Towards the integration of abduction and induction in artificial neural networks

Oliver Ray[1] and **Artur Garcez**[2]

Abstract. This paper presents a method for realising abduction in artificial neural networks (ANNs) by generalising existing neuro-symbolic approaches from normal logic programs to abductive logic programs (ALPs) in order to provide a more expressive formalism for representing and reasoning about partial knowledge and integrity constraints. The aim is to develop a massively-parallel methodology for abduction that can also be integrated with connectionist learning approaches to offer a finer degree of control over which assumptions can and cannot be made in learning. Existing methods for abduction in neural networks are not well suited to this task as they only apply to a restricted a class of abduction problems or they do not adequately address the issues of local minima and multiple solutions. This paper proposes an algorithm for translating ALPs into ANNs whereby no restrictions are imposed on the underlying programs and, if required, the network can systematically compute all abductive explanations or provide a guarantee when none exist. Moreover, since the topology of the network mirrors the structure of the program, it can be acquired and revised by standard neuro-symbolic training techniques and can also be exploited to impose a preference on the order in which the solutions are found.

1 Introduction

Neuro-symbolic integration [9, 6] aims to combine the respective benefits of artivicial neural networks (ANNs) and logic programming by providing practical methods of learning with declarative knowledge representations. This is achieved by translating logic programs into neural networks; either to provide an initial network which can be trained on further data using techniques such as back-propagation, as in [20], or to compute the consequences of the program under the stable model semantics by means of massively parallel deduction, as in [8]. But, state-of-the art approaches such as [20, 8, 6] only apply to logic programs with unique stable models and are not particularly well suited for representing and reasoning about partial knowledge that is inherent in learning. This limitation motivates the study of more powerful formalisms for expressing uncertainty and handling programs with more than one stable model.

Abductive logic programs (ALPs) [10] are an extension of logic programs that are more appropriate for representing and reasoning about partially complete knowledge. In particular, they allow the truth or falsity of some ground literals, known as *abducibles*, to be left unspecified subject to given integrity constraints. In contrast to normal logic programming, abductive proof procedures are free to assume any consistent set of abducibles when solving a goal. Thus, abduction does not merely determine whether a goal follows from a program, but computes a set of assumptions that, when aded to the program, ensure the goal succeeds. Each set of abducibles is called an *abductive explanation* and represents an extension of the program that is referred to as a *generalised stable model* [11]. By extending the program in this way, abduction can extrapolate potentially useful assumptions from partially complete theories.

The incompleteness of knowledge inherent in learning suggests inductive techniques may benefit from a facility for abduction. This claim is supported by logic-based machine learning systems show that abduction and induction can be combined to achieve superior reasoning capabilities, as evidenced for example in [15, 12, 4]. The benefits offered by neural networks over logical approaches in terms of noise-tolerance and massive-parallelism provide an even greater incentive to investigate the integration of abduction and induction at the sub-symbolic level. But, existing methods for abduction in neural networks are not well suited to this task as they only apply to a very restricted a class of abduction problems whose expressivity is limited to definite acyclic programs [7, 18, 2, 22] or they do not adequately address the issues of avoiding local minima and computing multiple solutions [13, 21, 14, 1].

This paper presents a novel methodology for abduction in neural networks by generalising existing neuro-symbolic approaches from normal logic programs to abductive logic programs. In particular, an algorithm is proposed for translating ALPs into ANNs such that the fixpoints of the network represent the generalised stable models of the program. The translation is introduced in three steps. First, a function θ is defined that maps logic programs into neural networks by adapting existing neuro-symbolic encoding methods. Second, a function ϕ is defined that maps acyclic abductive logic programs into neural networks by extending the program with some additonal clauses for abduction. Third, a function ψ is defined that maps any abductive logic program into a neural network using a simple pre-processing transformation which allows positive and negative cycles to be uniformly handled through abduction.

