Bounded Sceptical Reasoning

Isabelly Lourêdo Rocha

November 6th, 2017

Technische Universität Dresden

Department of Computer Science
International Center for Computational Logic
Knowledge Representation and Reasoning Group

Bounded Sceptical Reasoning

Isabelly Lourêdo Rocha

1. Reviewer Prof. Dr. Steffen Hölldobler
Department of Computer Science
Technische Universität Dresden, Germany

2. Reviewer Prof. Dr. Sergio Tessaris
KRDB Research Centre for Knowledge Representation
Free University of Bolzano, Italy

Supervisor Prof. Dr. Steffen Hölldobler

November 6th, 2017

Isabelly Lourêdo Rocha
Bounded Sceptical Reasoning
November 6th, 2017
Reviewers: Prof. Dr. Steffen Hölldobler and Prof. Dr. Sergio Tessaris
Supervisor: Prof. Dr. Steffen Hölldobler

Technische Universität Dresden
Knowledge Representation and Reasoning Group
International Center for Computational Logic
Department of Computer Science
Nöthnitzer Strasse 46
01187 and Dresden

Abstract

Well known psychological experiments have shown that classical logic does not seem to be adequate for modelling human reasoning. Therefore, we focus on an approach called the weak completion semantics which considers the least model of the weak completion of logic programs under the three-valued Łukasiewicz logic. This approach has shown how to adequately model many of the famous human reasoning tasks found in the literature, mostly by means of sceptical abduction. The reasoning is done by searching for all the abductive explanations for a given observation and computing which consequences can be derived from them.

The computation of sceptical consequences for an abductive problem has been encoded by means of a connectionist network. Abduction is known to be a computationally expensive problem, where computing the abductive sceptical consequences is DP-complete. The bottleneck of this problem is in the exponential number of candidate explanations that have to be considered. Therefore, we focus on the component of the connectionist network responsible for generating those candidates and try to optimise it.

We start by adding a minimality constraint to the generation of candidate explanations, which considerably reduces the number of candidates generated in practice, but the problem still remains the same in the worst case. We assume that humans do not consider all the candidate explanations when reasoning about skeptical abduction, but rather a subset of them. Therefore, we propose to substitute the current approach by a recurrent network such as Jordan or Elman networks which we have shown to be capable of learning arbitrary sequences of candidate explanations.

It is commonly required from computational models in artificial intelligence to exhibit a behaviour similar to the biological brain. Therefore, we propose some psychological experiments to confirm the assumptions we have made, bearing in mind that our approach has been designed in such a way that it could easily be aligned with these psychological experiments independent of their outcome.

Contents

1 Introduction

Many human reasoning tasks have been adequately modelled by a computational approach based on the weak completion semantics. In most of the cases, the models generated by this approach rely on the sceptical abductive consequences of an observation and the least model of a logic program representing the given background knowledge. This approach has been encoded as a connectionist network which is going to be the focus of this thesis. In this chapter, we show some of the mentioned reasoning problems as well as an overview of this network's architecture.

1.1 Sceptical Abduction

The notion of abduction was first introduced by the philosopher Pierce [30]. Let us illustrate the idea of abduction by means of an example. Consider a background knowledge consisting of the following conditionals:

If it rained last night, then the grass is wet.
If the sprinkler was on, then the grass is wet.
If the grass is wet, then the shoes are wet.

If we observe that the shoes are wet, we have several explanations. One explanation is that the grass is wet. However, we can further explain, why the grass is wet. Then, we obtain the explanation *it rained last night* and the explanation *the sprinkler was on*. The last two explanations are called basic, since we cannot further explain them. Another explanation is *the sprinkler was on and it rained last night*, which combines the last two explanations. Then, this explanation is not minimal. One meaningless explanation is *the shoes are wet*, i.e. the observation is explained by itself.

There are two forms of abduction which are well known in the literature: credulous and sceptical abduction. In credulous abduction, considering a specific background knowledge, humans would derive a conclusion if it follows from at least one of the explanations for the given observation. On the other hand, this is not sufficient in sceptical abduction where all explanations shall be considered. Coming back to the example, if we observe that *the grass is wet*, then we could credulously conclude that *it rained last night*. However, in the scenario where *the sprinkler was on*, our observation is also explained and not necessarily *it rained last night* has to be true.

Therefore, we can neither sceptically conclude that *it rained last night* nor that *the sprinkler was on*.

As we will discuss in more details later on, psychological experiments have shown that humans systematically deviate from the classical logically correct answers. Therefore, we will consider a non-classical approach to model human reasoning which is non-monotonic and based on the least model of the weak completion of logic programs. More precisely, here we focus focus on the component of this framework responsible for deciding which and in what order possible explanations should be considered.

1.1.1 Byrne's Suppression Task

A well-known psychological experiment by Byrne [3] has shown that, in certain circumstances, participants reject instances of the valid *modus ponens* and *modus tollens* inference form in conditional arguments. This experiment shows that people might suppress previously drawn conclusions when additional information becomes available. For example, when a conditional premise, such as *if she has an essay to finish, then she will study late in the library* is accompanied by the fact *she has an essay to finish*, 96% of the participants conclude that *she will study late in the library*. However, if we add the conditional premise *if the library stays open, she will study late in the library*, then the number of participants which conclude that *she will study late in the library* drops to 38%.

This experiment is a strong support for the assumption that humans reason non-monotonically. In the complete experiment, participants receive the following three conditionals:

Simple	*If she has an essay to finish, then she will study late in the library.*
Alternative	*If she has a textbook to read, then she will study late in the library.*
Additional	*If the library stays open, then she will study late in the library.*

The participants were divided into three groups such that each group received a different combination of these conditionals:

Group I	Simple.
Group II	Simple + Alternative.
Group III	Simple + Additional.

Fact	Conclusion	Group I	Group II	Group III
$e^{\top}$	$l^{\top}$	96%	96%	**38%**
$e^{\bot}$	$l^{\bot}$	46%	**4%**	63%
$l^{\top}$	$e^{\top}$	53%	**16%**	55%
$l^{\bot}$	$e^{\bot}$	69%	69%	**44%**

Table 1.1: Empirical results about suppression obtained by Byrne.

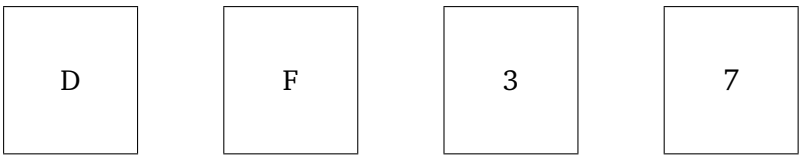
The task given to the participants consisted of two parts. In the first part the participants got either the fact *she has an essay to finish* ($e^{\top}$) or the negation of it, *she does not have an essay to finish* ($e^{\bot}$). In the second part the participants got either the fact *she will study late in the library* ($l^{\top}$) or the negation of it, *she will not study late in the library* ($l^{\bot}$). Based on the given information, the participants were asked to draw conclusions.

An overview of the empirical results obtained from this experiment is shown in Table 1.1. Percentages indicate the proportion of participants in each of the three groups that have drawn the respective conclusion from the indicated given fact and the conditionals. Where suppression took effect, the propositions are highlighted in bold. Similar results have been obtained by other researchers, see for example [9].

The suppression task is adequately modelled by the weak completion semantics using skeptical abduction. In particular, consider the case where we observe that *she will study late in the library* is true. If we model only the conditional given to Group I, the consequence that *she has an essay to finish* is derived, while if we model the conditionals given to Group II, this is no longer the case. If we observe the psychological results, our approach is able to suppress the conclusion when one more conditional to the background knowledge is added, reflecting the answers given by the participants.

1.1.2 Wason's Selection Task

In the original selection task [44] participants were told that each card had a letter on one side of the card and a number on the other side of the card. They were presented the following four cards the table:



The task of the participants was to evaluate the following conditional with respect to each of the four cards:

if there is a D on one side of the card, then there is 3 on the other side.

Which cards must be turned over in order to find out whether the conditional holds?

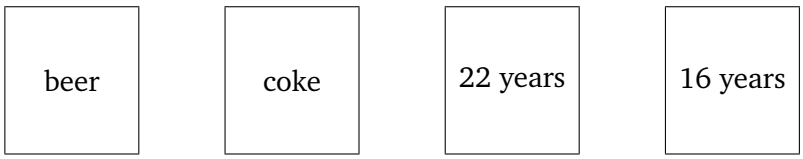
Assume the conditional is represented in classical propositional logic by the implication

$$3 \leftarrow D$$

where the propositional variable *3* represents the fact that the number 3 is shown and *D* represents the fact that the letter D is shown. Then, in order to verify the implication one must turn the cards showing D and 7. However, as repeated experiments have shown consistently (see Table 1.2a), participants believe differently.

Whereas 89% of the participants correctly determine that the card showing D must be turned (a number other than 3 on the other side would falsify the implication), 62% of the participants incorrectly suggests turning the card showing 3 (no relevant information can be found which would falsify the implication). Likewise, whereas only 25% of the participants correctly believe that the card showing 7 need to be turned (if the other side would show a D, then the implication is falsified), 16% incorrectly believe that the card showing F needs to be turned (no relevant information can be found which would falsify the implication). In other words, the overall correctness of the answers for the abstract selection task if modelled by an implication in classical two-valued logic is pretty bad.

The selection task was adapted to a social case [14]. Consider the following four cards:



Each card has the person's age on one side of the card and what the person is drinking on the other side of the card. Consider the conditional

if a person is drinking beer, then the person must be over 19 years of age.

The same question is then asked: Which cards must be turned over in order to find out whether the conditional holds?

D	F	3	7
89%	16%	62%	25%

(a) Abstract case.

beer	coke	22yrs	16yrs
95%	0.025%	0.025%	80%

(b) Social case.

Table 1.2: The results of the abstract (a) and social (b) cases of the selection task.

If the conditional is represented by the implication

$$over19 \leftarrow beer,$$

where $over19$ represents a person being older than 19 years and *beer* represents the person drinking beer. In order to verify the implication, one must turn the cards drinking beer and 16 years old. Participants usually solve the social version of the selection task classical logically correctly. Table 1.2b shows the results represented by Griggs and Cox [7] for the social case.

One explanation for the differences between both cases can be found in [22], namely that people view the conditional in the abstract case as a belief. For instance, the participants perceive the task to examine whether the rule is either *true* or *false*. On the other hand, in the social case, the participants perceive the rule as a social constraint, a conditional that ought to be *true*. People intuitively aim at preventing the violation of such a constraint, which is normally done by observing whether the state of the world complies with the rule.

Both abstract and social cases can be modelled by an approach based on the weak completion semantics shown in [8]. The fact that two conditionals following the same syntactic structure were evaluated differently gives a strong intuition that the semantics is also an important criteria to be considered when modelling those reasoning tasks. In the next section the discussion on the semantics of conditionals is developed in more details.

1.1.3 Conditionals Semantics

Conditionals can be categorised in many different types and classes, but there are two main groups of conditionals: indicative and subjunctive conditionals, the latter of which are also known as counterfactuals. In indicative conditionals both, the condition and the consequence, can be either true, false or unknown. But if the condition is true, then the consequence is asserted to be true. In counterfactuals, the consequence can again be either true, false or unknown, but the condition is known

to be false. Besides that, in the couterfactual circumstance of the condition being true, the consequence is asserted to be true.

In we focus on indicative conditionals, there are many subclasses to be considered. For example, they can be classified into *obligation* and *factual conditionals*. The difference between them is that in the *obligation conditionals* the consequence is obligatory, while in the *factual conditionals* this is not the case. More precisely, given an *obligation conditionals* it can never be the case that the condition happens and the consequence doesn't.

As explained in [4], conditionals where the consequent is denied are more likely to be evaluated to *true* if it is an *obligation conditional*. This happens because for this type of conditionals, people keep in mind a forbidden possibility where condition and not consequence happens together and, in this case, if not consequence is known to be *true*, then it cannot be the case that condition is *true* as well, otherwise the forbidden possibility is violated. Thus, *not condition* is concluded. But, since in a *factual conditional* this forbidden possibility does not exist, conditionals with the consequence denied should be evaluated to *unknown*.

Consider the following conditional:

If it rains, then the streets are wet.

Its consequence is obligatory. We cannot easily imagine a case where the condition the *it rains* is true and the consequence *the streets are wet* is not. We know streets, we know rain and its effects, and because the conditional is discussed without any specific context, there does not appear to be a case where the condition is true and the consequence is not. But if we now consider the following conditional:

If it rains, then she takes her umbrella.

Is consequence is not obligatory. Different than in the previous case, is the condition *it rains* is true, we can easily think about a scenario where the consequence *she takes her umbrella* is not. It is not so counterintuitive to imagine that it rained and she forgot to take her umbrella, for example.

In the selection task, the conditional considered in the abstract case is classified as a *factual conditional* as there is no obligation implied from the context. On the other hand, the conditional considered in the social case is classified as an *obligation conditional* as there is a clear obligation implied from the *must* condition. There are also semantic differences in the conditions of these conditionals which are discussed in [41].

1.1.4 Syllogistic Reasoning

A syllogism consists of two premises and a conclusion. Each of them is a quantified statement using one of the four quantifiers *all*, *no*, *some*, and *some are not* over sets of entities which we denote in the following by a, b, and c. An example of a syllogism is:

Some b are a
No b are c

In experiments, participants are normally expected to complete the syllogism by drawing a logical consequence from the first two premises, e.g. in this example *some a are not c*. Altogether, there are 64 syllogisms and, if formalised in first-order logic, we can compute their logical consequence in classical logic. The first two premises together with a consequence that follows classical logically is called a *valid syllogism*. Otherwise, it is called an *invalid syllogism*.

A meta-analysis by Khemlani and Johnson Laird [20] based on six experiments has shown that humans do not only deviate from the classical predictions of first-order logic, but from any other of at least twelve cognitive theories on the syllogistic reasoning. To understand the deviation from classical logic on the experimental results, let us consider an example of a syllogism discussed by Evans [12] which has an invalid but believable conclusion as well as a valid but unbelievable conclusion. Given the following two premises:

No addictive things are inexpensive.
Some cigarettes are inexpensive.

The conclusion that *some addictive things are inexpensive* is invalid, but believable. On the other hand, the conclusion that *some cigarettes are not addictive* is valid, but unbelievable.

From the cognitive theories available in the literature, the best overall results are achieved by Polk and Newell, the Verbal Models Theory [31], which predicts 84% of the participants responses, closely followed by Johnson-Laird, the Mental Model Theory [18], with 83%, whereas PSYCOP [33] by Rips only predicts 77% of the participants' responses.

Recently, a new cognitive theory, based on the weak completion semantics and skeptical abduction, to model the syllogistic reasoning task has been developed by Costa, Dietz, Hölldobler and Ragni [6]. This approach identifies seven principles, mostly motivated from findings in cognitive science, which are necessary to draw

the inferences. This approach has achieved the best results, compared to the results of other cognitive theories, with a prediction of 89%.

1.2 A Core Method for Sceptical Abduction

Hölldobler and Kencana Ramli [17] have shown that the computation of the least fixed point of the $\Phi_{\mathcal{P}}$ operator can be realised within a connectionist network, with the core-method [2]. A connectionist realisation of sceptical abduction under the weak completion semantics within the core method has been showed by Dietz Saldanha, Hölldobler, Kencana Ramli, and Palacios Medinacelli [39]. Figure 1.1 shows an overview of this connectionist network which consists of three main components.

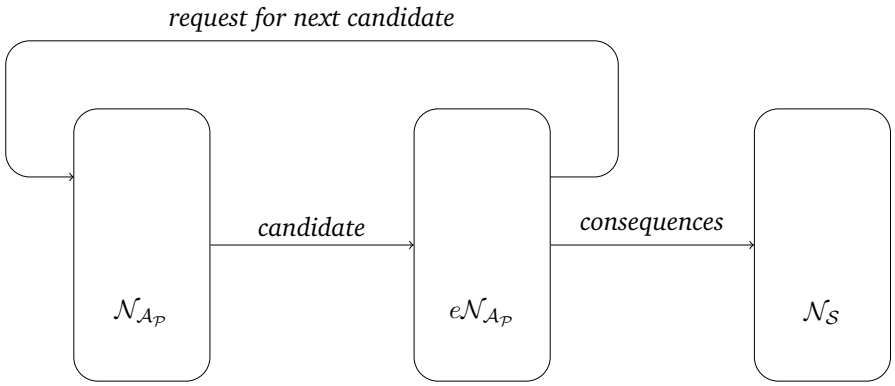


Figure 1.1: Overall view of the connectionist network to compute the sceptical consequences of a given abductive problem.

The first component of the network $\mathcal{N}_{\mathcal{A}_\mathcal{P}}$ sequentially generates the candidate explanations to be considered by the abductive framework one after the other and only upon request. The second component $e\mathcal{N}_{\mathcal{A}_\mathcal{P}}$ computes the least model of the weak completion of $\mathcal{P} \cup \mathcal{C}$, where $\mathcal{P}$ is the program considered as background knowledge and $\mathcal{C}$ is the current candidate explanation generated by $\mathcal{N}_{\mathcal{A}_\mathcal{P}}$. The network $e\mathcal{N}_{\mathcal{A}_\mathcal{P}}$ sends the request of a new candidate to network $\mathcal{N}_{\mathcal{A}_\mathcal{P}}$ once its task is done.

Besides the next candidate explanation, the network $\mathcal{N}_{\mathcal{A}_\mathcal{P}}$ also outputs the information whether the candidate generated is the last one. When that's the case, after the computation of the least model of this last candidate to be considered, the network $\mathcal{N}_\mathcal{S}$ computes all the sceptical consequences of the given observation by merging all the consequences followed by the program and the positive candidates.

The complexity of the task corresponding to component $\mathcal{N}_{\mathcal{A}_\mathcal{P}}$ is exponential with respect to the cardinality of the set of abducibles. The computation of the least fixed point of $\Phi_\mathcal{P}$, corresponding to component $e\mathcal{N}_{\mathcal{A}_\mathcal{P}}$, can be done in polynomial time as shown by Dietz Saldanha, Hölldobler, and Philipp [40]. The remaining component, denoted by $\mathcal{N}_\mathcal{S}$, has the task of computing the sceptical consequences, given the consequences of each positive candidate. This can be done by intersecting the given consequences, which can be reduced to the problem of sets intersection. Therefore, it can also be done in polynomial time.

Based on the complexity of each of those tasks, it is clear that the bottleneck of the problem is in the generation of candidate explanations. This component is encoded by means of a McCulloch Pitts network designed to generate all the non-complementary candidate explanations in a static and pre-defined order. The complexity results of the way we apply sceptical abduction is a strong indication that humans do not reason in the same way.

Motivated by this, we believe that humans do not generate all candidate explanation but rather a subset of them. Therefore, we are going to make some assumptions in the reasoning process which will allow us to reduce the number of generated candidate explanations. However, as there are so many open questions regarding this problem which have not yet been verified, we also propose to substitute the current approach by a more flexible one which is able to learn arbitrary sequences of candidate explanations and, therefore, can easily be adapted to the constraints later identified in psychological experiments.

1.3 Hypotheses

In order to optimise the process of generating candidate explanations and align it with the real cognitive process regarding sceptical abduction, we are going to make some assumptions as, for example, regarding minimal and bounded candidates. In this section we will discuss these hypotheses and their implications in more details.

1.3.1 Basic Candidate Explanations

Reconsidering the first example presented in Section 1.1, the only possible explanations which would be considered in our approach for the observation *the shoes are wet* are concerning that *it rained last night* or that *the sprinkler was on*, but not that the *the grass is wet*. This is a consequence of the abductive framework where we only consider the undefined atoms in the set of abducibles which are later going to be used for the generation of candidate explanations.

Therefore, our first hypothesis is that when humans reason about the sceptical consequences of a given problem they search for the basic candidate explanations and reason only with respect to them. An explanation is said to be basic if it cannot be explained by other facts or assumptions, i.e. it can only be explained by itself.

1.3.2 Sequential Generation

The second hypothesis we make is that humans do not generate several candidate explanations in parallel but they are sequentially generated and considered one by one in the reasoning process. Therefore, we do not have a distributed approach. Instead, the candidate explanations are generated after the other. Besides, a new candidate is generated only after our approach checks whether the current candidate is indeed an explanation for the given observation.

1.3.3 Non-complementary Candidate Explanations

Consider the following conditional:

If she studies, then she will pass the exam.

To explain, for instance, the observation that *she did not pass the exam*, we would not consider as candidate explanation the facts that *she studied* and that *she did not study* together. This type of candidate explanations are said to be complementary. Therefore, in our approach we also have the hypothesis that humans only consider non-complementary candidate explanations.

1.3.4 Minimal Candidate Explanations

Another hypothesis is that humans generate only minimal candidate explanations. If we reconsider Byrne's selection task presented earlier, for the observation that *she studies late in the library*, the minimal explanations would be either that *she has an essay to finish* or that *she has a text book to read*. The case where both explanations are combined also explains the given observation, but in a non-minimal way and, therefore, should not be considered. Moreover, the cases where *she has an essay to finish* but *she does not have a text book to read* and the other way around are also non-minimal explanations for the observation.

This hypothesis implies a constraint in the generation of the candidate explanations which is about the cardinality of the candidates. It is easy to see that if the candidate where *she has an essay to finish* and *she has a text book to read* are both true is

generated before a candidate where only one of these two facts is true and the other one is unknown, then there is no way to avoid this non-minimal candidate as it's subset is not known yet. Given this observation, we assume the generation of candidates has to be done from the smaller to the larger sets of candidates, where smaller and bigger is defined by their cardinality.

1.3.5 Bounded Sceptical Abduction

Although the minimality constraint reduces the number of candidate explanations generated in average, the complexity of the problem remains the same in the worse case. Imagine a ordering respecting the cardinality where the explanation *she does not have an essay to finish* and *she does not have a text book to read* is the last one. If we observe that *she does not study late in the library*, the only possible explanation for our observation would be the last one, which means that all the candidates would have to be generated just as before.

As mentioned before, it is hard to believe that humans would systematically generate all the candidate explanation to solve such a problem. Therefore, we have the hypothesis that there is a bound in the generation of candidate explanation and that the reasoning process is different for different types of conditionals as it has been investigated in [41]. This implies that arbitrary sequences of candidate explanations have to be considered.

1.4 Tasks

Considering the hypotheses we have made, we will now show what has to be done in the current approach to achieve such results. The tasks consist of two steps: first, we will show how to apply the minimality characterisation to the current McCulloch Pitts network, and, after this, we show a formal specification on how to substitute this component by a recurrent network which can learn arbitrary sequences of candidates.

1.4.1 Minimality

Given that the McCulloch Pitts network generates the candidate explanations in a static and pre-defined order, we can easily consider a sequence which respects the cardinality ordering. Given that the candidates are being generated from the smaller to the larger ones, we can store the information of which candidates were positive

and use this information to block the generation of further candidates which are supersets of them.

This can be done by adding a unit to the network corresponding to each of the candidates with cardinality larger than one. These units will be connected to all the units representing subsets of the respective candidate such that they are activated as soon as one of these candidates has been detected as being an explanation. The activation of these units will be stored for the further time steps and used later on to skip the generation of the non-minimal candidates. By this thechnique we ensure that only minimal candidate explanations are generated.

1.4.2 Arbitrary Sequences

In order to confirm our hypothesis on the bounded candidates and try to obtain more properties in the arbitrary sequence of candidates, we design some psychological experiments and propose their execution. Considering that there are so many open questions concerning this problem and that the psychological results are crucial to obtain a strong cognitive theory, we propose the substitution of the current approach by a recurrent network which can learn arbitrary sequences of candidate explanations and, therefore, easily adapt to the outcome of these experiments independent on what they are.

For this purpose, we will focus on Jordan and Elman networks and provide a formal specification on how they can be constructed, trained and tested for the problem of generating candidate explanations. This process evolves transforming our problem in a temporal domain problem and generating data which simulates the candidate explanations expected in different scenarios. We will start by showing that they can achieve the same results presented by the previous approach, but with the advantage of learning arbitrary sequences. We will design experiments to define the necessary number of hidden units such that they perform the task with a error rate close to zero.

1.5 Contributions

Optimisation of sceptical reasoning

Sceptical abductive reasoning is an expensive problem from the computational point of view and its bottleneck is the exponential number of candidate explanations which has to be considered. By introducing the minimality constraint, we have reduced the

number of candidate explanations generated in the average case, optimising with this the abductive sceptical reasoning process as a whole.

Flexibility

The current framework to generate candidate explanations is a McCulloch Pitts network which is designed to generated the candidate explanations as a static and pre-defined sequence. The replacement of this framework by a recurrent network which is able to learn arbitrary sequences of candidate explanations have given flexibility to our framework. As the sceptical reasoning approach is currently done based on several hypotheses which have not yet been tested, flexibility is crucial characteristic.

Bounded reasoning

From a cognitive aspect, it is hard to believe that humans generate this large number of candidate explanations in the reasoning process. We believe that they rather generate a subset of these candidate explanations. By introducing the minimality constraint, we have considerably reduced the number of candidate explanations generated in the average case. However, the problem remains the same in the worst case. Moreover, there is still a considerably large number of candidates even after applying minimality.

