory

This chapter gives an introduction to the Generic Animat research project, to vector space models for natural language understanding, how humans learn language and the human short term memory. The Generic Animat research project served as the basis for our model. Vector space models for natural language understanding were used in the evaluation. Human language learning served as an inspiration for the learning rules used in the project. And the human short term memory was the basis for how short term memory was implemented in the project.

2.1 Generic Animat Research Project

The artificial animals (Animats) in the Animat project learn and change their individual attributes during the their lifespans. The attributes of a single Animat are a dynamic graph (G), an activity on that graph (specifying which nodes are active) and a set of experience matrices [3]. Since our approach is inspired by the way a human child learns language (within its lifetime), we have focused on only some parts of the Generic Animat research project and excluded other parts such as the Animat's need to survive.

The graph (G) contains three different kinds of nodes; Perception nodes, Action nodes and Need nodes. Perception nodes are used to perceive different input from outside of the Animat. The Perception nodes can be sensor nodes that react to specific input that is present in the environment, or logical nodes ("AND", "NAND", "OR", "SEQ") that are used to represent more complicated situations such as combinations of input. Action nodes can be motor nodes representing simple actions (that will be used to produce single letters) or logical nodes ("AAND", "ASEQ") that are used to combine motor nodes into more complicated actions (such as producing words). Need nodes represent how 'healthy' the Animat is and can be used to reward or punish actions. However, the Need nodes were not used in our project, but are included in this section for the sake of completeness. An example of the kind of dynamic graph used by the Animats is shown in figure 2.1. At the time of the Animat's creation, its sensors and motors are already specified. The logical

nodes form as the Animat learns. New nodes can be added to the graph through different learning rules. The learning rules that are used in this project, and their application, will be described in chapter 3.

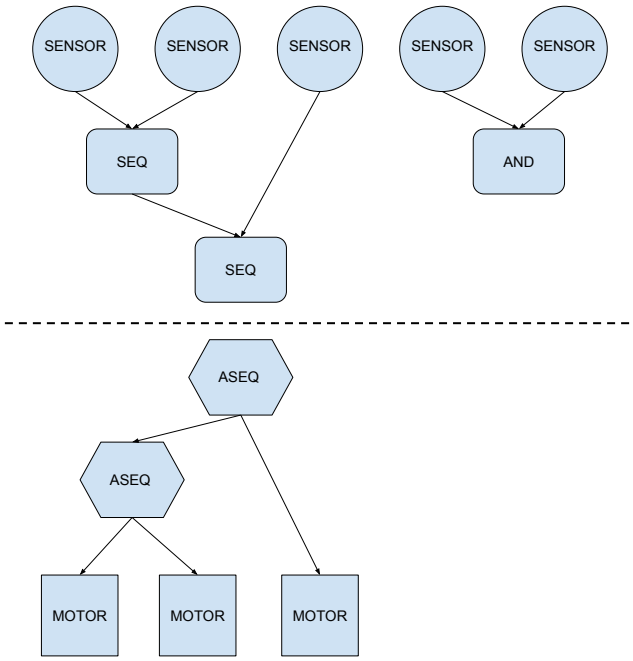


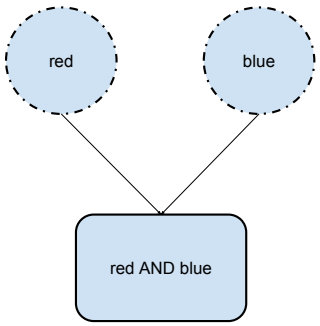
Figure 2.1: An illustration of how a Dynamic Graph could look in the Animat. Each node is labelled with its type to make the illustration clearer.

The Animat is given input, and updated, at discrete times. Each such time is referred to as a time-step. In a single time-step, a perception node can be active or inactive. A sensor is active when the stimuli/sensory input it can perceive is present in the environment. For example, a sensor for perceiving the colour red, *"SENSOR(red)"*, is activated when this colour is present in the environment.

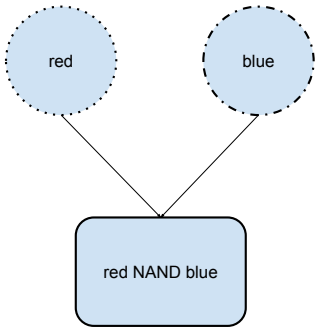
An AND node takes two other perception nodes as input, and is active when both of its input nodes are active. An AND node for perceiving the combination of the colours red and blue, *"SENSOR(red) AND SENSOR(blue)"*, see figure 2.2a, would be activated when both the colours red and blue are present in the environment and thus both perceived by the Animat's sensors.

A NAND node works the same way as the AND nodes, but is active in the opposite situations, that is, when at most one of its input nodes is active. Thus, *"SENSOR(red) NAND SENSOR(blue)"* would be active when either blue or red is not perceived, see figure 2.2b.

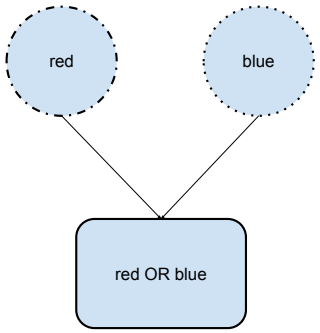
An OR node also takes two other perception nodes as input, and is active when either of its input nodes is active. An example would be *"SENSOR(red) OR SENSOR(blue)"* which is active when one or both of the colours red and blue are per-



(a) AND node (rectangle) for perceiving the combination of colours red and blue. The sensors (circles) for colours red and blue are both shown as active, but only the AND node is top active.



(b) NAND node (rectangle) for perceiving that the combination of colours red and blue is not present.



(c) OR node (rectangle) for perceiving that at least one of the colours red and blue is present in the environment.

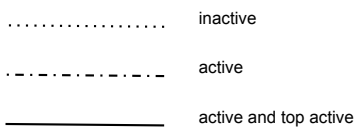


Figure 2.2: Examples of simple logical perception nodes. Sensors are displayed as circles and logical nodes as rectangles.

ceived, as can be seen in figure 2.2c.

A SEQ node takes two ordered perception nodes as input, and is active at time t if its first input node was active in time $t-1$ and its second input node is active in time t. Thus, *"SENSOR(red) SEQ SENSOR(blue)"* would become active if the Animat first perceives the colour red, and then in the next time-step perceives the colour blue, as can be seen in figure 2.3.

In addition to being active, a perception node can be top active. A perception node is top active if it is active and no node for which it is input is active. An example of this could be if the perception nodes *"SENSOR(red)"*, *"SENSOR(blue)"* and *"SENSOR(red) AND SENSOR(blue)"* are all active (as shown in figure 2.2a).

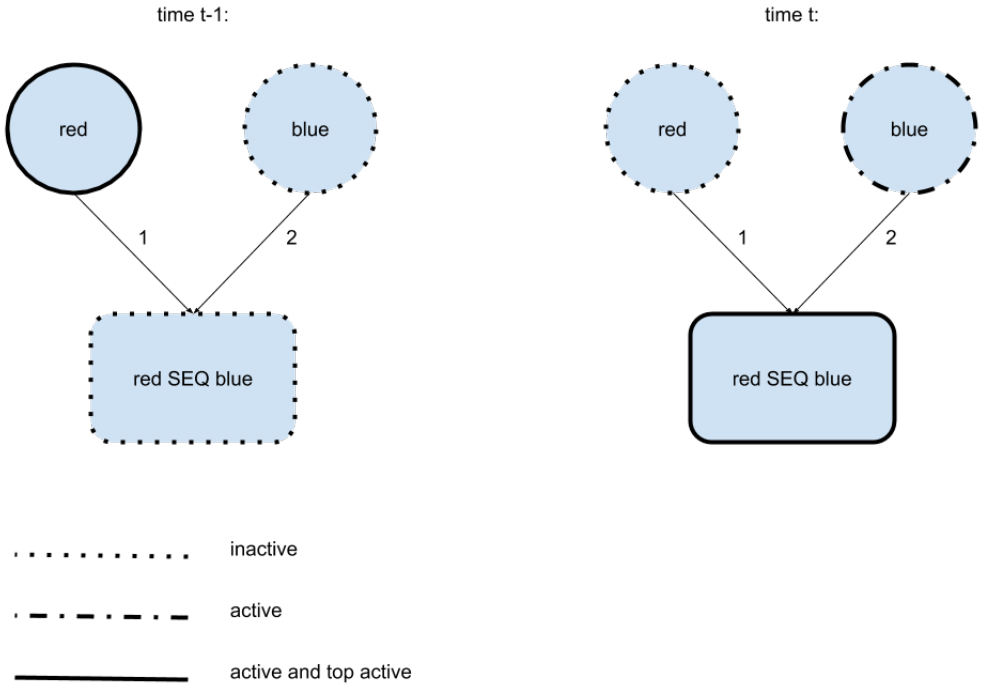
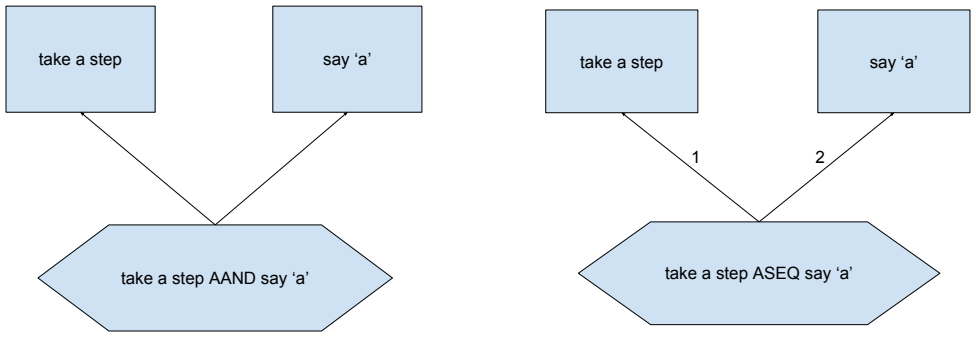


Figure 2.3: SEQ node (rectangle) for perceiving that red is followed by blue. In time-step $t-1$, the sensor for perceiving red is active. In the next time-step t, the sensor for perceiving blue is active, and the *"SENSOR(red) SEQ SENSOR(blue)"* also becomes active.

Then, *"SENSOR(red) AND SENSOR(blue)"* could be top active (provided that it is not input to some other node which is also active), as well as being active. However, neither *"SENSOR(red)"* nor *"SENSOR(blue)"* are top active, since they are input nodes to *"SENSOR(red) AND SENSOR(blue)"*.

When the Animat acts, it does so by activating an action node. Every motor node affects the environment in a predefined specific way. Examples of motor nodes could be *"MOTOR(take a step)"* which moves the Animat forward, or *"MOTOR(say 'a')"* which outputs the letter 'a' to the environment. An AAND ("Action AND") node activates two action nodes simultaneously, for example *"MOTOR(take a step) AAND MOTOR(say 'a')"* makes the Animat move forward and output 'a' at the same time, as can be seen in figure 2.4a. An ASEQ (Action SEQ) node activates two action nodes in sequence, for example *"MOTOR(take a step) ASEQ MOTOR(say 'a')"* (shown in figure 2.4b) makes the Animat first move forward, and then output 'a'.

The different matrices are used to keep track of when, and if, different nodes have been active in relation to each other. Not all of the matrices used in the Generic Animat research project were used in this project. Below, we will explain the function of the matrices that were relevant for our project, more specifically the "transition



(a) AAND node for performing two actions (to "take a step" and to "say 'a'") in the same time-step.

(b) ASEQ node for performing two actions in order: first "take a step" in time-step $t-1$ and then "say 'a'" in time-step t.

Figure 2.4: Examples of logical action nodes. Motors are displayed as rectangles and logical nodes as hexagons.

matrix", "sequence matrix" and "conditional matrix". The transition matrix consists of entries for all combinations of two perception nodes and one action node. For two perception nodes b and b' and an action a

$$Trans(b, a, b')$$

denotes the probability that b' becomes top active at time t given that b was active at the previous time-step $t-1$ and that the action a was performed at $t-1$.

The transition matrix is used, among other things, for determining what action nodes are generators for what perception nodes. A generator is an action that, with a high probability, results in a specific perception node becoming top active regardless of what node is active before. Therefore, if

$$Trans(True, a, b') = 1$$

the action a is said to be a generator of the node b'. For example, activating the motor node for producing a specific phoneme should always result in the Animat hearing that specific phoneme in the next time-step. To allow for some noise in the model, an inequality could be used instead of the equality above, exchanging the 1 for some constant $c \approx 1$, such as $c = 0.95$. This might be more realistic, since in real life, there are some situations where exceptions might occur. For example, if the environment is noisy, one might not hear oneself speak.

The sequence matrix has entries for all combinations of two perception nodes and is used for finding common sequences of nodes. Each entry stores information about how often one node has been followed by another. Thus:

$$Seq(b, b')$$

denotes the fraction over the last 100 time-steps that b has been top active at time-step t and b' has been top active at time-step $t-1$.