The paper is structured as follows. Section 2 recalls some basic notation and terminology relating to neural networks (with binary threshold neurons) and logic programs before introducing the task of abductive logic programming. Section 3 defines the functions θ and ϕ and shows how the networks they produce can compute the generalised stable models of acyclic abductive logic programs. Section 4 shows how the approach is extended to abductive logic programs with positive and negative cycles. The paper concludes with a summary and directions for future work. All of the examples have been implemented and tested using the *BrainBox* neural network simulator [5] and the configuration files may be downloaded from [16].

[1] Imperial College London, UK, email: or@doc.ic.ac.uk
[2] City University London, UK, email: aag@soi.city.ac.uk

2 Background

Threshold Neural Networks A *neural network*, or just *network* hereafter, is a graph (N, E) whose nodes N are called *neurons* and whose edges $E \subseteq N \times N$ are called *connections*. Each neuron is $n \in N$ labeled with a number $t(n)$ called its *threshold* and each connection $(n, m) \in E$ is labeled with a 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 state is 1 and it is said to be *inactive* if its state is 0. For each state s 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. The *least fixpoint* of the network, if it exists, is the fixpoint reached by repeatedly computing successor states starting from an initially relaxed network.

Normal Logic Programs A *rule* is an expression of the form $H \leftarrow B_1, \ldots, B_n, \neg C_1, \ldots, \neg C_m$, where the H, B_i and C_j are all atoms. 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 $H \leftarrow B_1, \ldots, B_n$. A rule with no body literals at all is called a *fact* and written H. A *normal logic program*, or just *program* hereafter, is a set of rules. 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 ground instances of the clauses in P. In additon, $\mathcal{A}_P^+$ and $\mathcal{A}_P^-$ denote, respectively, the sets of ground atoms that occur positively and negatively in $\mathcal{G}_P$; and $\mathcal{D}_P$ (the *dependency graph* of P) is the directed graph with signed edges whose nodes are the atoms in $\mathcal{A}_P^+ \cup \mathcal{A}_P^-$ and where there is a positive (resp. negative) edge from a to b iff there is a clause in $\mathcal{G}_P$ with a in the head and b occurring positively (resp. negatively) in the body. A cycle in $\mathcal{D}_P$ is *positive* if has no negative edges and is *negative* otherwise. A program P is said to be *acyclic* iff $\mathcal{D}_P$ contains no (positive or negative) cycles. 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 satisfied in I, and by deleting all of the negative literals in the remaining rules.

Abductive Logic Programs An *abductive logic program* [10] is a triple (T, IC, A) where T is a program (the *theory*), IC is a set of rules (*integrity constraints*) with the atom $\bot$ denoting logical falsity in their head, and A is a set of ground atoms (*aducibles*). Given a set G of ground atoms (the *goals*), the task of ALP is to compute a set $\Delta \subseteq A$ of abducibles such that G and IC are satisfied in some stable model of $T \cup \Delta$. In the terminology of [11], the goal G is said to be satisfied in the *generalised stable model* $T(\Delta)$; and Δ is said to be an *abductive explanation* of G with respect to T, IC and A.

To select between alternative explanations, additional preference criteria are often utilised. Two popular desiderata are the properties of *minimality* and *basicality*. Formally, an explanation Δ of G with respect to (T, IC, A) is *minimal* iff there is no $\Delta' \subset \Delta$ such that Δ' is an explanation of G, and is *basic* iff there is no $\Delta' \not\supseteq \Delta$ such that Δ' is an explanation of Δ. Intuitively, an explanation Δ is minimal if none of its atoms are redundant and is basic if none of its atoms can be further explained. For convenience the four inputs (T, G, IC, A) are collectively called an *abductive context*. A context is said to be definite, acyclic, etc, iff the theory T is definite, acyclic, etc.

