A Neural Network Approach for First-Order Abductive Inference

Oliver Ray and **Bruno Golénia**
Department of Computer Science
University of Bristol
Bristol, BS8 1UB, UK
{oray,goleniab}@cs.bris.ac.uk

Abstract

This paper presents a neural network approach for first-order abductive inference by generalising an existing method from propositional logic to the first-order case. We show how the original propositional method can be extended to enable the grounding of a first-order abductive problem; and we also show how it can be modified to allow the prioritised computation of minimal solutions. We illustrate the approach on a well-known abductive problem and explain how it can be used to perform first-order conditional query answering.

1 Introduction

Neurosymbolic research aims to combine neural inference methods with symbolic knowledge formalisms in order to better understand and exploit the cognitive functions of brain and mind. The integration of neural networks and logic programs is a major focus in this field. But, most existing work only deals with neural representations of propositional logic programs – which are not very well suited to applications with complex or incomplete information. Thus, our goal is to develop a connectionist approach for typed abductive logic programs – that are specifically intended for this purpose.

Abductive logic programs extend normal logic programs by allowing the truth of particular literals, called *abducibles*, to be assumed subject to certain conditions, called *integrity constraints*, when deciding which instances of a query, or *goal*, succeed from some program, or *theory*. Thus, any solution to a first-order abductive problem has two parts: a set of variable bindings, called an *answer*, denoting a successful instance of the goal; together with a set of abducibles, called an *explanation*, denoting a set of auxiliary assumptions. In this way, abductive logic programming can be seen as a form of conditional query answering.

This paper presents a neural network approach for first-order abductive inference. The approach generalises an existing method of Ray & Garcez [9] from propositional logic to the first-order case. Like its predecessor, the approach is based on translating an abductive problem into a neural network such that the fixpoints of the network are in one-to-one correspondence with the solutions of the original problem. Unlike other methods for neural abduction, such as [3; 10; 2;

13; 7; 12; 8; 1], our approach has the benefits of placing no restrictions on the use of negation or recursion and being able to systematically compute all required solutions.

Our main contributions are twofold: we show how the propositional method can be extended to enable the practical grounding of a first-order abductive problem; and we also show how it can be modified to allow the prioritised computation of minimal solutions. The former extension uses techniques from the field of Answer Set Programming (ASP) [6] to reduce the number of logically redundant clauses and literals in the ground program. The latter modification ensures the hypothesis space is searched in a way that gives higher priority to solutions with fewer abducibles. We illustrate the approach and explain how it can be used to perform first-order conditional query answering.

The rest of the paper is structured as follows: Section 2 recalls the relevant background material; Section 3 presents our approach; and the paper concludes with a summary and directions for future work.

2 Background

This section recalls some basic notions of neural networks and logic programs. The definitions closely follow those of Ray & Garcez [9], except that variables are now explicitly typed and may appear in any input to an abductive problem.

(Threshold) Neural Networks: A *neural network*, or just *network*, is a graph (N, E) whose nodes N are called *neurons* and whose edges $E \subseteq N \times N$ are called *connections*. Each neuron $n \in N$ is labelled with a real number $t(n)$ called its *threshold* and each connection $(n, m) \in E$ is labelled with a real number $w(n, m)$ called its *weight*. The *state* of a network is a function s that assigns to each neuron the value 0 or 1. A neuron is said to be *active* if its value is 1 and it is said to be *inactive* if its value is 0. For each state s of the network, there is a unique successor state s' such that a neuron n is active in s' iff its threshold is exceeded by the sum of the weights on the connections coming into n from nodes which are active in s. A network is said to be *relaxed* iff all of its neurons are inactive. A *fixpoint* of the network is any state that is identical to its own successor state. If a fixpoint t is reachable from an initial state s by repeatedly computing successor states, then t is referred to as the *fixpoint of s*.

$$T \;=\; \left\{ \begin{array}{l} wont\_start(C) \leftarrow flat\_battery(C) \\ wont\_start(C) \leftarrow no\_fuel(C) \\ wont\_start(C) \leftarrow spark\_plugs\_dirty(C) \\ head\_lights\_work(c1) \\ fuel\_gauge\_empty(c2) \\ spark\_plugs\_dirty(c3) \end{array} \right\} \cup \left\{ \begin{array}{l} car(c1) \\ car(c2) \\ car(c3) \end{array} \right\}$$

$$G \;=\; \{\; wont\_start(C) \wedge \neg spark\_plugs\_dirty(C) \;\}$$

$$IC \;=\; \left\{ \begin{array}{l} \leftarrow flat\_battery(C) \wedge head\_lights\_work(C) \\ \leftarrow no\_fuel(C) \wedge \neg fuel\_gauge\_empty(C) \wedge \neg broken\_gauge(C) \end{array} \right\}$$

$$A \;=\; \left\{ \begin{array}{l} flat\_battery(C) \\ no\_fuel(C) \\ broken\_gauge(C) \end{array} \right\}$$