Based on this, we believe that a bound is applied when generating candidate explanations and, therefore, we have introduce this feature to our framework. As the network has an anytime behaviour, it remains only to defined what kind of bounds are applied and their characterisation. For this purpose, we need psychological experiments which are beyond the scope of this master thesis. However, the approach in designed in such a way which that it can easily be adapted to the experiments outcome.

Cognitive adequacy

Throughout the development of this new approach, we have made several hypotheses which raises questions to be answered by means of psychological experiments. These questions have been discussed and an initial setup for experiments to test them has been proposed. Some of the hypotheses make sense from a computational point of view, but as we also want our approach to be cognitively adequate, carrying out these experiments is a crucial step.

1.6 Thesis Structure

Although we assume the reader to be familiar with logic, logic programming and connectionist networks, Chapter 2 introduces some of the general and basic concepts necessary as background knowledge for the work developed. The chapter is divided in two main parts: abduction under the weak completion semantics and connectionist networks. For the first part, we start by introducing some concepts on logic programs such as the weak completion, then we define which semantics we are going to consider throughout the thesis and finally de define the abductive framework as well as its complexity. For the second part, we start by introducing some general concepts of recurrent networks and focus then on Jordan and Elman networks which are going to be used later on.

In Chapter 3 we describe an approach based on McCulloch Pitts networks to generate non-complementary candidate explanations for the abductive problem. Later we introduce a possible extension of this approach such that it will now only generate minimal candidates. All the steps necessary for such modification are described by means of an example, but the algorithms on how to generalise to arbitrary problems is also presented.

After introducing the current approach for the generation of candidate explanations, in Chapter 4 we propose a new approach which is more flexible. We show that this approach can simulate the same behaviour as the previous one, with the advantage of learning arbitrary sequences of candidates. This is an important feature since we currently make assumptions on the process, but have not yet verified them by means of psychological experiments.

Finally, we want to evaluate the assumptions we have made and, as we already mentioned, a system that aims at being cognitively adequate has to be evaluated with respect to the way humans reason. Therefore, in Chapter 5 we suggest to carry out some psychological experiments. From their outcome, we expect to confirm or refute our hypothesis and extract more properties which can further improve our approach.

2 Preliminaries

This chapter introduces the general notions and terminologies that will be used throughout this thesis, which are based on [16, 23]. We start with Section 2.1 by introducing some of the basic concepts about logic programming, the three-valued semantics and the weak completion semantics which are necessary to understand the abductive framework. Then in Section 2.2 we introduce some of the basic concepts about connectionist networks and recurrent connectionist networks which are going to be used later for the generation of candidate explanations for the abductive framework. In particular, we will introduce the architecture and main concepts of Jordan and Elman networks.

2.1 Abduction under the Weak Completion Semantics

We assume the reader to be familiar with logic and logic programming. In the sequel, we will only consider propositional programs. If a program $\mathcal{P}$ is not propositional, then a propositional program $\mathcal{P}'$ corresponding to $\mathcal{P}$ can be obtained by grounding the program $\mathcal{P}$. We assume that each non-propositional program contains at least one constant symbol. The language $\mathcal{L}$ underlying a non-propositional program $\mathcal{P}$ contains precisely the predicate and constant symbols occurring in $\mathcal{P}$, and no others. Therefore, the following definitions can also be applied to programs which are not propositional. In those cases, we only need to compute the respective grounded program beforehand and reason with respect to it.

Therefore, we consider an *alphabet* that consists of finite disjoint sets of *propositional variables*, the *truth-value constants true* $\top$, *false* $\bot$ and *unknown* U, and the usual connectives *negation* $\neg$, *disjunction* $\vee$, *conjunction* $\wedge$, *implication* $\leftarrow$, *equivalence* $\leftrightarrow$. *Formulas* are constructed in the usual way from the propositional variables, the truth-value constants and the connectives. An atomic formula is called an *atom*. If A is an atom, then A and $\neg A$ are literals, called the *positive literal* and the *negative literal*, respectively. A *language* $\mathcal{L}$ given by an alphabet $\mathcal{A}$ consists of the set of all formulas constructed from the symbols of $\mathcal{A}$.

2.1.1 Logic Programs

Definition 1. A *clause* is a formula of the form:

$$A \leftarrow L_1 \wedge ... \wedge L_n. \tag{2.1}$$
$$A \leftarrow \top. \tag{2.2}$$
$$A \leftarrow \bot. \tag{2.3}$$

A is called *head* of the clause and the sub-formula to the right of the implication symbol is called *body* of the clause. Clauses are also called *rules* (2.1), *facts* (2.2) and *assumptions* (2.3).

A given pair of clauses is said to be *complementary* if they are of the following form

$$c \leftarrow \top. \qquad c \leftarrow \bot.$$

A set of clauses $\mathcal{C}$ is said to be *complementary* if it contains a complementary pair; otherwise, $\mathcal{C}$ is said to be *non-complementary*.

Definition 2. A *logic program* $\mathcal{P}$ is a finite set of clauses. A *propositional program* is a program where all clauses are propositional.

If $\mathcal{P}$ is a program, then $atoms(\mathcal{P})$ denotes the set of all atoms occurring in $\mathcal{P}$. An atom A is *defined* in $\mathcal{P}$ if and only if $\mathcal{P}$ contains a clause of the form

$$A \leftarrow Body.$$

A is *undefined in* $\mathcal{P}$ if and only if A is not defined in $\mathcal{P}$. The set of all atoms that are defined in $\mathcal{P}$ is denoted by $def(\mathcal{P})$. The set of all atoms that are undefined in $\mathcal{P}$, that is $atoms(\mathcal{P}) \setminus def(\mathcal{P})$, is denoted by $undef(\mathcal{P})$. The *definition* of A in $\mathcal{P}$ is given by

$$def(A, \mathcal{P}) = \{A \leftarrow Body \mid A \leftarrow Body \text{ is a clause in } \mathcal{P}\}.$$

A *normal program* – in the standard sense used in the literature on logic programming – is a program that does not contain assumptions. This means a program which consists only of rules and facts. The notion of assumptions, i.e., clauses of the form (2.3), seems to be counterintuitive at first sight. However, we have to consider that programs will be interpreted under their weak completion where the implication connectives are replaced by equivalence connectives. Under this circumstances, an assumption ensures that the head will be false.

When mechanisms of non-monotonic reasoning are applied to model human reasoning, it seems essential that only certain atoms are subjected to the closed-world assumption, while others are considered to follow the open-world assumption. Under the closed-world assumption all atoms are expected to be false if not stated otherwise.

Let $\mathcal{P}$ be a program and consider the following transformation:

1. For all $A \in atoms(\mathcal{P})$ replace $def(A, \mathcal{P}) = \{A \leftarrow body_1, \ldots, A \leftarrow body_n\}$, where $n \geq 1$, by $\{A \leftarrow body_1 \vee \cdots \vee body_n\}$.

2. For all $A \in undef(\mathcal{P})$ add $A \leftarrow \bot$.

3. Replace all occurrences of $\leftarrow$ by $\leftrightarrow$.

The resulting set of equivalences is the well-known Clark's *completion* of $\mathcal{P}$, denoted by $c\mathcal{P}$ [5]. If the second step is omitted, then the resulting set of equivalences is called the *weak completion* of $\mathcal{P}$, denoted by $wc\mathcal{P}$.

Weak completion semantics is the approach to consider weakly completed logic programs and to reason with respect to the least models of the weak completion of these programs. As we will see later, the weak completion of a program allows both, closed-world assumption and open-world assumption, to coexist within a logic program. In the following, we are interested in the weak completion of programs. Example 1 on page 18 clarifies the definitions just introduced.

2.1.2 Three-Valued Semantics

When it comes to modelling human reasoning tasks, there is an ongoing debate in psychology on whether and how logic could be used to describe the inference process. Many psychological experiments have shown that humans do not reason with respect to classical logic. That's the reason why some researchers have proposed the use of ternary logics to model these cognitive processes.

As it is shown in [32], two valued logics is not sufficient to model the Wason Selection Task [44]. However, it is possible to reproduce the pattern of the answers given by humans in this experiment if we make use of a three-valued logics approach. Based on this, the three-valued logics seems to be a more appropriated approach to model human reasoning tasks.

Example 1. Consider the program $\mathcal{P}$ consisting of the following four clauses:

$$\begin{aligned} a &\leftarrow b. \\ a &\leftarrow \neg c. \\ d &\leftarrow \top. \\ e &\leftarrow \bot. \end{aligned}$$

The first and second clauses are facts, the third clause is a fact and the fourth clause is an assumption. The set of atoms, defined atoms and undefined atoms, are

$$\begin{aligned} atoms(\mathcal{P}) &= \{a, b, c, d, e\}, \\ def(\mathcal{P}) &= \{a, d, e\}, \\ undef(\mathcal{P}) &= \{b, c\}. \end{aligned}$$

The completion of $\mathcal{P}$, $c\mathcal{P}$, consists of the following equivalences:

$$\begin{aligned} a &\leftrightarrow b \vee \neg c. \\ b &\leftrightarrow \bot. \\ c &\leftrightarrow \bot. \\ d &\leftrightarrow \top. \\ e &\leftrightarrow \bot. \end{aligned}$$

The weak completion of $\mathcal{P}$, $wc\mathcal{P}$, consists of the following equivalences:

$$\begin{aligned} a &\leftrightarrow b \vee \neg c. \\ d &\leftrightarrow \top. \\ e &\leftrightarrow \bot. \end{aligned}$$

In a three-valued logic, the truth values are not only *true* or *false*, symbolized by $\top$ and $\bot$, respectively. But there exists also a third value, which, in the sequel, we will call *unknown* and use the symbol U to denote it.

Definition 3. Under *two-valued semantics*, a *two-valued interpretation* $\mathcal{I}$ of a propositional program $\mathcal{P}$ consists of the set $\mathcal{T} = \{\top, \bot\}$ of truth values and a mapping $\alpha \to \mathcal{T}$ which is represented by the mapping $atoms(\mathcal{P}) \to \mathcal{T}$, assuming a standard interpretation for the connectives. The truth value of a given formula under a given interpretation is determined according to the corresponding logic. $\mathcal{I}(\mathcal{F}) = \top$ denotes that interpretation $\mathcal{I}$ maps formula $\mathcal{F}$ to $\top$. The same holds for the truth value $\bot$. A *two-valued model* $\mathcal{M}$ of $\mathcal{P}$ is a two-valued interpretation where for each clause $\mathcal{C}$ occurring in $\mathcal{P}$ it holds that $\mathcal{M}(\mathcal{C}) = \top$.

We extend two-valued semantics to three-valued semantics, where the corresponding truth values are $\top$, $\bot$ and U, which mean *true*, *false* and *unknown*, respectively. A (propositional logic) three-valued interpretation $\mathcal{I}$ consists of the set $\mathcal{T} = \{\top, \bot, \mathrm{U}\}$ of truth values and a mapping $\alpha \to \mathcal{T}$ which is represented by the mapping $atoms(\mathcal{P}) \to \mathcal{T}$, assuming a standard interpretation for the connectives. The truth value of a given formula under a given interpretation is determined according to the corresponding logic.

A three-valued interpretation is represented as a pair $\mathcal{I} = \langle \mathcal{I}^\top, \mathcal{I}^\bot \rangle$ of two disjoint sets of ground atoms, where

$$\mathcal{I}^\top = \{A \mid \mathcal{I}(A) = \top\} \text{ and } \mathcal{I}^\bot = \{A \mid \mathcal{I}(A) = \bot\}.$$

Atoms which do not occur in $\mathcal{I}^\top \cup \mathcal{I}^\bot$ are mapped to U.

The three-valued interpretations can be ordered in the following way. Given two interpretations $\mathcal{I} = \langle \mathcal{I}^\top, \mathcal{I}^\bot \rangle$ and $\mathcal{J} = \langle \mathcal{J}^\top, \mathcal{J}^\bot \rangle$:

$$\mathcal{I} \preceq \mathcal{J} \text{ if and only if } \mathcal{I}^\top \subseteq \mathcal{J}^\top \text{ and } \mathcal{I}^\bot \subseteq \mathcal{J}^\bot.$$

With this ordering, called the *knowledge ordering*, both the positive interpretations $I^\top$ and the negative interpretations $I^\bot$ are minimised. More details on this and other common ways to order three-valued interpretations is shown in [34].

A *three-valued model* $\mathcal{M}$ of $\mathcal{P}$ is a three-valued interpretation where for each clause $\mathcal{C}$ occurring in $\mathcal{P}$ it holds that $\mathcal{M}(\mathcal{C}) = \top$. Three-valued models that are minimal with respect to the knowledge ordering are *minimal models*. If there is only one minimal model, then this model is called the *least model*. In the sequel, we implicitly assume all interpretations and models to be three-valued.

F	$\neg F$
$\top$	$\bot$
$\bot$	$\top$
U	U

$\wedge$	$\top$	U	$\bot$
$\top$	$\top$	U	$\bot$
U	U	U	$\bot$
$\bot$	$\bot$	$\bot$	$\bot$

$\leftrightarrow$	$\top$	U	$\bot$
$\top$	$\top$	U	$\bot$
U	U	$\top$	U
$\bot$	$\bot$	U	$\top$

$\vee$	$\top$	U	$\bot$
$\top$	$\top$	$\top$	$\top$
U	$\top$	U	U
$\bot$	$\top$	U	$\bot$

$\leftarrow$	$\top$	U	$\bot$
$\top$	$\top$	$\top$	$\top$
U	U	$\top$	$\top$
$\bot$	$\bot$	U	$\top$

Table 2.1: The truth tables for the connectives under the three-valued Łukasiewicz logic, where $\top$, $\bot$, and U denote *true*, *false*, and *unknown*, respectively.

Since the first three-valued logic has been invented by Łukasiewicz [24], various different interpretations of the three-valued connectives have been proposed. For instance, Kleene has introduced a three-valued logics [21] which is identical to the Łukasiewicz logics except for the expressions U $\leftarrow$ U and U $\leftrightarrow$ U which are evaluated to *unknown* under Kleene, but evaluated to *true* under Łukasiewicz. In the following, we consider the three-valued Łukasiewicz (or Ł-) logic. Table 2.1 gives the truth tables for negation, conjunction, disjunction, implications and equivalence under the Ł-logic.

Definition 4. Two formulas $\mathcal{F}$ and $\mathcal{G}$ are said to be *semantically equivalent*, if and only if for all interpretations $\mathcal{I}$ it follows that $\mathcal{I}(\mathcal{F}) = \mathcal{I}(\mathcal{G})$. The semantic equivalence of two given formulas $\mathcal{F}$ and $\mathcal{G}$ is denoted by $\mathcal{F} \equiv \mathcal{G}$.

Some examples are given by the following semantic equivalences:

$$\mathcal{F} \equiv \neg\neg\mathcal{G}, \tag{2.4}$$
$$\mathcal{F} \wedge \mathcal{F} \equiv \mathcal{F}, \tag{2.5}$$
$$\mathcal{F} \vee \mathcal{F} \equiv \mathcal{F}, \tag{2.6}$$
$$(\mathcal{F} \wedge \mathcal{G}) \vee \mathcal{F} \equiv \mathcal{F}, \tag{2.7}$$
$$\mathcal{F} \wedge (\mathcal{G} \vee \mathcal{H}) \equiv (\mathcal{F} \wedge \mathcal{G}) \vee (\mathcal{F} \vee \mathcal{H}) \tag{2.8}$$
$$\mathcal{F} \leftrightarrow \mathcal{G} \equiv (\mathcal{F} \leftarrow \mathcal{G}) \wedge (\mathcal{F} \rightarrow \mathcal{G}) \tag{2.9}$$
$$\mathcal{G} \rightarrow \mathcal{F} \equiv \neg\mathcal{F} \rightarrow \neg\mathcal{G} \tag{2.10}$$
$$\mathcal{F} \rightarrow \mathcal{G} \equiv \neg\mathcal{F} \vee \mathcal{G} \tag{2.11}$$
$$(\mathcal{F} \rightarrow \mathcal{G}) \wedge (\mathcal{G} \rightarrow \mathcal{H}) \equiv \mathcal{F} \rightarrow \mathcal{H} \tag{2.12}$$

All the semantic equivalences shown above hold in the standard two-valued logic. However, when it comes to the three-valued Łukasiewicz logic, the semantic equivalences from (2.4) to (2.10) hold, but (2.11) and (2.12) do not hold.

A logic program can have several models, but the intended ones are often the least models, if they exist. Least models of logic programs can often be specified as least fixed points of appropriate semantic operators [1]. The least model of $wc\mathcal{P}$ can be obtained as the least fixed point of the semantic $\Phi_\mathcal{P}$ operator, denoted by lfp $\Phi_\mathcal{P}$ [42]. Each program as well as the weak completion of each program has a least model [16]. In the sequel, $\mathcal{M}_\mathcal{P}$ denotes the least model of $wc\mathcal{P}$.

Let $\mathcal{I} = \langle \mathcal{I}^\top, \mathcal{I}^\bot \rangle$ be an interpretation. $\Phi_\mathcal{P}(\mathcal{I}) = \langle \mathcal{J}^\top, \mathcal{J}^\bot \rangle$, where

$$\begin{array}{rcll} \mathcal{J}^\top & = & \{A & | \text{ there exists } A \leftarrow \mathit{Body} \in \mathcal{P} \text{ with } \mathcal{I}(\mathit{Body}) = \top\}, \\ \mathcal{J}^\bot & = & \{A & | \text{ there exists } A \leftarrow \mathit{Body} \in \mathcal{P} \text{ and} \\ & & & \text{for all } A \leftarrow \mathit{Body} \in \mathcal{P} \text{ we find } \mathcal{I}(\mathit{Body}) = \bot\}. \end{array}$$

Example 2 on page 23 illustrates the notions of interpretations and models, as well as shows the stepwise computation of the least fixed point of the semantic $\Phi_\mathcal{P}$ operator for a concrete case.

2.1.3 Abductive Framework

Definition 5. Under two-valued semantics a set of *integrity constraints* $\mathcal{IC}$, consists of clauses of the following form:

$$\bot \leftarrow \mathit{Body},$$

where *Body* is a conjunction of literals. $\mathcal{P}$ satisfies $\mathcal{IC}$ if and only if $\mathcal{P} \cup \mathcal{IC}$ is satisfiable. Under two-valued semantics a set of clauses is satisfiable if there exists a two-valued model. This implies that the *Body* of each clause in $\mathcal{IC}$ is mapped to false under this model.

If we now extend this concept to be considered under three-valued semantics, a set of *integrity constraints* consists of clauses of the following form:

$$\mathrm{U} \leftarrow \mathit{Body},$$

where *Body* is a conjunction of literals. An interpretation $\mathcal{I}$ *violates* a finite set $\mathcal{IC}$ of integrity constraints if and only if $\mathcal{IC}$ contains an integrity constraint U $\leftarrow$ *Body*

with $\mathcal{I}(Body) = \top$. Given an interpretation $\mathcal{I}$ and a set of integrity constraints $\mathcal{IC}$, $\mathcal{I}$ *satisfies* $\mathcal{IC}$ if and only if all clauses in $\mathcal{IC}$ are *true* under $\mathcal{I}$.

Definition 6. An *abductive framework* consists of a logic program $\mathcal{P}$ as knowledge base, a finite set of *abducibles* $\mathcal{A} \subseteq \mathcal{A}_\mathcal{P}$, where

$$\mathcal{A}_\mathcal{P} = \{A \leftarrow \top \mid A \in undef(\mathcal{P})\} \cup \{A \leftarrow \bot \mid A \in undef(\mathcal{P})\},$$

a finite set of integrity constraints $\mathcal{IC}$, and the entailment relation $\models_{wcs}$. We write $\mathcal{P} \models_{wcs} F$ if and only if formula $\mathcal{F}$ holds in $\mathsf{lfp}\,\Phi_\mathcal{P}$, which is identical to the least model of $wc\mathcal{P}$. An *abductive framework* is denoted by $\langle \mathcal{P}, \mathcal{A}_\mathcal{P}, \mathcal{IC}, \models_{wcs} \rangle$. In the sequel, we assume $\mathcal{IC} = \emptyset$, if not stated otherwise.

One should observe that each $\mathcal{P}$ and, in particular, each finite set of facts and assumptions has a least model of the weak completion. For the latter, this can be obtained by mapping all heads occurring in this set to *true*. Thus, explanations as well as the union of a program and an explanation are always satisfiable.

Definition 7. An *observation* $\mathcal{O}$ is a non-empty set of literals. $\mathcal{O}$ is *explainable* in the framework $\langle \mathcal{P}, \mathcal{A}_\mathcal{P}, \mathcal{IC}, \models_{wcs} \rangle$ if and only if there exists an $\mathcal{E} \subseteq \mathcal{A}$ called *explanation* such that $\mathcal{M}_{\mathcal{P} \cup \mathcal{E}} \models_{wcs} L$ for all $L \in \mathcal{O}$ and $\mathcal{P} \cup \mathcal{E}$ satisfies $\mathcal{IC}$. If $\mathcal{E}$ consists only of undefined atoms in $\mathcal{P}$, then $\mathcal{E}$ is said to be *basic*. For a given explanation $\mathcal{E}$ if there is no explanation $\mathcal{E}'$ such that $\mathcal{E}' \subseteq \mathcal{E}$, $\mathcal{E}$ is said to be *minimal*.

Definition 8. Given a program $\mathcal{P}$ and its respective set of abducibles $\mathcal{A}_\mathcal{P}$, a set of facts and assumptions denoted by $\mathcal{C}$ such that $\mathcal{C} \subseteq \mathcal{A}_\mathcal{P}$ is a possible explanation for a given observation $\mathcal{O}$. In the sequel, a possible explanation $\mathcal{C}$ is denoted by a *candidate explanation*. The set of all possible candidate explanations for a set of abducibles $\mathcal{A}_\mathcal{P}$, denoted by $\mathcal{C}_{\mathcal{A}_\mathcal{P}}$, is equal to the power set of $\mathcal{A}_\mathcal{P}$.

A candidate explanation $\mathcal{C}$ is said to be complementary if it contains a complementary pair of clauses. The set of non-complementary candidate explanations is given by the set $\mathcal{C}_{\mathcal{A}_\mathcal{P}}$ without the complementary candidates, denoted by $\mathcal{C}^{nc}_{\mathcal{A}_\mathcal{P}}$.

A candidate explanation $\mathcal{C}$ is said to be minimal if there is no other candidate explanation $\mathcal{C}'$ such that $\mathcal{C}'$ explains the given observation $\mathcal{O}$ and $\mathcal{C}' \subseteq \mathcal{C}$. The set of minimal candidate explanations is given by the set $\mathcal{C}_{\mathcal{A}_\mathcal{P}}$ without the non-minimal candidates, denoted by $\mathcal{C}^{min}_{\mathcal{A}_\mathcal{P},\mathcal{O}}$.

If a candidate explanation in fact explains the given observation, then the candidate is said to be *positive*. Otherwise, we have a *negative* candidate explanation.

Example 2. Consider the program $\mathcal{P}$ with the following two clauses:

$$a \leftarrow b.$$
$$b \leftarrow \top.$$

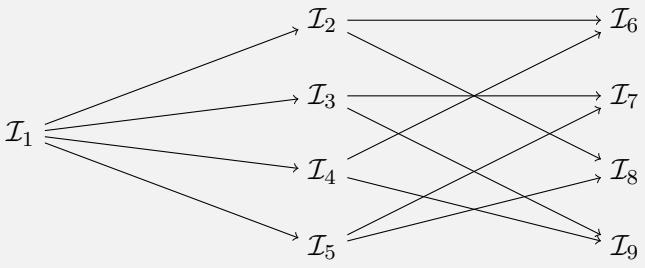
The weak completion of $\mathcal{P}$, $wc\mathcal{P}$, consists of the following equivalences:

$$a \leftrightarrow b.$$
$$b \leftrightarrow \top.$$

The set of atoms is $atoms(\mathcal{P}) = \{a, b\}$ and accordingly, under the three-valued logics, there are the following nine possible interpretations:

$$\begin{array}{lll} \mathcal{I}_1 = \langle \emptyset, \emptyset \rangle & \mathcal{I}_2 = \langle \{a\}, \emptyset \rangle & \mathcal{I}_3 = \langle \emptyset, \{a\} \rangle \\ & \mathcal{I}_4 = \langle \{b\}, \emptyset \rangle & \mathcal{I}_5 = \langle \emptyset, \{b\} \rangle \\ & \mathcal{I}_6 = \langle \{a, b\}, \emptyset \rangle & \mathcal{I}_7 = \langle \emptyset, \{a, b\} \rangle \\ & \mathcal{I}_8 = \langle \{a\}, \{b\} \rangle & \mathcal{I}_9 = \langle \{b\}, \{a\} \rangle \end{array}$$

The interpretation $\mathcal{I}_6$ is the only model of $\mathcal{P}$. The following graph shows the knowledge ordering of the nine interpretations, where $\mathcal{I}_i \rightarrow \mathcal{I}_j$ means that $\mathcal{I}_i \preceq \mathcal{I}_j$:



Let us compute $\Phi_{\mathcal{P}}$ with $\mathcal{I}_1 = \langle \emptyset, \emptyset \rangle$:

$$\begin{array}{lcll} \Phi_{\mathcal{P}}(\mathcal{I}_1) & = & \langle \{b\}, \emptyset \rangle & = \mathcal{I}_4, \\ \Phi_{\mathcal{P}}(\mathcal{I}_4) & = & \langle \{a, b\}, \emptyset \rangle & = \mathcal{I}_6, \\ \Phi_{\mathcal{P}}(\mathcal{I}_6) & = & \langle \{a, b\}, \emptyset \rangle & = \mathcal{I}_6. \end{array}$$

The interpretation $\mathcal{I}_6$ is least fixed point of $\Phi_{\mathcal{P}}$, e.g. $\mathsf{lfp}\,\Phi_{\mathcal{P}} = \mathcal{I}_6$, which also corresponds to the least model of $wc\mathcal{P}$.