Definition 2.1 (Abductive Context)**.** *An* abductive context *is a four-tuple (T, G, IC, A) where T is set of rules, G and A are sets of ground atoms, and IC is a set of integrity constraints.*

Example 2.1. *Consider the abductive context below describing an old car. The theory states that the car wont start if its battery is flat or if fuel tank is empty; that the battery is flat on wet days; that the car will overheat if its fan is broken; and that the lights of the car are on. The integrity constraint states that the lights cannot be on at the same time the battery is flat. The goal to that must be proved is $wont\_start$. The abducibles which may be assumed are wet_day, fan_broke, $fuel\_empty$.*

$$
\begin{aligned}
T &= \left\{ \begin{array}{l} wont\_start \leftarrow battery\_flat \\ wont\_start \leftarrow fuel\_empty \\ battery\_flat \leftarrow wet\_day \\ overheat \leftarrow fan\_broke \\ lights\_on \end{array} \right\} \\
G &= \{\ wont\_start\ \} \\
IC &= \{\ \bot \leftarrow battery\_flat, lights\_on\ \} \\
A &= \{\ fan\_broke, fuel\_empty, wet\_day\ \}
\end{aligned}
$$

There are two abuctive explanations of this context: $\Delta_1 = \{fuel\_empty\}$ and $\Delta_2 = \{fan\_broke, fuel\_empty\}$. The former is both minimal and basic, while the latter is neither minimal nor basic. These are the only correct explanations since all other sets of abducibles fail to satisfy either the goal or the integrity constraints.

3 Neural Network Abuction: Simple Case

This section presents a first methodology for realising abduction in neural networks by defining a translation which maps definite acyclic abductive logic programs into networks whose fixpoints correspond to the generalised stable models of the program. The initial restriction to acyclic programs is merely to simplify the presentation of the key ideas and is immediately lifted in the next section through some simple syntactic preprocessing of the inputs.

The proposed methodology builds upon existing neuro-symbolic techniques for transforming logic programs into neural networks and is easily adapted to suit any choice of encoding. In this paper, for ease of exposition, we introduce a translation based on multi-layer threshold networks, which is a slight variation of the approaches in [20, 8] and is easily generalised to the recurrent sigmoidal networks using the techniques in [6].

As formalised in Definition 3.1 below, the neural network $\theta(P)$ corresponding to a normal program P is obtained from the ground expansion $\mathcal{G}_P$ of P in the following way. For each rule $r = H \leftarrow B_1, \ldots, B_n, \neg C_1, \ldots, \neg C_m$ in G_P, add to the network

- a node with threshold $n - 1/2$ to represent the rule r
- a node with threshold $1/2$ for each atom H,B_i,C_j in the rule (which has not already been added through an earlier rule)
- an edge with weight 1 from r to the head atom H
- an edge with weight 1 from each unnegated body atom B_i to r
- an edge with weight -1 from each negated body atom C_j to r

Definition 3.1 (θ)**.** *If P is a program, then $\theta(P)$ is the network (N, E) such that*

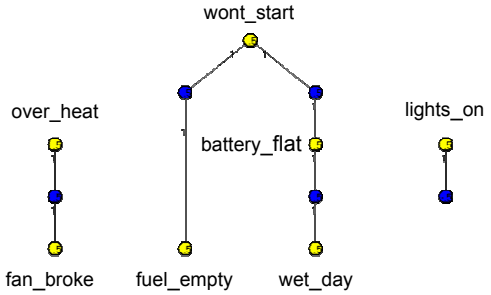
$$N = \bigcup_{r \in \mathcal{G}_P} \left\{ \begin{array}{l} r, H, B_1, \ldots, B_n, C_1, \ldots, C_m \\ \mid r = H \leftarrow B_1, \ldots, B_n, \neg C_1, \ldots, \neg C_m \end{array} \right\}$$

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

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

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

Example 3.1. *If P is the program T in Example 2.1 above, then $\theta(P)$ is the network below. For convenience, nodes representing atoms are lightly shaded and are annotated with the name of the atom, while nodes corresponding to the rules in the program are darkly shaded. The threshold of each neuron and the weight of each connection are also shown.*



The translation algorithm above produces a neural network encoding of a given program. In common with other approaches, it can be shown that if the program is acyclic then the least fixpoint of the network exists and corresponds the unique stable model of the program. But, to perform abduction, this procedure must be supplemented with some way of representing goals, integrity constraints and some means of activating different combinations of abducibles. As formalised in Definition 3.2 below, the required abductive machinery can be obtained by transforming an abductive context (T, G, IC, A) into a logic program with some clauses (T', G', IC', A') representing the context and others (C, K, L) representing some additional logic to ensure the fixpoints of the network correspond to the generalised stable models of the theory.