The conditional matrix has entries for all combinations of two perception nodes and is used for storing the probability that two nodes are top active at the same time. For two perception nodes b and b'

$$Cond(b, b')$$

denotes the probability that b is top active in time-step t given that b' is top active in time-step t.

2.2 Vector Space Models for Natural Language Understanding

One of the approaches commonly used in natural language processing is vector space models. In this method words are grouped based on the context in which they usually occur in large quantities of text. This is often done by looking at the words before and after, and considering words that often have the same neighbours as similar. A good explanation of this approach can be found in [4] by Stephen Clark. We will give a brief summary of the main points of this approach here (following the same structure as Clark), as it is highly relevant for the evaluation of our project, but refer to Clark's text for a more thorough explanation.

Vector space models for lexical meaning, has several similarities to vector space models for document retrieval. The document retrieval problem in Information Retrieval (as described in [13]) can here be formulated as "given a query -typically represented as a set of query terms - return a ranked list of documents from some set ordered by relevance to the query" [4]. The basic approach here is to represent both the query and the documents as vectors in a multi-dimensional space, where each base vector in that space represents one word. The vectors for the query and documents are then given by having the coefficient for each base vector be the number of times that word occurs in the query/document [4]. The similarity between a document and the query is then given by the overlap between the two vectors, which in turn is given by the dot product between them:

$$Sim(\vec{d}, \vec{q}) = \vec{d} \cdot \vec{q}$$

This approach can then be refined by accounting for how common different words are, and how long the documents are. Accounting for how common different words are can be done by multiplying the coefficients with the inverse document frequency (IDF) [14]. Clark uses the equivalent approach of dividing by how common a word is. The result (in either case) is that less relevance is given to very common words

like "the", and more relevance is given to uncommon words. Finally, by accounting for how long the documents are by dividing by the product of the lengths of the document vector and the query vector, we obtain the following equation for the similarity [4]:

$$Sim(\vec{d}, \vec{q}) = \frac{\vec{d} \cdot \vec{q}}{\| \vec{d} \| \cdot \| \vec{q} \|} = Cosine(\vec{d}, \vec{q}) \tag{2.1}$$

From this equation a Term-Document Matrix can be created [4]. This is done by letting each word be represented by a row, and each document be represented by a column. The columns are thus the document vectors, but the rows become vector representations of the different words. As such, the word vectors can be compared in the same way as the documents in equation 2.1. The use of documents for the columns can be changed to some other kind of defined context (such as the x words before and after a target word), giving a more general approach for creating the word vectors. Thus, this approach gives a way for a program to create representations of words that can be compared with each other, and does not require any prior knowledge of the grammatical structure of the language.

While there are several different ways to improve upon this approach, this captures the main idea behind vector space models for lexical meaning. Several of the improvements also utilise the grammatical structure of the language. As this thesis focuses on first language learning, methods that require such knowledge about the language are not comparable and will thus not be explained here. For a more detailed description of vector space models for lexical meaning see Stephen Clark's paper in [4].

2.3 Human Language Learning and Babbling

The way that human children acquire their first language has been studied by psychologists for many years. While both the genetics and the environment that allow humans to learn language are more complicated than the Animat and its world, there are some aspects of human children's language learning that could be relevant also for an Animat learning language. For example, it is believed that by babbling and listening to the sounds they produce, human babies map which movements of their speech articulators result in which sounds [15, p.124].

An interesting thing to note is that human children tend to imitate the sounds made by adults [15, p.124]. Very young children (4-6 months old) start off producing sounds from all languages. Later, however, their babbling becomes focused on only those phonemes that appear in the language that they have been exposed to [16, p.395].

2.4 Short Term Memory and Chunking

Human memory and its mechanisms have been studied within the field of cognitive psychology. One famous finding within this field is that the short term memory of humans appears to have limited capacity; it seems that there is a limit on how much information we can hold in our short term memory without forgetting parts of it. Different researchers have explained this limited capacity of the short term memory differently; some suggest that there is a limit on the number of items or chunks of information that can be stored, while others have suggested that there might be a time limit, and that we forget because an item can be kept in the short term memory only for a limited time without repetition. Different numbers for the limit, of the maximum number of items that can be stored in the short term memory, have also been proposed. One such example is the number 7 plus or minus 2, suggesting that people can remember 5 to 9 items. Another number which has been suggested is 4, or 3 to 5 items [17].

To remember more information, individual items of information may be combined together into larger units (chunks) that have meaning. This is what is known as "chunking". Often used examples of chunking are how the long sequence of digits in a telephone number is often remembered as several groups of numbers (chunks), and how it is easier to remember a sequence of letters if you can find acronyms in it. For a concrete example of the latter, in the paper “The magical number 4 in short-term memory: A reconsideration of mental storage capacity” by Cowan [17] the sequence “fbicbsibmirs” is used to illustrate this, as this sequence of letters is more easily remembered chunked into the familiar acronyms “fbi”, “cbs”, “ibm” and “irs”.

Chunking has also been explored in relation to language learning. A string of words contains more information than a string of letters, in that each word in itself is a string of letters. Since these letters are chunked into a word, remembering words allows us to keep a larger quantity of information. The ability to chunk information (for example letters into words) is thus important for our ability to process information, and can prove to be useful for learning. One study on this theme was made by Xu in [18], who explored how chunking affects Second Language Acquisition. In this study, a group of students at Qingdao University of Science and Technology were given lectures on the theory of short term memory and chunking. The students were also taught to apply chunking to listening comprehension. The results of the study suggest that the students who received these lectures performed better at listening and reading in their English examinations than those students who had not been taught how to use chunking.

3

Method

To achieve the goals of the project, we implemented two slightly different versions of an Animat, both using the Python programming language. Both versions existed in a simpler kind of environment, since an extended spatial world was not required for testing the Animat's language skills. Instead the focus of the implementation was on the functionality necessary for learning and testing language. The functionality of our first Animat version was implemented and tested in several steps, where the last two were the goals of the project. After this work was complete, we made some modifications to create our second version, and tested it using the same evaluation method.

The two versions of the Animat are very similar, with the difference between them being how sequences of individual letters are processed into words. The first version of the Animat (the Temporal Animat) does this through an extension of the sequence nodes which we have called temporal sequence nodes. The second version of the Animat (the Chunking Animat) has no temporal sequence nodes, and instead uses a chunking rule to bind letters into known sequences (words) in the Animat's short term memory.

We explain our implementations in the order in which they were created, and therefore start this chapter by describing the basic implementation of the Temporal Animat. We explain how the functionality for meeting the two goals of the project was implemented, going over the different steps of the work in chronological order. Thereafter, we describe how our implementation was modified to create the Chunking Animat. Finally, we describe the vector space model which was used for evaluation.

3.1 Base Implementation of the Animat

This section describes the basic implementation of the Temporal Animat. The design mostly follows the structure used in the Animat project [3] and described in chapter 2.1. Where deviations have been made, the changes are described and compared to

the original version.

We will begin by explaining how the Temporal Animat relates to its environment and how it can receive information about its world through its sensors. Next, the Animat's state at the time of its instantiation is explained. Thereafter a more detailed subsection on the Animat's sequence nodes follows, since these nodes are of special interest in this project, as they are used for representing words. Lastly, we will describe the matrices that have been modified and/or added to this implementation of the Animat.

3.1.1 Environment

The Animat was made to exist in an environment which could contain the different concepts that the Animat could potentially perceive. Since our Animat did not need to move through a world to learn words, we did not model the environment to have different positions in space. Instead, in its simplest form, the environment only had a set of sensory input that the Animat could perceive. This sensory input could be text symbols that the Animat should learn to recognise as parts of a language, but also other concepts an animal could perceive, such as heat, cold, light, colours, smells, etc. The content of the environment was made so that it could change over time. An Animat could perceive what was in the environment at a given time, given that it had the sensors for perceiving these specific things. When the Animat acted, for example by speaking, it could create output (such as a text symbol or a word), which would appear in the environment in the next time-step. In this way, the Animat could perceive the effects of its actions. The Animat could only create output that it had motors for, or sequences of such output. In this project the Animat was limited to only have motors for letters. The environment could also be made to contain sensory input which was provided by the user. This was not limited to letters; it could also be sensations representing temperature, colours etc., for example the feeling cold or the colour green. If several Animats would exist in the same environment, they would all be exposed to the same sensory input at any given time. All content in the environment can be accessed directly, allowing for a human user to see what the Animat sees. Thus, both the input to, and output from, the Animat can be printed and analysed.

The environment was also extended to contain a list of input, called 'temporal input', with the purpose of having ordered input within a time-step, such as the letters in a word. This temporal input list was used to represent input that the Animat should be able to perceive within a single time-step, but that should still be ordered. Each such temporal input would be given to the Animat in, what we will refer to as, a temporal time-step within a single time-step. For example, a word would consist of a number of letters, and the order of the letters is relevant here, but it is perceived within one single time-step. The reason for this was to be able to give the Animat a word in one time-step, regardless of how long the word was. How the Animat

processed this 'temporal input' will be explained later in this section.

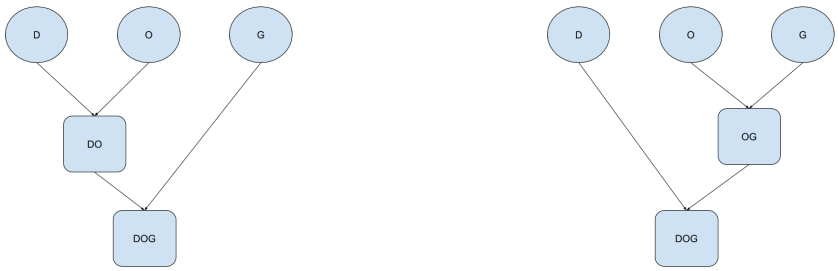
3.1.2 Instantiation of the Animat

As previously mentioned in section 2.1, an Animat has a graph consisting of perception nodes and action nodes. As a new instance of an Animat was created, the graph was made to consist only of some specified set of sensors and motors, all of which were unconnected. Other than the sensors, each of which could perceive some specific type of input (for example the letter 'a', the feeling 'cold', or the color 'green'), the Animat was also given a single True node, which is a sensor node that is always active, regardless of what is present in the environment. As the Animat gains new experiences and learns, new composite nodes (such as SEQ nodes, AND nodes, ASEQ nodes and AAND nodes) could be added and removed. How nodes were added, for the Animat to achieve the goals of the project, will be further described in the subsections 3.3 and 3.4.

In addition to the sensor and motor nodes, the Animat instance also had a set of matrices for storing information about the frequency of different perceptions over many time-steps, and two slightly different short term memories that keep track of only very recent perceptions. The matrices will be described in section 3.1.4, and how they were used will be explained in the rest of this chapter. The two short term memories of the Animat are a short term memory and a temporal short term memory. The short term memory stores which perception nodes have been top active in recent time-steps. In the Temporal Animat, the size of the short term memory is defined through a parameter specifying for how many time-steps back the top active nodes are stored. Similarly, the temporal short term memory stores top active perception nodes for a number of temporal time-steps. The short term memory is updated at each tick, while the temporal short term memory is updated at each temporal tick. For a node to be stored in the short term memory, it needs to be top active at the end of a time-step, thus not storing the nodes that were top active during the earlier temporal time-steps of the time-step.

3.1.3 Sequence nodes

In this implementation, the SEQ nodes were made to work in a way that allowed for, what was deemed as, a more natural tree structure of sequences. More precisely, the trees formed by the sequence nodes have been altered so that their subtrees can have arbitrary size, while in the original version there could be some problems if the right subtree did not consist of a sensor. To understand why this is, we consider an example. Each perception node has an activation time, which is the number of time-steps it takes for it to become active. For a sequence node representing a word, the activation time should be equal the length of the word, since one letter is given in each time-step. For example, a sensor for the letter "d" has activation



(a) Figure showing a sequence node taking a sequence as its first input, and a sensor as its second input.

(b) Figure showing a sequence node taking sensor as its first input, and a sequence as its second input.

Figure 3.1: Displaying sensors as circles (with the sign they detect written on them) and sequences as squares (with the sequence they detect written on them)

time 1, a sequence node for recognising "do" would have activation time 2, while the sequence node for recognising word "dog" would have an activation time of 3. The original version of the sequence nodes combined two nodes that were active in time-step $t-1$ (the left input) and in time-step t (the right input). This meant that to combine two non-overlapping parts of a word, the right input node could take no longer than one time-step to become active. Therefore the word "dog" could be formed by the nodes for "do" and "g" as seen in figure 3.1a, with the sensor for "d" being active in time $t-2$, "do" being active in time $t-1$ and "g" being active in time-step t. However, the word "dog" would not be recognised by a "dog" sequence node formed by "d" (left input) and "og" (right input), as seen in figure 3.1b. For this node to become active, "d" would have to be active in time-step $t-1$, and "og" in time-step t, meaning "o" would have to be heard at time $t-1$, at the same time as "d". In other words, the original version of the SEQ nodes would become active if and only if the second input was active exactly one time-step after the first input was active. If the second input was a sequence it would need to begin before the first input became active. To avoid this, the second node had to be a sensor with activation time 1.