For a given set of candidate explanations $\mathcal{C}$, the set of positive candidate explanations, denoted by $\mathcal{C}^{pos}$, among the set of candidates is defined as

$$\mathcal{C}^{pos} = \{\mathcal{C}_j \mid \mathcal{C}_j \in \mathcal{C} \text{ and } \mathcal{P} \cup \mathcal{C}_j \models_{wcs} \mathcal{O}\}.$$

The set of negative candidate explanations, denoted by $\mathcal{C}^{neg}$, is given by $\mathcal{C} \setminus \mathcal{C}^{pos}$.

Proposition 1. Let $\langle \mathcal{P}, \mathcal{A}_\mathcal{P}, \models_{wcs} \rangle$ be an *abductive framework*, $\mathcal{O}$ an observation, and $\mathcal{E} \subseteq \mathcal{A}_\mathcal{P}$ an explanation for $\mathcal{O}$ which contains a complementary pair $c \leftarrow \top, c \leftarrow \bot$. Then, $\mathcal{E}' = \mathcal{E} \setminus \{c \leftarrow \bot\}$ is also an explanation for $\mathcal{O}$ and $\mathcal{M}_{\mathcal{P}\cup\mathcal{E}} = \mathcal{M}_{\mathcal{P}\cup\mathcal{E}'}$.

Proof: [39]. □

Proposition 2. Explanations are *monotonic*. Let $\langle \mathcal{P}, \mathcal{A}_\mathcal{P}, \models_{wcs} \rangle$ be an abductive framework and $\mathcal{O}$ an observation. If $\mathcal{E}$ is an explanation for $\mathcal{O}$, then any set $\mathcal{E}'$ such that $\mathcal{E} \subseteq \mathcal{E}' \subseteq \mathcal{A}$ is also an explanation for $\mathcal{O}$.

Proof: [16]. □

There are two possible ways of abductive reasoning: credulous and sceptical. In credulous reasoning all the consequences which follow from the program and at least one of the explanations for the observation are valid conclusions. On the other hand, in sceptical reasoning, a conclusion is only valid if it follows from the program and all the explanations for the observation.

Let $\langle \mathcal{P}, \mathcal{A}_\mathcal{P}, \mathcal{IC}, \models_{wcs} \rangle$ be an abductive framework, $\mathcal{O}$ an observation and $\mathcal{F}$ a formula:

- $\mathcal{F}$ follows credulously from $\mathcal{P}$ and $\mathcal{O}$ if and only if there exists an explanation $\mathcal{E}$ for $\mathcal{O}$ such that $\mathcal{P} \cup \mathcal{E} \models_{wcs} \mathcal{F}$.

- $\mathcal{F}$ follows sceptically from $\mathcal{P}$ and $\mathcal{O}$ if and only if for all explanations $\mathcal{E}$ for $\mathcal{O}$ we find $\mathcal{P} \cup \mathcal{E} \models_{wcs} \mathcal{F}$.

The set of sceptical consequences $\mathcal{S}_\mathcal{E}$ of a given set of explanations $\mathcal{E}$ is defined as follows

$$\begin{aligned} \mathcal{S}_\mathcal{E} &= \bigcap_{\mathcal{E}' \in \mathcal{E}} \mathcal{M}_{\mathcal{P}\cup\mathcal{E}'} \\ &= \langle \bigcap_{\mathcal{E}' \in \mathcal{E}} \mathcal{M}^{\top}_{\mathcal{P}\cup\mathcal{E}'}, \bigcap_{\mathcal{E}' \in \mathcal{E}} \mathcal{M}^{\bot}_{\mathcal{P}\cup\mathcal{E}'} \rangle. \end{aligned}$$

All the notions about the abductive framework, including the definition of candidate explanations and how to reason with respect to the weak completion of programs is presented in more details by Example 3 on page 26.

2.1.4 Complexity

Proposition 3. Computing lfp $\Phi_{\mathcal{P}}$ can be done in polynomial time for acyclic logic programs $\mathcal{P}$.

Proof: See [40], Proposition 12. □

Proposition 4. Deciding whether an observation $\mathcal{O}$ has an abductive explanation based on a given program $\mathcal{P}$ and its respective set of abducibles $\mathcal{A}_{\mathcal{P}}$ is NP-complete.

Proof: See [40], Proposition 13. □

Proposition 5. Deciding whether $\mathcal{P} \cup \mathcal{E} \models_{wcs} \mathcal{F}$ does not hold for all explanations $\mathcal{E}$ given $\mathcal{A}_{\mathcal{P}}$ is NP-complete.

Proof: See [40], Proposition 14. □

Proposition 6. Let $\mathcal{L} \subseteq \sum^*$ be a language. Then $\mathcal{L}$ is NP-complete if and only if $\mathcal{L}$ is coNP-complete.

Proof: See [29], Proposition 10.1. □

Proposition 7. Deciding whether $\mathcal{P} \cup \mathcal{E} \models_{wcs} \mathcal{F}$ holds for all explanations $\mathcal{E}$ given $\mathcal{A}_{\mathcal{P}}$ is coNP-complete.

Proof: The opposite problem is shown to be NP-complete by Proposition 5. By Proposition 6, deciding the above problem is coNP-complete. □

Definition 9. A Language $\mathcal{L}$ is in the complexity class DP if and only if there are two languages $\mathcal{L}_1 \in$ NP and $\mathcal{L}_2 \in$ coNP such that $\mathcal{L} = \mathcal{L}_1 \cap \mathcal{L}_2$.

Proposition 8. The question whether $\mathcal{F}$ follows sceptically from an abductive problem is DP-complete.

Proof: See [40], Proposition 17. □

Example 3. Consider program $\mathcal{P}$ consisting of the following clauses:

$$c \leftarrow a.$$
$$c \leftarrow b.$$

Assume that $\mathcal{IC} = \emptyset$ and that $\mathcal{O} = \{c\}$. Let us consider this observation in the abductive framework $\langle \mathcal{P}, \mathcal{A}_\mathcal{P}, \mathcal{IC}, \models_{wcs} \rangle$, where the set of abducibles $\mathcal{A}_\mathcal{P}$ consists of the following facts and assumptions:

$$a \leftarrow \top. \qquad a \leftarrow \bot.$$
$$b \leftarrow \bot. \qquad b \leftarrow \bot.$$

There are the following five non-complementary explanations for $\mathcal{O}$:

$$\begin{aligned}
\mathcal{E}_1 &= \{a \leftarrow \top\},\\
\mathcal{E}_2 &= \{b \leftarrow \top\},\\
\mathcal{E}_3 &= \{a \leftarrow \top, b \leftarrow \bot\},\\
\mathcal{E}_4 &= \{a \leftarrow \bot, b \leftarrow \top\},\\
\mathcal{E}_5 &= \{a \leftarrow \top, b \leftarrow \top\}.
\end{aligned}$$

The only minimal explanations for $\mathcal{O}$ are $\mathcal{E}_1$ and $\mathcal{E}_2$. As a and b does not follow from all the explanations, it does not follow sceptically, but only credulous, from $\mathcal{P}$ and $\mathcal{O}$.

The set of candidate explanations $\mathcal{C}$ for $\mathcal{P}$ and $\mathcal{O}$ is the power set of $\mathcal{A}_\mathcal{P}$. The set of non-complementary candidate explanations $\mathcal{C}^{nc}_{\mathcal{A}_\mathcal{P}}$ consists of the following nine candidates:

$$\begin{aligned}
&\mathcal{C}_1 = \emptyset,\\
&\mathcal{C}_2 = \{a \leftarrow \top\}, && \mathcal{C}_2 = \{a \leftarrow \bot\},\\
&\mathcal{C}_4 = \{b \leftarrow \top\}, && \mathcal{C}_5 = \{b \leftarrow \bot\},\\
&\mathcal{C}_6 = \{a \leftarrow \top, b \leftarrow \top\}, && \mathcal{C}_7 = \{a \leftarrow \top, b \leftarrow \bot\},\\
&\mathcal{C}_8 = \{a \leftarrow \bot, b \leftarrow \top\}, && \mathcal{C}_9 = \{a \leftarrow \bot, b \leftarrow \bot\}.
\end{aligned}$$

The set of minimal candidate explanations for $\mathcal{A}_\mathcal{P}$ and $\mathcal{O}$, $\mathcal{C}^{min}_{\mathcal{A}_\mathcal{P},\mathcal{O}}$, consists of the candidates $\mathcal{C}_1$, $\mathcal{C}_2$, $\mathcal{C}_3$, $\mathcal{C}_4$, $\mathcal{C}_5$ and $\mathcal{C}_9$. The set of positive candidate explanations $\mathcal{C}^{min,pos}_{\mathcal{A}_\mathcal{P},\mathcal{O}}$ consists only of $\mathcal{C}_2$ and $\mathcal{C}_4$. Finally, the sceptical consequences of $\mathcal{C}^{min,pos}_{\mathcal{A}_\mathcal{P},\mathcal{O}}$ is given by

$$\mathcal{S}_{\mathcal{C}^{min,pos}_{\mathcal{A}_\mathcal{P},\mathcal{O}}} = \langle \{a, c\} \cap \{b, c\}, \emptyset \cap \emptyset \rangle = \langle \{c\}, \emptyset \rangle$$

2.2 Connectionist Networks

A connectionist network consists of interconnected processing units. The general model of a processing unit consists of a summing part followed by an output part. The summing part receives N input values, weights each value, and computes a weighted sum. The weighted sum is called the *activation value*. The sign of the weight for each input determines whether the input is *excitatory* (positive weight) or *inhibitory* (negative weight). The output part produces a signal from the activation value.

We can divide these connectionist networks into two large classes. One class consists of the feed-forward networks and the other class consists of the recurrent networks. In general, whether a network belongs to one class or another can be decided based on its overall connectivity. If the network has one or more cycles, i.e., it is possible to follow a path from a unit to itself, then the network is referred to as recurrent. Feed-Forward networks have no cycles.

We assume the reader to be familiar with connectionist networks as, for example, it is defined in [13]. In particular, we will use McCulloch-Pitts networks of threshold units as defined in [25] and multi-layer feed-forward networks as defined in [36]. We use modified connections of the following form:

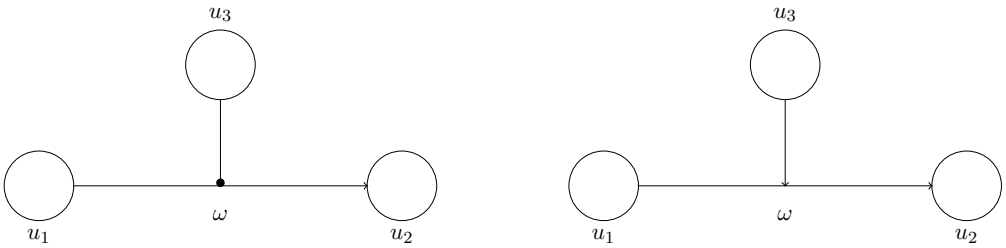
- In case of a *positively modified connection*, the input $i = wv$ of a unit is only received if the modifier m is 1, where v is the output of the sending unit and w is the weight of the connection:

$$i = \begin{cases} wv & \text{if } m = 1, \\ 0 & \text{if } m = 0. \end{cases}$$

- In case of a *negatively modified connection*, the input $i = wv$ is only received if the modifier m is 0:

$$i = \begin{cases} 0 & \text{if } m = 1, \\ wv & \text{if } m = 0. \end{cases}$$

The modifier m is the output of another unit. Figure 2.1 shows the graphical representations of a positively and a negatively modified connection.



(a) Negative modification. **(b)** Positive modification.

Figure 2.1: The output v of unit u_1 is received by the input i of unit u_2 only if the modifier m of unit u_3 is 0 (a) or 1 (b), respectively.

2.2.1 Recurrent Networks

The following notions introduced here are based on [15, 26]. If the input in a feed-forward network is held constant, the trajectory of the network in state space will remain at a single point. Clearly, in order to archive interesting sequential behaviour in the presence of a constant input, there must be recurrent connections in the network. Therefore, because our main goal is to generate a sequence of candidate explanations for the abductive framework which characterises a sequential behaviour, we will focus on recurrent networks.

Recurrent networks were created in the 1980's and have been an important focus of research and development during the 1990's. They are designed to learn sequential or time-varying patterns, because each unit can use its internal memory to maintain information about the previous input. In cases of language, for example, if we would like to predict the next word in a sentence it is very important to know which words came before it. Thus, allowing the network to gain a deeper understanding of the statement is a great property when it comes to problems like this one. Language modelling is only one of the many applications for these recurrent networks. They have also been applied to image processing, pattern recognition, classification, clustering, data mining and so on.

The topology of a recurrent network can be very general, since they allow feedback connections. This means that every unit can be connected to every unit, including to itself. Allowing the presence of feedback connections among units brings an important characteristic to this networks, a time-dependent behaviour. Because of this, they can also be classified as dynamic networks. The state of a network, at one moment in time, also depends on the state at a previous moment in time. The dynamics of a recurrent neural network can be continuous or discrete in time. When a continuous time system is simulated, it is usually converted into a set of simple first order difference equations, which is formally identical to a discrete system. Therefore, we will concentrate on discrete time networks.

The dynamics of a recurrent network can be described, for instance, by a discrete time system

$$x(k+1) = f(x(k)), \quad k \geq 1, 2, ...,$$

where x denotes the state of the corresponding network and f is some suitable mapping.

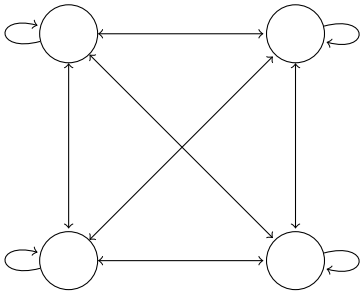
The architecture of the recurrent network can vary from fully interconnected (Figure 2.2a) to partially connected (Figure 2.2b). Fully connected networks do not have distinct input layers of nodes, and each unit has a weighted connection to every other unit in the architecture as well as a single feed-back connection to itself.

Simple partially recurrent networks (Figure 2.2b) have distinct input layer of nodes and not every unit must be connected to every other unit. However, although some nodes are part of a feed-forward structure, other units provide the sequential context and receive feed-back connections from other units. These units are denoted the context units and composed what we will later refer to as the context layer. Weights from the context units (e.g., C1 and C2 in out example) are processed like those for the input units, for example, using back propagation. The context units receive time-delayed feedback from, in the case of Figure 2.2b, the second layer units. Training data consists of inputs and their desired successor outputs. These type of networks can be trained, for example, to predict the next letter in a string of characters or to validate a string of characters.

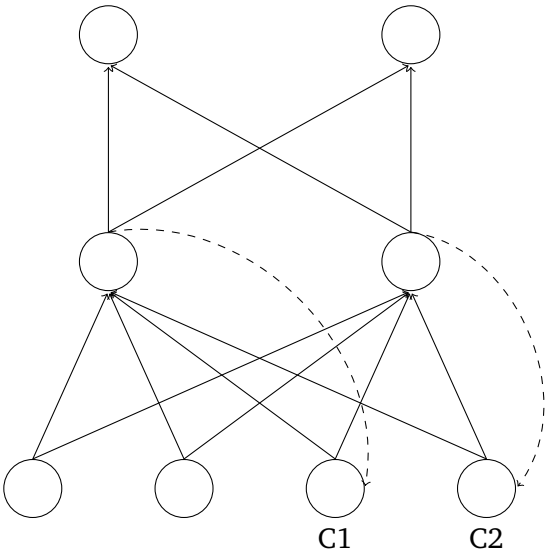
Back-propagation in feed-forward networks moves backward from the final error through the outputs, weights and inputs of each hidden layer, assigning those weights responsibility for a portion of the error by calculating their partial derivatives. These derivatives are then used by the learning rule, e.g. gradient descent, to adjust the weights up or down, depending on which direction decreases error.

Recurrent networks rely on an extension of back-propagation called back-propagation through time. Time, in this case, is simply expressed by a well-defined, ordered series of calculations linking one time step to the next, which is all back-propagation needs to work. Note that, Neural Networks, recurrent or not, are simply nested composite functions like $f(g(h(x)))$. Adding a time element only extends the series of functions for which we calculate derivatives with the chain rule.

In the sequel, the algorithm used will be the one developed in [37]. The algorithm is a gradient search in weight space for a set of weights which implements the desired function. In [37] the algorithm is demonstrated for nonrecurrent networks, but it can also be applied to recurrent networks.



(a) Fully connected.



(b) Partially connected.

Figure 2.2: Examples of a fully connected (a) and a partially connected (b) recurrent network. Solid lines represent trainable connections.

To illustrate some of the ideas introduced here, a simple partially connected Recurrent Network consisting of two units is shown in Example 4 on page 33. Now we will now introduce two fundamental ways which can be used to add feed-back into feed-forward multi-layer networks, i.e. Jordan Networks and Elman networks.

2.2.2 Jordan Networks

A theory of serial order which describes how sequences of actions might be learned and performed is presented in [19]. The theory is embodied in the form of a parallel distributed processing [35] or connectionist network [13]. Such networks are composed of many simple processing units that are connected though weighted links. The structure of the network consists of plan units, state units, hidden units and output units. The plan units and the state units together serve as the input units for a network which implements the output function f through weighted connections from the plan and state units to the output units. The next-state function is implemented with recurrent connections from the output units to the state units. This allows the current state to depend on the previous state and on the previous output (which is itself a function of the previous state and the plan). The full overall structure of the network is shown in Figure 2.3.

In the proposed network, there is no explicit representation of temporal order and no explicit representation of action sequences. This is due to the fact that there is only one set of output units for the network, so that at any point in time, only one output vector is present. The network proposed by Jordan essentially implements the output function which relates patterns as a network associative memory in which many associations are stored in the same set of weights. The learning and generalisation of abilities demonstrated for such networks [35, 38] are just those that are needed for the output function.

Although it is possible that the next-state function is learned, this is not necessary for the system as a whole to be able to learn to produce sequences. Therefore, the recurrent connections have a fixed weight of 1.0 and all the other connections are trained. Learning is realised as an error-correcting process in which parameters of the network are incrementally adjusted based on the difference between the actual output of the network and the desired output. Essentially, the next-state function provides a time-varying state vector, and associations are learned from this state vector and plan vector to desired output vectors.

The form that desired output vectors are assumed to take is a generalisation of the approach used in traditional error-correlation schemes. Instead of assuming that a value is specified for each output unit, it is assumed that there are constraints

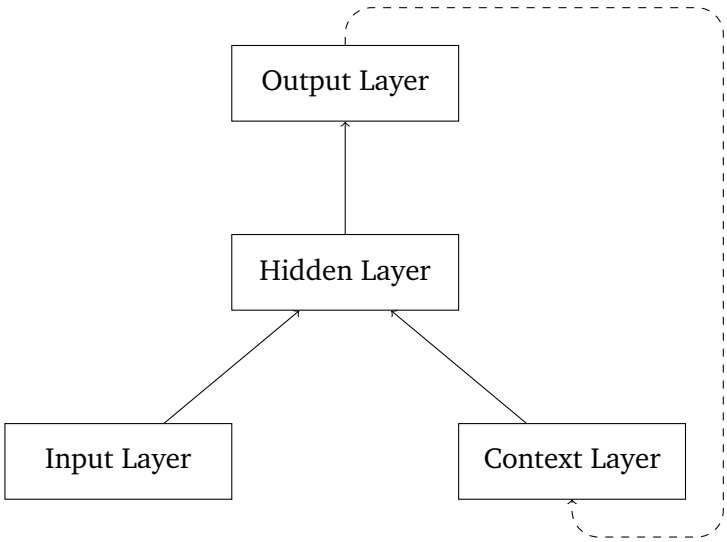


Figure 2.3: A simple recurrent network in which activations are copied from the output layer to the context layer on a one-for-one basis, with fixed weight of 1.0. Solid lines represent trainable connections.

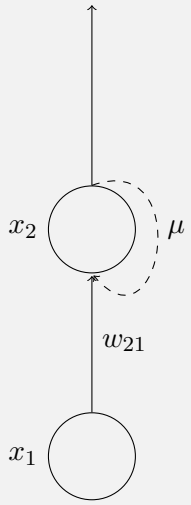
specified on the values of the output units. Constraints may specify a range of values that an output unit may have, a particular value, or no value at all. It is also possible to consider constraints that are defined among output units. For example, the sum of the activations of a set of units might be required to take on a particular value. More details on how these constraints enter into the learning process is given in [19].

2.2.3 Elman Networks

As introduced in [11] and [10], an Elman Network is a three-layered recurrent network, with an additional set of units called the context layer. Both the input units and the context units activate the hidden units. The hidden units then feed forward to activate the output units and also feed back to activate the context units. This constitutes the forward activation. The output is then compared with the expected output for the given input and back propagation is used to adjust connection strengths incrementally. Recurrent connections are fixed at 1.0 and are not subject to adjustments. The full overall structure of the network is shown in Figure 2.4.

As one may notice, this approach is an variation of the Jordan Networks introduced in the previous Section. The only difference between them is on the source of the recurrent connections, which is the output layer and the hidden layer for Jordan and Elman networks, respectively. This is also the main characteristic of the Elman networks, which is an internal representation of time.

Example 4. An example of a simple partially connected recurrent network consisting of two units is shown below.



The recurrent connection of the output unit back to itself means that the output of the network depends not only on the input, but also on the state of the network at the previous time step.

For example, letting μ be the value of the recurrent weight, and assuming for simplicity that the units are linear ,i.e., the function is the identity function, the activation of the output unit at time t is given by

$$x_2(t) = \mu x_2(t-1) + w_{21}x_1(t)$$

where $x_1(t)$ is assumed to be constant over time.

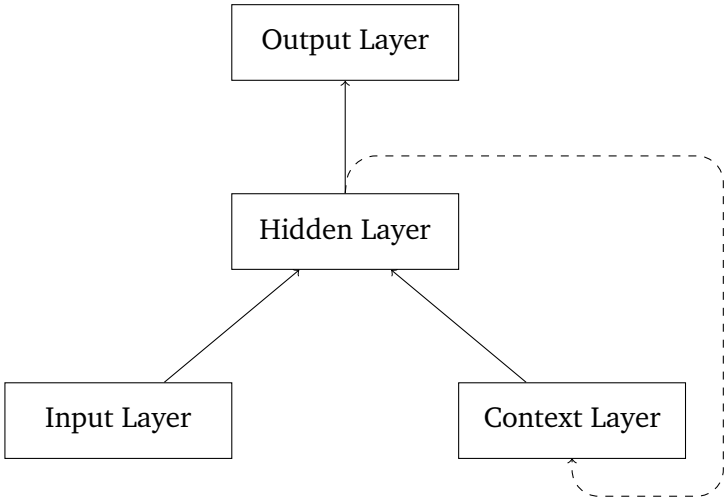


Figure 2.4: A simple recurrent network in which activations are copied from the hidden layer to the context layer on a one-for-one basis, with fixed weight of 1.0. Solid lines represent trainable connections.

In feed-forward networks employing hidden units and a learning algorithm, the hidden units develop internal representations for the input patterns that recode those patterns in a way which enables the network to produce the correct output for a given input. In the architecture presented in [11], the context units remember the previous internal state. Therefore, the hidden units have the task of mapping both an external input and the previous internal state to the desired output. Because the patterns on the hidden units are saved as context, the hidden units must accomplish this mapping and at the same time develop representations which are useful encodings of the temporal properties of the sequential input. Thus, the internal representations are now sensitive to temporal context.

3 Generating a Static Sequence of Candidate Explanations

In the introduction we have briefly discussed a method using connectionist networks to compute the sceptical consequences of a given program $\mathcal{P}$ and its respective set of abducibles $\mathcal{A}_\mathcal{P}$ by means of abduction, which was introduced in [39]. This chapter will focus on the component of this network responsible for generating the candidate explanations for $\mathcal{A}_\mathcal{P}$. In particular, this component is a McCulloch-Pitts network [25] which sequentially generates all non-complementary candidate explanations in a static pre-defined sequence. After explaining how this has been done, we would like to improve this approach by extending the current version of this network such that it will now generate only minimal candidate explanations.