Definition 3.2 (ϕ)**.** *Let (T, G, IC, A) be an abductive context. Let N be the number of abducibles in A. Let P be the length of the longest directed path in $\mathcal{D}_T$ with no repeated nodes. Let M be the smallest integer greater than or equal to $\frac{1}{2}(P + 2N + 3)$. Let $goal$, ic, $soln$, $next$, $done$, $sync$, $nogood$, $hold$, a_i, b_i, c_i, d_i and k_j be propositions not appearing in (T, G, IC, A) for all $0 \leq i \leq N$ and for all $0 \leq j \leq M$. Then $\phi(T, G, IC, A)$ is the network $\theta(T' \cup G' \cup IC' \cup A' \cup C \cup K \cup L)$ where*

$$\begin{array}{lcl} T' & = & T \\ G' & = & \{goal \leftarrow B_1, \ldots, B_n \mid \{B_1, \ldots B_n\} = G\} \\ IC' & = & \{ic \leftarrow L_1, \ldots, L_m \mid \bot \leftarrow L_1, \ldots, L_m \in IC\} \\ A' & = & \{A_i \leftarrow a_i \mid A_i \in A\} \end{array}$$

$$C = \bigcup_{i=1}^{N} \left\{ \begin{array}{l} a_i \leftarrow a_i, \neg c_i \\ a_i \leftarrow d_i \\ b_i \leftarrow a_i \\ c_i \leftarrow b_{i-1}, \neg a_{i-1}, a_i \\ d_i \leftarrow b_{i-1}, \neg a_{i-1}, \neg a_i \end{array} \right\} \cup \left\{ \begin{array}{l} b_0 \leftarrow next \\ done \leftarrow b_N, \neg a_N \\ done \leftarrow done \end{array} \right\}$$

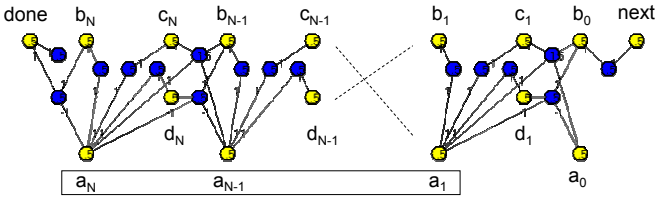
$$K = \bigcup_{i=1}^{M} \{\ k_i \leftarrow k_{i-1}\ \} \cup \left\{ \begin{array}{l} k_0 \leftarrow \neg hold, \neg k_M \\ sync \leftarrow k_0, \neg k_1 \end{array} \right\}$$

$$L = \left\{ \begin{array}{l} nogood \leftarrow ic \\ nogood \leftarrow \neg goal \\ soln \leftarrow sync, \neg nogood \\ soln \leftarrow soln, \neg nogood \\ hold \leftarrow soln \\ hold \leftarrow done \\ next \leftarrow sync, nogood \end{array} \right\}$$

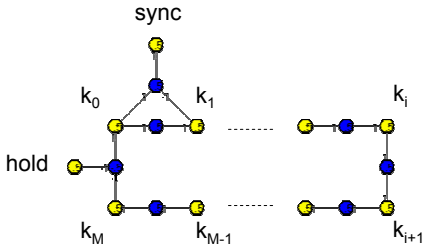
The first four theories passed to θ are a representation of the abductive context in which $goal$ is true when the goal is satisfied, ic is true when an integrity constraint is violated, and each abducible A_i is true when the corresponding atom a_i is true. More formally, T' is the theory T; G' comprises a single clause with $goal$ in the head and the atoms of G in the body; IC' is obtained by inserting ic into the head of each constraint in IC; and A' contains one clause of the form $A_i \leftarrow a_i$ for each abducible $A_i \in A = \{A_1, \ldots, A_n\}$.

The last three theories passed to θ denotes some control logic for activating different combinations of abducibles until an explanation is found or all possibilities are exhausted. When a solution is found, the network will enter a stable state in which $soln$ is activated and the a_i indicate which abducibles are contained in the explanation. If $next$ is briefly activated, the network will leave this stable state and look for the next solution. Once all possibilities have been tried, the network will enter a stable state in which $done$ is activated.