$$D \;=\; \{\; car(C) \;\}$$

Figure 1: Abductive context for the *classic cars* problem

(Typed) Logic Programs: A *typed logic program*, or just *program*, is a set of rules of the form $H \leftarrow B_1 \wedge \ldots \wedge B_n \wedge \neg C_1 \wedge \ldots \wedge \neg C_m$ in which the H, B_i and C_j are atoms and in which any variable is associated with a unary predicate called its *type* or *domain*. The atom to the left of the arrow is called *head* of the rule, while the literals to the right comprise the *body*. The head atom H and the positive body atoms B_i are said to occur *positively* in the rule, while the negated body atoms C_j are said to occur *negatively*. A rule with no negative body literals is called a *definite clause* and written $H \leftarrow B_1 \wedge \ldots \wedge B_n$. A rule with no body literals is called a *fact* and simply written H. A rule with no head literal is called a *denial* and written $\leftarrow B_1 \wedge \ldots \wedge B_n$. If P is a program, then $\mathcal{B}_P$ (the *Herbrand base* of P) is the set of all atoms built from the predicate and function symbols in P; and $\mathcal{G}_P$ (the *ground expansion* of P) is the program comprising all *well-typed* ground instances of the clauses in P. In addition, $\mathcal{A}_P^+$ and $\mathcal{A}_P^-$ denote, respectively, the sets of ground atoms that occur positively and negatively in $\mathcal{G}_P$. A *stable model* of P is a Herbrand interpretation $I \subseteq \mathcal{B}_P$ that coincides with the least Herbrand model of the definite program P^I obtained by removing from $\mathcal{G}_P$ each rule containing a negative literal not in I, and by deleting all of the negative literals in the remaining rules.

Abductive Logic Programs: An *abductive logic program* [4] is a triple (T, IC, A) where T is a program (the *theory*), IC is a set of denials (*integrity constraints*), A is a set of facts (*abducibles*). Given a conjunction G of literals (*goals*), the task of ALP is to compute a set θ of variable substitutions (an *answer*) and a set Δ of ground abducibles (an *explanation*) such that there is a stable model of $T \cup \Delta$ (which, in the terminology of [5], is called a *generalised stable model* of T) that satisfies all of the denials in IC and all of the literals in $G\theta$. To specify an abductive problem, one must state the theory T, goal G, constraints IC, and abducibles A. If needed, the types of any variables can be explicitly given as a set of facts D (*domain declarations*) with one atom $p(X)$ for every variable X of type p. For convenience, we will refer to the collection of five inputs (T, G, IC, A, D) as a *(typed) abductive context*. A context is propositional if it contains no variables. When a context has many different solutions it is usual to prefer explanations with the fewest number of abducibles. Intuitively, this corresponds to the principle of Occam's Razor, which favours the simplest hypotheses that fit the data. In practice, this means that subset-minimal and/or cardinality-minimal explanations are usually required.

Example 2.1. *Consider the abductive context in Figure 1. The theory T describes a collection of three* classic cars. *It states that a car C wont start if its battery is flat, if its fuel tank is empty, or if its spark plugs are dirty. It also states that the headlights of the first car $c1$ are working, that the fuel gauge of the second car $c2$ is showing empty, and that the spark plugs of the final car $c3$ are dirty. The constraints IC state, firstly, that the battery of a car cannot be flat if the headlights of that car are working and, secondly, that it is impossible for a car to have no fuel if its fuel gauge is not empty and not broken. The abducibles A allow us to assume that any car has a flat battery, has no fuel, and/or has a broken fuel gauge. The goal G asks for which cars C it is possible to show that (1) the car does not start and that (2) the car does not have dirty spark plugs. The domain declarations D assert that all occurrences of the variable C represent* cars.

This problem has many solutions. There are two cardinality-minimal solutions which bind C to $c2$ after abducing $no\_fuel(c2)$ or $flat\_battery(c2)$, respectively. There is one more subset-minimal solution which binds C to $c1$ after abducing $no\_fuel(c1)$ and $broken\_gauge(c1)$. Note that, by the second constraint, $no\_fuel(c1)$ can only be abduced if $broken\_gauge(c1)$ is also abduced (at its gauge is not showing empty). Note also, by the first constraint, $flat\_battery(c1)$ cannot be abduced (as its headlights work). More than a hundred non-minimal solutions can be obtained by adding redundant abducibles to the explanations above.

As non-minimal solutions are often of little practical use, this paper will develop a neural approach for preferentially finding minimal solutions of first-order abductive problems.

3 First order Neural network abduction

The approach in this section builds on well-known methods for translating propositional logic programs into neural networks whose fixpoints correspond to stable models. In actual fact, we build upon an extension of these methods proposed by Ray and Garcez [9] for translating propositional abductive contexts into neural networks whose fixpoints correspond to generalised stable models. This is done by rewriting negative literals as abducibles and adding clauses that, when translated into a neural network, perform a systematic search through the space of abductive solutions.