3.1 Non-Complementary Candidate Explanations

Consider a program $\mathcal{P}$, its respective set of abducibles $\mathcal{A}_\mathcal{P}$, an observation $\mathcal{O}$ and an explanation $\mathcal{E} \subseteq \mathcal{A}_\mathcal{P}$ for $\mathcal{O}$ which contains a complementary pair of clauses, e.g. $c \leftarrow \top$ and $c \leftarrow \bot$. As it is stated in Proposition 1, $\mathcal{E}' = \mathcal{E} \setminus \{c \leftarrow \bot\}$ is also an explanation for $\mathcal{O}$ and $\mathcal{M}_{\mathcal{P}\cup\mathcal{E}} = \mathcal{M}_{\mathcal{P}\cup\mathcal{E}'}$. From Proposition 1 we can then directly conclude that complementary explanations do not need to be considered. Because of this, we will start by focusing only on the generation of non-complementary candidate explanations.

For a given a program $\mathcal{P}$ containing n undefined atoms, the set of abducibles $\mathcal{A}_\mathcal{P}$ consists of $2n$ facts and assumptions. There are 3^{2n} candidate explanations and 3^n non-complementary candidate explanations for $\mathcal{A}_\mathcal{P}$. As mentioned before, we will focus on the non-complementary candidates only which is already a significant optimisation on the number of candidates. This is illustrated in Example 5 on page 36. These candidate explanations can be represented by means of n-bit binary vectors, where $n = |\mathcal{A}_\mathcal{P}|$ and each bit represents one of the n atoms or literals in $\mathcal{A}_\mathcal{P}$. For example, each candidate explanation in $\mathcal{C}_{\mathcal{A}_{\mathcal{P}_{sup}}}$ can be represented by a 4-bit binary vector.

Example 5. Consider Byrne's suppression task discussed in the introduction. If we focus on the conditionals given Group II (Simple + Alternative), they can be represented by the program $\mathcal{P}_{sup}$ consisting of the following four clauses:

$$\begin{array}{lcl} l & \leftarrow & e \wedge \neg ab_1. \\ l & \leftarrow & t \wedge \neg ab_2. \\ ab_1 & \leftarrow & \bot. \\ ab_2 & \leftarrow & \bot. \end{array}$$

Where the atoms l, e, and, t stand for *she sill study late in the library*, *she has an essay to finish*, and *she has a text book to read*, respectively. Moreover, the atoms ab_1 and ab_2 stand for the abnormalities.
We can observe that only the atoms e and t are undefined. Thus, the set of abducibles $\mathcal{A}_{\mathcal{P}_{sup}}$ consists of the following four facts and assumptions:

$$\begin{array}{ll} e \leftarrow \top. & e \leftarrow \bot. \\ t \leftarrow \top. & t \leftarrow \bot. \end{array}$$

There are eighty-one candidates explanations for $\mathcal{A}_{\mathcal{P}_{sup}}$, where nine of them are non-complementary. The set of non-complementary candidate explanations $\mathcal{C}_{\mathcal{A}_{\mathcal{P}_{sup}}}$ consists of the following nine candidates:

$$\begin{array}{ll} \mathcal{C}_0 = \emptyset, & \\ \mathcal{C}_1 = \{e \leftarrow \top\}, & \mathcal{C}_2 = \{e \leftarrow \bot\}, \\ \mathcal{C}_3 = \{t \leftarrow \top\}, & \mathcal{C}_4 = \{t \leftarrow \bot\}, \\ \mathcal{C}_5 = \{e \leftarrow \top, t \leftarrow \top\}, & \mathcal{C}_6 = \{e \leftarrow \top, t \leftarrow \bot\}, \\ \mathcal{C}_7 = \{e \leftarrow \bot, t \leftarrow \top\}, & \mathcal{C}_8 = \{e \leftarrow \bot, t \leftarrow \bot\}. \end{array}$$

Candidate Explanations	$e^{\top}$	$e^{\bot}$	$t^{\top}$	$t^{\bot}$
$\emptyset$	0	0	0	0
$\{e \leftarrow \top\}$	1	0	0	0
$\{e \leftarrow \bot\}$	0	1	0	0
$\{t \leftarrow \top\}$	0	0	1	0
$\{t \leftarrow \bot\}$	0	0	0	1
$\{e \leftarrow \top, t \leftarrow \top\}$	1	0	1	0
$\{e \leftarrow \top, t \leftarrow \bot\}$	1	0	0	1
$\{e \leftarrow \bot, t \leftarrow \top\}$	0	1	1	0
$\{e \leftarrow \bot, t \leftarrow \bot\}$	0	1	0	1

Table 3.1: Binary vector definitions of the non-complementary candidate explanations for the set of abducibles $\mathcal{A}_{\mathcal{P}_1}$.

The candidate explanation

$$\mathcal{C}_0 = \emptyset$$

is represented by 0000, and the candidate explanation

$$\mathcal{C}_1 = \{e \leftarrow \top\}$$

is represented by 1000. Table 3.1 shows the 4-bit representation for each of the nine non-complementary candidate explanations in $\mathcal{C}_{\mathcal{A}_{\mathcal{P}_1}}$. Given this, the goal is to construct a neural network which sequentially generates the binary vectors shown in Table 3.1.

We start by formally specifying a finite automaton with state output (Moore machine) [27] as shown in Figure 3.1 on page 38. This automaton has the set $\{0, 1\}$ of input symbols, the set $\{0000, 0001, 0010, 0100, 1000, 0101, 1001, 0110, 1010\}$ of output symbols, the set $\{q_0, q_1, q_2, q_3, q_4, q_5, q_6, q_7, q_8\}$ of states with q_0 being the initial state, the following transition function δ and the output function λ:

δ	0	1
q_0	q_0	q_1
q_1	q_1	q_2
q_2	q_2	q_3
q_3	q_3	q_4
q_4	q_4	q_5
q_5	q_5	q_6
q_6	q_6	q_7
q_7	q_7	q_8
q_8	q_8	q_0

$state$	λ
q_0	0000
q_1	1000
q_2	0100
q_3	0010
q_4	0001
q_5	1010
q_6	1001
q_7	0110
q_8	0101

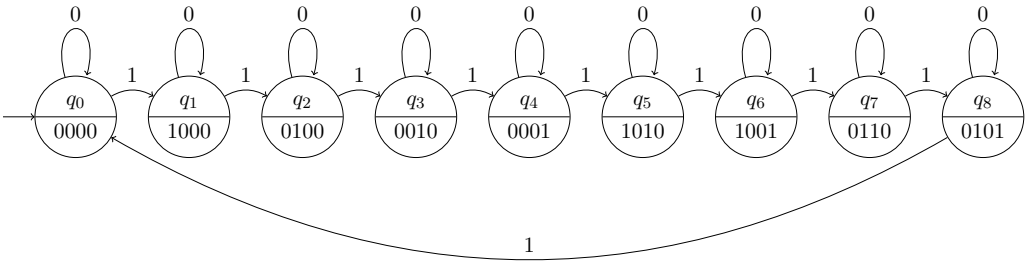


Figure 3.1: A finite automaton generating all possible and non-complementary candidate explanations for $\mathcal{A}_{\mathcal{P}_2}$.

Warren McCulloch and Walter Pitts have shown in [25] that each finite automaton with state output can be turned into a neural network of binary threshold units (sometimes called a *McCulloch-Pitts network*) receiving an input via its input units and producing an output via its output units. In particular, a McCulloch-Pitts network to sequentially generate all the non-complementary candidate explanations for $\mathcal{A}_{\mathcal{P}_1}$ can be obtained from the finite automaton shown in Figure 3.1.

The network has one input unit *next* triggering the state transitions; nine internal units, which are called the hidden units; five output units, consisting of $e^{\top}$, $e^{\bot}$, $t^{\top}$, $t^{\bot}$ and *done*, representing the four elements in the the set of abducibles $\mathcal{A}_{\mathcal{P}_1}$ and whether all the candidate explanations have been generated. The output unit *done* is active when the last candidate is generated, and passive, otherwise. The network $\mathcal{N}_{\mathcal{A}_{\mathcal{P}_1}}$ is shown in Figure 3.2 and more details on how it has been constructed can be found in [39].

Note that, because our focus here is on the generation of candidate explanations, we only concentrate on one of the networks which compose the overall network described in [39]. The main goal of this network is to compute the sceptical abductive consequences for a given program $\mathcal{P}$ and observation $\mathcal{O}$.

Among the other components of this overall network, there is a network denoted by $e\mathcal{N}_{\mathcal{P}}$ which encodes the given program $\mathcal{P}$ and computes the lfp $\Phi_{\mathcal{P}}$. The network $e\mathcal{N}_{\mathcal{P}}$ has as input the output of the network $\mathcal{N}_{\mathcal{A}_{\mathcal{P}}}$, which is a given non-complementary candidate explanation $\mathcal{C}$ for the set of abducibles $\mathcal{A}_{\mathcal{P}}$. The network $e\mathcal{N}_{\mathcal{P}}$ uses this to compute the lfp $\Phi_{\mathcal{P}\cup\mathcal{C}}$ and its result is given as output of the network.

Once this network is done with the computation of lfp $\Phi_{\mathcal{P}\cup\mathcal{C}}$ and outputs the expected results, the overall network then requests a new candidate explanation by activating unit *next* of the network $\mathcal{N}_{\mathcal{A}_{\mathcal{P}}}$ and the process is repeated until unit *done* (*done*) is active and there is no more candidate explanations to be considered. In this moment, there is another component of the network which considers all the outputs given by

the network network $e\mathcal{N}_{\mathcal{P}}$ together with a given observation $\mathcal{O}$ and computes the sceptical consequences of it.

Considering the behaviour of the overall network and because we know that the computation of $\mathsf{lfp}\,\Phi_{\mathcal{P}\cup\mathcal{C}}$ is not done in a fixed number of time steps, it is clear that units *next* and *done* are crucial to assure communications among the several components as well as to make sure that the candidate explanations will be generated on demand.

Initially, unit q_0 will be active and all other units passive. As soon as unit *next* is externally activated, then unit q_0 will become passive and unit q_1 will become active. Moreover, unit q_1 will activate unit $e^{\top}$ representing the first non-complementary and non-empty candidate explanation

$$\mathcal{C}_1 = \{e \leftarrow \top\}.$$

Upon further external activation of unit *next*, this process will continue until unit q_8 becomes active. Unit q_8 will activate units $e^{\perp}$ and $t^{\perp}$ representing the last candidate explanation to be considered

$$\mathcal{C}_8 = \{e \leftarrow \perp,\ t \leftarrow \perp\}.$$

Another activation of unit *next* will return the network into its initial state. Because the environment needs to know when all possible explanations have been generated, the output unit *done* will also become active once unit q_8 becomes passive. Unit *done* will remain active unless unit *next* is externally activated again.

While the approach described here restricted the candidate explanations for a given set of abducibles to the non-complementary ones, it is well known that abduction often restricts this explanations to the minimal ones. Given this, we would like to introduce an extension of the current approach such that only the non-complementary and minimal candidate explanations are generated.

3.2 Minimal Candidate Explanations

As mentioned before, it is often the case that one is interested in reasoning w.r.t. the minimal explanations. Intuitively, minimal explanations are interesting in the sense that they are not too general and do not trivially imply the given observation. In other words, the minimality property prevents from over-restricting the models of the abductive explanations.

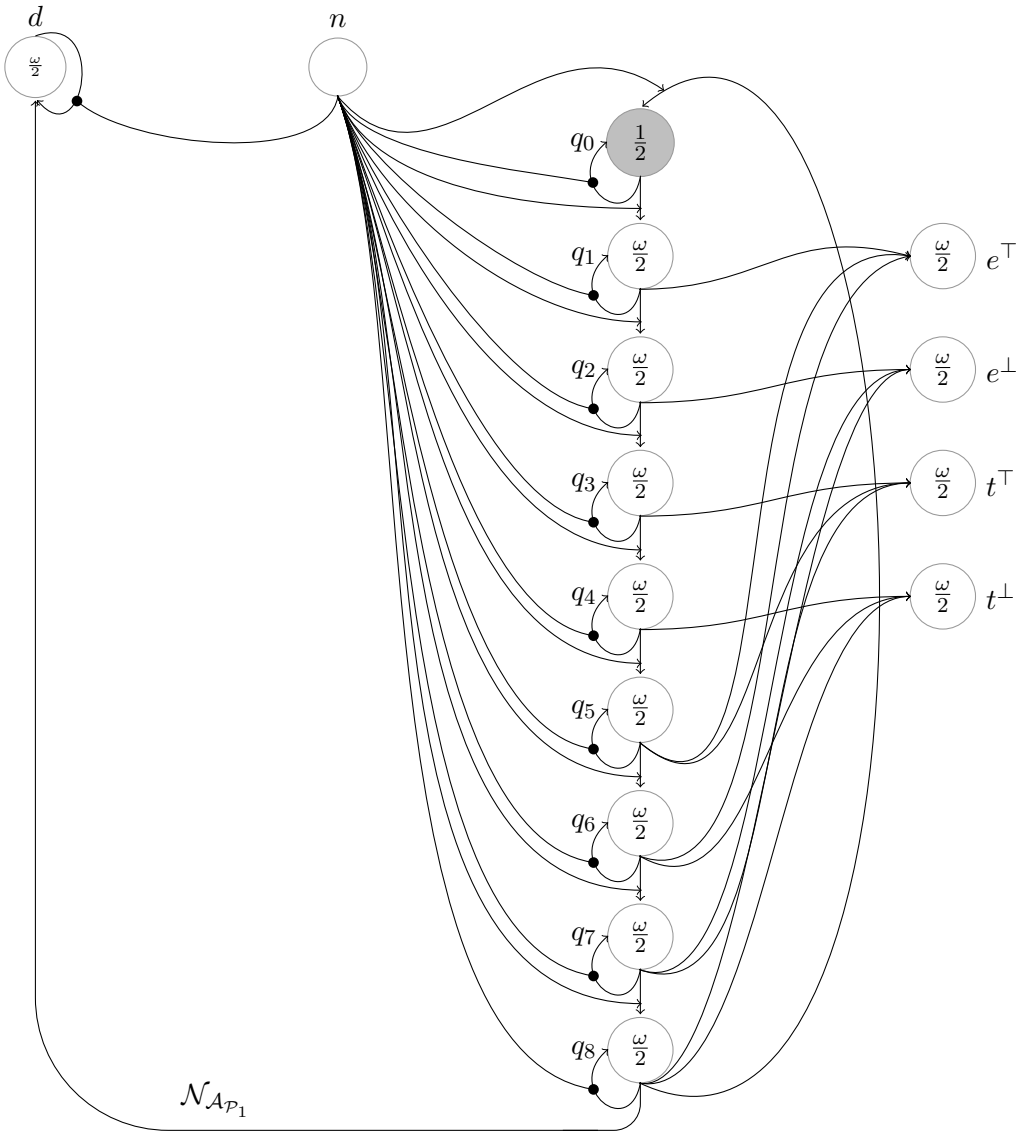


Figure 3.2: McCulloch-Pitts network $\mathcal{N}_{\mathcal{A}_{\mathcal{P}_1}}$ designed to sequentially generate all the non-complementary candidate explanations for $\mathcal{A}_{\mathcal{P}_1}$.

From Proposition 2 it directly follows that, for a given program $\mathcal{P}$, its respective set of abducibles $\mathcal{A}_\mathcal{P}$ and an observation $\mathcal{O}$, if $\mathcal{E} \subseteq \mathcal{A}_\mathcal{P}$ is an explanation for $\mathcal{O}$, i.e. $\mathcal{P} \cup \mathcal{E} \models_{wcs} \mathcal{O}$, then the explanation $\mathcal{E}$ with additional facts and assumptions is still an explanation for $\mathcal{O}$. This means that, when generating the non-complementary candidate explanations, we can avoid the generation of candidates which are a superset of those candidates known to have explained the given observation. In this way, no candidate $\mathcal{C}$ such that $\mathcal{E} \subseteq \mathcal{C}$ and $\mathcal{P} \cup \mathcal{E} \models_{wcs} \mathcal{O}$ should be generated.

Thus, the goal now is to construct a network which will restrict the generation of non-complementary candidate explanations to the minimal ones. One way to archive this is by extending the network $\mathcal{N}_{\mathcal{A}_{\mathcal{P}_1}}$ shown in Figure 3.2. For this purpose, reconsider the program $\mathcal{P}_{sub}$ from Example 5 on page 36. Because the minimal explanations are defined w.r.t. a given observation, let's consider a specific observation, e.g. *she goes to the library*,

$$\mathcal{O}_1 = \{l\}.$$

There are only two minimal explanations for $\mathcal{O}_1$ given $\mathcal{P}_1$, i.e.

$$\begin{aligned} \mathcal{E}_1 &= \{e \leftarrow \top\}, \\ \mathcal{E}_2 &= \{t \leftarrow \top\}. \end{aligned}$$

In this case, instead of sequentially generating all the nine binary vectors representing each of the non-complementary candidate explanations as it is done by the network $\mathcal{N}_{\mathcal{A}_{\mathcal{P}_1}}$, we would like to have a network which would avoid the generation of those binary vectors representing the candidate explanations which are supersets of candidates known to be explanations. The candidate explanations $\mathcal{C}_1$ and $\mathcal{C}_3$ are equal to the minimal explanations $\mathcal{E}_1$ and $\mathcal{E}_2$, respectively. Because of this, we know that they are explanations for the observation $\mathcal{O}_1$ and the extended network should not generate the candidate explanations

$$\begin{aligned} \mathcal{C}_5 &= \{e \leftarrow \top, t \leftarrow \top\}, \\ \mathcal{C}_6 &= \{e \leftarrow \top, t \leftarrow \bot\}, \\ \mathcal{C}_7 &= \{e \leftarrow \bot, t \leftarrow \top\}, \end{aligned}$$

since $\mathcal{C}_1 \subseteq \mathcal{C}_5$, $\mathcal{C}_1 \subseteq \mathcal{C}_6$, $\mathcal{C}_3 \subseteq \mathcal{C}_5$, and $\mathcal{E}_3 \subseteq \mathcal{C}_7$.

This means that after generating the candidate explanation

$$\mathcal{C}_4 = \{t \leftarrow \bot\},$$

the network is expected to avoid the generation of candidates which are a superset of already known explanations, which in this case are $\mathcal{C}_1$ and $\mathcal{C}_2$, and directly jump to the generation of the last candidate explanation

$$\mathcal{C}_8 = \{e \leftarrow \bot, t \leftarrow \bot\}.$$

Moreover, the network should then return to its initial state. Given this, the extended network is then expected to generate the following sequence of 4-bit binary vectors: 0000, 1000, 0100, 0010, 0001 and 0101.

Now, in order to obtain the network $\mathcal{N}'_{\mathcal{A}_{\mathcal{P}_1}}$ which will be an extension of $\mathcal{N}_{\mathcal{A}_{\mathcal{P}_1}}$ such that the generation of candidate explanations is no longer restricted only to non-complementary, but also to the minimal ones, we will apply the following three steps:

1. Access the information of whether the last candidate explanation generated has indeed explained the given observation and keep it during the upcoming time steps until all the candidate explanations have been generated;

2. For each candidate explanation, store the information of whether any other candidate explanation which is a subset of it has been in fact an explanation for the given observation;

3. Use the information from step 2 to avoid the generation of non-minimal candidate explanations.

The first modification can be done by simply adding a new input unit e (*explanation*), which will be externally activated when the last possible explanation generated was in fact an explanation for the given observation. This means that unit e will be active when that is the case, and passive, otherwise. Unit e also has a recurrent connection which is negatively modified by unit *next*. With this we make sure that, once unit e has been activated, it will remain active also during the following time steps where unit *next* is passive and, on the next time step where unit *next* is active again, e just remains active if it is externally activated.

Regarding the second step, the first candidate explanation to be generated

$$\mathcal{C}_0 = \emptyset$$

is a subset of all the other candidates. Because of this, we will add a connection from unit q_0 to unit *done* which will be positively activated by units *next* and e. This means that when the candidate $\mathcal{C}_0$ is an explanation for the given observation $\mathcal{O}$, no

further candidate explanations will be generated and the network is done. Besides this, we also make the connection between q_0 and q_1 being positively modified by unit *next* and negatively modified by unit e. Meaning that, for the case where $\mathcal{C}_0$ is not an explanation for the given observation $\mathcal{O}$, the network will proceed as before and unit q_1 will be activated.

For the candidate explanations with cardinality equal or greater than two, the second step will be done by adding a corresponding unit h_i for each of this candidates. Each of these units will be passive when its respective possible explanation should be generated and active, otherwise. This will be done by connecting each unit q_j responsible for generating a subset of a given set to its respective unit h_i. By doing this, a given unit h_i will be active as soon as any of its subsets is known to be an explanation.

Another important point to be considered in the second step is that, once these units h_i have been activated by one of units q_i, they need to remain active in the further time steps. This is important because the candidate explanations which are supersets of a given candidate explanation may not be generated directly after them, but in a few time steps later. Thus, this information should also be available in further time steps.

Besides this, since we're creating new units representing only the candidate explanations which have cardinality greater than one, this candidates have at least two non-empty subsets, but only one of them being an actual explanation is sufficient to keep this candidate explanation from being generated. Thus, once one of the subsets of the given candidate is an explanation, unit should be active and remain active independent of the other subsets being indeed explanations or not.

In practice, this can be done by simply adding a recurrent connection to each of these units. Each of this recurrent connections is negatively modified by unit *done* such that they all become passive as soon as the last possible explanation has been generated.

In our example, four units h_5, h_6, h_7 and h_8 will be added for each of its respective candidate explanation

$$\begin{aligned}
\mathcal{C}_5 &= \{e \leftarrow \top, t \leftarrow \top\},\\
\mathcal{C}_6 &= \{e \leftarrow \top, t \leftarrow \bot\},\\
\mathcal{C}_7 &= \{e \leftarrow \bot, t \leftarrow \top\},\\
\mathcal{C}_8 &= \{e \leftarrow \bot, t \leftarrow \bot\}.
\end{aligned}$$

Because units q_1 and q_3 are responsible for, respectively, generating the two candidate explanations

$$\mathcal{C}_1 = \{e \leftarrow \top\},$$
$$\mathcal{C}_3 = \{t \leftarrow \top\},$$

there will be a connection from units q_1 and q_3 to unit h_5, since both $\mathcal{C}_1$ and $\mathcal{C}_3$ are subsets of $\mathcal{C}_5$ which is represented by unit h_5. For analogous reasons, there will be connections from units q_1 and q_4 to unit h_6, from units q_2 and q_3 to unit h_7, and from units q_2 and q_4 to unit h_8. Besides this, the four units h_5, h_6, h_7 and h_8 will have a self connection negatively modified by unit *done*.

Finally, the third step regarding the fact that the network should be able to jump over the non-minimal possible explanations will be done in the following way. Each unit q_i connected to a unit q_{i+1} responsible for activating the generation of a possible explanation of cardinality greater than one will no longer be connected only to the next unit q_{i+1}, but to unit *done* and to all units q_j such that j = 0 or j > i. Besides this, we will make use of the h_i units to positively or negatively modify each of these connections depending on the case. A connection from a unit q_i to a unit q_j will be positively modified by all units h_k such that k < j, and negatively modified by unit h_j. Algorithm 1 shows the general steps for this modification, where PositiveConnection(a,b,c) and NegativeConnection(a,b,c) are used to express when a connection from unit a to unit b is positively and negatively modified by c, respectively. In our case, for example, unit q_4 will no longer be connected only to unit q_5, but to units q_5, q_6 q_7, q_8, q_0 and *done*. Unit q_5 will now be connected to units q_6, q_7 q_8, q_0 and *done*. And so on, until unit q_7 which will be connected to units q_8, q_0 and *done*. In our example, the algorithm would use *min* equals to five and max equals to eight. To illustrate how the algorithm works, let's consider the connections from unit q_4 and see how they would be modified by units h_i.

First of all, the connection from unit q_4 to unit q_5 will be negatively activated by unit h_5. This means that unit h_5 is passive when none of the following two candidate explanations are actual explanations for the observation $\mathcal{O}$:

$$\begin{aligned} \mathcal{C}_1 &= \{e \leftarrow \top\}, \\ \mathcal{C}_3 &= \{t \leftarrow \top\}. \end{aligned}$$

In that case, unit q_5 is activated and the next candidate explanation to be generated is, as before,

$$\mathcal{C}_5 = \{e \leftarrow \top, t \leftarrow \top\}.$$

The connection from unit q_4 to unit q_6 is positively modified by unit h_5 and negatively modified by unit h_6. This means that unit h_5 is active when one of the candidate explanations $\mathcal{C}_1$ or $\mathcal{C}_2$ (or both) were actual explanations for the observation $\mathcal{O}$.

Algorithm 1. How to make the new connections between units q_i and from them to unit *done* be positively or negatively modified by units h_i.