The modification to the sequence nodes made it possible for a sequence node to use another sequence node, with a longer activation time than 1, as the second input, and become active if and only if that node becomes active as soon as possible after the first input. That is, if the second input node represents a sequence of length 3, then the sequence node taking it as input will become active if its first input was active 3 time-steps before the second input (the sequence of length 3) became active. To be able to create these nodes, the Animat made use of its short term memory to look at what nodes were active a number of time-steps previously. This however meant that the length of the short term memory determined the maximum time that could pass between the first and second input to a new sequence node.

In addition to the SEQ-nodes, the implementation of the Temporal Animat had a type of sequence nodes called 'temporal sequence nodes' (TSEQ), which were used to perceive the 'temporal input' in the environment. The TSEQ nodes function in the same way as the SEQ nodes described above, except that they can be updated several times within a time-step, more specifically once per element in the temporal input list. Each such update of all the TSEQ nodes will in this report be referred to as a 'temporal time-step'. As such, a single word could be processed by the Animat within a single time-step, but through several temporal time-steps. In one temporal time-step, one letter was processed, and temporal sequence nodes could thus be used for recognising single words as sequences of letters within a time-step.

3.1.4 Matrices

As mentioned in section 2.1, the Animat stores information about how often nodes have been active/top active in relation to each other in a set of matrices. This information is used for predicting, based on past experiences, the outcomes of different actions. The graph and the matrices are updated in each time-step (or in some cases, in each temporal time-step), to add the experiences of the last time-step. This is what constitutes learning. The matrices we used in the Temporal Animat implementation were a simplified version of the transition matrix and a conditional matrix (as described in section 2.1). Furthermore, we also implemented a temporal sequence matrix and an extended conditional matrix. In addition to this, the Animat had a list of generators.

In our project, we used the transition matrix only for finding generators for sensors, i.e. to find which actions results in which perceptions. How this was done will be described in section 3.2. For this purpose, it was enough to use a simplified version of the transition matrix described in 2.1. This version had entries only for the True node as the first index. This was done to save memory space and make the simulations run faster, without any loss of generality, as the normal matrix could be used just as well in the code.

Generators for perception nodes that consist of more than one sensor, such as sequence nodes, could be added without looking at the transition matrix, as will be described in section 3.4. However, all generators a would be removed from the list of generators for a perception node b if the transition matrix indicates that the execution of action a does not result in b being active in the next time-step.

To be able to create temporal sequence nodes (TSEQ nodes) we added a new matrix, called the temporal sequence matrix. The temporal sequence matrix is similar to the sequence matrix described in 2.1, but the temporal sequence matrix is updated in each temporal time-step. It has entries for combinations of two perception nodes. For two perception nodes b and b'

$$TempSeq(b, b')$$

denotes the fraction over the last 100 temporal time-steps that b has been top active at temporal time-step t and b' has been top active at temporal time-step $t - A(b)$, where $A(b)$ denotes the time it takes for b to become active.

To be able to associate between words, we wanted to store which perception nodes had been active at the same time, or at almost the same time. For this purpose, an extended conditional matrix was added. The extended conditional matrix is similar to the conditional matrix described in 2.1, but while the conditional matrix stores the probability that two perception nodes are top active at the *exact* same time, the extended conditional matrix instead stores the probability that two perception nodes are active at *approximately* the same time. For perception nodes b and b'

$$ExtendedCond(b, b')$$

denotes the probability that b is top active in time-step t given that b' is top active in some time-step t' which is within some fixed number of time-steps from t. This number of time-steps was determined by the size of the short term memory, as the short term memory was used to store the information on what nodes were active in these time-steps.

3.2 Step 1 - Mapping Motor Nodes to Letters

Once the Animat had the basic functionality required for processing text, we proceeded with the first step in teaching the Animat language, that is, learning how to produce what letters. Since this step was about teaching the Animat to anticipate the result of activating single motor nodes, no new nodes were added in this step, but the Animat's transition matrix was updated. This was done through a method inspired by human children's babbling. To begin this process, the Animat is set to activate a randomly chosen action node (in this case a motor node, since there are no other action nodes to start with) in each time-step. The result of this is that the Animat "hears" the text symbol which is produced by this motor in the next time-step; that is, the sensor for perceiving the text symbol is activated. As part of the Animat's functionality for general learning, the probabilities in the transition matrix (along with other information structures) are updated in each time-step, to take into account the results of the action performed in the last time-step.

The transition matrix is then used by the Animat to update its list of generators. This is done by the Animat looking at the transition matrix for transitions from the True-node, $Trans(True, a, b')$. Every action is added to the list of generators of each perception node that has always been active after that action has been taken. That is, an action a is added to the list of generators, as a generator for the node b', if $Trans(True, a, b') = 1$. If an action a which is listed as a generator of perception node b' has been taken, and $Trans(True, a, b') \neq 1$, then a is removed from the list, as a generator for b'.

Below the flow of the algorithm for mapping motor nodes to letters is shown as a list.

1. Update sensor nodes to perceive input
2. Update Transition matrix for previous action (if any)
3. Update list of generators based on Transition matrix
4. Activate random motor
5. Store the random motor as the previous action

3.3 Step 2 - Recognising Words

The next step in making the Animat capable of learning language was to make it able to recognise words. The Animat's ability to recognise a word or situation is dependent on what nodes it has, that become active in that situation. Therefore, the purpose of this step was to add functionality enabling the Animat to add nodes for common sequences of letters to its graph.

The way that the Animat was made to add nodes, was through probabilistic learning. This meant that, in each time-step, the Animat decided to learn with some probability $p_{seq-formation}$. This parameter could be lowered to make the Animat be more tolerant to noise, or increased to make the Animat learn quicker. If the Animat decided to learn, it would pick a pair of nodes b and b' with probability proportional to the value of the entry $TempSeq(b, b')$ and such that the node "(b TSEQ b')" did not yet exist. If such a node already existed for the selected pair, the Animat would attempt the same thing again, until success or until a fixed number of attempts had been made. A parameter 'SEQ formation attempts' was used to specify how many attempts an Animat would make. If a pair (fulfilling the requirements stated above) was successfully found, then the Animat would add a new node, "(b TSEQ b')", to its graph.

In this way the Animat was then able to learn sequences of letters (or sequences of sequences), thus forming words. For example, the word "cat" would be recognised by a sequence node of the form (writing the sensors as the letters they react to): "($'c'$ TSEQ ($'a'$ TSEQ $'t'$))" or "(($'c'$ TSEQ $'a'$) TSEQ $'t'$)".

Below the flow of the algorithm for creating perception nodes for recognising words is shown as a list.

1. With probability $p_{seq-formation}$ attempt to create a new TSEQ-node

1.1. If the maximum number of attempts is exceeded, go to step 2

1.2. Select a pair of nodes b and b' with probability proportional to $TempSeq(b, b')$

1.3. If a node (b TSEQ b') already exists for the selected pair, return to step 1.1

1.4. Create a new node (b TSEQ b')

2. Update sensor nodes to perceive input

3. Update experience matrices

3.4 Step 3 - Producing Words

The next step in the implementation of the Animat was to add functionality for producing words, that is, for activating sequences of motors. This was done by extending the algorithm which adds a perception node for recognising a learnt word to also, whenever possible, add an action node for producing the same word.

In practice this meant that whenever a temporal sequence node (b TSEQ b') was added for a new sequence of perceptions b and b', the Animat tried to create a corresponding action node ($g(b)$ ATSEQ $g(b')$) where $g(b)$ and $g(b')$ were generators for b and b'. This action node was then added as a generator for the newly created perception node (b TSEQ b'), adding a new way for the Animat to find new generators. This could of course only be done if the Animat had known generators for b and b'. If it did not, no action node was added, meaning the Animat could recognise the new word but not produce it. In this case, a generator for the word could still later be found by looking at the transition matrix in the way described in section 3.2, if a such a node had been added.

Below the flow of the algorithm for creating perception nodes for recognising words and action nodes for producing them is shown as a list.

1. With probability $p_{seq-formation}$ attempt to create a new TSEQ-node

1.1. If the maximum number of attempts is exceeded, go to step 2

1.2. Select a pair of nodes b and b' with probability proportional to $TempSeq(b, b')$

1.3. If a node (b TSEQ b') already exists for the selected pair, return to step 1.1

1.4. Create a new perception node (b TSEQ b') and an action node ($g(b)$

ATSEQ $g(b)'$)

2. Update sensor nodes to perceive input

3. Update experience matrices and list of generators

3.5 Goal 1 - Associating Words to Words

After the three steps needed for the goals were completed, work on Goal 1 could begin. The idea was that the Animat should learn to associate words to words in a way inspired by vector space models for lexical meaning (see section 2.2). However, due to the way that the Animat learns words, it needed to do it in a slightly different way.

To be able to make associations, the Animat used the extended conditional matrix (which stores the probabilities of two nodes b and b' being top active at approximately the same time). The first step in adding functionality for associations was thus to implement the updating of the extended conditional matrix. A temporal short term memory had already been implemented in the earlier steps, and in a similar way a normal short term memory was now implemented. This short term memory was updated each time-step to store what nodes had been top active during which of the last X time-steps (where X was a variable defined when the Animat was initialized). This short term memory was then used to update the extended conditional matrix. At the end of every time-step the entries of the extended conditional matrix were updated for every pair of nodes b and b', where b was top active that time-step and b' was among the nodes stored in the short term memory. If a node was stored as top active during several different time-steps in the short term memory, the entry would be updated several times as well.

By constructing the extended conditional matrix this way it became similar to the Term-Document Matrix described in section 2.2, but with some key differences. One of the differences was that the rows and columns represented nodes, that did not necessarily represent words. Another difference was that it could handle several nodes being top active at the same time. And thirdly, it contained rows and columns for all the parts that made up the words (and possibly several nodes that represented the same word in different ways), making it larger than the Term-Document Matrix.

Despite the differences from the Term-Document Matrix, the associations from a word were made in a similar way. The Animat was given a function that would return the 10 most highly related nodes to the (longest) currently top active node. When the Animat was asked for associations it would use the rows of the extended conditional matrix as vector representations of the meanings of the nodes. It would then find the most similar rows to the row of the target word, by using the Cosine measure between the rows (vectors).

The Animat had an update method that was called once for every time-step. It also had a number of parameters that could be modified during a run to control its behaviour, such as if it should babble, if it should learn associations and the probability for forming new SEQ nodes. In this way the Animat could, for example, be made to babble for the first time-steps, and then be made to stop when we wanted to start giving it words to learn. The update method in the Animat worked as follows:

1. Only update if it has not updated this time-step.
2. If the Animat is not set to babble:
    - Use probabilistic learning.
3. Update which nodes were active last time-step.
4. If the Animat is not set to babble, then for each element in the temporal part of the environment:
    4.1. Update (temporal) all temporal nodes.
    4.2. Update the temporal sequence matrix.
5. Update (normal) all nodes.
6. Update the transition matrix and the list of generators.
7. If the Animat is set to learn associations:
    - Update the conditional matrix.
8. If the animat is set to babble:
    - Activate a random action node.

3.6 Goal 2 - Associating Words to Sensory Input

The second goal of the project was to make the Animat capable of making multimodal associations. The idea was that this should be done in a similar way to the first goal. However, some slight changes were needed in order for the Animat to be capable of making both word-to-word associations and multimodal associations in the same way.

There were two major differences between the words associated between in the first

goal, and the senses associated from in this goal. The first was that the words were represented by 'temporal sequence nodes' that took two nodes as input and were updated in temporal time-steps, whereas the senses were represented by sensor nodes that could be updated in both kinds of time-steps. The second difference was that the Animat had generators for the words (and could thus produce them), but did not have generators for the other senses. This meant that the Animat could no longer produce all the associations that it made.

Due to the general way that the algorithms used in the model had been designed, no change was needed to functions in the Animat to account for the first difference. To account for the second difference, the association function in the Animat needed to be further generalised.

As the idea was that the Animat should return only associations that it could generate/produce, some of the associations made by the Animat would no longer be valid as output from the association function. As such, some nodes needed to be removed from the associations returned. Namely those nodes for which the Animat did not have any generator. To ensure that only valid associations were returned, the Animat was made to return a list of Action nodes instead of a list of Perception nodes. In the first goal this did not make any difference, as all perception nodes had a generator (see Step 1 and Step 3), and as such this generalisation would not change the results of Goal 1.

For the purpose of evaluating the Animat's understanding of words, a reverse method was also implemented that returned the highest associated perception nodes that the Animat did not have generators for. This was used to make the Animat associate from a word to its meaning, i.e., non letter sensors.

As such the functionality implemented for Goal 2, was a generalisation of the functionality implemented for Goal 1. Thus the Animat could make associations between words and other sensory input in the same way as it could make associations between words and words.