In principle, one could try and solve a first-order problem by applying the method of [9] to the propositional context obtained by taking all possible well-typed ground instances of the abductive inputs. But, in practice, this naive approach is not feasible as it generates neural networks that are too large and take too long to find the minimal solutions.

To address these limitations, we developed an improved method for more effectively computing the minimal solutions of first-order problems. The new approach exploits an ASP system called LPARSE [11] to allow the efficient grounding of a first-order program; and it also ensures the prioritised computation of minimal solutions by modifying the search strategy of the original method.

Given a typed first-order abductive context, the new method has six main steps: First it introduces new predicates into the language to represent abducibles and negations. Then it translates the typed abductive context into a normal propositional abductive context using LPARSE together with some appropriate pre- and post-processing. Next it rewrites any negative literals in the theory as abducibles subject to some simple integrity constraints that preserve their logical meaning. After that, it adds some clauses that allow the resulting network to compute abductive solutions. At this point, it translates the resulting context into a neural network. Finally it allows the network to compute the abductive solutions in order of minimality. These steps are now described in turn.

STEP 0: Extending the Language

The logical transformations used in our method require the introduction of predicates to represent abducibles and negations. For every predicate p in a given abductive context, we assume two new predicates, denoted $p^\dagger$ and p^*, which represent the *assumption* of p and the *negation* of p, respectively.

STEP 1: Obtaining a Propositional Context

Our approach for translating a typed first-order abductive context into a normal propositional abductive context uses a system called LPARSE which, as described in [11], is practical tool for grounding typed logic programs. Compared to a naive propositionalisation approach, LPARSE further simplifies the ground program by removing literals that are known to be true from the body of a clause and by removing clauses whose bodies are known to be false from the program. Since LPARSE is not specifically designed to work with abductive logic programs, some pre- and post-processing is needed to make the approach work. Thus we have three sub-steps:

• **Pre-processing:** We first turn an abductive context into a logically equivalent program by temporarily employing negative cycles to represent abducibles. This is needed to prevent LPARSE from treating abducibles as undefined domain predicates, which would otherwise be removed from the ground program. Then domain declarations of the form $\#domain\ p(V)$ are added for each variable V of type p in D (which causes LPARSE to add the atom $p(V)$ into the body of every clause containing V). Finally, the goal G is turned into a clause by using a new propositional atom $goal$ as the head. This is carried out by the function ζ below.

Definition 3.1 (ζ). *Let* $X = (T, G, IC, A, D)$ *be a typed abductive context and let* G', A' *and* D' *be as follows*
$$G' = \{goal \leftarrow L_1 \wedge \ldots \wedge L_n \,|\, L_1 \wedge \ldots \wedge L_n = G\},$$
$$A' = \left\{ \begin{array}{c} a \leftarrow \neg a^* \\ a^* \leftarrow \neg a \end{array} \,\middle|\, a \in A \right\}$$
$$D' = \{\#domain\ p(V) \,|\, p(V) \in D\}$$
Then $Y = \zeta(X)$ *is the logic program* $T \cup G' \cup IC \cup A' \cup D'$.

• **Grounding:** The resulting program is now grounded and simplified by LPARSE, as indicated by the function χ below.

Definition 3.2 (χ). *Let* Y *be a logic program (possibly containing domain declarations). Then* $Z = \chi(Y)$ *is the ground logic program obtained by running* LPARSE *on* Y.

• **Post-processing:** After running LPARSE, any surviving ground instances of the negative cycles introduced by the pre-processing phase are converted back into explicit abducibles. This is done by replacing each cycle of the form $a \leftarrow \neg a^*$ and $a^* \leftarrow \neg a$ with a bridging clause $a \leftarrow a^\dagger$ and making the ground atom $a^\dagger$ abducible. It is interesting to note that this re-conversion is not strictly necessary as any negations will be replaced by abducibles and integrity constraints in the next step. But, it is more efficient to exploit the fact that each cycle represents a single abducible in order to avoid the unnecessary introduction of one additional abducible and two additional integrity constraints. The bridging clauses allow to distinguish instances of an atom whose truth is abduced from those whose truth is implied. In addition, any resulting ground instances of the goal clause are added to the theory T_1 and their head atom $goal$ becomes the new propositional goal. This process is performed by the function μ below.

Definition 3.3 (μ). *Let* Z *be any ground logic program and*
$$
\begin{array}{rcl}
Z_{IC} & = & \{\leftarrow L_1 \wedge \ldots \wedge L_n \in Z\} \\
Z_G & = & \{goal \leftarrow L_1 \wedge \ldots \wedge L_n \in Z\} \\
Z_A & = & \{a \leftarrow \neg a^* \in Z\} \cup \{a^* \leftarrow \neg a \in Z\} \\
Z_T & = & Z \setminus (Z_{IC} \cup Z_G \cup Z_A)
\end{array}
$$
define