Input: min = the smallest i from the h_i units
max = the greatest i from the h_i units

Output: Positive and negative modifications by units h_i of the new connections between units q_i and to unit *done*

1: **Function** AddModifiers(min, max) **do**
2: **for** $i \in \{min - 1, \ldots, max - 1\}$ **do**
3: **for** $j \in \{i + 1, \ldots, max\}$ **do**
4: **for** $k \in \{i + 1, \ldots, j - 1\}$ **do**
5: PositiveConnection(q_i, q_j,h_k)
6: **end for**
7: NegativeConnection(q_i, q_j,h_j)
8: **end for**
9: **for** $j \in \{min, \ldots, max\}$ **do**
10: PositiveConnection(q_i, q_0,h_j) and PositiveConnection(q_i, d,h_j)
11: **end for**
12: **end for**
13: **EndFunction**

Consequently, the next candidate explanation should no longer be $\mathcal{C}_5$. If that is case, since the connection from unit q_4 to unit q_5 is negatively modified by unit h_5, unit q_5 will indeed not be activated. Besides this, unit q_6 is activated when none of the following two candidate explanations are actual explanations for the observation $\mathcal{O}$:

$$\begin{aligned} \mathcal{C}_1 &= \{e \leftarrow \top\}, \\ \mathcal{C}_4 &= \{t \leftarrow \bot\}. \end{aligned}$$

If that is the case, the next candidate explanation to be generated is now

$$\mathcal{C}_6 = \{e \leftarrow \top, t \leftarrow \bot\}.$$

And so on, until the connections from unit q_4 to units q_0 and *done*, which will be positively modified by units h_5, h_6, h_7 and h_8, meaning that when none of the remaining candidate explanations should be generated, i.e. all the four h_i units are active, units *done* and q_0 will be activated. Moreover, the activation of unit *done* will deactivate units h_i and the network is back to its initial state. The connections from unit q_6 to units q_7, q_8, q_0 and *done*, from unit q_7 to units q_8, q_0 and *done*, and from unit q_8 to units q_0 and *done* will be positively or negatively modified by units h_i following the same structure as explained above.

The connection from unit q_4 to unit q_6 is positively modified by unit h_5 and negatively modified by unit h_6. This means that unit h_5 is active when one of the candidate explanations $\mathcal{C}_1$ or $\mathcal{C}_2$ (or both) were actual explanations for the observation $\mathcal{O}$. Consequently, the next candidate explanation should no longer be $\mathcal{C}_5$. If that is case, since the connection from unit q_4 to unit q_5 is negatively modified by unit h_5, unit q_5 will indeed not be activated. Besides this, unit q_6 is activated when none of the following two candidate explanations are actual explanations for the observation $\mathcal{O}$:

$$\begin{aligned} \mathcal{C}_1 &= \{e \leftarrow \top\}, \\ \mathcal{C}_4 &= \{t \leftarrow \bot\}. \end{aligned}$$

If that is the case, the next candidate explanation to be generated is now

$$\mathcal{C}_6 = \{e \leftarrow \top, t \leftarrow \bot\}.$$

And so on, until the connections from unit q_4 to units q_0 and *done*, which will be positively modified by units h_5, h_6, h_7 and h_8, meaning that when none of the remaining candidate explanations should be generated, i.e. all the four h_i units are active, units *done* and q_0 will be activated. Moreover, the activation of unit *done* will deactivate units h_i and the network is back to its initial state. The connections from unit q_6 to units q_7, q_8, q_0 and *done*, from unit q_7 to units q_8, q_0 and *done*, and from unit q_8 to units q_0 and *done* will be positively or negatively modified by units h_i following the same structure as explained above.

A fragment of the network $\mathcal{N}_{\mathcal{A}_{\mathcal{P}_1}}$ is shown in Figure 3.3 with all the necessary modifications to extend this network such that it will only generate the minimal candidate explanations. In order to get the final version of the extended network $\mathcal{N}'_{\mathcal{A}_{\mathcal{P}_1}}$, one needs to overlap the network $\mathcal{N}_{\mathcal{A}_{\mathcal{P}_1}}$ with the extensions shown in Figure 3.3.

So, after applying all the above described modifications to the network $\mathcal{N}_{\mathcal{A}_{\mathcal{P}_1}}$, we get the extended network $\mathcal{N}'_{\mathcal{A}_{\mathcal{P}_1}}$. The network $\mathcal{N}'_{\mathcal{A}_{\mathcal{P}_1}}$ has the same two input units *done* and *next* as before and an extra input unit e (*explanation*), which gives us the information whether the previous candidate was indeed an explanation. The hidden units now consist of units q_i as before and the extra units h_i, which will be used to positively or negatively modify certain connections of the network. Units h_i are the most important modification of the network and are the ones responsible for blocking non-minimal candidate explanations from being generated. The output units remain the same, i.e. five output units $e^\top$, $e^\bot$, $t^\top$, $t^\bot$ and *done*. The connections between units q_i and the output units $e^\top$, $e^\bot$, $t^\top$, $t^\bot$ also remain the same. The network $\mathcal{N}'_{\mathcal{A}_{\mathcal{P}_1}}$ generates all the minimal candidate explanations for the program $\mathcal{P}_1$ and a given observation $\mathcal{O}$.

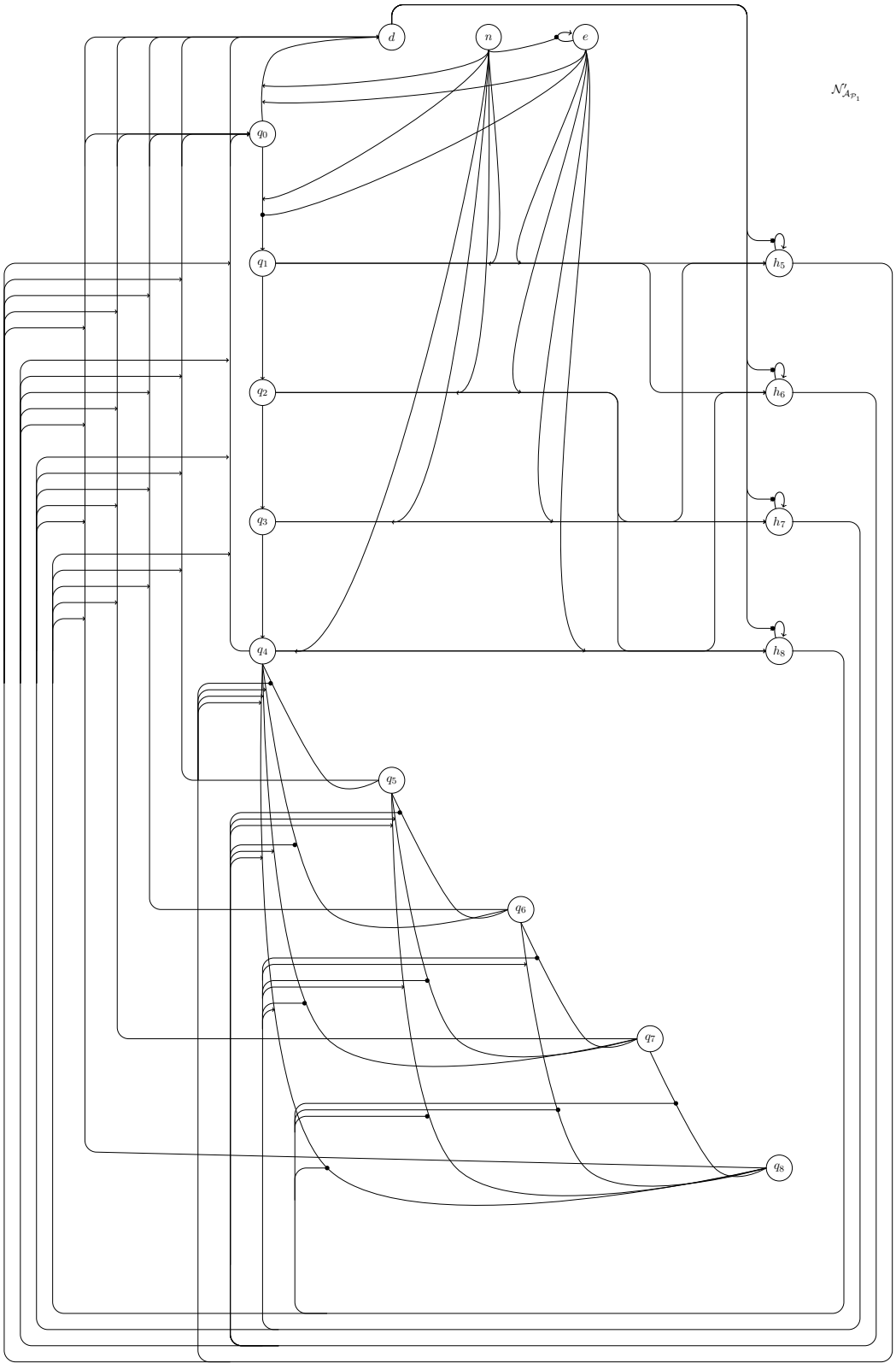


Figure 3.3: Fragment of the McCulloch-Pitts network from Figure 3.2 consisting of the necessary modifications to sequentially generate all the minimal candidate explanations for $\mathcal{A}_{\mathcal{P}_1}$.

Let's observe how the network $\mathcal{N}'_{\mathcal{A}_{\mathcal{P}_1}}$ behaves for the observation

$$\mathcal{O}_1 = \{l\}.$$

As before, unit q_0 will be initially active and all other units passive. As soon as unit *next* is externally activated, then unit q_0 will become passive and unit q_1 will become active. Moreover, unit q_1 will activate unit $e^\top$ representing the first minimal and non-empty candidate explanation

$$\mathcal{C}_1 = \{e \leftarrow \top\}.$$

Because $\mathcal{C}_1$ is an explanation for the observation $\mathcal{O}_1$, the next time unit *next* is activated, unit e will also be activated. Thus, units q_2, h_5 and h_6 will become active and unit q_1 will become passive. Moreover, unit q_2 will activate unit $e^\bot$ representing the candidate explanation

$$\mathcal{C}_2 = \{e \leftarrow \bot\}.$$

In the next activation of unit *next*, because $\mathcal{C}_2$ is not an explanation for the observation $\mathcal{O}_1$, unit e will be passive. Unit q_2 will become passive, unit q_3 will become active and units h_5 and h_6 will remain active. Moreover, unit q_3 will activate unit $t^\top$ representing the candidate explanation

$$\mathcal{C}_3 = \{t \leftarrow \top\}.$$

In the next activation of unit *next*, because $\mathcal{C}_3$ is also an explanation for the observation $\mathcal{O}_1$, unit e will also be activated. Unit q_3 will become passive, units q_4 and h_7 will become active, and units h_5 and h_6 will remain active. Moreover, unit q_4 will activate unit $t^\bot$ representing the candidate explanation

$$\mathcal{C}_4 = \{t \leftarrow \bot\}.$$

In the next time unit *next* is activated, because $\mathcal{E}_4$ is not an explanation for the observation $\mathcal{O}_1$, unit e will be passive. Unit q_4 will become passive and units h_5, h_6 and h_7 will remain active. Because units h_5, h_6 and h_7 are active, but unit h_8 is passive, unit q_4 will activate unit q_8. Unit q_8 will activate units $e^\bot$ and $t^\bot$ representing the candidate explanation

$$\mathcal{C}_8 = \{e \leftarrow \bot, t \leftarrow \bot\}.$$

Finally, the next time unit *next* is activated, because $\mathcal{C}_8$ is not an explanation for the observation $\mathcal{O}_1$, unit e will be passive. Moreover, unit q_8 will activate units q_0 and *done*, and unit q_8 will become passive. Because unit *done* is active, all units h_i which were active, e.g. h_5, h_6 and h_7, will become passive, returning the network to its

Input		Hidden													Output				
t	e	q_0	q_1	q_2	q_3	q_4	q_5	q_6	q_7	q_8	h_5	h_6	h_7	h_8	$e^\top$	$e^\bot$	$t^\top$	$t^\bot$	d
0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
1	1	0	0	1	0	0	0	0	0	0	1	1	0	0	0	1	0	0	0
1	0	0	0	0	1	0	0	0	0	0	1	1	0	0	0	0	1	0	0
1	1	0	0	0	0	1	0	0	0	0	1	1	1	0	0	0	0	1	0
1	0	0	0	0	0	0	0	0	0	1	1	1	1	0	0	1	0	1	1

Table 3.2: State of the input, hidden and output units of the network $\mathcal{N}'_{\mathcal{A}_{\mathcal{P}_1}}$ for the observation $\mathcal{O}_1 = \{l\}$ on its initial state and on the activation of the input unit *next* for five consecutive time steps.

initial state and informing the environment that all possible explanations have been generated. This steps are shown in Table 3.2.

With this we have shown in details one possible way of extending the network $\mathcal{N}_{\mathcal{A}_{\mathcal{P}_1}}$ to a network $\mathcal{N}'_{\mathcal{A}_{\mathcal{P}_1}}$ which will only compute the minimal candidate explanations for the given program $\mathcal{P}_1$ and any given observation $\mathcal{O}$. We have also explained how the network $\mathcal{N}'_{\mathcal{A}_{\mathcal{P}_1}}$ behaves for the observation

$$\mathcal{O}_1 = \{l\}.$$

Although we have focused in the previous example, one should remember that the network here presented is a general approach for any given program $\mathcal{P}$ which has a respective set of abducibles of size four. Figure 3.4 shows the different outcomes the network $\mathcal{N}'_{\mathcal{A}_{\mathcal{P}}}$ can provide depending on the given program $\mathcal{P}$ and observation $\mathcal{O}$. We use dashed arrows to denote that the previous candidate explanations did not explain the given observation and solid arrows to denote the opposite. For example, in the example where we consider the program $\mathcal{P}_1$ and the observation $\mathcal{O}_1$, the network output corresponds to the following path:

$$0000 \dashrightarrow 1000 \rightarrow 0100 \dashrightarrow 0010 \rightarrow 0001 \dashrightarrow 0101 \dashrightarrow done.$$

Example 6 on page 51 shows another example of a program with a corresponding set of abducibles of size four and how the same network could be reused. Moreover, a network for a given program $\mathcal{P}$ which has a correspondent set of abducibles of arbitrary size can be constructed following the same structure. The details on how this can be done is explained earlier in this section where we present Algorithm 1 on page 45.

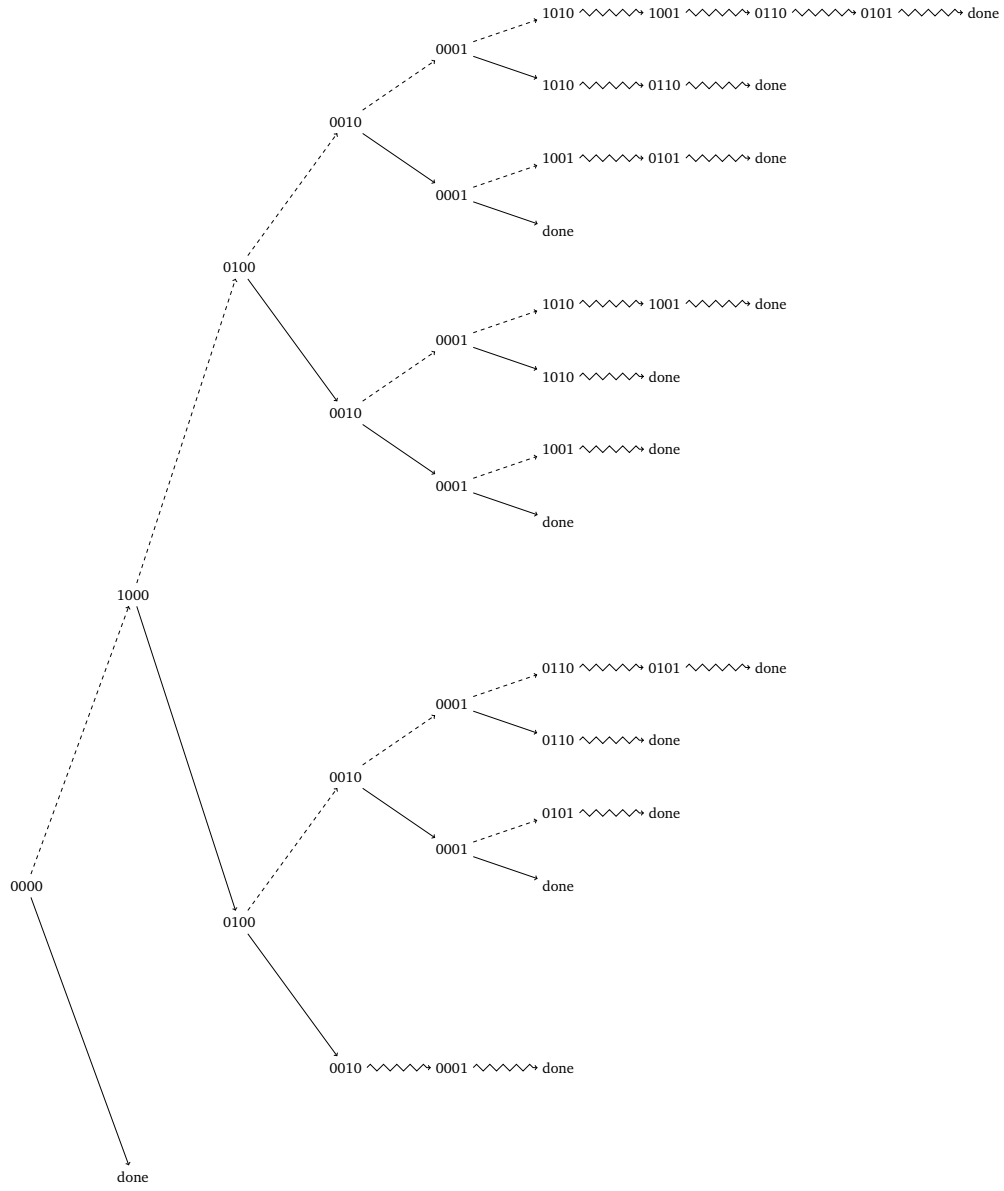


Figure 3.4: All the possible sequences of minimal candidate explanations for potential programs having a respective set of abducibles of size four taking into account potential arbitrary observations. Solid and dashed arrows denote positive and candidates, respectively. Wavy arrows denote the cases where both are possible.

Example 6. Consider program $\mathcal{P}$ consisting of the following clauses:

$$\begin{aligned} a &\leftarrow b \\ a &\leftarrow \neg b \\ a &\leftarrow c \end{aligned}$$

The corresponding set of abducibles $\mathcal{A}_{\mathcal{P}_2}$ consists of the following four facts and assumptions:

$$\begin{aligned} b &\leftarrow \top \qquad & b &\leftarrow \bot \\ c &\leftarrow \top \qquad & c &\leftarrow \bot \end{aligned}$$

It is easy to see that the network would look the same, with the only difference that units before representing $e^{\top}$, $e^{\bot}$, $t^{\top}$ and $t^{\bot}$ now represent $b^{\top}$, $b^{\bot}$, $c^{\top}$ and $c^{\bot}$, respectively. If we now consider the observation

$$\mathcal{O}_2 = \{a\},$$

as the minimal explanations for $\mathcal{O}_2$ given $\mathcal{P}_2$ are the following

$$\begin{aligned} \mathcal{E}_1 &= \{b\}, \\ \mathcal{E}_2 &= \{\neg b\}, \\ \mathcal{E}_3 &= \{c\}, \end{aligned}$$

one can see that, in this case, the output of the network would correspond to the following path from the tree shown in Figure 3.4:

$$0000 \dashrightarrow 1000 \rightarrow 0100 \rightarrow 0010 \rightarrow 0001 \dashrightarrow done.$$

3.3 Conclusion

In this chapter we have focused on how to generate possible abductive explanations for a given program $\mathcal{P}$ and observation $\mathcal{O}$ using connectionist networks. First, we have looked into an approach to generate non-complementary explanations introduced in [39]. Then, we proposed a way to improve this approach by applying some modifications to it such that only minimal explanations are generated. Our new approach produces all possible explanations in a specific order such that minimal explanations can be detected and all non-minimal possible explanations can be discarded. This was a relevant improvement, since detecting minimality is not a trivial task, but it is still not the ideal solution to our problem for the following reasons.

First, although a formal specification of how to construct and extend this network is provided, the construction of different networks may be required for different sets of abducibles. This is certainly not the most convenient solution to our problem and therefore we would like to come up with a more general approach.

Besides this, we do not believe that humans test all possible explanations in a systematic way. It appears to us that in particular reasoning episodes some possible explanations are systematically tested whereas others are not considered at all. Our approach to generate the minimal candidate explanations already partially solves this problem in the sense that many unnecessary candidate explanations are discarded and, even for the same set of abducibles, the possible explanations vary depending on the given observation. However, the process of generating this possible explanations is still somehow done in a systematic way.

Given that, we would like to have a more general approach not only in the sense that it should be flexible and work for different sets of abducibles, but also by generating as possible explanations only those which are more plausible to actually explain our observations.

4 Learning Arbitrary Sequences of Candidate Explanations

In Section 3 we have presented an approach for generating candidate explanations in a fixed given order. However, as we cannot assume that humans generate candidates in a static sequence, having an approach which is more abstract and can learn arbitrary sequences would be the ideal scenario. Therefore, in this chapter we will focus on recurrent networks, more specifically in Jordan and Elman networks. We will show how these networks can be used to archive the same results presented for the previous approach, but also how they can easily be generalised for arbitrary sequences.

4.1 Non-Complementary Candidate Explanations

In Section 3 we have shown by means of an example how a predefined sequence of non-complementary candidate explanations can be generated based on connectionist networks. Now we would like to start this section by showing that the same result can also be archived based on recurrent networks. In order to do so, we will begin by constructing, training and testing a recurrent network for the same example used before. Thus, reconsider the program $\mathcal{P}_{sup}$ presented in Example 5 on page 36.

In the first version of the connectionist network presented before, each of the nine candidate explanations is generated one after the other in a fixed order. After generating the last one, unit *done* is activated and the network returns to its initial state. Every time a new candidate explanation is generated, the state of the network remains the same until a next candidate is requested by the activation of unit *next*. This means that the expected output is the previous candidate explanation when unit *next* is passive and the next one, otherwise.

One possible way to translate this problem into a temporal domain problem is by constructing a sequence of binary vectors consisting of five bits. The first four bits (from left to right) will correspond to the four output units of the connectionist network, representing $e^{\top}$, $e^{\bot}$, $t^{\top}$ and $t^{\bot}$. The last bit (from left to right) corresponds to unit *done* of the connectionist network, representing whether all the non-complementary

candidate explanations have been generated. Each of these bits will be equal to *one* when its respective unit is active and equal to *zero* when it is passive.

Given this, there are different sequences we could obtain depending on the oscillation of the values of the request unit *next*. As explained in Chapter 3, we are just working with one component of a network which generated the sceptical consequences of a given program $\mathcal{P}$ and observation $\mathcal{O}$. Therefore, the activation of unit *next* will come from the other components of this overall network which is responsible for checking weather a given candidate explanation $\mathcal{C}$ explains $\mathcal{O}$ given $\mathcal{P}$.

If we consider the scenario where the network only needs one time step to decide whether the candidate was an explanation, then next candidates would be requested one after another without a break. In that case we have the most basic sequence, which is the request sequence where unit *next* is active for nine time steps in a row. For this sequence of values for the request unit, the output sequence expected is the following:

$$00000, 10000, 01000, 00100, 00010, 10100, 10010, 01100, 01011.$$

However, this is not always the case since we don't know how many time steps the external network will need before requesting a next candidate explanation. Moreover, this steps are not fixed, meaning that for different candidates it might take a different number of time steps. Therefore, we can also obtain any sequence which has the following form:

$$\begin{array}{ccccc} (00000)^+, & (10000)^+, & (01000)^+, & (00100)^+, & (00010)^+, \\ & (10100)^+, & (10010)^+, & (01100)^+, & (01011)^+. \end{array}$$

The main task here is to predict, at each point in time, the next binary vector in the sequence which represents the next candidate explanation. The value of this vector will depend on two factors: the last candidate explanation generated and the value of unit *next*. Thus, our input will also consist of a 5-bit binary vector where the first four bits (from left to right) also correspond to $e^\top$, $e^\bot$, $t^\top$ and $t^\bot$, but the last bit (from left to right) corresponds to unit *next* of the connectionist network, representing the request of a new candidate explanation. This bit will be equal to *one* when a new candidate explanation should be generated and *zero*, otherwise. All possible inputs and its respective expected outputs can be seen in Table 4.1.

Now we would like to define a recurrent network corresponding to the behaviour described above. Since we are dealing with a temporal domain problem, the most suitable type of network in our case would be a recurrent network. As discussed

Input	Output
00000	00000
10000	10000
01000	01000
00100	00100
00010	00010
10100	10100
10010	10010
01100	01100
01010	01011

(a) Cases where the fifth bit in the input is *zero*.

Input	Output
00001	10000
10001	01000
01001	00100
00101	00010
00011	10100
10101	10010
10011	01100
01101	01011
01011	00000

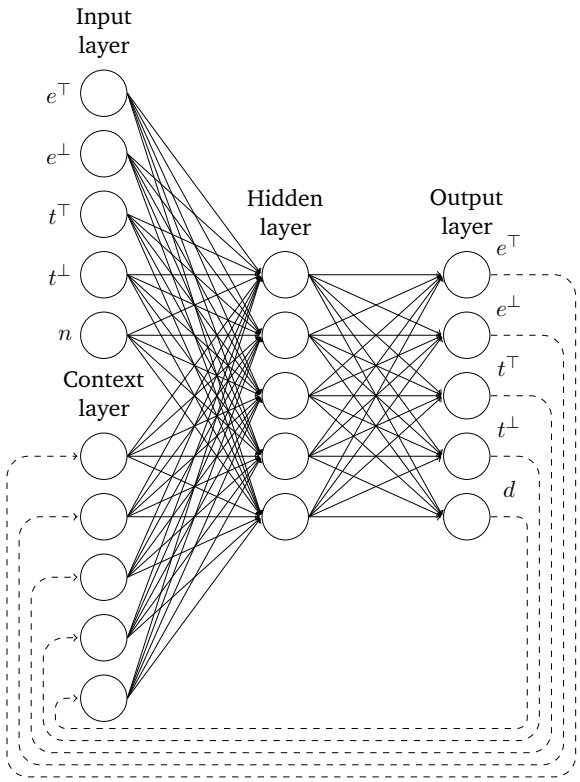
(b) Cases where the fifth bit in the input is *one*.