3.7 The Chunking Animat

A second version of the Animat, which did not have TSEQ nodes and temporal time-steps, was also implemented and evaluated. In this implementation, each letter of a word was given within one time-step. All sequences consisted of SEQ-nodes, and could theoretically extend beyond words and parts of words to phrases, sentences, etc.

The idea behind this version of the Animat was to only keep the longest recognised sequence in the short term memory, and to discard pieces of information that are part of a longer sequence. In other words, the Animat would chunk the contents of

its short term memory together when they were part of the same sequence.

As a basic example, we can consider a Chunking Animat which has learned the word "cat" as a sequence of a perception node which recognises the sequence "ca" followed by the sensor for recognising "t". At the beginning of the word the sensor node for recognizing "c" would become top active. This node would then be stored in the short term memory, along with information about the time-step in which it was top active. In the next time-step, the Animat would hear "a" and then the sequence node for recognising "ca" would become top active. Since the "c" node is input to the "ca" node, the "c" node would be removed from the short term memory, i.e. the information would be chunked. Next, the Animat would hear "t", the node for "cat" would become top active, and then "ca" would be discarded. Then the Animat would simply remember "cat" having been top active, and keep no record of "c" and "ca" also having been top active. This would mean that the word "cat" only takes up one space in the short term memory, even as it was heard over three time-steps. This chunking functionality enables the Animat to keep several long sequences in its short term memory, and thereby retain information about which words and phrases have been heard after one another, while treating all sequences in the same way, regardless of whether they make up words, parts of words, or multiple words in phrases.

The algorithm the Animat used for chunking was to, in each time-step, call a chunking method before inserting a new element into the short term memory. Each element in the short term memory contained a set of nodes, and also stored how many time-steps ago it happened. In the Temporal Animat, the nodes in an element at position n in the short term memory were simply the nodes that were top active n time-steps previously. In the Chunking Animat, which nodes were inserted into the short term memory in each time-step, was decided by the chunking algorithm. The chunking algorithm worked as follows:

1. Find the currently top active node (b) with the longest activation time.
2. For every element (e) in the short term memory:
    2.1. Find the node (b') in e with the longest activation time.
    2.2. If the start of the sequence b was before the start of the sequence b':
        - Remove e from the short term memory.
3. Create a new element to insert into the short term memory, containing:
    3.1. All currently top active nodes.
    3.2. All active nodes that are input to the node (b) and have an activation time of 1.

4. Return the newly created element.

The reason that some nodes that were not top active were added to the new element in the short term memory, was to avoid the Animat getting stuck when learning some words. That problem could otherwise arise if the Animat learnt two overlapping parts of a word, such as the "ca" and "at" parts of the word "cat". The problem was that if the Animat only stored the nodes that were top active, it would store the nodes for the sequences "ca" and "at". But no combination of these two nodes can create the sequence "cat", and as such the Animat would not learn the word "cat". This problem did not exist before chunking was implemented as the Animat would then still remember the sensor for "c" being active before the two sequences, and be able to combine it with the sequence "at" to form the word "cat". The fix made it so that the last node in a top active sequence would still be stored in the short term memory, allowing for the Animat to combine the "ca" sequence with the sensor for "t", and thus creating the sequence "cat".

While the Temporal Animat could simply update its extended conditional matrix at the end of every time-step, this would be problematic in the Chunking Animat. The problem would be that this would make the Chunking Animat update the extended conditional matrix for parts of words while hearing them. To avoid this, the Chunking Animat was simply made to only update its extended conditional matrix when it couldn't chunk. The reason for this was that after perceiving a known sequence, the Animat would not be able to chunk it with the start of the next. Thus the end of every sequence would result in the Animat being unable to chunk. Therefore, the Animat being unable to chunk means that it is likely to be at the end of a word. And that is when the Animat should ideally update its extended conditional matrix. As the Animat now had a new element in its short term memory, it would update the extended conditional matrix for all elements in the short term memory except the newest. The newest might be a single letter word, but it could also be just the start of a word. Therefore the Animat should not update its extended conditional matrix for this element, until it was unable to chunk it with some later input.

For the Chunking Animat, two pieces of extra functionality were added. Each could be used independently of the other. The extra functionality was not needed for the algorithm to work, but were introduced to possibly improve the Animat's performance.

The first was a way to enforce the Animat to only use sequence nodes where the second input node had an activation time of one. The reasons for this were that the chunking algorithm worked better when all sequence nodes had this structure, and that adding this functionality would enable evaluation of how big impact the structure of the sequence nodes had on the performance of the Animat.

The second functionality was a way to filter out some nodes from being added to the short term memory if they represented the end of a longer top active sequence.

An example of how this would work can be seen if the word “cat” was represented as the sequence “ca” followed by the letter “t”, and the Animat also had a node for the sequence “at”. Then both the node for the sequence “cat” and the node for the sequence “at” would be topactive when the Animat had heard the word “cat”. But as the sequence “at” only is a part of the longer sequence “cat” this functionality would make the Animat not add it to the short term memory. This would result in only the node for the sequence “cat” being added to the short term memory.

3.8 Implementation of a Vector Space Model

To assess the quality of the associations made by the two Animat versions, they were compared to the associations made by a vector space model trained on the same data. For this purpose, we implemented a simple vector space model, as described in section 2.2. This vector space model was trained on a text containing X unique words.

How often two unique words occur in the same context in the input text is stored in a $X \cdot X$ matrix. Two words occur together if they are within a fixed number of words from each other (either before or after) in the text. This gives each word a context of twice this size. In this way, each entry $e(i, j)$ represents how many times the word j has occurred close to the word i. This number is then divided by the total number of times that j has appeared in the text, to account for some words being more common than others. Each row of the matrix can be seen as a vector representing one word. By comparing the distance $Cosine(\vec{u}, \vec{v})$ between the vectors $\vec{u}$ and $\vec{v}$ representing different words u and v, a value of the similarity between these two words is obtained. Similarity between words is here seen as being equivalent to their being strongly associated, and this value is then used for ordering the other words that occur in the input data by how similar they are to a target word.

As output, the vector space model gives, for each target word, a list of the ten words which are most strongly associated to this word. This list could then compared to the ten words that were deemed as most strongly associated by the Animat model.

4

Results

As stated in chapter 3, the project was split into three steps and two goals. The first section of this chapter presents the evaluation and results of the steps, along with some discussion, for the Temporal Animat. The second section presents the result of the goals in the same way. The last section presents the evaluation and results of the Chunking Animat, and compares the Chunking Animat to the Temporal Animat.

4.1 Result of Steps

The three steps (3.2 "Mapping Motor Nodes to Letters", 3.3 "Recognising Words" and 3.4 "Producing Words") were evaluated separately to get a view of how well the Animat managed the different tasks needed for the goals. The results of these evaluations are presented here. The following subsections will present the evaluation and results for each of them, along with some discussion of the result.

4.1.1 Result of Step 1

The Animat's algorithm for learning generators was evaluated by testing how well the Animat could find the generators for producing the letters of the English alphabet, through babbling. The Animat was given a sensor and a motor for each letter in the alphabet, in a random order. The Animat was then set to babble for a fixed number of time-steps. The number of sensors, for which it had correctly identified the corresponding generators, was recorded. This was then averaged over a 100 runs to account for stochastic noise. The tests were repeated for different values of how many time-steps the Animat babbled. The results are shown in figure 4.1.

As can be seen in figure 4.1, in the test where the Animat babbled for 100 time-steps, it identified all generators in 53% of the runs, and found on average 98% of the generators. When babbling for 200 time-steps, the Animat found all generators in 95% of the runs. And when babbling for 220 or more time-steps, the Animat

always found all generators.

The expected time to find all generators in this way can be found analytically by modeling the problem as an instance of the so called "the coupon collector problem". The expected time is then given by:

$$n \sum_{k=1}^{n} \frac{1}{k}$$

where n is the number of generators to be found [19]. As the number of generators to be found are 26, the expected time is between 100 and 101 time-steps. As stated, the Animat managed to find all the generators in 100 or less time-steps in 53% of the runs. From this it is clear that the Animat performs in accordance with the theoretical estimate. Thus the Animat is as fast as can be expected at learning generators through the process of babbling. And while this test was made for the letters of the English alphabet, due to the generality of the algorithm, the same approach can be used equally effectively to learn generators for any kind of sensor.

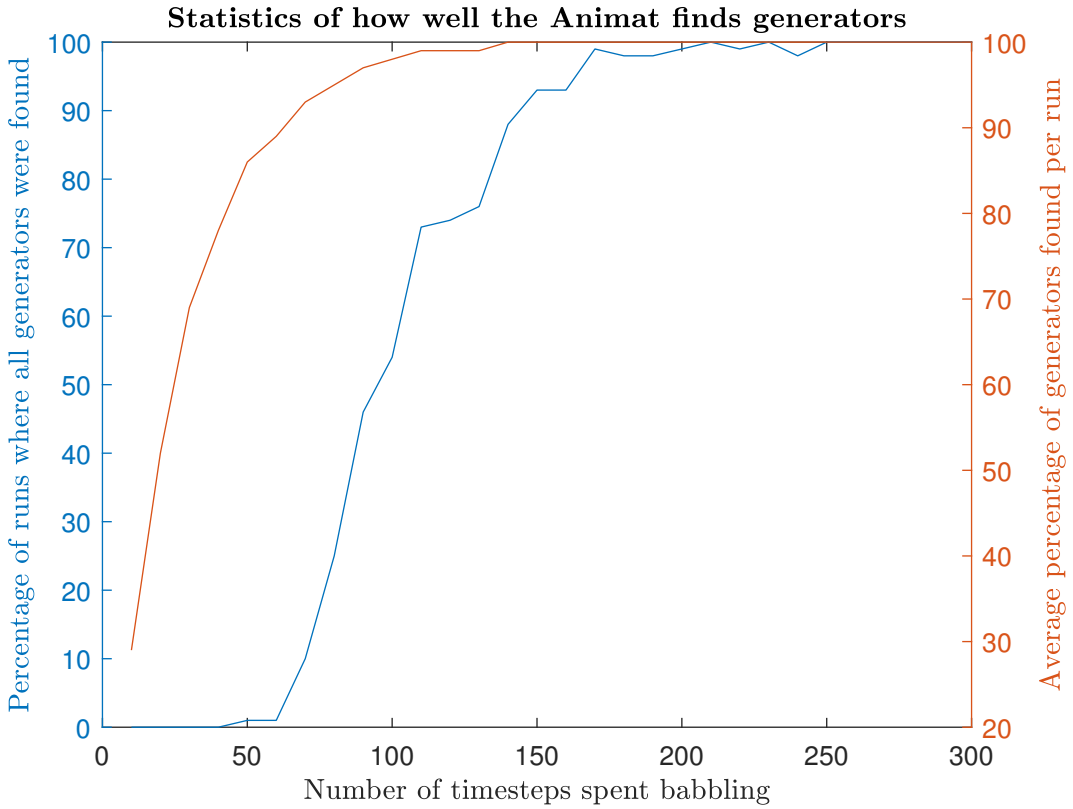


Figure 4.1: Plot showing how often the Animat is capable of finding all generators in the different tests (blue). And showing the average number of generators found in each test (orange).

4.1.2 Result of Step 2

The algorithm for learning words, as described in section 3.3, was tested on a set of 100 words. The Animat was updated $100 \cdot 20 \cdot 2$ times, so that every word was repeated on average 20 times. In every other time-step, the Animat was given a single random word drawn from the set with uniform probability. In the time-steps between, the Animat was given a single space/pause as input, thus separating the different words (and the reason for the 2 in the equation above). The value of $p_{seq-formation}$ was set to 1, as this model was free from noise. The Temporal short term memory capacity was set to 7, and the number of attempts to form a new (Temporal) SEQ node was set to 10. The values of the parameters used, are shown in table 4.1.

Table 4.1: The parameters used when testing the Animat's ability to learn to recognise words.

Parameter:	**Value:**
Total number of words	100
Average number of occurrences	20
SEQ formation probability	1
Temporal short term memory capacity	7
SEQ formation attempts	10

Figure 4.2 shows the results of such a test run. In this run, the Animat learned 95 of 100 words. The top right plot shows the distribution of how often the different words occured, and that they all appeared approximately 20 times each, and were thus treated equally. The bottom right plot shows the length distribution of the words, showing that the lengths of the words ranged from 3 to 12. The total number of perception nodes (red line in left plot) and the number of perception nodes representing real words that were to be learned (blue line in left plot) increased during the first 2132 time-steps. As can be seen, the program eventually reaches a point where no new perception nodes are added, even though there are more words left to learn.