Then $W = \mu(Z)$ *is the propositional abductive context* $(T_1, G_1, IC_1, A_1, D_1)$ *such that*
$$
\begin{array}{rcl}
T_1 & = & Z_T \cup Z_G \cup \{a \leftarrow a^\dagger \,|\, \{a \leftarrow \neg a^*\} \in Z_A\} \\
G_1 & = & \{goal\} \\
IC_1 & = & Z_{IC} \\
A_1 & = & \{a^\dagger \,|\, \{a \leftarrow \neg a^*\} \in Z_A\} \\
D_1 & = & \emptyset
\end{array}
$$

$$T_1 = \left\{\begin{array}{l} car(c1) \\ car(c2) \\ car(c3) \\ wont\_start(c1) \leftarrow flat\_battery(c1) \\ wont\_start(c2) \leftarrow flat\_battery(c2) \\ wont\_start(c3) \leftarrow flat\_battery(c3) \\ wont\_start(c1) \leftarrow no\_fuel(c1) \\ wont\_start(c2) \leftarrow no\_fuel(c2) \\ wont\_start(c3) \leftarrow no\_fuel(c3) \\ \boxed{wont\_start(c1) \leftarrow spark\_plugs\_dirty(c1)} \\ \boxed{wont\_start(c2) \leftarrow spark\_plugs\_dirty(c2)} \\ wont\_start(c3)\boxed{\leftarrow spark\_plugs\_dirty(c3)} \\ spark\_plugs\_dirty(c3) \\ fuel\_gauge\_empty(c2) \\ head\_lights\_work(c1) \end{array}\right\} \cup \left\{\begin{array}{l} broken\_gauge(c1) \leftarrow broken\_gauge^\dagger(c1) \\ broken\_gauge(c2) \leftarrow broken\_gauge^\dagger(c2) \\ broken\_gauge(c3) \leftarrow broken\_gauge^\dagger(c3) \\ no\_fuel(c1) \leftarrow no\_fuel^\dagger(c1) \\ no\_fuel(c2) \leftarrow no\_fuel^\dagger(c2) \\ no\_fuel(c3) \leftarrow no\_fuel^\dagger(c3) \\ flat\_battery(c1) \leftarrow flat\_battery^\dagger(c1) \\ flat\_battery(c2) \leftarrow flat\_battery^\dagger(c2) \\ flat\_battery(c3) \leftarrow flat\_battery^\dagger(c3) \end{array}\right\}$$

$$\cup \left\{\begin{array}{l} goal \leftarrow wont\_start(c1)\boxed{\wedge \neg spark\_plugs\_dirty(c1)} \\ goal \leftarrow wont\_start(c2)\boxed{\wedge \neg spark\_plugs\_dirty(c2)} \\ \boxed{goal \leftarrow wont\_start(c3) \wedge \neg spark\_plugs\_dirty(c3)} \end{array}\right\}$$

$$IC_1 = \left\{\begin{array}{l} \leftarrow flat\_battery(c1)\boxed{\wedge head\_lights\_work(c1)} \\ \boxed{\leftarrow flat\_battery(c2) \wedge head\_lights\_work(c2)} \\ \boxed{\leftarrow flat\_battery(c3) \wedge head\_lights\_work(c3)} \\ \leftarrow no\_fuel(c1) \wedge broken\_gauge^*(c1)\boxed{\wedge \neg fuel\_gauge\_empty(c1)} \\ \boxed{\leftarrow no\_fuel(c2) \wedge \neg broken\_gauge(c2) \wedge \neg fuel\_gauge\_empty(c2)} \\ \leftarrow no\_fuel(c3) \wedge broken\_gauge^*(c3)\boxed{\wedge \neg fuel\_gauge\_empty(c3)} \\ \leftarrow broken\_gauge^*(c1) \wedge broken\_gauge^*(c1) \\ \leftarrow \neg broken\_gauge^*(c1) \wedge \neg broken\_gauge^*(c1) \\ \leftarrow broken\_gauge^*(c3) \wedge broken\_gauge^*(c3) \\ \leftarrow \neg broken\_gauge^*(c3) \wedge \neg broken\_gauge^*(c3) \end{array}\right\}$$

$$G_1 = \{\ goal\ \}$$

$$A_1 = \{\ no\_fuel^\dagger(c1), flat\_battery^\dagger(c1), broken\_gauge^\dagger(c1), broken\_gauge^*(c1)\ \}$$
$$\cup\ \{\ no\_fuel^\dagger(c2), flat\_battery^\dagger(c2), broken\_gauge^\dagger(c2)\ \}$$
$$\cup\ \{\ no\_fuel^\dagger(c3), flat\_battery^\dagger(c3), broken\_gauge^\dagger(c3), broken\_gauge^*(c3)\ \}$$

Figure 2: Propositional abductive context for the *classic cars* example (where boxes identify clauses and literals removed by the grounding process)

Example 3.1. *The result of applying the above transformation to the abductive context of Example 2.1 is shown in Figure 2 — which, for convenience, also shows, within boxes, the clauses and literals removed by* LPARSE. *Note that the omitted rules can never be true and the omitted constraints can never be violated.*

STEP 2: Obtaining a Definite Theory

To avoid potential problems resulting from the unrestricted use of negation in the theory (which could result in networks where some states have no fixpoints) we use a well-known equivalence between negation and abduction which allows us to treat each negative literal $\neg p(t_1, \ldots, t_n)$ as an abducible $p^*(t_1, \ldots, t_n)$ subject to integrity constraints stating that an atom and its negation cannot both be true or both be false in the same model. Intuitively, we are free to assume the falsity of any atom if it is consistent to do so. As formalised by the function η, below, we add such constraints for all ground atoms appearing negatively in the context. For convenience, this function uses some notation for representing the positive form of an expression obtained by replacing all negative literals by their abducible proxies.