Table 4.1: Each possible input and its respective expected output represented as 5-bit binary vectors for the generation of non-complementary candidate explanations for the given program $\mathcal{P}_{sup}$.

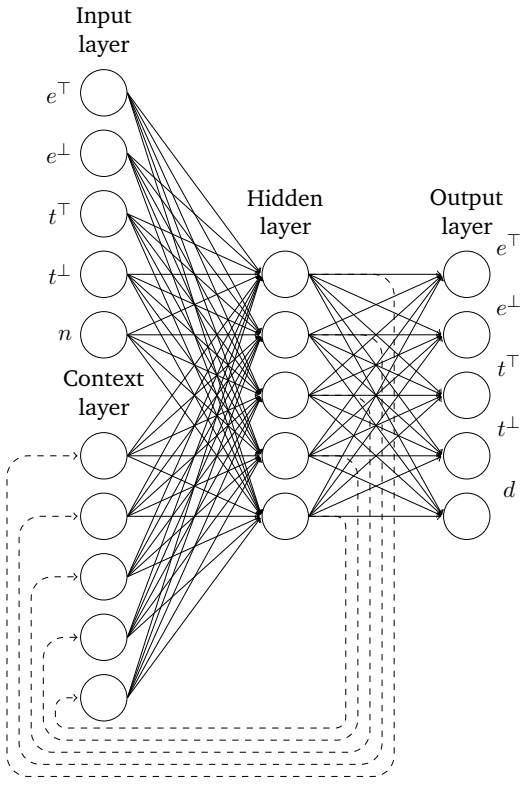
in Section 2, there are many different types of recurrent networks but here we will focus only on two of them, which are the Jordan and the Elman networks. The Elman network is an variation of the Jordan network and the only difference between them is in the source of the connections to the context layer. In an Elman network, the connections to the context units come from the hidden units instead of the output units as it is done in a Jordan network. Because of this, it is clear that the input and output specifications described above can be equally applied for both types of networks.

When it comes to the structure of the network, the 5-bit binary vector input described above can be mapped into the input layer of the network where each bit corresponds to a unit. Thus, the input layer consists of five units. The same applies to the output layer which will then also consist of five units. As mentioned before, these two layers are exactly the same for both the Elman and the Jordan network. On the other hand, the context layer has different dependencies in each case. For the Jordan network, the context layer is a copy of the output layer and, thus, also has five units. But in the Elman network, the context layer is a copy of the hidden layer. If we consider a hidden layer of size five, then the context layer of the Elman network would also consist of five units. The structure of the both networks following this specifications is show in Figure 4.1.

The implementation of the Jordan networks and the Elman networks which we are going to use when performing the experiments later on can be found in [28]. Those network implementations are part of a library consisting of some standard networks written in Python. It follows the exact structure shown in Figure 4.1, where all the trainable weights are randomly initialised and learned though the training phase



(a) Jordan Network.



(b) Elman Network.

Figure 4.1: Structure of the Jordan (a) and Elman (b) networks for the generation of all non-complementary candidate explanations for the program $\mathcal{P}_{sup}$. Note that the number of hidden units may vary, and, consequently, the context layer in the Elman netwoks may vary as well. Solid lines are trainable.

where back propagation is applied. More details on this process was introduces in Chapter 2.

For sake of simplicity, we have considered a hidden layer of size five in the example shown in Figure 4.1. But defining the number of units in the hidden layer is not an easy task. It is well known that if the hidden layer has too few units, the network can present high training error and high generalisation error due to underfitting. On the other hand, if the network has too many hidden units, it may present low training error but still have high generalisation error due to overfitting. Thus, the only way to determine the best number of hidden units is by training several networks with different sizes for the hidden layer and estimate the generalisation error for each of them.

Our next step is then to set up an experiment which consists of creating these several networks with different number of hidden units, training and testing each of them by means of a k-fold cross-validation. In k-fold cross-validation, the original sample is randomly partitioned into k equal sized subsamples. From the k subsamples, a single subsample is retained as the validation data for testing the model, and the remaining $k-1$ subsamples are used as training data. The cross-validation process is then repeated k times, with each of the k subsamples used exactly once as the validation data. The k results from the folds are then averaged to produce a single estimation. In our experiments we will use k-fold cross-validation with $k = 10$.

The validation measure selected was the *mean absolute error* (*MAE*), which is computed for each configuration of the networks. The mean absolute error is given by

$$\begin{aligned} \text{MAE} &= \frac{1}{n}\sum_{i=1}^{n}|y_i - x_i| \\ &= \frac{1}{n}\sum_{i=1}^{n}|e_i|, \end{aligned}$$

where y_i is the i^{th} predicted value, x_i is the i^{th} expected value, and n is the number of predictions. In this case, e_i denotes then the distance between y_i and x_i.

The process of cross-validation is repeated ten times for each network configuration and the final result is given by the mean absolute error averaged over these ten trials. Based on this we will have the adequate grounds to define the optimal number of hidden units for both types of networks considered here. This is done for the both types of networks, i.e. Jordan and Elman networks, such that we can also compare their performance. Algorithm 2 on page 58 shows the overall structure of this experiment.

Algorithm 2. Overall experiment to find optimal number of hidden units for Jordan and Elman networks.

Input: trials = number of times to test the networks
folds = number of slices the data will be slipted
passes = number of passes through the training data
min_units; max_units = interval for variation in the hidden units
candidates = sequence of candidate explanations to be learned

Output: mean absolute error for each different number of hidden units

```
1:  Function Experiment(trials, folds, passes) do
2:    request_data = GenerateRequestData
3:    cross_validation = CrossValidation(request_data, folds)
4:    test_data = cross_validation.test_data
5:    train_data = cross_validation.train_data
6:    for t ∈ {0, ..., trials} do
7:      for h ∈ {min_units, ..., max_units} do
8:        for f ∈ {0, ..., folds} do
9:          for p ∈ {0, ..., passes} do
10:           train(jordan, candidates, train_data[j])
11:           train(elman, candidates, train_data[j])
12:         end for
13:         test(jordan, candidates, test_data[j])
14:         test(elman, candidates, test_data[j])
15:       end for
16:     end for
17:   end for
18: EndFunction
```

The process of generating the request data, as well as the training and testing phase, are explained in more details later in this section. Relating to our example, in order to learn the sequence described previously, we use Algorithm 2 with the following set up of the parameters:

$$\begin{aligned} trials &= 10, \\ folds &= 10, \\ passes &= 12, \\ min\_units &= 1, \\ max\_units &= 10, \\ candidates &= \{00000, 10000, 01000, 00100, 00010, 10100, 10010, 01100, 01011\}. \end{aligned}$$

In practice, the information of the previous explanation for the input layer will come from the output layer. Thus, the input unit *next* will be the only one which is externally activated. Because of this, the training and testing data must consist only of possible values for unit *next*. In order to obtain this data, we generate several binary sequences of different length s such that among the s bits exactly nine are equal to one. This means that, in every sequence considered, each of the nine candidate explanations will be requested within arbitrary time intervals for different bounds s. For instance, if s is equal to nine (which is the minimal possible value in this case), the only possible sequence of values for unit *next* is 111111111. While for s equals to ten, each of the following sequence of values could be considered for unit *next*:

$$\begin{aligned} &0111111111, 1011111111, 1101111111, 1110111111, 1111011111, \\ &1111101111, 1111110111, 1111111011, 1111111101, 1111111110. \end{aligned}$$

Algorithm 3 on page 60 shows how the request data of n candidate explanations is generated for given minimum and maximum bounds. For the example we are considering (i.e. set of abducibles of size four), there are nine non-complementary candidate explanations to be requested. Thus, the request data is generated as explained above with $min = 9$, $max = 90$, $m = 10$ and $n = 9$. We vary the request bound s between nine and ninety. Ten different and arbitrary sequences of length s are generated for each value of s, giving us the following total different sequences of arbitrary size:

$$(max - min) * m = 810.$$

All the sequences are then shuffled and divided into training and testing sets by means of a 10-fold cross-validation.

Algorithm 3. A list containing the request data of n candidate explanations.

Input: min; max = interval of request bound
m = number of sequences with the same bound
n = number of candidate explanations

Output: A list containing the request data of n candidate explanations. Each sequence has a bound varying from min to max, and each bound is repeated m times.

```
1:  Function GenerateRequestData(min, max, m, n) do
2:      request_data = []
3:      for i ∈ {min, ..., max} do
4:          for j ∈ {0, ..., m} do
5:              sequence = {s_0, ..., s_n, s_(n+1), ..., s_i}
6:              s_0, ..., s_n = 1; s_(n+1), ..., s_i = 0
7:              shuffle sequence
8:              append sequence to request_data
9:          end for
10:     end for
11:     shuffle request_data
12:     return request_data
13: EndFunction
```

The network is trained such that every time the input unit *next* is active, a new candidate explanation will be generated, and, when the input unit *next* is passive, the last candidate explanation will be generated again. With this we assure that the output units will remain the same while the input unit *next* is passive. Thus, the training phase consists of externally making the input bit *next* active or passive and comparing the output with the expected output, which will be the next or the last candidate explanation depending on the current value of the input bit *next*. The details of the training process for a given network and given set of abducibles $\mathcal{A}_\mathcal{P}$ is shown in Algorithm 4 on page 62.

The training phase consisted of presenting each 5-bit input vector, one at a time, in order. The task on each input cycle was to predict the next 5-bit vector corresponding to the next candidate explanation. The sequence was wrapped around, such that the first sequence was presented after the last one. So at the end of the 810 sequence, the process began again, without a break, starting with the first value of the sequence. The training continued in this way until the network had experienced twelve complete passes through the sequence.

The network was then tested on another sequence that obeyed the same regularities, but was created from a different initial randomisation. For the testing phase, the validation data was also used in the same way as the training data. With the only difference that now instead of back propagating the error, we just compute the mean absolute error between the expected output for the given input and the actual output given by the networks. Algorithm 5 on page 62 shows the testing phase in more details.

Given this, we performed the experiment described above for both the Jordan and Elman networks with the number of hidden units varying from one to ten and compute the corresponding prediction error. Training was done through back-propagation with the following parameters:

$$\begin{aligned} \textit{learning rate} &= 0.001, \\ \textit{momentum} &= 0.01. \end{aligned}$$

The approximate training time of one network over one epoch is 15 seconds. As we train the network over 12 epochs, we need approximately $15 * 12 = 180$ seconds for the whole training phase. The testing phase only takes approximately one second. As we do a 10-fold cross-validation, this process is repeated 10 times and the error is averaged over the 10 folds. Therefore, to obtain one error value we need $180 * 10 = 1800$ seconds, i.e. 30 minutes.

Algorithm 4. How to train a given network for a given sequence of candidate explanations and request data.

Input: network = network to be trained
candidates = sequence of candidate explanations to be learned
request_data = sequence of values for the request unit

Output: The given network is trained to generate the candidate explanations in the given order.

```
1: Function train(network, candidates, request_data) do
2:   nextId = 0
3:   for request in request_data do
4:     network.propagate_forward(request + candidates[nextId][0:4])
5:     if request == 1 then
6:       nextId = (nextId+1)%length(candidates)
7:     end if
8:     network.propagate_backward(candidates[nextId])
9:   end for
10: EndFunction
```

Algorithm 5. How to test a given network for a given sequence of candidate explanations and request data.

Input: network = network to be trained
candidates = sequence of candidate explanations to be tested
request_data = sequence of values for the request unit

Output: The given network is trained to generate the candidate explanations in the given order.

```
1: Function test(network, candidates, request_data) do
2:   expected = []; output = []; expectedId = 0; last_output = (0,0,0,0,0);
3:   for request in request_data do
4:     last_out = network.propagate_forward(request + last_out[0:4])
5:     if request == 1 then
6:       expectedId = (expectedId+1)%length(candidates)
7:       expected.add(candidates[expectedId])
8:       output.add(last_output)
9:     end if
10:  end for
11:  return MAE(output, expected)
12: EndFunction
```

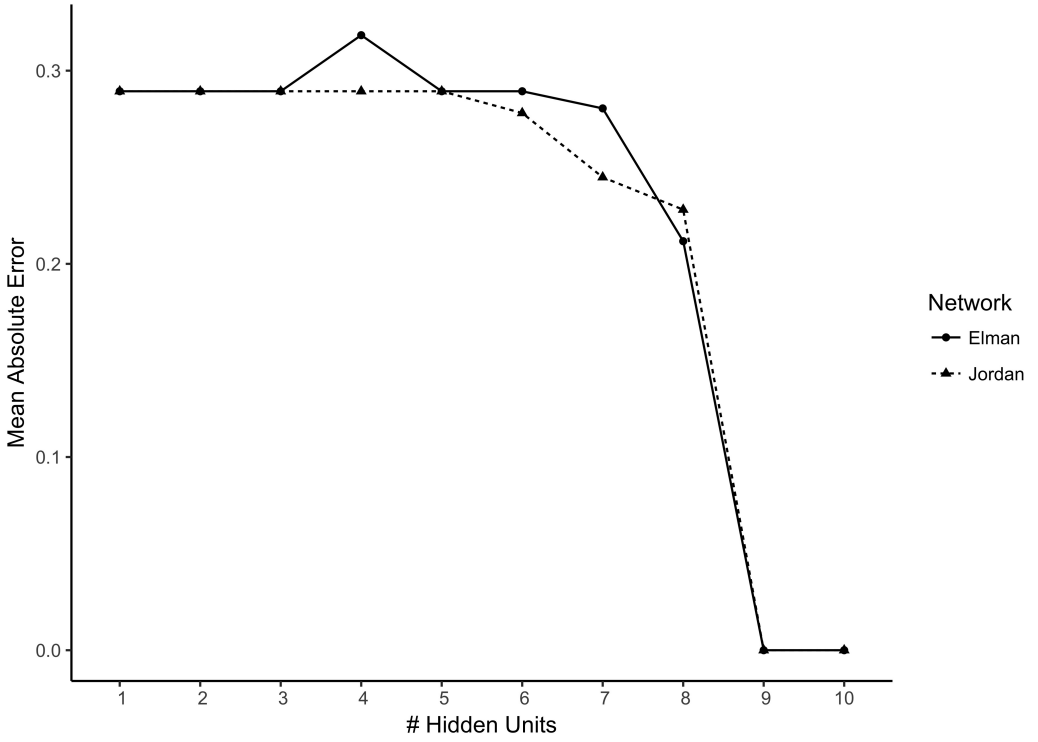


Figure 4.2: Results of an experiment to define the appropriated number of hidden units in a recurrent network to generate non-complementary candidates explanations for the program $\mathcal{P}_{sup}$. The size of the hidden units vary from one to ten. The graph plots the mean absolute error (MAE) for a 10-fold cross-validation averaged over ten trials for each network configuration.

As this process is done for two different types of network with each of them varying in 10 different configurations (number of hidden units), training, testing and computing the error for each network configuration takes $30 * 2 * 10 = 600$ minutes, i.e. 10 hours. Moreover, the final error is the averaged over 10 trials which means that the whole experiment takes $10 * 10 = 100$ hours to run.

The result of this experiment is shown in Figure 4.2. The graph plots the mean absolute error (MAE) for a 10-fold cross-validation of a data constructed in the way previously described averaged over ten trials. As one can observe from this graph, the behaviour of the Jordan and the Elman networks are quite similar. Besides this, it is also clear that the prediction error drops significantly for the networks with a hidden layer of size greater or equal than nine.

With this we can conclude that both a Jordan and an Elman network constructed in the way described above have one hundred percent of accuracy for the task of generating all the possible non-complementary candidate explanations for the program $\mathcal{P}_{sup}$ and its respective set of abducibles $\mathcal{A}_{\mathcal{P}_{sup}}$. Although we have focused on showing the results for the set of abducibles $\mathcal{A}_{\mathcal{P}_{sup}}$, it is easy to observe that the experiment can be repeated in the exact same way for any set of abducible which

has cardinality four. The only difference is on which abducible each of the four bits represent which in our case was $e^{\top}$, $e^{\top}$, $e^{\top}$, and $e^{\top}$, but the representation can easily be substituted by any other set of abducibles without affecting the procedure.

Moreover, as it is shown in the algorithms presented earlier, the process is specified in an abstract level such that it should work for a given set of abducibles of any size. Therefore, we would like to abstract from the previous example and show how one could follow the same steps for any other program with corresponding set of abducibles of different sizes. For this purpose, consider an arbitrary program $\mathcal{P}$ and its corresponding set of abducibles $\mathcal{A}_{\mathcal{P}}$, where $|\mathcal{A}_{\mathcal{P}}| = \text{N}$. Both the input and output layers would consist of N+1 units. As explained previously, the ideal number of hidden units can vary from case to case and one way to define this if by reproducing the same experiment developed above. Both the Jordan and Elman networks can be structured in the same way, with the difference on the recurrent connections only.

To show that this is indeed the case we will reproduce the experiment and analyse its output for other values of N. In order to do so, we will show example of programs which have a respective size of abducibles of size N. However, note that, just as we have explained earlier, the programs are only for illustrative purpose but the results shown here can be generalised for any set of abducibles of corresponding cardinality N. Consider the variations of program $\mathcal{P}_{sup}$ shown in Example 7 on page 65.

A network that sequentially generates the candidate explanations in $\mathcal{C}_{\mathcal{A}_{\mathcal{P}_{sup'}}}$ consists of three input units and three output units. In order to define the adequate number of units in the hidden layer, we reproduce the experiment for Jordan and Elman networks now following the configurations described here. As it is shown in Figure 4.3a, again the Jordan and the Elman networks have presented a similar behaviour and a hidden layer consisting of four units seems to be the most adequate for both types of networks. For the candidate explanations $\mathcal{C}_{\mathcal{A}_{\mathcal{P}_{sup''}}}$, the network consists of seven input units and seven output units. Regarding the hidden units, as it is shown in Figure 4.3b, the Jordan and Elman networks need thirteen and fourteen hidden units, respectively. This last case is also a strong indication on the generalisation capacity of such networks, since there are twenty-seven candidate explanations and less than the half of this number is needed in hidden units.

With this we can conclude that creating Jordan and Elman networks for the generation on non-complementary candidate explanations can be done for sets of abducibles of different sizes by following the steps defined in this section. Moreover, the experiment to define which of the two networks is more suitable and which is the most adequate number of hidden units can also easily be reproduced for the different sets of abducibles.

Example 7. Consider a subset $\mathcal{P}_{sup'}$ of the program $\mathcal{P}_{sup}$ only consisting of the following two clauses:

$$l \leftarrow e \wedge \neg ab_1. \qquad ab_1 \leftarrow \bot.$$

The repesctive set of abducibles $\mathcal{A}_{\mathcal{P}_{sup'}}$ consists of the following fact and assumption:

$$e \leftarrow \top. \qquad e \leftarrow \bot.$$

The set of non-complementary candidate explanations $\mathcal{C}_{\mathcal{A}_{\mathcal{P}_{sup'}}}$ consists of the following three elements:

$$\mathcal{C}_0 = \emptyset, \qquad \mathcal{C}_1 = \{e \leftarrow \top\}, \qquad \mathcal{C}_2 = \{e \leftarrow \bot\}.$$

Consider a program $\mathcal{P}_{sub''}$ which is the program $\mathcal{P}_{sup}$ with an additional clause, e.g. *if she has a presentation to prepare, then she will study late in the library*. Thus, we have the program $\mathcal{P}_{sub''}$ consisting of all the four clauses in $\mathcal{P}_{sup}$ plus the following two:

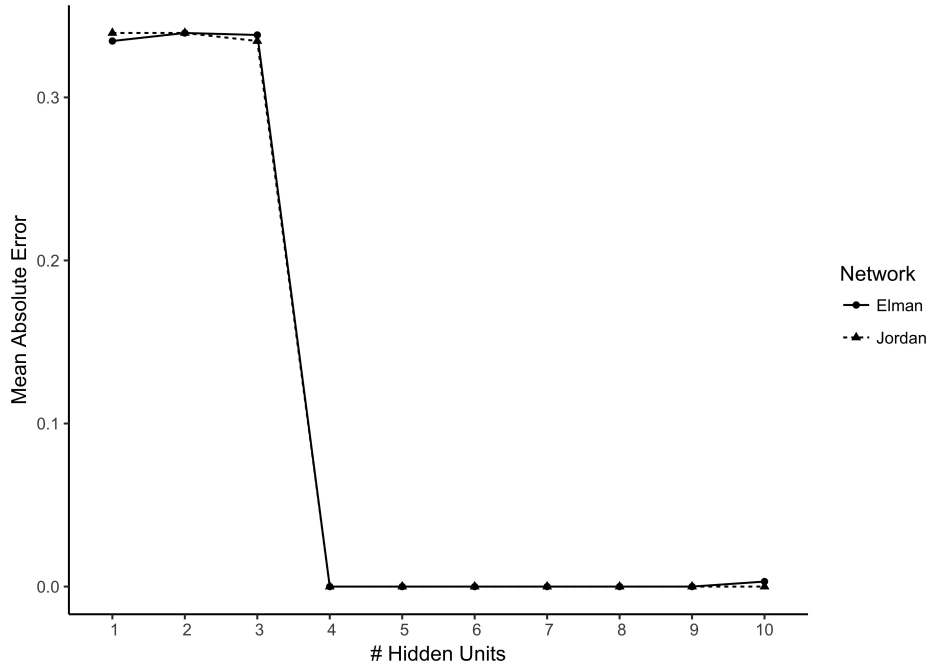
$$l \leftarrow p \wedge \neg ab_3. \qquad ab_3 \leftarrow \bot.$$

Where the new atom p stands for *she has a presentation to prepare* and ab_3 for the abnormality, as usual. Its repesctive set of abducibles $\mathcal{A}_{\mathcal{P}_{sup''}}$ consists of the following six elements:

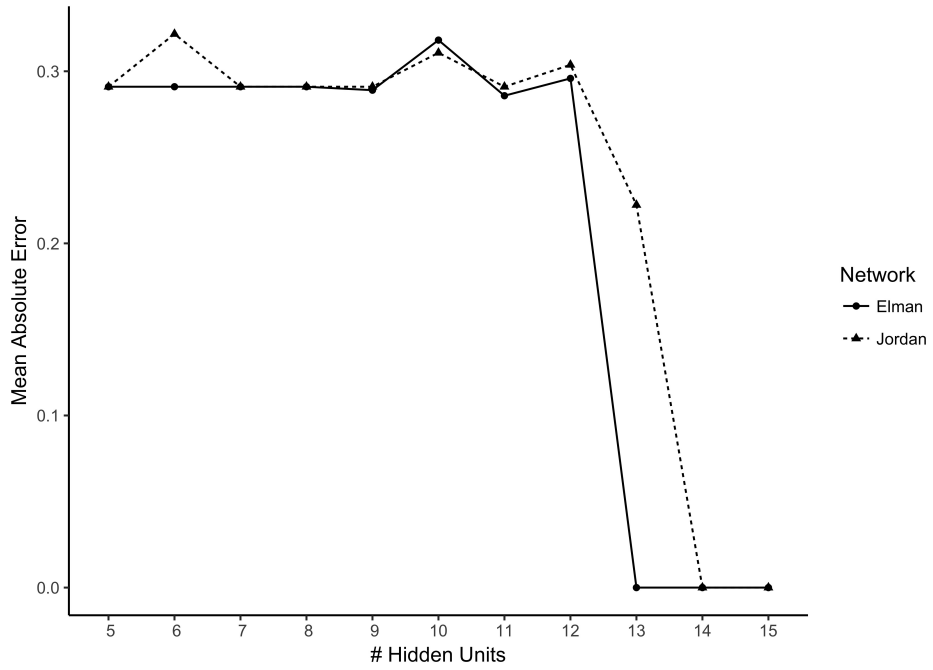
$$\begin{array}{lll} e \leftarrow \top. & t \leftarrow \top. & p \leftarrow \top. \\ e \leftarrow \bot. & t \leftarrow \bot & p \leftarrow \bot. \end{array}$$

The set of non-complementary candidate explanations $\mathcal{C}_{\mathcal{A}_{\mathcal{P}_{sup''}}}$ consists of the following twenty seven elements:

$$\begin{array}{lll} \mathcal{C}_1 = \emptyset, & & \\ \mathcal{C}_2 = \{e \leftarrow \top\}, & \mathcal{C}_3 = \{e \leftarrow \bot\}, & \\ \mathcal{C}_4 = \{t \leftarrow \top\}, & \mathcal{C}_5 = \{t \leftarrow \bot\}, & \\ \mathcal{C}_6 = \{p \leftarrow \top\}, & \mathcal{C}_7 = \{p \leftarrow \bot\}, & \\ \mathcal{C}_8 = \{e \leftarrow \top, t \leftarrow \top\}, & \mathcal{C}_9 = \{e \leftarrow \top, t \leftarrow \bot\}, & \\ \mathcal{C}_{12} = \{e \leftarrow \bot, t \leftarrow \top\}, & \mathcal{C}_{13} = \{e \leftarrow \bot, t \leftarrow \bot\}, & \\ \mathcal{C}_{14} = \{e \leftarrow \bot, p \leftarrow \top\}, & \mathcal{C}_{15} = \{e \leftarrow \bot, p \leftarrow \bot\}, & \\ \mathcal{C}_{16} = \{t \leftarrow \top, p \leftarrow \top\}, & \mathcal{C}_{17} = \{t \leftarrow \top, p \leftarrow \bot\}, & \\ \mathcal{C}_{18} = \{t \leftarrow \bot, p \leftarrow \top\}, & \mathcal{C}_{19} = \{t \leftarrow \bot, p \leftarrow \bot\}, & \\ \mathcal{C}_{20} = \{e \leftarrow \top, t \leftarrow \top, p \leftarrow \top\}, & \mathcal{C}_{21} = \{e \leftarrow \top, t \leftarrow \top, p \leftarrow \bot\}, & \\ \mathcal{C}_{22} = \{e \leftarrow \top, t \leftarrow \bot, p \leftarrow \top\}, & \mathcal{C}_{23} = \{e \leftarrow \top, t \leftarrow \bot, p \leftarrow \bot\}, & \\ \mathcal{C}_{24} = \{e \leftarrow \bot, t \leftarrow \top, p \leftarrow \top\}, & \mathcal{C}_{25} = \{e \leftarrow \bot, t \leftarrow \top, p \leftarrow \bot\}, & \\ \mathcal{C}_{26} = \{e \leftarrow \bot, t \leftarrow \bot, p \leftarrow \top\}, & \mathcal{C}_{27} = \{e \leftarrow \bot, t \leftarrow \bot, p \leftarrow \bot\}. & \end{array}$$



(a) Set of abducibles of cardinality two.