The reason that the program reaches such a point is that some words that are longer than the Animat's short term memory capacity (7) can be blocked from being learned if parts of them are learned in a specific order. This problem occurs when the Animat first learns the last part of a long word, meaning this will be top active at the end of the word, rather than a shorter part of the word. For example, this can happen with the word 'elephant'. If the Animat has learnt the sequence 'lephant', then as it receives the last 't' of the word 'elephant' in temporal time-step 8, the node for recognizing 'lephant' will be top active. But since 'e' was top active a fairly long time ago (in temporal time-step 1), the limitation of the short term memory of the Animat means this information is already forgotten. Thus, even if the Animat has suitable parts to combine to form the entire word, such as perception nodes for recognising 'eleph' (active in temporal time-step 5) and 'ant' (active in

temporal time-step 8), 'ant' will never be top active if it is input to the longer word 'lephant', since 'lephant' is also active in temporal time-step 8. In this way, having learned 'lephant' may block the Animat from learning the entire word 'elephant'.

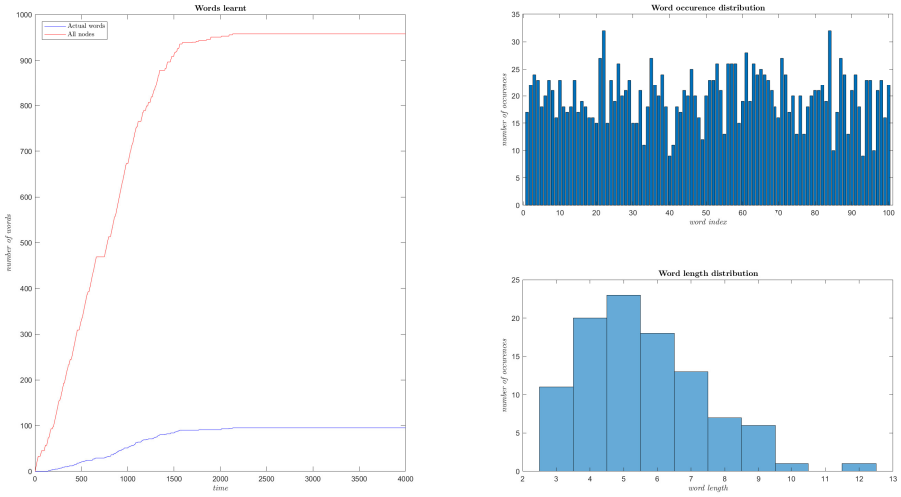


Figure 4.2: Left plot shows how the total number of perception nodes, and the number of perception nodes representing actual words, changes over time. Top right plot shows the distribution of how often the different words were given as input. Bottom right plot shows the length distribution of the words. (Scaled up versions of all three plots can be found in Appendix A.)

4.1.3 Result of Step 3

To evaluate the Animat's functionality for learning to produce words, the Animat was given input through a process similar to that used in previous steps: it was allowed to babble to map motor nodes and sensor nodes, and was then given 100 words to learn in the same way as in section 4.1.2. How many of the learnt words the Animat could also produce was then tested by using a repeat-method created specifically for this evaluation. This method would update the Animat, but also hardcoded the Animat to activate the generator (if any) of the top active node with the longest activation time. The reason that the node with the longest activation time was selected was that it would be the most detailed description of the situation. By calling this method, the Animat was thus made to try to repeat whatever word it heard last. The number of words that it could correctly produce was counted and compared to the number of words it had learned to recognise.

The result was that the Animat always learned to produce all the words that it had learned to recognise. As the Animat was set to babble until it had learnt all generators, it was never exposed to a word/sequence containing sensor input that it could not generate.

The reason that the Animat was set to babble until it had found all generators was that if the Animat did not have generators for all the parts of the sequence, it would not be able to create a generator for the sequence. As the Animat was capable of learning to produce all words as soon as it had learnt to recognise them, we can conclude that this part of the Animat's learning algorithm worked as fast as possible. As no assumptions were made about the sensors/motors in the sequences, this algorithm can be applied equally well to any sequence and not just words.

4.2 Result of Goals

The two goals of the project (3.5 "Associating Words to Words" and 3.6 "Associating Words to Sensory Input") were evaluated by comparison with the vector space model implemented in 3.8. The following two subsections explain the evaluation processes in more detail and present the results and discussion, first for Goal 1 and then for Goal 2.

It should be noted that execution time the vector space model was considerably shorter than that of the Animat. As the Animat was not optimized for speed, this was not considered one of the main points of the evaluation. Therefore it is mentioned here, but will not be discussed in detail in this chapter.

4.2.1 Goal 1 - Associating Words to Words

This section will describe how the Temporal Animat's performance at associating words to words was measured. We will begin by describing the the evaluation method used, and then continue by presenting the results. Lastly, we will discuss the Animat's performance and present some possible explanations for why its associations sometimes deviate from those made by the vector space model.

4.2.1.1 Evaluation method

To test how well the Animat fulfils the first goal, it was compared to the vector space model implemented for this purpose (see section 3.8). The Animat was first made to babble until it learned all generators for the sensors (letters) in the same way as in section 4.1.1. Then the Animat was given all the words in the input text that would be used for learning associations, in a random order. Each word was given to the Animat approximately 15 times, similarly to when the Animat was given words when evaluating step 2 (see 4.1.2). When learning the associations, the Animat and the vector space model were given identical input texts. The text used consisted of simple sentences, which were repeated in random order, and contained a limited

number of unique words. The idea was to use the kind of simple language which might be spoken to a human child learning its first words. The size of the context for each word was set to 4 (two words to either side), and the short term memory of the Animat was set to 6. As the Animat was given a space (as the only input in a time-step) between words, this meant that the number of words each model would take into account (when updating the row for a target word) was the same.

A score representing the accuracy of the Animat's associations was calculated based on the overlap between its associations and the associations of the vector space model. For each target word, the associations were compared. If a word y which was among the top ten associations made by the vector space model, was also in the Animat's top ten associations, the score was increased by one. By doing this for all target words, a score in the range $[0, 10 \cdot X]$ was attained (where X is the number of unique words). The score was then divided by the max score $(10 \cdot X)$ to get a percentage result. A probability distribution was extrapolated from the results of 14 independent runs, and is shown in blue in figure 4.3.

A second score was also calculated in a slightly different way. Instead of increasing the score by one for each word that appeared in the top ten associations by both models, the increase in score was dependent of how highly that word was associated by the vector space model. If the top word from the vector space model's associations was among the Animat's associations, the score would increase by ten. If the second highest word from the vector space model's associations was among the Animat's associations, the score would be increased by nine. Similarly for all the top ten words in the associations made by the vector space model until the last witch would increase the score by one. The second score was thus in the range $[0, 55 \cdot X]$. The score was then divided by the maximum score, and a probability distribution was extrapolated in the same way as for the first score. This distribution is shown in red in figure 4.3.

4.2.1.2 Result

As can be seen in figure 4.3, the mean of the first distribution (blue) is at 81.3833% overlap. The lowest score was 78.17%, and the highest score was 85.95%. For the second distribution (red) the mean was 91.2681%, the lowest score was 88.55% and the highest score was 93.51. The max value, min value, standard deviation and mean of the two evaluation calculations are shown in table 4.2.

4.2.1.3 Discussion

From the results above we can conclude that the Animat's associations mostly overlap with those made by the vector space model. Particularly, we can see from the red plot in figure 4.3 that the top associations of the vector space model almost always

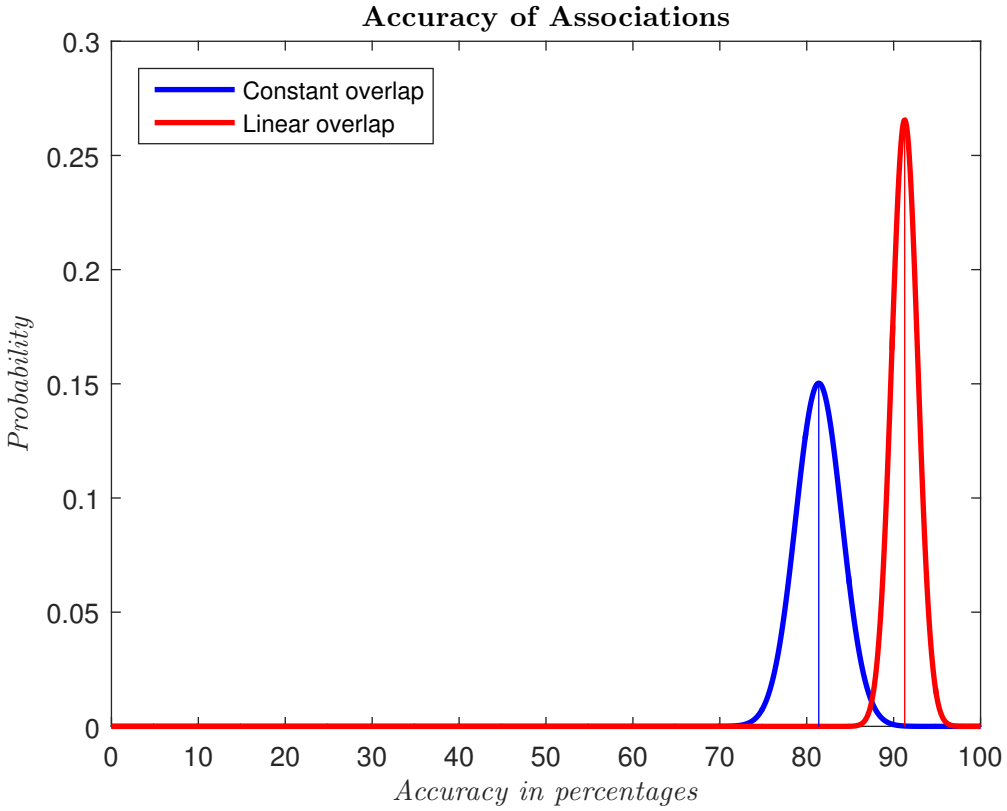


Figure 4.3: Plot showing the results of Goal 1. The blue plot shows the probability distribution of the Animat's accuracy when the same score is given for all associations of the vector space model, along with its mean. The red plot shows the distribution and mean for when the associations were worth different amounts depending on how highly they were associated by the vector space model.

were among the associations made by the Animat. Thus, it is clear that the Animat managed to give the most relevant associations with a high probability. As the same kind of mechanism for associating was used in both models (cosine distance), this is not a surprising result. However, it does give a measure of how much impact the noise introduced by the generality of the Animat's learning rules has. There are several explanations accounting for the differences that do appear between their associations. Some of these become evident when studying the associations for each word, and will be discussed in the rest of this section.

Firstly, the Animat sometimes returns the same word several times. This is because one word can be stored as several sequences. For example the sequence "cat" can be represented as the sequence "ca" followed by "t", or as "c" followed by the sequence "at". Having the same word several times is of course unnecessary. A way to get around this problem could be to extend the Animat's functionality to be able to delete one node if two nodes are equivalent (which would here mean that they recognise the same sequence). This could also potentially be solved through the use of a forgetting rule, as there would be no advantage to having two equivalent nodes.

Table 4.2: Table showing the max value, min value, standard deviation and mean of the two evaluation calculations for Goal 1 (Associating words to words).

	Constant Overlap	**Linear Overlap**
Minimum Value	78.17	88.55
Maximum Value	85.95	93.51
Standard Deviation	2.65449	1.50194
Mean	81.3833	91.2681

Secondly, by studying additional data which is stored in each of the runs, we can see that Animat does not learn all words, as has been discussed in section 4.1.2. However, it is worth noting that as the Animat uses probabilistic learning it can fail to learn words due to chance. And as the number of words increase the probability of this increases as well. One reason for this increase is that the Animat in the current implementation can select two nodes that are already combined, and thus fail to create a new node. If this possibility was removed by using a slightly more refined algorithm for selection, this issue would be resolved. Due to the limited scope of this project however, the current algorithm is deemed sufficient. If the Animat has not learnt a word, this word can of course not be returned as an association. This means that if measures were taken to improve the Animat's learning of words, it is likely that this could also improve the result of Goal 1.

Thirdly, with the data sets used for evaluation, some words might be equally associated by both the Animat and the vector space model. In these cases, the order of these words in the lists of returned associations would be arbitrary. And if the 10th and 11th most highly associated words to a target word were equally highly associated, the Animat might return a different word than the vector space model. However, it would still be an equally valid association as the one returned by the vector space model, as both would have the same cosine distance to the target word.

To summarise, we conclude that even if the Animat does not score perfectly, the results suggest that the method of using cosine distance for associations works in the Animat model, producing a similar result as it does in vector space models for lexical meaning. This suggests that the noise introduced by the generality of the learning rules in the Animat model has a fairly small impact on the accuracy of the Animat's associations.

4.2.2 Goal 2 - Associating Words to Sensory Input

This section describes how the Animat's performance at associating words to other forms of sensor input was measured. First, the evaluation method is explained. Thereafter, the resulting scores showing the Animat's accuracy are presented. Thereafter we discuss the result and the evaluation method.