The theory C represents a binary counter whose outputs $a_N a_{N-1} \ldots a_1$ each drive one abducible. The network encoding of C is shown below. The counter advances each time the node $next$ is briefly activated and it activates the node $done$ when the counter overflows. Each bit of the counter uses for nodes, a_i, b_i, c_i and d_i, to implement a divide by two register that toggles the state of a_i whenever the state of a_{i-1} changes from on to off: with the nodes c_i and d_i signaling a_i to turn off and on, respectively.



The theory K represents a clock whose output $sync$ is used to advance the counter if the current state is not a solution. The network encoding of K is shown below. The nodes k_i form a loop where the sate of each one follows that of its predecessor; except for the first, which opposes the last. The period of the clock is proportional to the number of nodes $M + 1$, which is chosen to give the rest of the network sufficient time to stabilise between successive signals. The clock is disabled when $hold$ is active. The output $sync$ is active when k_0 is is on but k_1 is not,



The theory L represents some simple control logic that uses $sync$ to advance the counter or to suspend the clock according to whether the current abducibles are a valid explanation. $nogood$ indicates when the goal is not satisfied or one of the integrity constraints is violated. When $sync$ becomes active, either $next$ or $soln$ will be activated depending on the state of $nogood$. The first case will advance the network into the next state while the second will force the network to stabilise.

Example 3.2. *If (T, G, IC, A) is the context in Example 2.1 above, then $\phi(T, G, IC, A)$ is the network shown in Figure 1(a). The theories T', G', IC', A' are shown below. There are $N = 3$ abducibles in A and the longest simple path in $\mathcal{G}_T$ is $(wet\_day, fuel\_empty, fan\_broke)$ with length $P = 3$. The least upper bound of $\frac{1}{2}(P + 2N + 3)$ is $M = 6$.*

$$
\begin{array}{lcl}
T' & = & T \\
G' & = & \{goal \leftarrow wont\_start\} \\
IC' & = & \{ic \leftarrow battery\_flat, lights\_on\} \\
A' & = & \left\{ \begin{array}{l} fan\_broke \leftarrow a_1 \\ fuel\_empty \leftarrow a_2 \\ wet\_day \leftarrow a_3 \end{array} \right\}
\end{array}
$$

For any acyclic abductive context (T, G, IC, A) it can be shown that the least fixpoint of the network $\phi(T, G, IC, A)$ exists and is computed in a finite time. If $soln$ is active in this state, then the network represents a generalised stable model $T(\Delta)$ of T that satisfies G and IC, where Δ consists of the active abdicubles. All other solutions can be computed by asserting $next$ to force the network to search for the next stable state, which also exists and is computed in finite time. If $done$ is active, then no further solutions exist.

In the case of Example 3.2 above, it is easily verified [3] that the initially relaxed network rejects the initial hypothesis $\{fan\_broke\}$ and converges instead to the solution $\Delta_1 = \{flat\_battery\}$. If a signal is manually applied to $next$, then network will converge to the next solution $\Delta_2 = \{flat\_battery, fan\_broke\}$. If another signal is applied to $next$, then network will reject all remaining hypotheses and converge to the final $done$ state, indicating that no other solutions exist for this context.

4 Neural Network Abduction: General Case

This section shows how the methodology introduced above can be extended to abductive logic programs with cycles using a simple preprocessing transformation. But, before doing so, it is instructive to illustrate why programs with cycles are potentially problematic.

[3] The reader can use the software available from [5] with the data at [16] to run the network in Fig 1(a) by holding down ctrl-F1 to advance the network one time point and double clicking neuron 98 to apply a signal to $next$. Note that the data file contains some redundant neurons which merely serve to ensure that the connections between neurones follow the same easy-to-read layout as shown in the figure above.