Definition 3.4 (*). Let $C = H \leftarrow B_1 \wedge \ldots \wedge B_n \wedge \neg C_1 \wedge \ldots \wedge \neg C_m$ be a clause. Then C^* is the (definite) clause $C^* = H \leftarrow B_1 \wedge \ldots \wedge B_n \wedge C_1^* \wedge \ldots \wedge C_m^*$.

Definition 3.5 (η). Let $W = (T_1, G_1, IC_1, A_1, D_1)$ be a propositional abductive context. Then $\eta(W)$ is the abductive context, $(T_2, G_2, IC_2, A_2, D_2)$ such that

$$
\begin{aligned}
T_2 &= \{C^* \,|\, C \in T_1\} \\
G_2 &= \{C^* \,|\, C \in G_1\} \\
IC_2 &= \{C^* \,|\, C \in IC_1\} \\
&\cup \left\{ \begin{array}{c} \leftarrow a \wedge a^* \\ \leftarrow \neg a \wedge \neg a^* \end{array} \,\middle|\, a \in \mathcal{A}^-_{T_1 \cup IC_1 \cup G_1} \right\} \\
A_2 &= A \cup \{a^* \,|\, a \in \mathcal{A}^-_{T_1 \cup IC_1 \cup G_1}\} \\
D_2 &= D_1
\end{aligned}
$$

STEP 3: Adding Clauses for Abduction

We now add some extra clauses that, when translated into a neural network, will perform a systematic search through the space of abductive solutions.

As in the method of Ray & Garcez [9], we use a binary counter whose output determines which abducibles are assumed and which are not. In earlier work, the output of this counter followed a simple binary sequence: 000, 001, 010, 011, 100, etc. But, in this paper, we wish to keep the number of abducibles as small as possible for as long as possible. Thus we use a modified search sequence (called a Banker's sequence) where the number of true bits increases monotonically: 000, 100, 010, 001, 110, etc.

In fact, we use two cascaded Banker's sequences, one for pure abducibles $a^\dagger \in A_2$ and one for (the negations of) the negative literals $a^* \in A_2$. The clauses which define these counters are formalised in the theory C below. Firstly, C^{1,n_1} denotes a counter with n_1 bits $a^1_{n_1}, \ldots, a^1_1$ bits corresponding to the abducibles $a^\dagger_{n_1}, \ldots, a^\dagger_1$. Secondly, C^{2,n_2} denotes a counter with n_2 bits $a^2_{n_2}, \ldots, a^2_1$ bits corresponding to the abducibles $a^*_{n_2}, \ldots, a^*_1$. Each counter i has $HOLD^i$

and $MOVE^i$ conditions that determine its behaviour when a global $next$ signal is applied to the network. If $HOLD^i$ is true, it will keep its current value. If $MOVE^i$ is true, it will advance to the next value in the Banker's sequence. If neither is true, the counter will reset to zero.

The definition of each counter $C^{i,n}$ is split into five macros $C^{i,n}_1$ to $C^{i,n}_5$ that together provide a logical specification of the counter state transition table. In brief, b^i_j is true iff the jth bit of counter i is true; c^i_j is true iff b^i_j is true and b^i_{j-1} is false; d^i_j is true iff c^i_j is true and c^i_k is false for all $k < j$; e^i_j is true iff b^i_j is false and b^i_{j-1} is true; and all^i is true iff all the bits of counter i are true.

$$ C = C^{1,n_1} \cup C^{2,n_2} \quad \text{where} $$

$$ C^{i,n} = \bigcup_{k=1}^{5} C^{i,n}_k \quad \text{such that} $$

$$ C^{i,n}_1 = \bigcup_{k=0}^{n} \left\{ \begin{array}{l} b^i_k \leftarrow \bigwedge_{j=1}^{n} \neg d^i_j \wedge b^i_{n-k} \wedge MOVE^i \\ b^i_k \leftarrow \bigwedge_{j=1}^{n} \neg d^i_j \wedge e^i_{n-k} \wedge MOVE^i \\ b^i_k \leftarrow b^i_k \wedge \neg next \\ b^i_k \leftarrow b^i_k \wedge HOLD^i \end{array} \right\} $$

$$ C^{i,n}_2 = \left\{ \begin{array}{l} b^i_n \leftarrow \bigwedge_{j=0}^{n} \neg b^i_j \wedge MOVE^i \\ all^i \leftarrow \bigwedge_{j=0}^{n} b^i_j \end{array} \right\} $$