4.2.2.1 Evaluation method

To evaluate how well the Animat can associate between words and non-word sensor input, two forms of tests were conducted. The first test compared the associations the Animat made from sensor input to words, with a predefined set of words. In the second test the Animat was made to associate from words to sensor input. For comparison, the vector space model was also made to associate between the sensor input and the words, and the associations it made were compared with those made by the Animat. However, as the vector space model was made specifically for words, it needed to get the input in a different way, as will be explained later in this section.

To conduct the tests a set of keywords were chosen, to which other forms of sensor input could be connected. For each keyword, a set of connected non-language sensor input was defined. For example, for the keyword “winter”, the sensor input was "cold" and "white". Such input, which represents perceptions for non-language will in this section be referred to as “sensations”, to distinguish them from the sensor input which activates the sensors for perceiving letters.

In the first test, the Animat was given an input text, and when a keyword such as “winter” occured in this text, the Animat’s environment would also be made to contain the perceptions defined in the set of sensations connected to this keyword. With the keyword “winter” the sensations would be “cold” and “white”. Thus, as the Animat perceived the word “winter”, it would also perceive the feeling “cold” and the colour “white”. One specific sensation could occur with several keywords, for example the Animat could also perceive the colour “white” whenever the word “white” occurred in the input text. The Animat was then made to associate to each sensation it had been exposed to, returning generators for perception nodes it associated to it. Since the Animat only had motors for producing letters, only generators for words and parts of words existed, and thus only generators for producing associated words (and no sensations) were returned. For each sensation, the associated words were compared to the list of keywords which were connected to this sensation. If a keyword in this list was among the top k associations returned by the Animat, the Animat’s score was increased, where k is the length of the list of keywords connected to the sensation. The maximum score would be reached if, for every sensation, the keywords connected to that sensation were at the top of the Animat’s list of associations. This max score was calculated and the Animat’s actual score was compared to it.

In addition to this, a second score was calculated, based on the reverse evaluation process, in which the Animat was given a keyword and then made to return an ordered list of all associations which consisted of perception nodes for which there were no generators (i.e. sensations).

To be able to compare the Animat’s performance to another system, the same text was given to the vector space model. Since the vector space model only takes texts as input, it could not simultaneously be given other input (sensations). Instead, the

vector space model implementation was trained on the input text repeated several times over. The first time, the text was unaltered, but as it was repeated, the keywords were alternately used and substituted with their connected sensations written within brackets. For example, the sentence "In winter the snow falls" would appear as such the first time. The second time it would be replaced by "In [cold] the snow falls", and the third time it would be "In [white] the snow falls". The fourth time it would again be "In winter the snow falls". The vector space model was modified to return, as associations for a word, a list the most highly associated words that were not in brackets (corresponding to the Animat's inability to produce non-words concepts). The associations made by the vector space model for all of the defined sensations was then compared to those made by the Animat. Then the vector space model was also used for the second test in a similar fashion. But in this test it was made to return associations that were words with brackets (i.e. sensations), for all the predefined keywords.

The results were averaged over seven runs, and probability distributions for the accuracy of the two tests were extrapolated. These distributions are shown in figure 4.4.

4.2.2.2 Result

As can be seen in figure 4.4, the mean of the Animat's score when associating from a sensation to word (blue, solid line) is 59.11. The standard deviation is 4.25. For comparison, the vector space model's score on this task (blue, dashed line) is 43.75. The mean of the Animat's score when associating from a keyword to a sensation (the red, solid line) is 93.57, with a standard deviation of 1.12. The vector space model's corresponding score (red, dashed line) is 80.0.

4.2.2.3 Discussion

From these results, we can conclude that on average the Animat performs better than the vector space model. However, it is important to note that due to lack of time the tests have been conducted on a limited set of input, and that further tests are necessary to further verify that these early results hold for different input data.

Another aspect to keep in mind is that for some sensations, there was only one or a few predefined corresponding keywords. As we evaluate by comparing to these predefined keywords, this could end up being a quite strict evaluation method. For example, in one test run there was one instance where the sensation "[light]" had keywords "summer", "day" and "sun". The Animat returned the word "lamp" among its associations, which was not defined as a keyword, and thus did not give contribute to raising the Animat's score. However, one could argue that intuitively, "lamp" is a reasonable association, as it often appears in the same context as light.

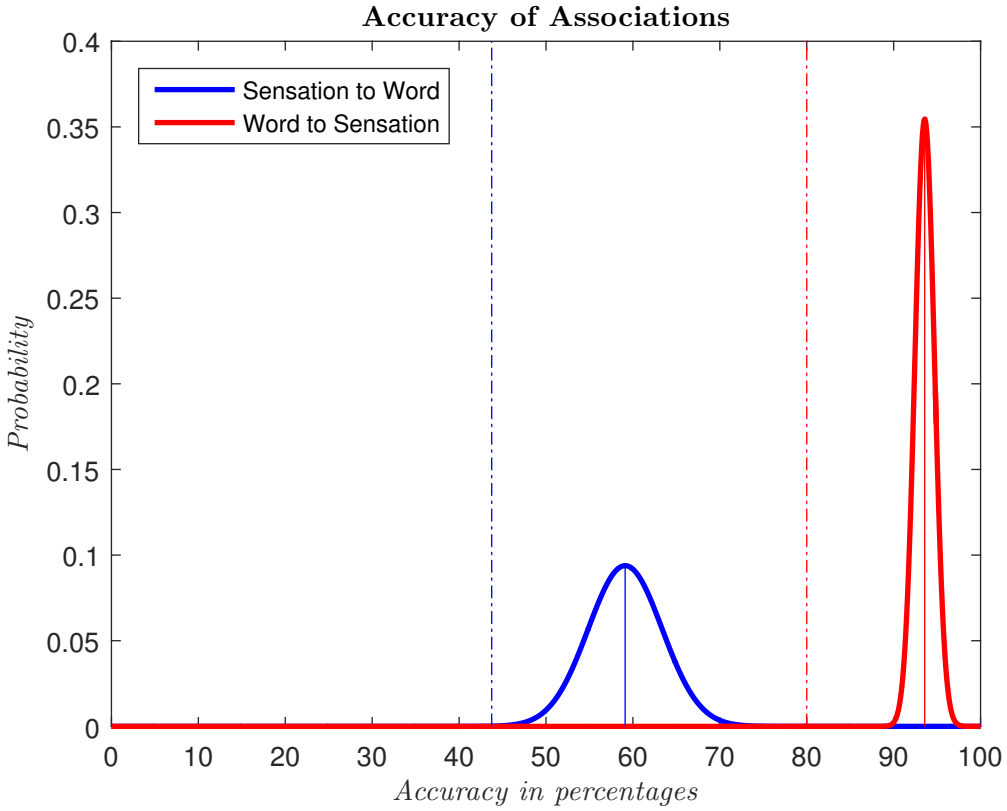


Figure 4.4: Plot showing the results of Goal 2. The solid lines show the probability distributions of the Animat's accuracy, with the blue line representing the result of the associations from sensations to keywords, and the red line representing result of the associations from keywords to sensations. The corresponding results for the vector space model is shown with dashed lines.

Considering this, it is worth keeping in mind that the evaluation method used might not perfectly capture what is to be measured.

It is also interesting to note that the Animat scored higher when made to associate from a keyword to sensations (word to other sensor input), than when made to associate from a sensation to words (other sensor input to word). The higher score here can be explained by the Animat not having been given as many sensors for non-language input as it was given words to learn. Thus, the Animat would had a limited number of nodes representing sensations, and a fairly large number of perception nodes representing words. Thus, when made to return the associated sensations to a keyword, the Animat simply had fewer nodes to choose from, and was therefore less likely to return an irrelevant association. To be able to draw more conclusions, further tests should be run, using an Animat with a larger number of sensors for non-language. The python dictionary used to connect words to sensations when giving the Animat and the vector space model input can be found in Appendix B.

When comparing to the vector space model implementation, it is also important to

remember that it does not get the input in exactly the same way, since the Animat can receive a word and multiple other perceptions at the same time. The vector space model of course was not designed for this purpose. In this comparison, the Animat had a limited number of sensors for non-language perceptions. If more such sensors were added for each keyword, it would eventually become difficult to adjust this to the vector space model. The way that the vector space model receives the input in this test would not work very well if the text was expanded and the number of sensations was increased; it would not be feasible to give the vector space model a sentence one more time for each added sensation that could be added to the Animat model.

In short, we can conclude that the Animat has a clear advantage against the vector space model when performing the kind of task given in Goal 2. While these results are encouraging, it would be necessary to do more tests, using more complicated sets of keywords and sensations, to determine how well it would work on a larger scale.

4.3 Performance of the Chunking Animat

This section will describe the performance of the Chunking Animat. We will start by describing the evaluation process, and then present the results. Finally we will discuss them, and compare to vector space models and to the Temporal Animat.

4.3.1 Evaluation Method

The Chunking Animat was evaluated for the first goal in much the same way as the Temporal Animat. It was not evaluated for the different steps independently, as they were parts of Goal 1. From the Chunking Animat's performance when evaluating Goal 1, it was therefore possible to discern how well it managed the different steps. As when evaluating the Temporal Animat, the context window of the vector space model was smaller than the short term memory of the Animat, as the Animat would store the spaces between words. The size of the context window was 4 (two words to either side), and the length of the Animat's short term memory was set to 6.

As stated in section 3.7, there were two pieces of functionality in the Chunking Animat that could be enabled or disabled to examine how they affected the performance of the Animat. The first was to make the Animat only create sequence nodes where the second input node had an activation time of one. The second was to make the Animat not add nodes to the short term memory if they represented a sequence of nodes that was the same as the end of the longest topactive node. The Animat was evaluated with different combinations of these extra functionalities. However, due to time constraints the number of runs for each combination was low. As the Ani-

mat has not displayed any large variance in earlier evaluations, the results obtained are still expected to be representative of the Animat's performance. The number of runs for each combination are shown in the results shown in table 4.3.

4.3.2 Result

As stated, how well the Chunking Animat managed the different steps could be discerned when evaluating Goal 1. In all runs, the Chunking Animat managed to find all generators within approximately the same number of time-steps as the Temporal Animat (with the differences in time being due to the randomness of the algorithm). The Chunking Animat was also capable of learning all words in some runs, but missing some in other runs due to random chance. Just as the Temporal Animat, the Chunking Animat was capable of learning to produce all words that it had learnt to recognise.

The results of the evaluation of Goal 1 for the Chunking Animat are shown in table 4.3 for the four different combinations of which of the extra functionalities were used. The results shown are the mean of the results obtained in different averaging runs.

Table 4.3: Results for the Chunking Animat for four different combinations of which extensions to use.

Extensions used	**Overlap**	**Number of runs**
None	27.73%	2
Limited Sequence Nodes	29.45%	2
Ignoring Subsequences	99.21%	2
Both	99.61%	2

4.3.3 Discussion

As can be seen in table 4.3, the Chunking Animat performed poorly in the runs without either of the extra functionality. One reason for this was that when the Animat failed to chunk parts of words, it would fill its short term memory with parts of words, and start to forget (remove from the short term memory) the words that came before. However, the main reason for the low score was that the Animat would update its extended conditional matrix when failing to chunk. This would occur at the end of a word, as was intended, but it would also occur during a word if the representation of the word contained at least one sequence node that had another sequence node as it's second input. The Animat would then, for at least one time-step, be unable to chunk the short term memory. This would result in the first input to any such node getting a vector similar to the word it was part of. This would in turn would lead to the Animat associating that part when asked for associations for

the whole word. An example of this would be if the word “winter” was represented as the combination of the sequences “wint” and “er”. The Animat would not be able to chunk “wint” with “e”, and therefore would update its conditional matrix for the node representing the sequence “wint”. This would make the sequence “wint” have a vector representation similar to that of “winter”. As the vector representations of the sequences “wint” and “winter” would be similar, the Animat would associate them to each other. This problem was expected, and the reason that the option to limit what sequence nodes the Animat created was introduced.

When the Animat was limited to only create sequence nodes that took a node with an activation time of one as their first input, it obtained an accuracy of 29.45%. While slightly better than the first evaluation, this is still a low degree of accuracy. This is because, while the Animat manages to chunk entire words together, the Animat now has a higher probability of having top active nodes representing the last parts of sequences. For example, as the word “winter” would be represented as the sequence “winte” followed by the letter “r”, the Animat could also have nodes for sequences such as “er” or “ter” as top active at the end of the word. And as they were top active at the end of the word, their rows in the extended conditional matrix would be updated. This would result in the rows for “winter”, “ter” and “er” becoming very similar. While this problem could also occur when the Animat was not limited in its creation of sequence nodes in this way, the probability for this happening was noticeably increased when this functionality was added. This issue was the reason that functionality for making the animat not add subsequences to the short term memory, was added.

When the Animat was also made to use the functionality for keeping nodes representing the end of top active sequences from being added to the short term memory, the result improved considerably. We now see that the Chunking Animat performed almost perfectly, with an overlap of 99.61%. While extra functionality was needed for this to work, all added functionality is made to be general and can be used for other tasks than language. However, the enforcement of all sequence nodes using only nodes with activation time one as their second input is a severe limitation of the model, as it drastically increases the amount of nodes needed.