(b) Set of abducibles of cardinality six.

Figure 4.3: Results of an experiment to define the appropriated number of hidden units in a recurrent network to generate non-complementary candidates explanations for a given set of abducibles with cardinality two (a) and six (b). The size of the hidden units vary from one to ten. The graph plots the mean absolute error (MAE) for a 10-fold cross-validation averaged over ten trials for each network configuration.

Remember that for a set of abducibles $\mathcal{A}_\mathcal{P}$ of cardinality $2n$, there are 3^n non-complementary candidate explanations $\mathcal{C}_{\mathcal{A}_\mathcal{P}}$. These 3^n candidates can be ordered as $3^n!$ different sequences. For example, in the most basic case which is $n = 1$, we have that cardinality of $\mathcal{A}_\mathcal{P}$ is two, there are three candidate explanation which can be ordered in six different sequences, i.e.

$$\begin{array}{lcllcl}
\mathcal{C}^1_{\mathcal{A}_\mathcal{P}} & = & \{\mathcal{C}_1, \mathcal{C}_2, \mathcal{C}_3\}, & \mathcal{C}^2_{\mathcal{A}_\mathcal{P}} & = & \{\mathcal{C}_1, \mathcal{C}_3, \mathcal{C}_2\}, \\
\mathcal{C}^3_{\mathcal{A}_\mathcal{P}} & = & \{\mathcal{C}_2, \mathcal{C}_3, \mathcal{C}_1\}, & \mathcal{C}^4_{\mathcal{A}_\mathcal{P}} & = & \{\mathcal{C}_2, \mathcal{C}_1, \mathcal{C}_3\}, \\
\mathcal{C}^5_{\mathcal{A}_\mathcal{P}} & = & \{\mathcal{C}_3, \mathcal{C}_1, \mathcal{C}_2\}, & \mathcal{C}^6_{\mathcal{A}_\mathcal{P}} & = & \{\mathcal{C}_3, \mathcal{C}_2, \mathcal{C}_1\}.
\end{array}$$

Similarly, there are several different sequences for the sets of non-complementary candidate explanations regarding different cardinalities of the abducibles. In order to train the network with this different orders, we only have to set up the parameter *candidates* in the algorithms shown earlier and everything else works in the same way.

For instance, when solving Byrne's selection task represented by $\mathcal{P}_{sup}$, we have used the same sequence from Chapter 3 for sake of simplicity. However, any other sequence could have been used, e.g.

$$\begin{array}{lcl}
candidates^1 & = & \{00000, 10100, 10000, 01000, 00100, 00010, 10010, 01100, 01011\}, \\
candidates^2 & = & \{00000, 10100, 01011, 10000, 01000, 00100, 00010, 10010, 01100\}, \\
candidates^3 & = & \{00000, 10100, 10010, 10000, 01000, 00100, 00010, 01100, 01011\}, \\
candidates^4 & = & \{00000, 00100, 10010, 10000, 01000, 10100, 00010, 01100, 01011\}, \\
\ldots & & \\
candidates^{9!} & = & \{00100, 00010, 01100, 01011, 10100, 10010, 10000, 01000, 00000\}.
\end{array}$$

This shows that not only the same result as the one from the previous approach can be archived, but also that arbitrary sequences can be easily considered. However, as explained before, generating the non-complementary candidate explanations is not our main goal. Thus, we will now show how the same networks can also be used to generate the minimal candidate explanations.

4.2 Minimal Candidate Explanations

In this section will show that recurrent networks can be used not only to generate the non-complementary candidate explanations, as shown in Section 4.1, but also to generate the minimal candidate explanations for a given observation. The most important information necessary for this modification is whether the last candidate explanation has in fact explained the given observation or not.

We again show that this is indeed the case by means of an example. Thus, reconsider the program $\mathcal{P}_{sup}$ from Example 5 on page 36. The only necessary modification in the structure of the Jordan and Elman networks presented in Section 4.1 is to add one bit in the input vector which will now consist of six bits instead of five. This new bit gives us the informations of whether the last candidate was an explanation or not. This is reflected in the network by an additional unit in the input layer. The new version of both networks is shown in Figure 4.4.

The activation of this new unit e (*explanation*) is done by the external network responsible for checking whether a candidate explanation has explained the given observation or not. This is the same network which activates the unit *next* (*next*), which means that the request of a new candidate explanation and the activation of unit e are synchronised.

Also reconsider the observation

$$\mathcal{O}_1 = \{l\},$$

we know that its minimal explanations given program $\mathcal{P}_{sup}$ and the observation $\mathcal{O}_1$ are the following:

$$\begin{aligned} \mathcal{E}_1 &= \{e \leftarrow \top\}, \\ \mathcal{E}_2 &= \{t \leftarrow \top\}. \end{aligned}$$

Thus, when generating the candidate explanations, all the supersets of the two explanations $\mathcal{E}_1$ and $\mathcal{E}_2$ should not be taken into account. Thus, if we are also considering that the candidate explanations will be generated following the same order as in the previous sections, the following candidates explanations are expected to be generated:

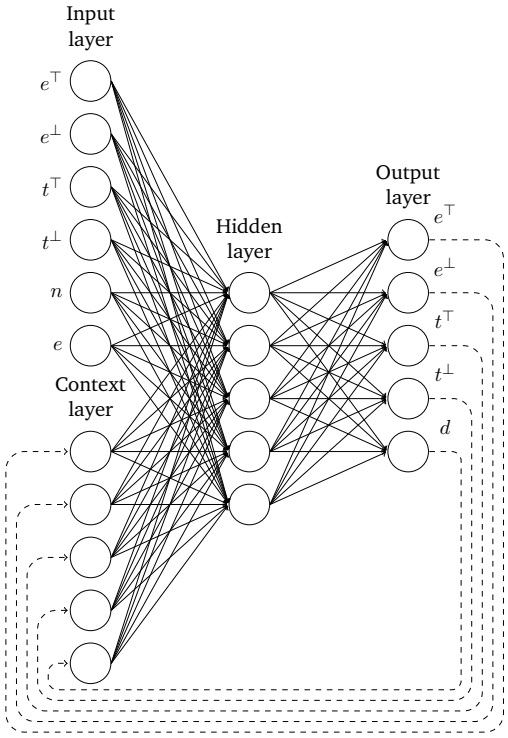
$$\begin{aligned} \mathcal{E}_0 &= \emptyset, \\ \mathcal{E}_1 &= \{e \leftarrow \top\}, \\ \mathcal{E}_2 &= \{e \leftarrow \bot\}, \\ \mathcal{E}_3 &= \{t \leftarrow \top\}, \\ \mathcal{E}_4 &= \{t \leftarrow \bot\}, \\ \mathcal{E}_8 &= \{e \leftarrow \bot, t \leftarrow \bot\}. \end{aligned}$$

In terms of the 5-bit binary vectors corresponding to the expected output the expected sequence if we have the next unit activated for six time steps in a row would be:

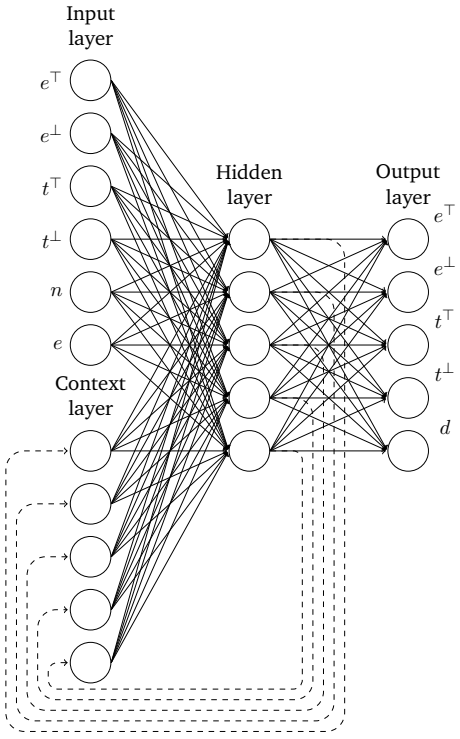
$$00000, 10000, 01000, 00100, 00010, 01011.$$

But of course, just as we had before, this is not always the case and, depending on the activation of the next unit, any sequence of the following form would be valid:

$$(00000)^+, (10000)^+, (01000)^+, (00100)^+, (00010)^+, (01011)^+.$$



(a) Jordan Network.



(b) Elman Network.

Figure 4.4: Structure of the Jordan (a) and Elman (b) networks for the generation of all non-complementary candidate explanations for the program $\mathcal{P}_{sup}$. Solid lines are trainable.

Input	Output
000000	00000
100001	10000
010000	01000
001001	00100
000100	00010
010100	01011

(a) Cases where the fifth bit in the input is *zero*

Input	Output
000010	10000
100011	01000
010010	00100
001011	00010
000110	01011
010110	00000

(b) Cases where the fifth bit in the input is *one*

Table 4.2: Each possible input and its respective expected output for the generation of minimal candidate explanations for the program $\mathcal{P}_{sup}$ and observation $\mathcal{O}_1 = \{l\}$ considering a specific ordering.

All possible inputs and its respective expected outputs for this example is shown in Table 4.2.

Note that, when we assume the candidates to be minimal, there is a cardinality constraint implicit in the ordering of the candidates. For the program $\mathcal{P}_{sup}$ which we are discussing, for example, the following ordering could no longer be considered:

$$candidates^2 \quad = \quad \{00000, 10100, 01011, 10000, 01000, 00100, 00010, 10010, 01100\}.$$

The candidate explanation 10100 representing $\{e^\top, t^\top\}$ is a positive non-minimal candidate explanation which the generation could not have been avoided if this order is considered. Therefore, we assume that for the generation of minimal candidate explanations, the sequences will respect the cardinality order. However, this does not mean that the order is now fixed. For a given set of candidates $\mathcal{C}$, there are the following number of sequences respecting the cardinality constraint:

$$c_0! * c_1! * \cdots * c_n!$$

where c_i represents the number of candidates in $\mathcal{C}$ with cardinality i. In our example, the six minimal candidates can be ordered as

$$1! * 4! * 1! = 16$$

different sequences, as there is one candidate with cardinality zero, four with cardinality one and one with cardinality two.

The training and the testing phase of the Jordan and Elman networks are done in the same way as described before. The only difference now is in the input of the network and the training and testing data. The input of the network will now contain an aditonal bit representing the request of next candidate explanation. The

data will now be generated considering all the possibilities of candidates being explanations and the resulting minimal set of candidate explanations for each of this cases. Table 4.2 shows one of this potential cases for a given program having a correspondent set of abducibles of size four.

As we know that in the case where the fifth bit is zero only the identity function is computed, we will only consider the cases where the fifth bit is one to demonstrate all the possible sixteen sequences for potential programs $\mathcal{P}$ and observations $\mathcal{O}$, such that the set of abducibles $\mathcal{A}_{\mathcal{P}}$ has cardinality four. Figure 3.4 on page 50 shows all the possible outcomes and paths for each of these cases and Table 4.3 shows the data we use in the training process such that the Jordan and Elman networks can reflect this behaviour.

So the new training and testing data for a given program $\mathcal{P}$ with a respective set of abducibles $\mathcal{A}_{\mathcal{P}}$ of size four will not only have a fixed sequence of size nine being generated, but all the ten different sequences of different sizes. This different sequences are due to the possible variations in the activation of unit e. We assume a corresponding program and observation which which leads to these expected data can be easily found. Therefore, in each pass of the training phase as well as in the testing phase, we choose one arbitrary mapping of potential input to expected output from the sixteen shown in Table 4.3 on page 72.

The same experiment described in the previous section is then performed again, but now taking into account the modifications in the generation of the training and verification data. In practice, all the algorithms shown before work exactly in the same way with the only difference that instead of having one fixed sequence of candidate explantions to be learned, we now have a set of possible candidate explanations. An arbitrary one is chosen from this set in the beginning of every new training phase.

The performance measure considered is also the same, e.g. mean absolute error. Figure 4.5a on page 73 shows the result of this experiment for the generation of minimal candidate explanation for a given program $\mathcal{P}$ with set of abducibles $\mathcal{A}_{\mathcal{P}}$ of size four. This network could then be used, for instance, to generate the minimal candidate explanations for a given observation $\mathcal{O}$ given the program $\mathcal{P}_{sup}$, since its respective set of abducibles $\mathcal{A}_{\mathcal{P}_{sup}}$ consists of four facts and assumptions.

If we filter the results shown in Figure 4.5a to see only the mean absolute error with respect to the cases shown in Table 4.2 which corresponds to the given observation $\mathcal{O}_1$, we get the results shown in Figure 4.5b on page 4.5a.

Input	Output
000010	10000
100010	01000
010010	00100
001010	00010
000110	10100
1010 10	10010
100110	01100
011010	01011
010110	00000

Input	Output
010110	00000
000010	10000
100011	01000
010010	00100
001010	00010
000110	01100
011010	01011

Input	Output
100110	00000
000010	10000
100010	01000
010011	00100
001010	00010
000110	10100
1010 10	10011

Input	Output
010110	00000
000010	10000
100010	01000
010010	00100
001011	00010
000110	10010
100110	01011

Input	Output
011010	00000
000010	10000
100010	01000
010010	00100
001010	00010
000111	10100
1010 10	01101

Input	Output
010110	00000
000010	10000
100011	01000
010010	00100
001011	00010
000110	01011

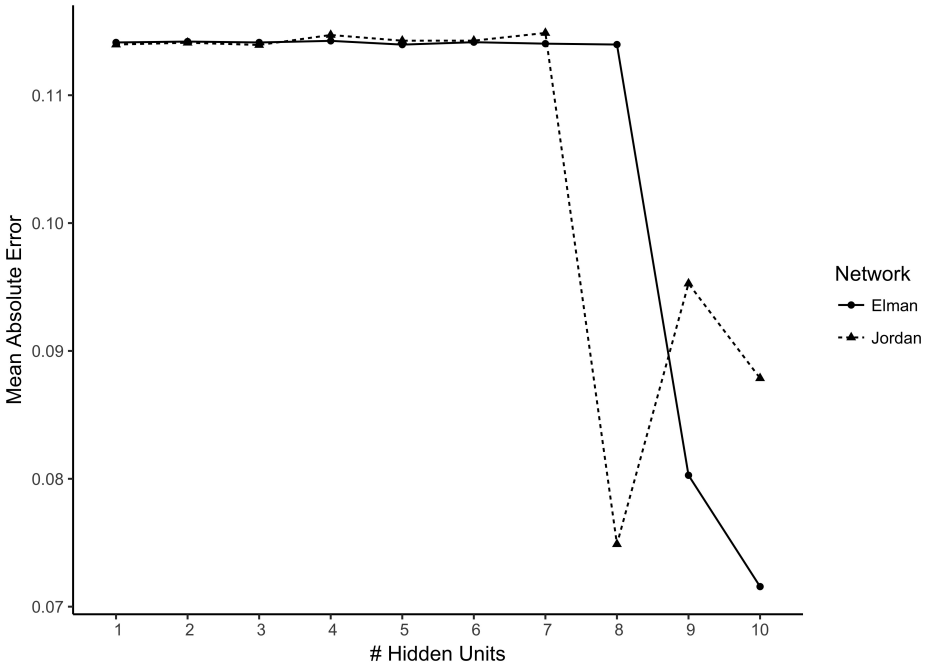
Input	Output
011010	00000
000010	10000
100011	01000
010010	00100
001010	00010
000111	01101

Input	Output
100110	00000
000010	10000
100010	01000
010011	00100
001011	00010
000110	10011

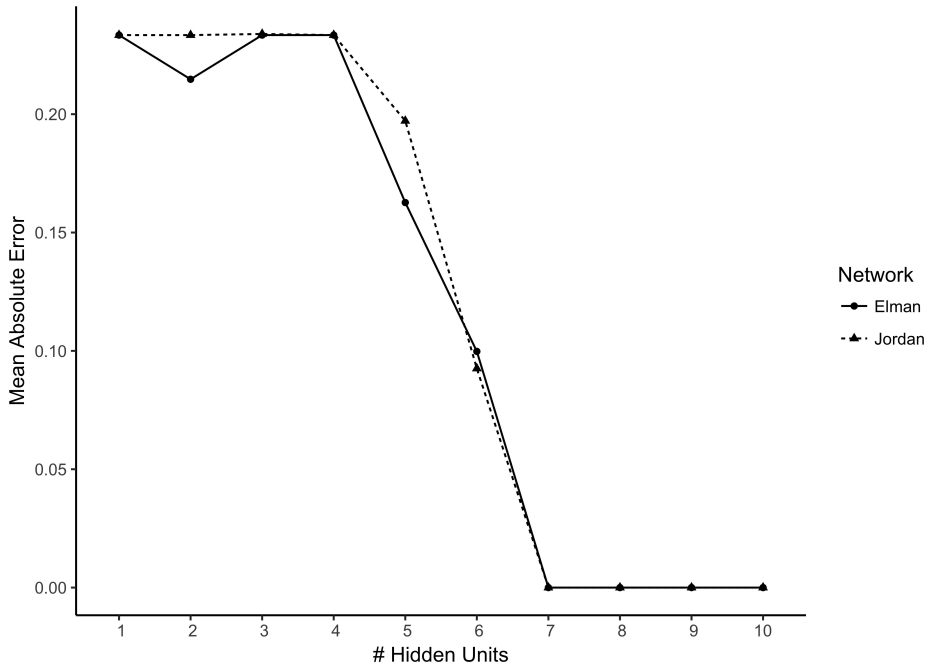
Input	Output
010110	00000
000010	10000
100010	01000
010011	00100
001010	00010
000111	01011

Input							Output
000111	000111	000111	000111	000111	000110	000110	00000
000010	000010	000010	000010	000010	000010	000010	10000
100011	100010	100010	100011	100011	100011	100011	01000
010011	010010	010011	010010	010011	010011	010011	00100
001011	001011	001011	001011	001010	001011	001010	00011

Table 4.3: Input and expected output for all the sixteen possible combinations of the fifth bit representing $explanation$.



(a) General results.



(b) Results filtrated by the test cases considering program $\mathcal{P}_{sup}$ and observation $\mathcal{O}_1$.

Figure 4.5: Results of an experiment to define the appropriated number of hidden units in a recurrent network to generate non-complementary candidates explanations for a set of abducibles of cardinality four, considering arbitrary programs and potential explanations. The size of the hidden units vary from one to ten. The graph plots the mean absolute error (MAE) for a 10-fold cross-validation averaged over ten trials for each network configuration.

With this we have shown that the Jordan and Elman networks can also be used to generate only the minimal candidate explanations for observation $\mathcal{O}_1$ given program $\mathcal{P}_{sup}$ with accuracy of one hundred percent starting from seven hidden units on.

Let us now consider another observation, e.g.

$$\mathcal{O}_2 = \{\neg l\}$$

In this case, the only explanation for the observation $\mathcal{O}_2$ given the program $\mathcal{P}_{sup}$ would be the explanation

$$\mathcal{E}_9 = \{e \leftarrow \bot, t \leftarrow \bot\}.$$

Therefore, all the nine candidate explanations would be generated just like in the network presented in the previous section, since only the last candidate

$$\mathcal{C}_9 = \mathcal{E}_9 = \{e \leftarrow \bot, t \leftarrow \bot\}.$$

is an explanation for the given observation. However, as we will see later, this is not the case for programs with a larger respective set of abducibles. Moreover, because these networks can easily learn different sequences of candidate explanations, we will all propose some cognitive experiments to define which order is the most appropriate one.

Just like in the previous section, the process shown here for a set of abducible consisting of four facts and assumptions can be easily repeated for sets of abducibles of different cardinalities. The only difference from the previous cases is that now the observation also plays a role in the sequence of candidate explanations to be generated, so even the same program could have different outcomes for different observations. This was illustrated by the example shown earlier.

Concerning the learning of arbitrary sequences of candidate explanations, throughout this section we have assumed a specific order for didactic reasons and to let clear how this networks can simulate the previous approach. However, any of the possible sequences mentioned earlier could have been chosen instead of the one which has been used. In fact, we have also repeated the tests shown in this section considering different orderings for those candidate and the results presented here were not affected.

4.3 Conclusion

In this chapter we have shown how to make use of Jordan and Elman networks to perform the task of generating candidate explanations. We started by defining how the non-complementary candidate explanations can be generated and then we restricted this candidates to the minimal ones. We have performed experiments to find the optimal number of hidden units for each of the two cases and also compared the results given by each of the two networks considered here.

As one may observe from these results, in all tested cases, there are some small differences on the number of hidden units necessary for the Jordan and Elman networks to perform the task, but, in general, they have shown to perform equally good. Concerning the generalisation capacity of these networks, this characteristic is not clear when considering the sets of the abducibles with small cardinality. However, this already changes for cardinality equals six, where there are twenty-seven candidates to be generated but only thirteen hidden units is enough to perform the task.

Given this, we can conclude that the approach presented in this chapter can perform as good as the one shown in Chapter 3 with the advantage that we abstracted the structure of the network. This means that we can generate the same sequence of candidate explanation as before, but with a fixed and well defined structure which can be reused for sets of abducibles of different sizes. In the current approach, instead of constructing a new network from the stretch for each possible cardinality of the set of abducibles, we reuse the same network structure and only have to learn the different sequences for each cardinality.

Moreover, as we have also shown, the process of creating and training such a network to generate the candidate explanations is totally automatised. The only necessary steps are the set up of parameters such as cardinality of the set of abducibles and which network to be used. Besides of this, since the generation of candidates is now done by means of learning, we can easily consider different orders for the sequence of candidates. As we will see later, the order in which the candidates are generated can play a role on the sceptical consequences if we are considering bounded reasoning. This gives our current approach one more advantage over the previous one.

5 Psychological Experiments

It is beyond the scope of this master thesis to specify and execute psychological experiments. In order to so, we need the support of cognitive scientists with a background in reasoning and memory models. Nevertheless, it is clear that just modelling is not satisfying and, as argued by Strube [43], knowledge engineering should also aim at being cognitively adequate. Our current approach relies on several assumptions and, therefore, in order to verify its cognitive adequacy, these assumptions should be tested. Hence, in this chapter we will consider various open questions and discuss preliminary possible setups for psychological experiments. These questions arise from a modelling point of view rooted in the weak completion semantics and its connectionist encoding as discussed in the previous chapters.

The first assumption we make is that humans do not consider several candidate explanations in parallel, but rather one at a time. Moreover, we require candidate explanations to be non-complementary and, later on, to be minimal as well. Based on the exponential number of candidate explanations, we also assume that humans do not consider all but rather a subset of them. If all the candidate explanations are considered, their generation order does not affect the outcome. However, if the hypothesis of bounded candidates is confirmed, then different orderings might lead to different outcomes and, therefore, we would also be interested in checking which order is preferred. Another important aspect would be which kind of bounds are applied by humans and based on what.

Some of these assumptions are intuitive and seem to make sense also from the cognitive point of view. For example, it is hard to imagine humans testing several candidates at the same time. On the other hand, some assumptions are merely motivated by the computational aspect, as, for example, requiring candidates to be minimal significantly reduces the number of candidates generated in the average case. However, it is not that difficult to imagine a scenario where a non-minimal explanation would be plausible. In either case, executing these psychological experiments is an important step which will give us the necessary support to verify, correct and improve our approach.