$$ C^{i,n}_3 = \bigcup_{k=2}^{n} \bigcup_{j=0}^{k-1} \{ b^i_{k-2-j} \leftarrow b^i_j \wedge d^i_k \wedge MOVE^i \} $$

$$ C^{i,n}_4 = \bigcup_{k=1}^{n} \left\{ \begin{array}{l} e^i_k \leftarrow \neg b^i_k \wedge b^i_{k-1} \\ c^i_k \leftarrow b^i_k \wedge \neg b^i_{k-1} \\ d^i_k \leftarrow \bigwedge_{j=1}^{k-1} \neg c^i_j \wedge c^i_k \\ b^i_{k-1} \leftarrow d^i_k \wedge MOVE^i \\ a^i_k \leftarrow ABD^i_k \end{array} \right\} $$

$$ C^{i,n}_5 = \bigcup_{k=1}^{n} \bigcup_{j=k+1}^{n} \{ b^i_j \leftarrow b^i_j \wedge d^i_k \wedge MOVE^i \} $$

$$
\begin{aligned}
MOVE^1 &= next \wedge all^2 \\
MOVE^2 &= next \wedge \neg all^2 \\
HOLD^1 &= \neg all^2 \\
HOLD^2 &= false \\
ABD^1_k &= b^1_k \text{ for all } k \\
ABD^2_k &= \neg b^2_k \text{ for all } k
\end{aligned}
$$

The 2nd counter increments until its maximum is reached, whence it resets. The 1st counter holds until the 2nd resets, whence it increments. To preferentially *minimise* the number of assumed abducibles the outputs $a^1_{n_1} \ldots a^1_1$ of the 1st counter are obtained by copying the corresponding bits $b^1_{n_1} \ldots b^1_1$. To preferentially *maximise* the number of atoms assumed false, the outputs $a^2_{n_2} \ldots a^2_1$ of the 2nd counter are obtained by negating the corresponding bits $b^2_{n_2} \ldots b^2_1$.

6

As in [9], we advance the counter if a fixpoint fp is reached that violates the goal $goal$ or integrity constraints ic. Previously, this was done by a clock with a constant period determined by worst case propagation delay through the network generated by the program. For efficiency, we now add clauses to detect as soon as a fixpoint is reached. These are given by the theory S below, which uses four new propositions $z', z^+, z^-, z^{\$}$ to compare the current state of each atom z (in a set Z of atoms) with their previous states.

In brief, z' denotes the previous state of z; z^+ is true iff the current and previous states are both true; z^- is true iff they are both false; and $z^{\$}$ is true iff they are the same. The $w1$-$w4$ are wait signals that just give the counters enough time to compute their next value before the network decides whether the current abducibles are a solution (in which case the signal $soln$ is activated) or whether another set of abducibles are needed (in which case the signal $next$ is activated). fp indicates when a fixpoint has been reached. The signal $done$ indicates when all solutions are exhausted.

$$S \quad = \quad S_1^Z \cup S_2^Z \cup S_3^Z \quad \text{where}$$

$$S_1^Z \quad = \quad \bigcup_{z \in Z} \left\{ \begin{array}{l} z' \leftarrow z \\ z^+ \leftarrow z \wedge z' \\ z^- \leftarrow \neg z \wedge \neg z' \\ z^{\$} \leftarrow z^+ \\ z^{\$} \leftarrow z^- \end{array} \right\}$$

$$S_2^Z \quad = \quad \left\{ \begin{array}{l} w_1 \leftarrow next \\ w_2 \leftarrow w_1 \\ w_3 \leftarrow w_2 \\ w_4 \leftarrow w_3 \\ \\ fp \leftarrow \bigwedge_{z \in Z} z^{\$} \wedge \bigwedge_{i=1}^{4} \neg w_i \wedge \neg next \end{array} \right\}$$

$$S_3^Z \quad = \quad \left\{ \begin{array}{l} next \leftarrow \neg next \wedge fp \wedge ic \\ next \leftarrow \neg next \wedge \neg goal \wedge fp \\ soln \leftarrow \neg next \wedge goal \wedge fp \wedge \neg ic \\ done \leftarrow all^1 \wedge all^2 \wedge fp \end{array} \right\}$$

To properly specify these additional clauses, it is necessary to state the parameters n_1, n_2 and Z in C and S. This is done by the function δ below, which defines n_1 as the number of abducibles $a^\dagger$, n_2 as the number of negations a^*, and Z as the set of all atoms appearing (positively or negatively) in the context. In addition, this function adds a special atom ic into the head of every integrity constraint, and adds the literal $\neg next$ into the body of every rule to ensure the network is fully relaxed whenever a new solution is attempted. The resulting network architecture is summarised in Figure 3.