When the Animat used the functionality for avoiding adding subsequences to the short term memory, but was allowed to create sequence nodes using nodes of any length as their second input, it obtained an overlap of 99.23%. While not quite as good as when using both of the added functionalities, it still produces almost identical associations to the ones made by the vector space model. And the small difference from the previous settings could potentially be due to random chance, as the number of runs for the simulations were low. As this version does not limit how the Animat creates sequence nodes, it is more general and therefore the overall better version.

From the results, we see that when only using the same basic functionality as the Temporal Animat, and the simple chunking rule, the Chunking Animat performs

quite poorly. However, when extended with a simple rule for not adding subsequences to the short term memory, it manages to obtain almost perfect results. As this functionality was not included in the Temporal Animat, the results of the two models can not be directly compared. It is likely that adding such functionality to the Temporal Animat would have improved its results as well. The Chunking Animat does not need to use any special sequence nodes (TSEQ), and is therefore a more simple and clean approach. Thus we conclude that the Chunking Animat is the preferable model of the two, as it is more general.

5

Conclusion

We start this chapter by providing some general discussion about the project. We then continue by looking into what possible extensions and further work might be interesting. We finish with a conclusion and summary of the entire project.

5.1 Discussion

We start this discussion by evaluating the strengths and weaknesses of our model, and by motivating some of the decisions that were made during the project. We then look at the evaluation process, and consider its strengths and weaknesses. Finally we give a general discussion about our results.

5.1.1 Model

In this section we will discuss the way that sequence nodes were used in this project, and why they are of interest in a more wide setting. We also consider the handling of nodes in general and how the model scaled as the number of nodes increased. We discuss what advantage the Animat has when compared to vector space models. Finally we motivate why the Chunking Animat was of interest, and what advantages it has over the Temporal Animat.

5.1.1.1 Why we have focused on sequence nodes

As our thesis explores language learning within the Animat model, much of the focus has been put on the sequence nodes of the Animat, which have been used for modelling words as sequences of letters. While this might seem like a specific purpose, the structure of the sequence nodes could be used for sequences in general, thereby enabling learning of many other concepts in the same way. For example, melodies can be modelled as sequences of notes, and for navigation it could be

beneficial to recognize a series of landmarks along a path as a sequence of sensor input.

5.1.1.2 Handling of nodes and size of matrices

One issue which limits the testing that can be done at the present state is the amount of nodes that the Animat learns. The more nodes that are formed, the bigger the matrices become. This eventually slows down the execution of the program. While some improvement of the performance can be achieved by changing the implementation to optimize the number of calculations needed (through the use of sparse matrices, dictionaries, etc), another approach is to try to minimize the number of nodes. In the current model, the same sequence might form in several ways, for example the sequence "cat" might be "ca" followed by a "t" as well as a "c" followed by a "at". This means that we actually get multiple nodes for some words, which is of course not necessary, and contributes to the large size of the matrices. For this reason, it might be beneficial to be able to compare nodes and remove those which produce duplicate sequences. Functionality for finding and removing equivalent nodes could therefore be a useful extension. Also, as all nodes are currently kept regardless of whether they are used or not, there may be nodes for some sequences that basically never occur. To deal with this, adding some kind of forgetting rule for removing nodes that rarely become active could also be an improvement. This would also be in line with the biological approach, considering that knowledge tends to fade from human minds without repetition [20].

5.1.1.3 Advantages of the Animat model

An advantage of the Animat is that it can be given several types of sensor input at the same time. While ordinary vector space models look at words preceding and succeeding the target word, the Animat can be given more sensory input which it can associate to a word as it hears it. As the Animat model aims to model more than just language, an Animat can be given the ability to perceive much more. This is interesting as it makes it possible for an Animat to associate words to different kinds of perceptions. It allows for grounding the understanding of words in the actual experiences that the words are intended to represent in the Animat's world. This is in line with the ideas discussed in [11] and [12]. In a biologically inspired project on AGI, such as the Generic Animat project, this is an interesting approach since it actually resembles the way that biological creatures such as humans can use language for communicating their experiences to each other.

5.1.1.4 Advantages of the Chunking Animat

The implementation of the Chunking Animat is cleaner than the Temporal Animat, in that it eliminates the need for temporal time-steps and TSEQ nodes for forming words. The use of chunking makes the sequence nodes more general, as it treats all segments of sensor input (words and phrases as well as sequences of sensations such as “cold”, “warm” etc.) in the same way, by modelling all forms of sequences using only SEQ nodes. While our project has focused on the learning of words and associations to them, an interesting extension to explore might be if SEQ nodes could be formed not just for individual words, but for sentences of varying length.

5.1.2 Evaluation

The vector space model implemented for the evaluation is very simple. However, the association function in the Animat is equally simple. As the purpose of the comparison was to get a measure of how the way that the Animat learns and stores words affects the Animat's ability to make associations, the simplicity of the implemented vector space model is not a problem.

The evaluation of Goal 2 is, as mentioned, very strict in that it demands that only the sets of predefined words are given as associations. This issue could be handled by using several different evaluation methods, and by comparing with several different models. Different evaluation methods would allow for a more broad view of how the Animat performs, thus better representing its accuracy. As vector space models are not made to handle this kind of input, it could be of interest to compare with other models. It could also be possible to compare the Animat's associations with those made by a group of humans, as this would be the most accurate benchmark. Furthermore, the way the input is given is deterministic in that every time the Animat hears a defined keyword, it will also perceive the connected sensations. This is not the case in the real world, where words can be used in varying contexts. Therefore it would be interesting to see how well the Animat would perform if, when given a keyword, it would be given the connected sensations as input with a high probability, but not necessarily every single time.

The largest problem with the current evaluation is the limited size of the datasets used for training. As the implementation of the Animat model developed in this project was not optimised for speed, it is much slower than the vector space model in terms of execution time. Due to the limited time and resources of this project, it was therefore not possible to make evaluations on any larger datasets. Because of this, further work is needed to ensure that the Animat is capable of making reasonable associations on larger datasets. The results of the performed evaluations suggest that this would be the case, but they cannot be taken as a guarantee. The dataset used for the evaluation of Goal 2 was also limited in that only a small number of sensors (for other input than letters) were defined. While both the Animat and the

vector space model could have handled larger sets of sensors, even though the way input was given to the vector space model did not scale as well, no such datasets were available within the limited time of the project. With more time and resources, a larger dataset could be produced and used for evaluation.

From this we conclude that further evaluation on larger datasets is needed. But the current evaluation manages to give some interesting results, as well as an indication of how the Animat model would perform when tested on larger datasets.

5.1.3 Interpretation of Results

The results of the steps show that our approach can be successfully used in the Animat model for the purpose of learning to recognise and produce words. That the Animat is capable of making relevant associations to the words that it has learnt to recognise, based on the context in which it has been exposed to them, can also be seen. Furthermore, the results show that the general way through which the Animat learns words only has a small impact on the quality of its associations. One can also see that the Animat is capable of making relatively accurate multimodal associations, by using the same general association method as when associating words. We can therefore conclude from the results that our approach can successfully be used for language learning on a small scale, and that it has potential to work on a larger scale if improved upon.

5.2 Further Work

The results presented in this thesis project must be considered preliminary, in that many more tests would need to be performed to evaluate the potential of this method for language learning on a larger scale. We have found that the Animat can indeed learn words and connect them to other perceptions, however there are many more areas that need further investigation. In this section we will list the ones that we find most important and interesting.

The most important further work that is needed is, as stated previously, to test the model on larger sets of data. This would be needed to see how well the Animat performs when it needs to learn more words, and when the words appear in more complex contexts. It would also be interesting to see how adding some noise/errors to the input would affect the quality of the Animat's associations. We believe that the Animat would still do equally well when compared to the vector space model, but these further tests would be needed to ensure this.

The second most important further work is to see how the Animat would perform when given a larger set of sensors for other input than letters. This would in theory

make the Animat capable of grounding its understanding of words further in the sensory input that they are meant to represent. It would also be relevant to examine how the Animat handles these multimodal associations when exposed to sensations that are present for words some of the time, but not always. The preliminary results suggest that the Animat should be capable of handling the increase in sensors. The generality of how the Animat makes associations suggest that the Animat should be able to handle sensations that don't always appear together with a word, just as well as it handles the words that don't always appear in each others context. Since sensations and words are all represented by perception nodes and treated the same way by the algorithm for associations, increasing the number of sensors for non-language sensations should not introduce more problems than increasing the number of words to learn does. But further work is needed to verify these indications from the current results.

A third area to explore is how the Animat would perform if tested on more complicated language than single words. As the Chunking Animat treats all sequences in the same way, it would be possible to extend the evaluation method to test if the model can be used for learning phrases and sentences as well. In addition to this, the spaces between words could be removed entirely to see how well the Animat performs when given no indication of when words begin and end. This is of interest, since pauses between words tend to be less clearly marked in spoken language, and is actually not always used in written text either [21].

As previously mentioned, one possible way to improve the performance of the Animat could be to introduce functionality for forgetting; thereby removing nodes that either are not used, or are equivalent to some other node in what they represent. For an Animat which exists in a more realistic world, where it can be exposed to more noise (misspellings, made up words, etc.), and where the environment may change over time, this could be particularly useful. In a world which changes, knowledge which was once necessary might eventually become deprecated and useless to hold onto.

Another interesting extension to the current work would be to see what other mechanisms can be used for learning language in the Animat model. Currently, only probabilistic learning is used when learning to recognise words. And when the Animat learns associations, it is only based on words (or other sensations) appearing in similar contexts. It would be interesting to see how well other mechanisms would work when applied to these tasks. One such mechanism is reinforcement learning, where the Animat could be rewarded when producing a word that it has heard. Further reinforcement learning could be used to reward the Animat when giving good associations, and punishing it when making bad ones. Another learning mechanism that could be of interest is imitation learning, where the Animat would 'record' a sequence and then create a sequence node representing the entire recorded sequence. This could be done in many different ways, such as the Animat combining all the elements in its short term memory or combining any pair of nodes that were active in two consecutive time-steps. The addition of such learning mechanisms could

improve the performance of the Animat, and would therefore be of great interest.

As has been stated previously, the model implemented in this project was not optimised for speed, and as such it would be of value to improve the implementation in order to make it run faster and scale better. While this would not reveal any interesting results on its own, it might be necessary in order for some of the extensions mentioned above.

5.3 Conclusion

We have explored how and to what extent a model for general artificial intelligence can be used for the task of first language learning. This has been done through the design and implementation of algorithms for first language learning in the Generic Animat model, taking care to ensure that all learning rules employed are general and can be used for other tasks than language learning. Two models (referred to as the "Temporal Animat" and the "Chunking Animat") have been developed and evaluated by comparison with vector space models for lexical meaning.

The first model (Temporal Animat) achieved an average overlap of approximately 80% with vector space models, in its top ten associations for words. It also performed, on average, better than the vector space models when making associations both to and from sensations other than words. The second model (Chunking Animat) achieved an overlap of approximately 29% with vector space models, in its top ten associations for words. When extended with a simple method for not adding some subsequences to its short term memory, the second model achieved a overlap of approximately 99%.

The two models are simple implementations, made for evaluating if and how well our approach works on a small scale. In their current forms our Animat implementations do not reach a level where they can be considered generally intelligent in the sense that a human or animal is intelligent. The graph of nodes does not scale in a way that enables the Animat to learn enough to be that versatile, resulting in larger texts with many unique words taking too long to process. This limits the implementations from being able to handle enough versatile input to reach a human-like language proficiency. However, as the results suggest that the learning rules function as intended, further improvements and optimisations of the model could possibly make it reach a level of language proficiency equal to that of a young child learning its first words. The rules that have been used in this project are all general, in that they are made to be able to function for any sequences, and not just for words.

We conclude that our approach works well for the learning of words and simple associations. However the model has only been evaluated on limited sets of words and simple texts, and further work is therefore needed to fully explore how well this approach to first language learning works. Therefore, we believe that it would be

interesting to continue exploring to what extent this approach can be used, both in regard to first language learning and for other purposes.

Bibliography

[1] Marta Koch. Artificial intelligence is becoming natural. *Cell*, 173(3):531 – 533, 2018.

[2] Alessio Mauro Franchi, Lorenzo Sernicola, and Giuseppina Gini. Linguistic primitives: A new model for language development in robotics. In Elio Tuci, Alexandros Giagkos, Myra Wilson, and John Hallam, editors, *From Animals to Animats 14*, pages 207–218, Cham, 2016. Springer International Publishing.

[3] C. Strannegård et al. The animat path to artificial general intelligence, 2017.

[4] S. Clark. *Vector Space Models of Lexical Meaning*, pages 493–522. John Wiley & Sons, Ltd, 2015.

[5] Kevin Mote. Natural language processing - a survey. 2012.