5.1 Hypotheses

Hypothesis I: Only basic candidate explanations are considered

As it is defined in the abductive framework, the set of abducibles consists only of the undefined atoms. Consequently, the set of candidate explanations is also only based on the undefined atoms which means that our approach only considers the basic candidate explanations. This means that, if we have from the background knowledge the conditionals *if a, then b* and *if b, then c*, and we observe that *c* holds, the only possible explanation is regarding *a*, but not *b*.

This is the first hypothesis made in our approach and, therefore, it would be appropriated to start the phycological experiments by verifying this assumption. A simple way to do so would be by giving the participants a background knowledge as the following:

If it rained last night, then the grass is wet.
If the grass is wet, then the shoes are wet.

If we observe that *the shoes are wet*, in our approach, because we only consider basic explanations, the only possible explanation for this observation would be *it trained last night*. Therefore, if the participants are given this observation and asked whether *it trained last night* follows, they would reply yes in the case our hypothesis is correct. This experiment alone is not enough to confirm our assumption, but it is an initial setup and would give us an indication on whether the hypothesis holds.

Hypothesis II: The generation of candidate explanations is done sequentially

Although it is hard to imagine that humans would consider several candidate explanations at the same time and reason with respect to them in parallel, this is an assumption which underlies our whole approach. Therefore, although it seems to be a plausible assumption, it would be appropriated to verify that this is indeed the case by means of phycological experiments.

Hypothesis III: Complementary candidate explanations are not considered

Our approach considers a positive information to be stronger than a negative information. This is an underlying assumption of the weak completion semantics as the clause $a \leftarrow \top$ and $a \leftarrow \bot$ are combined into the clause $a \leftrightarrow \top \vee \bot$, which is semantically equivalent to $a \leftrightarrow \top$. Therefore, from the computational point of view, considering complementary candidate explanations does not make sense as the negative part of the complementary pair is anyways overwritten.

Moreover, considering only non-complementary candidate explanations seems to make sense from the cognitive point of view as well. If we take their semantics into account, a complementary candidate would be contradictory as it contains a fact and an assumption with the same head. It is hard to imagine that one would try to explain a given observation by stating that something is true and false at the same time.

For example, let us reconsider Byrne's Suppression Task which we have repeatedly discussed in the previous chapters. If we observe that *she goes to the library* is true, we probably will not think that *she has an essay to write* and *she does not have an essay to write* is a reasonable explanation for our observation. We might think about them separately as explanations or together with other facts and assumptions, but probably not in the same candidate explanation.

Although this assumption seems to make sense from a cognitive point of view, it is still a consequence of the weak completion semantics which has not been tested by a psychological experiment and, therefore, we do not have the grounds to confirm its accuracy. Given this, including a test on the generation of only non-complementary candidates is desired.

Hypothesis IV: Only minimal candidate explanations are considered

Even after filtrating the candidate explanations to consider only the ones which are non-complementary, there is a considerably large number of candidates left to be considered. If the set of abducibles has cardinality $2n$ we obtain 3^n non-complementary candidate explanations. It is hard to imagine that humans systematically try all those candidates and, therefore, we try to reduce this number by adding another constraint which rules out the non-minimal candidate explanations as well. In the average case, this minimality constraint indeed significantly reduces the number of candidate explanations, but in the worst case the problem remains the same. For example, if the observation has only one explanation which is the last one in the sequence being considered, all the candidates are still generated.

Besides this, the minimality assumption is motivated by the fact that explanations are monotonic, which alone is not enough to state that humans exclude the non-minimal candidates when trying to explain an observation. Moreover, it is not so trivial to see that minimality also makes sense from a cognitive point of view. First of all, the minimality property implies a predefined ordering based on the cardinality of the candidates. As mentioned earlier, the ordering of the generation of candidate explanations is also an open question and, therefore, we can not only assume that humans also follow this order in their way of reasoning without verifying it.

Given this, an experiment which tests whether humans indeed consider only minimal candidate explanations is crucial. We could start by testing the hypothesis that humans only consider the minimal explanations, which can already give an indication on the minimality of the candidates. If this hypothesis is refuted, then we can already discard the minimality constraint. Otherwise, although it is a strong indication on the minimality of candidates, it could still be the case that non-minimal candidates are generated first and revised afterwards. Therefore, in the case the pre-hypothesis is confirmed, another experiment testing on the cardinality ordering of candidates is still strongly suggested.

One possible way to test this would be by not giving the participants enough time to generate all the explanations and ask them for the consequences. Based on this, we can observe if the given consequences are derived from smaller candidate explanations and the cardinality ordering is indeed considered. For example, consider that participants are given the conditionals in Byrne's suppression task as background knowledge, the fact *she will study late in the library* as an observation and asked if it follows that *she has an essay to write*. If participants are given the time to generate only one explanation and the cardinality constraint is not applied, then the first explanation would be that *she has an essay to write* and *a text book to read* and in this case the majority of the participants would say that *she has an essay to write* follows. In this case, our assumption would be refuted.

However, if participants consider the cardinality constraint, then the first explanation would be either that *she has an essay to write* or that *she has a text book to read*. In this case, the percentage of participants who would reply that *she has an essay to write* follows should drop to approximately 50%. If this is the case, them the experiment would support our hypothesis. The sequence would be affected by different factors, e.g. order in which the conditionals are presented, and, therefore, variations on the configurations of those factors should also be applied to the experiment for a more accurate result.

Hypothesis V: Candidate explanations are generated up to a bound

Even if hypothesis III is confirmed, minimality does not reduce the number of candidate explanations to be considered is all the cases. However, as mentioned before, we do not believe that it might be the case that humans generate all candidate explanations, but rather a subset of them. Therefore, there must be other factors affecting the subset of candidates considered. Later on we would like to investigate this factors as well, but first of all we would like to confirm the assumption that the generation of candidate explanations is indeed bounded.

One potential way of doing so could be, for example, by simply giving an abductive problem to the participants, asking them the skeptical consequences and, afterwards, analysing if those consequences can only follow in the scenario where all candidate explanations have been considered. If this is not the case, we have a strong indication that there is a bound and, therefore, the following hypotheses should be tested as well in order to better understand such bounds.

Hypothesis VI: Candidate explanations are generated in different orders

Let us recall the number of different orderings in which the candidate explanations might be considered. For a set of candidate explanations $\mathcal{C}$ of cardinality n there are $n!$ different ways of ordering these n candidate explanations. If we consider the minimality constraint in the ordering, the number of sequences can be reduced, but will still be significantly large specially when n increases. When applying the minimality constraint to $\mathcal{C}$, as we need to consider the cardinality ordering, we have the following number of orderings:

$$c_0! * c_1! * \cdots * c_n!,$$

where c_i represents the number of elements in $\mathcal{C}$ with cardinality i.

Given that, even after applying the minimality constraint, there are so many possible orderings, it is trivial to see that we cannot just assume one fixed and pre-defined sequence as it was done by the previous approach. Rather, we would like to observe how this generation is done by humans and, based on this, abstract properties which will allow us to restrict these possibilities.

As shown in Example 8 on page 83, the two different orderings considered for the same bound presents different sceptical consequences. Assume we would like to answer the question whether e follows sceptically from $\mathcal{P}_{sup}$ and $\mathcal{O}_1$. The answer for this question in the ordinary scenario is that e does not follow sceptically. However, we have seen that e follows skeptically for the first but not for the second ordering considered that the bound is equal to three.

In order to illustrate our assumptions, we use here small examples where the number of abducibles is not that large only for didactic reasons. In these examples, the bound might not make a lot of sense as it is totally plausible to imagine that humans would consider, for example, all the nine candidate explanations when reasoning about Byrne's suppression task. However, we have to consider that this number grows exponentially and, with four abducibles there are already eighty-one candidate explanations.

Hypothesis VII: Different bounds are applied depending on different factors

As has been mentioned before, there are many different ordering considered for the same set of candidate explanations. Therefore, if we consider a bound k for a given set of candidates $\mathcal{C}$, different orderings of $\mathcal{C}$ would result in different sceptical consequences for the same bound k. Moreover, the same ordering of $\mathcal{C}$ bounded by k assuming different values would also result in different sceptical consequences. For instance, in Example 8 on page 83, e follows skeptically for one of the orderings if a bound equal to three is considered, but this no longer holds if the bound is increased to four.

Considering that different skeptical consequences may follow for different bounds, it is crucial to understand the factors in which the definition of such a bound is based on so we can incorporate these factors in our approach. Instead of designing an experiment which focuses on the bound itself, we would rather propose several experiments concerning different background knowledges which allow us to derive the different bound considered and then analyse this data in order to abstract the factors out of it.

One way to do so would be by running a pre-experiment which defines the time needed for a given participant to generate only one candidate explanation. After this, we would measure the time needed for the same participant to solve the different given tasks and based on the outcome of the previous experiment derive which bound has been used. If we can identify patterns between the characteristics of the given problems and the bounds used, we might be able to define the factors which influences the process of defining these bounds.

5.2 Outlook

In this chapter we give a general overview of the hypothesis which are already considered in our approach as well as the ones we strongly believe and could easily incorporate in our framework. As the cognitive adequacy of our framework is a crucial aspect, we would like to execute psychological experiments which are going to confirm or refute our hypotheses.

In the case those hypothesis are confirmed, then we have a strong support for our theory. However, even if any of these hypotheses is refuted, we have designed our framework in such a flexible way that it could easily be adapted to any of the potential experimental outcomes.

Example 8. Reconsider program $\mathcal{P}_1$, observation $\mathcal{O}_1$ and their correspondent set of non-complementary and minimal candidate explanations $\mathcal{C}$. Note that $|\mathcal{C}| = 6$. If we do not take the cardinality constraint into account, there are $6! = 720$ possible sequences for $\mathcal{C}$. Otherwise, there are $1! * 4! * 1! = 24$. From these possible sequences, let us consider the following two:

$$\begin{aligned}\mathcal{C}_1 = &\ \{\emptyset, \{e \leftarrow \top\}, \{e \leftarrow \bot\}, \{t \leftarrow \top\}, \{t \leftarrow \bot\}, \{e \leftarrow \bot, t \leftarrow \bot\}\},\\ \mathcal{C}_2 = &\ \{\emptyset, \{t \leftarrow \top\}, \{t \leftarrow \bot\}, \{e \leftarrow \top\}, \{e \leftarrow \bot\}, \{e \leftarrow \bot, t \leftarrow \bot\}\}.\end{aligned}$$

The set of actual explanations for $\mathcal{C}_1$ and $\mathcal{C}_2$ are:

$$\begin{aligned}\mathcal{E}_{\mathcal{C}_1} = &\ \{\{e \leftarrow \top\}, \{t \leftarrow \top\}\},\\ \mathcal{E}_{\mathcal{C}_2} = &\ \{\{t \leftarrow \top\}, \{e \leftarrow \top\}\}.\end{aligned}$$

If we now consider bounded reasoning with bound $k = 3$ and $k = 4$, we have:

$$\begin{aligned}\mathcal{C}_1^3 = &\ \{\emptyset, \{e \leftarrow \top\}, \{e \leftarrow \bot\}\}, \qquad \mathcal{C}_1^4 = \ \{\emptyset, \{e \leftarrow \top\}, \{e \leftarrow \bot\}, \{t \leftarrow \top\}\},\\ \mathcal{C}_2^3 = &\ \{\emptyset, \{t \leftarrow \top\}, \{t \leftarrow \bot\}\}, \qquad \mathcal{C}_2^4 = \ \{\emptyset, \{t \leftarrow \top\}, \{t \leftarrow \bot\}, \{e \leftarrow \top\}\}.\end{aligned}$$

The set of actual explanations for bounded $\mathcal{C}_1$ and $\mathcal{C}_2$ are:

$$\begin{aligned}\mathcal{E}_{\mathcal{C}_1^3} = &\ \{\{e \leftarrow \top\}\},\\ \mathcal{E}_{\mathcal{C}_2^3} = &\ \{\{t \leftarrow \top\}\},\\ \mathcal{E}_{\mathcal{C}_1^4} = \mathcal{E}_{\mathcal{C}_2^4} = &\ \{\{t \leftarrow \top\}, \{e \leftarrow \top\}\}.\end{aligned}$$

The sceptical consequences for the bounded candidate explanations are:

$$\begin{aligned}\mathcal{S}_{\mathcal{E}_{\mathcal{C}_1^3}} = &\ \langle\{e, l\}, \{ab_1, ab_2\}\rangle,\\ \mathcal{S}_{\mathcal{E}_{\mathcal{C}_2^3}} = &\ \langle\{t, l\}, \{ab_1, ab_2\}\rangle.\\ \mathcal{S}_{\mathcal{E}_{\mathcal{C}_1^4}} = \mathcal{S}_{\mathcal{E}_{\mathcal{C}_2^4}} = &\ \langle\{l\}, \{ab_1, ab_2\}\rangle.\end{aligned}$$

The facts e and t follow sceptically from $\mathcal{P}_1$ and $\mathcal{O}_1$ if the orderings $\mathcal{C}_1^3$ and $\mathcal{C}_2^3$ are respectively considered with bound three. However, this is no longer the case if the bound is increased to four.

6 Conclusion

Psychological experiments, as Byrne's suppression task and Wason's selection task, have shown that human deviate from classical logic when performing reasoning tasks. considering this, we have focused on a non-monotonic approach based on the weak completion of logic programs to formalise episodes of human reasoning. More precisely, we have worked with a connectionist network which encodes such an approach and generates the sceptical consequences of an abductive problem.

This network consists of three main components which are responsible for generating the candidate explanations, computing the consequences of each explanation, and, finally, deriving the sceptical consequences for the given observation. Deciding whether a formula follows sceptically from an abductive framework is DP-complete, a complexity that is outside of NP. Because the bottleneck of this problem is the exponential number of candidate explanations which has to be considered, we have focused on the component of the network responsible for their generation and proposed some ways of optimising it.

In Chapter 3 we have presented the current state of the network which generates all the non-complementary candidate explanations. Later in this chapter we have shown one possible way to reduce the number on candidate explanations which is introducing the minimality constraint. In this chapter, we have considered a static sequence of candidate explanations. However, the large amount of possible sequences in which these candidate explanations can be considered is a strong indicator that humans do not always consider a static predefined sequence.

Based on this, in Chapter 4 we have proposed to substitute the current approach by a recurrent network, such as Jordan or Elman, which has shown to perform perfectly in the task of learning arbitrary sequences of candidate explanation. These networks have presented an error rate of zero in all the cases tested and we have shown how general this approach is, allowing us to easily adapt to generate different sequences of non-complementary, minimal and even bounded candidate explanations.

The complexity of this task gives us the strong impression that humans do not consider all the candidate explanations but rather a subset of it. One possible way of reducing this number is by applying the minimality constraint which we have

previously discussed. However, this minimality implies a cardinality order which we have not verified. Besides this, even if the minimality constraint reduces the number of candidates in practice, the complexity of the problem remains the same in the worst case. Therefore, we strongly believe that there are other parameters which are taken into account when generating the candidate explanation and we define a bound based on them.

Considering this, we have proposed the setup of some psychological experiments in Chapter 5 which verify the assumptions we have made and also try to identify some other properties in the ordering and bounds of the candidate explanations. As mentioned before, the new approach proposed here is designed in such a way that it could be aligned with these experiments independent on their outcome.

As future work, we propose the realisation of the psychological experiments discussed earlier. If the results of these experiments do not confirm the assumptions we have made, then we also propose to apply the necessary modifications to our approach such that it is aligned with the cognitive process regarding the task of solving an abductive problem. Moreover, the experimental results might also allow us to add more constraints to the process of generating candidate explanations, concerning for example the bound or ordering of the candidates.

Summing up, we have come up with an approach for the generation of candidate explanations which can be easily adapted to arbitrary sequences. As discussed before, from the psychological point of view, there are still many open questions concerning sceptical abduction. Therefore, having such a versatile approach is the ideal solution considering the current state of the problem.

List of Tables

List of Figures

List of Symbols

$\top$	Truth value *true*.
$\bot$	Truth value *false*.
U	Truth value *unknown*.
$\mathcal{P}$	Logic program.
$c\mathcal{P}$	Completion of $\mathcal{P}$.
$wc\mathcal{P}$	Weak completion of $\mathcal{P}$.
$atoms(\mathcal{P})$	Set of all atoms occurring in $\mathcal{P}$.
$def(\mathcal{P})$	Set of all defined atoms occurring in $\mathcal{P}$.
$undef(\mathcal{P})$	Set of all undefined atoms occurring in $\mathcal{P}$.
$\mathcal{A}_\mathcal{P}$	Set of abducibles for a program $\mathcal{P}$.
$\mathcal{IC}$	Set of integrity constraints.
$\mathcal{O}$	Observation.
$\mathcal{E}$	Explanation.
$\mathcal{C}$	Candidate explanation.
$\mathcal{C}_{\mathcal{A}_\mathcal{P}}$	Set of candidate explanations for the set of abducibles $\mathcal{A}_\mathcal{P}$.
$\models$	Consequence relation.
$\models_{wcs}$	Consequence relation under the weak completion semantics.
lfp	Least fixed point.
$\Phi_\mathcal{P}$	Semantic operator introduced by Stenning and van Lambalgen.
$\mathcal{I}$	Interpretation.
$\mathcal{I}^\top$	Interpretation consisting of the atoms that are mapped to $\top$.
$\mathcal{I}^\bot$	Interpretation consisting of the atoms that are mapped to $\bot$.

Bibliography

[1] Krzysztof R Apt and Maarten H Van Emden. „Contributions to the theory of logic programming“. In: *Journal of the ACM (JACM)* 29.3 (1982), pp. 841–862.

[2] S Bader and S Hölldobler. „The core method: Connectionist model generation“. In: *Artificial Neural Networks–ICANN 2006* (2006), pp. 1–13.

[3] Ruth M J Byrne. „Suppressing valid inferences with conditionals“. In: *Cognition* 31.1 (1989), pp. 61–83.

[4] Ruth M J Byrne. *The rational imagination: How people create alternatives to reality*. MIT press, 2007.

[5] Keith L Clark. „Negation as failure“. In: *Logic and data bases*. Springer, 1978, pp. 293–322.

[6] Ana Oliveira da Costa, Emmanuelle-Anna Dietz Saldanha, Steffen Hölldobler, and Marco Ragni. „A computational logic approach to human syllogistic reasoning“. In: (2017).

[7] James R Cox and Richard A Griggs. „The effects of experience on performance in Wason's selection task“. In: *Memory & Cognition* 10.5 (1982), pp. 496–502.

[8] Emmanuelle-Anna Dietz, Steffen Hölldobler, and Marco Ragni. „A computational logic approach to the abstract and the social case of the selection task“. In: *Proceedings of the 11th International Symposium on Logical Formalizations of Commonsense Reasoning, COMMONSENSE*. 2013.

[9] Kristien Dieussaert, Walter Schaeken, Walter Schroyens, and Géry d'Ydewalle. „Strategies during complex conditional inferences“. In: *Thinking & reasoning* 6.2 (2000), pp. 125–160.

[10] Jeffrey L Elman. „Finding structure in time“. In: *Cognitive science* 14.2 (1990), pp. 179–211.

[11] Jeffrey L Elman. „Representation and structure in connectionist models“. In: (1989).

[12] Jonathan St BT Evans. *Reasoning, rationality and dual processes: Selected works of Jonathan St BT Evans*. Psychology Press, 2013.

[13] Jerome A Feldman and Dana H Ballard. „Connectionist models and their properties“. In: *Cognitive science* 6.3 (1982), pp. 205–254.

[14] Richard A Griggs and James R Cox. „The elusive thematic-materials effect in Wason's selection task“. In: *British Journal of Psychology* 73.3 (1982), pp. 407–420.

[15] John A Hertz, Anders S Krogh, and Richard G Palmer. *Introduction to the theory of neural computation*. Vol. 1. Basic Books, 1991.

[16] Steffen Hölldobler and Carroline DP Kencana Ramli. „Logic programs under three-valued Łukasiewicz semantics“. In: (2009), pp. 464–478.

[17] Steffen Hölldobler and Carroline DP Kencana Ramli. „Logics and networks for human reasoning“. In: *International Conference on Artificial Neural Networks*. Springer. 2009, pp. 85–94.

[18] PN Johnson-Laird. *Mental models: Towards a cognitive science of language, inference, and consciousness*. 6. Harvard University Press, 1983.

[19] Michael I Jordan. „Serial order: A parallel distributed processing approach“. In: *Advances in psychology* 121 (1997), pp. 471–495.

[20] Sangeet Khemlani and PN Johnson-Laird. *Theories of the syllogism: A meta-analysis.* 2012.

[21] Stephen Cole Kleene, NG de Bruijn, J de Groot, and Adriaan Cornelis Zaanen. *Introduction to metamathematics*. Vol. 483. van Nostrand New York, 1952.

[22] Robert Kowalski. *Computational logic and human thinking: how to be artificially intelligent*. Cambridge University Press, 2011.

[23] John W Lloyd. *Foundations of logic programming*. Springer Science & Business Media, 2012.

[24] Jan Łukasiewicz. „On three-valued logic“. In: *The Polish Review* (1968), pp. 43–44.

[25] Warren S McCulloch and Walter Pitts. „A logical calculus of the ideas immanent in nervous activity“. In: *The bulletin of mathematical biophysics* 5.4 (1943), pp. 115–133.

[26] Larry Medsker and Lakhmi C Jain. *Recurrent neural networks: design and applications*. CRC press, 1999.

[27] Edward F Moore. *Gedanken-experiments on sequential machines. Automata studies, annals of mathematical studies, No. 34*. 1956.

[28] P. Rougier Nicolas. *neural-networks*. `https://github.com/rougier/neural-networks`. 2012.

[29] Christos H Papadimitriou. *Computational complexity*. John Wiley and Sons Ltd., 2003.

[30] Charles Sanders Peirce. *Collected papers of charles sanders peirce*. Vol. 5. Harvard University Press, 1974.

[31] Thad A Polk and Allen Newell. „Deduction as verbal reasoning.“ In: *Psychological Review* 102.3 (1995), p. 533.

[32] M Ragni, E A Dietz, I Kola, and S Hölldobler. „Two-Valued Logic is Not Sufficient to Model Human Reasoning, but Three-Valued Logic is: A Formal Analysis.“ In: *Bridging 2016 – Bridging the Gap between Human and Automated Reasoning*. Ed. by U. Furbach and C. Schon. CEUR Workshop Proceedings. CEUR-WS.org, 2016, pp. 61–73.

[33] Lance J Rips. *The psychology of proof: Deductive reasoning in human thinking*. Mit Press, 1994.

[34] Carolina Ruiz and Jack Minker. „Computing stable and partial stable models of extended disjunctive logic programs“. In: *International Workshop on Nonmonotonic Extensions of Logic Programming*. Springer. 1994, pp. 205–229.

[35] David E Rumelhart and James L Mcclelland. *Parallel distributed processing: Explorations in the microstructure of cognition: Foundations (Parallel distributed processing)*. 1986.

[36] David E Rumelhart, Geoffrey E Hinton, James L McClelland, et al. „A general framework for parallel distributed processing“. In: *Parallel distributed processing: Explorations in the microstructure of cognition* 1 (1986), pp. 45–76.

[37] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. *Leanrning representations by back-propagating errors*. na, 1986.

[38] DE Rumelhart and DA Norman. „Parallel associative models of memory: Where do they fit“. In: *Hinton and Anderson* 45 (1981).

[39] Emmanuelle-Anna Dietz Saldanha, Steffen Hölldobler, Carroline DP Kencana Ramli Kencana Ramli, and Luis Palacios Medinacelli. „A Core Method for the Weak Completion Semantics with Skeptical Abduction“. In: *Journal of Artificial Intelligence Research (accepted)* (2017).

[40] Emmanuelle-Anna Dietz Saldanha, Steffen Hölldobler, and Tobias Philipp. „Contextual abduction and its complexity issues“. In: *Proceedings of the 4th International Workshop on Defeasible and Ampliative Reasoning (DARe) co-located with the 14th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR)*. Vol. 1872. 2017, pp. 58–70.

[41] Emmanuelle-Anna Dietz Saldanha, Steffen Hölldobler, and Isabelly Lourêdo Rocha. „Obligation versus Factual Conditionals under the Weak Completion Semantics“. In: *YSIP2*. Ed. by S Hölldobler, A Malikov, and C Wernhard. Vol. 1837. CEUR-WS.org. 2017, pp. 55–64.

[42] Keith Stenning and Michiel Van Lambalgen. *Human reasoning and cognitive science*. MIT Press, 2012.

[43] Gerhard Strube. „The role of cognitive science in knowledge engineering“. In: *Contemporary knowledge engineering and cognition*. Springer, 1992, pp. 159–174.

[44] Peter C Wason. „Reasoning about a rule“. In: *The Quarterly journal of experimental psychology* 20.3 (1968), pp. 273–281.

Declaration

I declare that I have developed and written the enclosed Master Thesis completly by myself, and have not used sources or means without declaration in the text. Any thoughts from others or literal quotations are clearly marked. The Master Thesis was not used in the same or in a similar version to achieve an academic grading or is being published elsewhere.

Dresden, November 6th, 2017

Isabelly Lourêdo Rocha