Definition 3.6 (δ). *Let $(T_2, G_2, IC_2, A_2, D_2)$ be a propositional abductive context with $G_2 = \{goal\}$ and $D_2 = \emptyset$. Now let $n_1 = |\{a^\dagger \in A_2\}|$, let $n_2 = |\{a^* \in A_2\}|$, let $Z = A_{T_2 \cup G_2 \cup IC_2}^+ \cup A_{T_2 \cup G_2 \cup IC_2}^-$, and let R be the program*

$$R \quad = \quad \left\{ \begin{array}{l} H \leftarrow B_1 \wedge \ldots \wedge B_n \wedge \neg next \\ | H \leftarrow B_1 \wedge \ldots \wedge B_n \in T_2 \end{array} \right\}$$

$$\cup \quad \left\{ \begin{array}{l} ic \leftarrow B_1 \wedge \ldots \wedge B_n \wedge \neg next \\ | \leftarrow B_1 \wedge \ldots \wedge B_n \in IC_2 \end{array} \right\}$$

Then, $P = \delta(T_2, G_2, IC_2, A_2)$ is the program $R \cup C \cup S$.

STEP 4: Obtaining a neural network

Now we translate the clauses obtained so far into a network using the method of Ray & Garcez [9], recalled below, which is based on well-known neurosymbolic techniques.

Definition 3.7 (θ). *Let P be a logic program, then $\theta(P)$ is the network (N, E) such that*

$$N = \bigcup_{r \in \mathcal{G}_P} \left\{ \begin{array}{l} r, H, B_1, \ldots, B_n, C_1, \ldots, C_m \\ | r = H \leftarrow B_1 \wedge \ldots \wedge B_n \wedge \neg C_1 \wedge \ldots \wedge \neg C_m \end{array} \right\}$$

$$E = \bigcup_{r \in \mathcal{G}_P} \left\{ \begin{array}{l} (r, H), (B_1, r), \ldots, (B_n, r), (C_1, r), \ldots, (C_m, r) \\ | r = H \leftarrow B_1 \wedge \ldots \wedge B_n \wedge \neg C_1 \wedge \ldots \wedge \neg C_m \end{array} \right\}$$

and for all $r = H \leftarrow B_1 \wedge \ldots \wedge B_n \wedge \neg C_1 \wedge \ldots \wedge \neg C_m \in \mathcal{G}_P$

$$t(r) = n - 1/2 \qquad \begin{array}{ll} t(H) = 1/2 & w(r, H) = 1 \\ t(B_i) = 1/2 & w(B_i, r) = 1 \\ t(C_j) = 1/2 & w(C_j, r) = -1 \end{array}$$

STEP 5: Computing Abductive Solutions

In the last step, we compute the answers with the neural network. Starting from a relaxed network, we activate the node $next$. Then, on every other subsequent state, we check whether the network has reached a fixpoint where $soln$ activated. If it has, then the current abducibles are recorded and we see which of the $goal$ rules is activated in order to extract the successful ground instances of the query. If more solutions are required then we activate $next$ in order to seek the next answer. We stop this process when a suitable answer is computed or $done$ is true, meaning the entire search space has been explored.