[6] Alexander Clark, Chris Fox, and Shalom Lappin. *The handbook of computational linguistics and natural language processing*. John Wiley & Sons, 2013.

[7] Arianna D'Ulizia, Fernando Ferri, and Patrizia Grifoni. A survey of grammatical inference methods for natural language learning. *Artificial Intelligence Review*, 36(1):1–27, Jun 2011.

[8] Gernot A. Fink and SpringerLink (e-book collection). *Markov models for pattern recognition: from theory to applications*. Springer, London, 2nd 2014;second; edition, 2008;2014;.

[9] J. Rissanen. A universal data compression system. *IEEE Transactions on Information Theory*, 29(5):656–664, 1983.

[10] Marcel H. Schulz, David Weese, Tobias Rausch, Andreas Döring, Knut Reinert, and Martin Vingron. Fast and adaptive variable order markov chain construction. In Keith A. Crandall and Jens Lagergren, editors, *Algorithms in Bioinformatics*, pages 306–317, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg.

[11] Joe Saunders, Caroline Lyon, Frank Forster, Chrystopher L. Nehaniv, and Ker-

stin Dautenhahn. A constructivist approach to robot language learning via simulated babbling and holophrase extraction. In *2009 IEEE Symposium on Artificial Life*, pages 13–20. IEEE, 2009.

[12] Angelo Cangelosi. Grounding language in action and perception: From cognitive agents to humanoid robots. *Physics of Life Reviews*, 7(2):139–151, 2010.

[13] Christopher D Manning, Prabhakar Raghavan, Hinrich Schütze, et al. *Introduction to information retrieval*, volume 1. Cambridge university press Cambridge, 2008.

[14] Karen Sparck Jones. A statistical interpretation of term specificity and its application in retrieval. *Journal of documentation*, 28(1):11–21, 1972.

[15] A. Gopnik, A. Meltzoff, and P. Kuhl. *How Babies Think: The Science of Childhood*. Weidenfeld & Nicolson, London, 1999.

[16] M. Passer et al. *Psychology: The Science of Mind and Behaviour*. McGraw-Hill Higher Education, Maidenhead, european e edition, 2009.

[17] Nelson Cowan. The magical number 4 in short-term memory: A reconsideration of mental storage capacity. *Behavioral and Brain Sciences*, 24(1):87–114; discussion 114–85, 02 2001.

[18] Fang Xu. Short-term working memory and chunking in sla *. *Theory and Practice in Language Studies*, 6(1):119–126, 01 2016.

[19] John. A. Rice. *Mathematical Statistics and Data Analysis*. Brooks/Cole, Belmont, CA, 3 edition, 2007.

[20] Jaap M. J. Murre and Joeri Dros. Replication and analysis of ebbinghaus' forgetting curve. *PLoS One*, 10(7), 07 2015.

[21] F. Coulmas. *Writing Systems: An Introduction to Their Linguistic Analysis*. Cambridge Textbooks in Linguistics. Cambridge University Press, 2003.

A

Step 2 Plots

Plots for the evaluation of Step 2 - Recognising words.

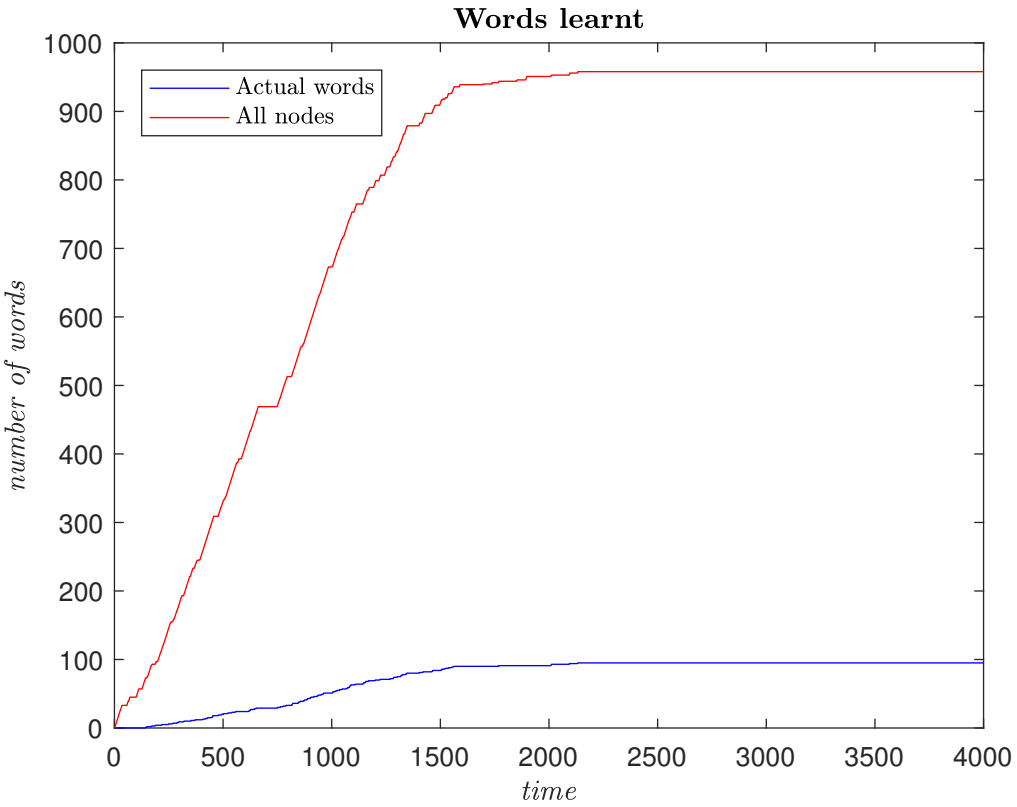


Figure A.1: Plot showing how the total number of perception nodes, and the number of perception nodes representing actual words, changes over time.

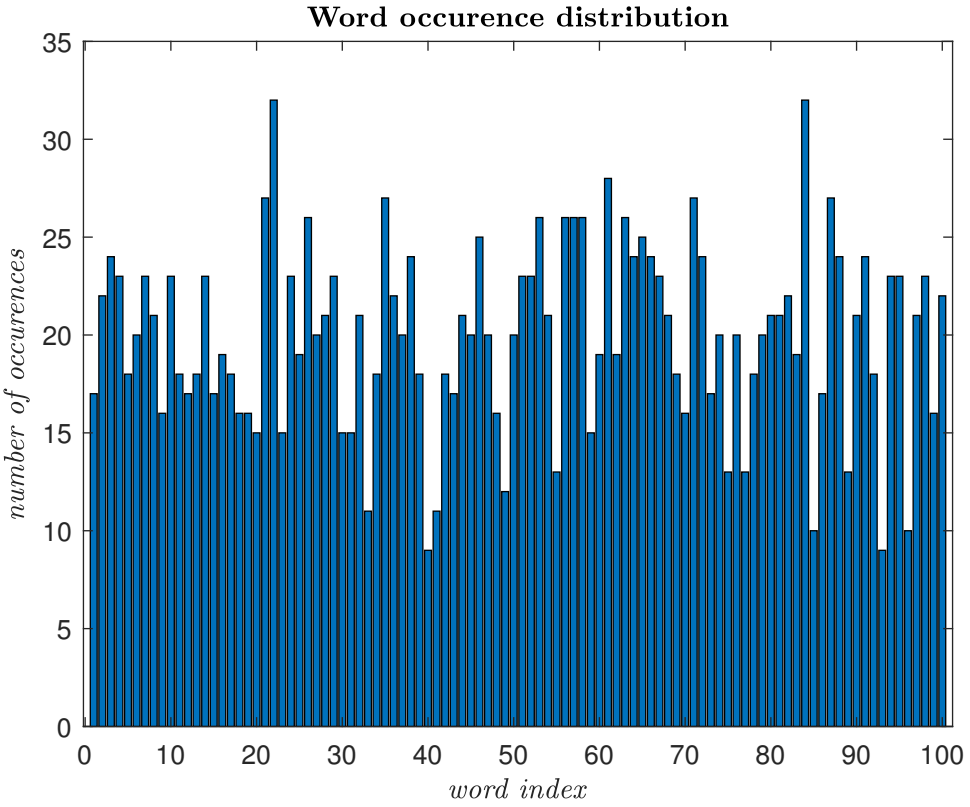


Figure A.2: Plot showing the distribution of how often the different words were given as input.

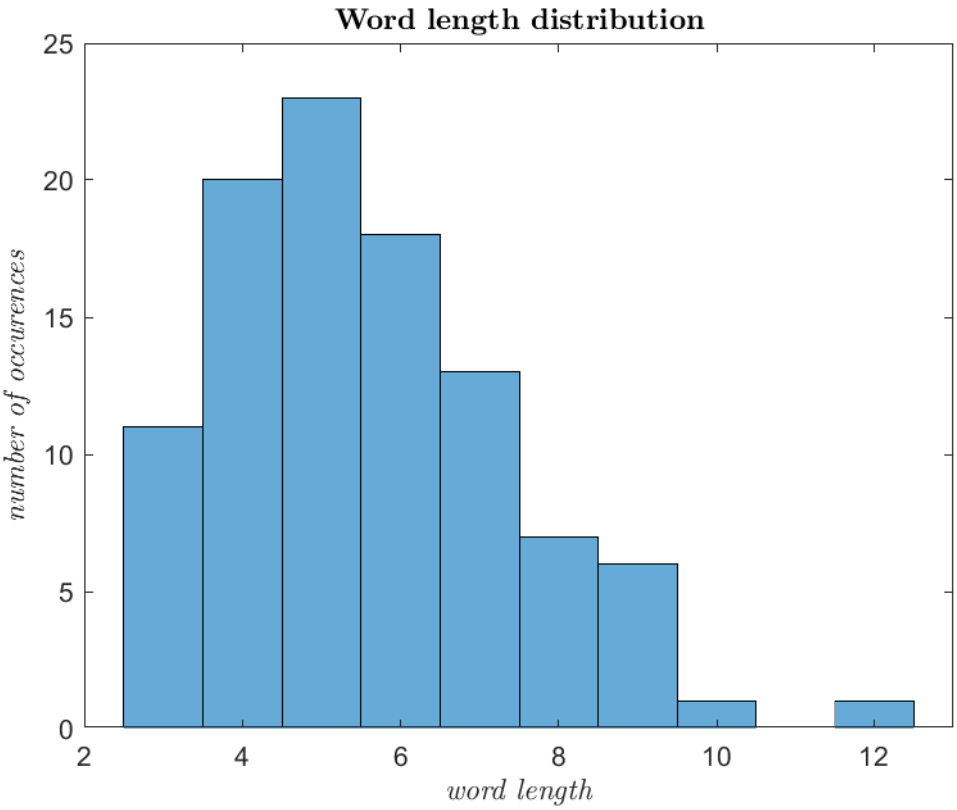


Figure A.3: Plot showing the length distribution of the words.

B

Goal 2 Dictionary

```
sensations = {}
sensations["winter"] = {"[cold]", "[white]"}
sensations["summer"] = {"[warm]", "[light]", "[green]"}
sensations["cat"] = {"[soft]", "[warm]", "[small]", "[sharp]"}
sensations["dog"] = {"[soft]", "[warm]", "[loud]", "[big]"}
sensations["kitten"] = {"[soft]", "[warm]", "[tiny]", "[sharp]"}
sensations["puppy"] = {"[soft]", "[warm]", "[loud]", "[small]"}
sensations["bird"] = {"[small]", "[loud]", "[tiny]"}
sensations["berry"] = {"[sweet]", "[soft]", "[tiny]", "[red]"}
sensations["milk"] = {"[white]", "[wet]", "[cold]", "[sweet]"}
sensations["water"] = {"[wet]", "[blue]", "[cold]", "[warm]"}
sensations["ice"] = {"[cold]", "[hard]", "[blue]", "[white]"}
sensations["snow"] = {"[cold]", "[white]", "[soft]"}
sensations["rain"] = {"[cold]", "[wet]"}
sensations["sun"] = {"[light]", "[warm]", "[yellow]", "[red]"}
sensations["night"] = {"[dark]", "[quiet]", "[black]", "[cold]"}
sensations["day"] = {"[light]", "[loud]", "[warm]"}
sensations["green"] = {"[green]"}
sensations["white"] = {"[white]"}
sensations["black"] = {"[black]"}
sensations["red"] = {"[red]"}
sensations["wet"] = {"[wet]"}
sensations["cold"] = {"[cold]"}
sensations["frozen"] = {"[cold]", "[hard]"}
sensations["leaves"] = {"[green]", "[small]"}
sensations["tree"] = {"[green]", "[big]"}
sensations["bush"] = {"[green]", "[small]", "[soft]", "[sharp]"}
sensations["grass"] = {"[green]", "[soft]", "[tiny]"}
sensations["lake"] = {"[wet]", "[blue]", "[big]"}
sensations["river"] = {"[wet]", "[blue]"}
```