4 Conclusion and future work

We believe that the integration of abductive and inductive inference is necessary to develop improved learning and reasoning approaches. In this paper we presented a neurosymbolic approach for first-order abductive inference. Unlike most other such approaches, we do not impose any restrictions on the use of negation or recursion. However, the ASP grounding procedure assumes that all variables have finite domains, which limits the use of function symbols. We have implemented our method using the C++ programming language and applied it to the abductive context shown in Figure 1. In this way, we correctly obtained the minimal solutions followed by all of the other non-minimal solutions. Even in this simple example we note that the search time increases significantly if a naive grounding is used in place of our more efficient ASP approach. We also note that the time needed to compute all minimal solutions is significantly less than the time needed to compute all (non-minimal) solutions. In this sense we have improved existing methods of neurosymbolic abduction. However, the complexity of the approach is still exponential in the number of abducibles, which limits its practical use. In future work, we will compare our neural approach with related symbolic approaches and investigate ways of parallelising our method.
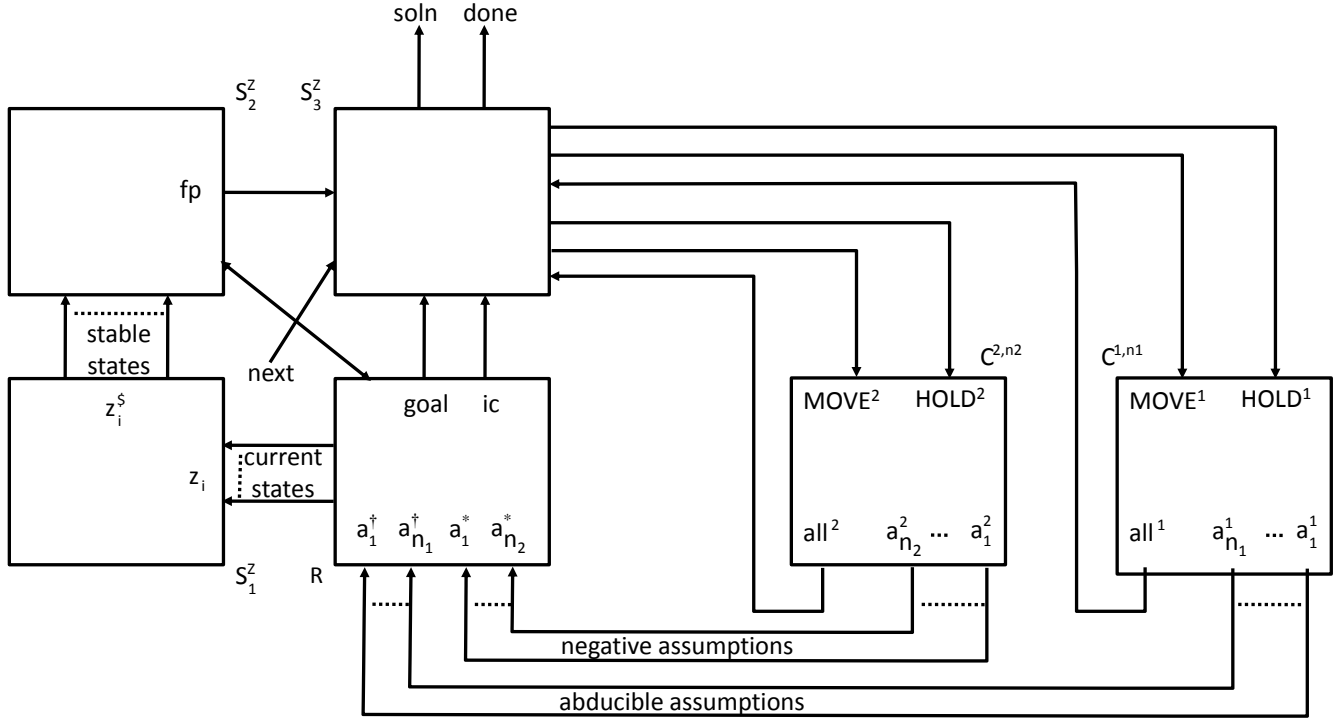
Figure 3: Network architecture showing the logical interactions between the counters C^{1,n_1}, C^{2,n_2} control blocks S_1^Z, S_2^Z, S_3^Z, and the ground program R

References

[1] A. Abdelbar, M. El-Hemaly, E. Andrews, and D. Wunsch II. Recurrent neural networks with backtrack-points and negative reinforcement applied to cost-based abduction. *Neural Networks*, 18(5-6):755–764, 2005.

[2] B. Ayeb, S. Wang, and J. Ge. A Unified Model For Neural Based Abduction. *IEEE Transactions on Systems, Man and Cybernetics*, 28(4):408–425, 1998.

[3] A. Goel and J. Ramanujam. A Neural Architecture for a Class of Abduction Problems. *IEEE Transactions on Systems, Man and Cybernetics*, 26(6):854–860, 1996.

[4] A.C. Kakas, R.A. Kowalski, and F. Toni. Abductive Logic Programming. *Journal of Logic and Computation*, 2(6):719–770, 1992.

[5] A.C. Kakas and P. Mancarella. Generalized Stable Models: a Semantics for Abduction. In *Proceedings of the 9th European Conference on Artificial Intelligence*, pages 385–391. Pitman, 1990.

[6] V. Lifschitz. Answer set programming and plan generation. *Artificial Intelligence*, 138(1-2):39–54, 2002.

[7] P. Lima. Logical Abduction and Prediction of Unit Clauses in Symmetric Hopfield Networks. In *Artificial Neural Networks, 2*, volume 1, pages 721–725. Elsevier, 1992.

[8] J. Medina, E. Mérida-Casermeiro, and M. Ojeda-Aciego. A neural approach to abductive multiad-joint reasoning. In *Proceedings of the 10th International Conference on Artificial Intelligence: Methodology, Systems, and Applications*, volume 2443 of *Lecture Notes in Computer Science*, pages 213–222, 2002.

[9] O. Ray and A. d'Avila Garcez. Towards the integration of abduction and induction in artificial neural networks. In *Proceedings of the ECAI'06 Workshop on Neural-Symbolic Learning and Reasoning*, pages 41–46, 2006.

[10] J. Reggia, Y. Peng, and S. Tuhrim. A Connectionist Approach to Diagnostic Problem-Solving Using Causal Networks. *Information Sciences*, 70:27–48, 1993.

[11] P. Simons, I. Niemelä, and T. Soininen. Extending and implementing the stable model semantics. *Artificial Intelligece*, 138(1-2):181–234, 2002.

[12] R. Vingrálek. A connectionist approach to finding stable models and other structures in nonmonotonic reasoning. In *Proceedings of the Second International Workshop on Logic Programming and Non-Monotonic Reasoning*, pages 60–81. MIT Press, 1993.

[13] C. Zhang and Y. Xu. A Neural Network Model for Diagnostic Problem Solving with Causal Chaining. *Neural Networks and Advanced Control Strategies*, 54:87–92, 1999.