First consider positive cycles by supposing that the rule $fan\_broke \leftarrow over\_heat$ is added to T in Example 2.1 and the constraint $\bot \leftarrow over\_heat$ is added to IC. The problem is that the cycle between fan_broke and $over\_heat$ introduces a memory into the network that causes a permanent violation of integrity. Once $over\_heat$ is activated by fan_broke, they both remain high, and so does ic. Hence, the one correct solution is rejected due to the memory of the violation caused by first hypothesis to be tested.

One solution to this problem is to relax the sub-networks $T*$, $G*$ $IC*$ and $A*$ after each set of abducibles is tried. This is easily realised by adding a special abducible $true$ to the body of each rule that it is always connected to the least significant bit a_1 of the counter to ensure that its state is continuously alternating with respect to the other abducibles. In this way, any self-sustaining loops are systematically deactivated before the next set of abducibles is presented to the network.

Next consider negative cycles by supposing that the rules $door\_open \leftarrow \neg door\_closed$ and $door\_closed \leftarrow \neg door\_open$ are added to T in Example 2.1 and the atom $door\_open$ is added to G. The problem is that the cycle between $door\_open$ and $door\_closed$ introduces an instability into the network that prevents any fixpoint being reached from the initially relaxed state. Instead of converging to a stable state in which $door\_open$ is active and $door\_closed$ is inactive, these atoms continually force each other to change state.

Following [3], one answer to this problem involves re-writing negative literals as positive abducibles and to implement negation through abduction. This is achieved by introducing a new abducible predicate p_i^* to denote the negation $\neg p_i$ of each predicate p_i in the context and adding integrity constraints to ensure that for any ground terms $t_1, \ldots, t_n$ exactly one of $p(t_1, \ldots, t_n)$ and $p^*(t_1, \ldots, t_n)$ is true. As shown in [11], there is a 1-1 correspondence between the generalised stable models of the original and transformed contexts.

These solutions are formalised together in Definition 4.1 below, which transforms an arbitrary context (T, G, IC, A) into a definite context (T'', G'', IC'', A'') before using ϕ to generate the network. Since the latter context is definite, there are no potential instabilities in the network caused by negative cycles; and assuming that ϕ maps $true$ to a_1, there will be no residual memory in the network caused by positive cycles. Thus, it can be shown that $\phi(T'', G'', IC'', A'')$ computes exactly the generalised stable models of (T, G, IC, A).

Definition 4.1 (ψ)**.** *Let (T, G, IC, A) be an abductive context not containing the proposition $true$. Let $R = \{p_1, \ldots, p_k\}$ be the set of predicates p_i appearing in (T, G, IC, A) and let $S = \{p_1^*, \ldots, p_k^*\}$ be a set of predicates p_i^* not appearing in (T, G, IC, A). For each atom C of the form $p_i(t_1, \ldots, t_n)$, let C^* denote the atom $p_i^*(t_1, \ldots, t_n)$. Recall that $\mathcal{A}^-_{T \cup IC}$ denotes the set of atoms that appear negated in the ground expansion of the program $T \cup IC$. Then $\psi(T, G, IC, A)$ is the network $\phi(T'', G'', IC'', A'')$ such that*

$$
\begin{array}{lcl}
T'' & = & \left\{ \begin{array}{l} H \leftarrow true, B_1, \ldots, B_n, C_1^*, \ldots, C_m^* \\ \mid H \leftarrow B_1, \ldots, B_n, \neg C_1, \ldots, \neg C_m \in T \end{array} \right\} \\
G'' & = & G \cup \{true\} \\
IC'' & = & \left\{ \begin{array}{l} \bot \leftarrow B_1, \ldots, B_n, C_1^*, \ldots, C_m^* \\ \mid \bot \leftarrow B_1, \ldots, B_n, \neg C_1, \ldots, \neg C_m \in IC \end{array} \right\} \\
 & \cup & \{ \bot \leftarrow C, C^* \mid C \in \mathcal{A}^-_{T \cup IC} \} \\
 & \cup & \{ \bot \leftarrow \neg C, \neg C^* \mid C \in \mathcal{A}^-_{T \cup IC} \} \\
A'' & = & A \cup \{true\} \cup \{C^* \mid C \in \mathcal{A}^-_{T \cup IC}\}
\end{array}
$$

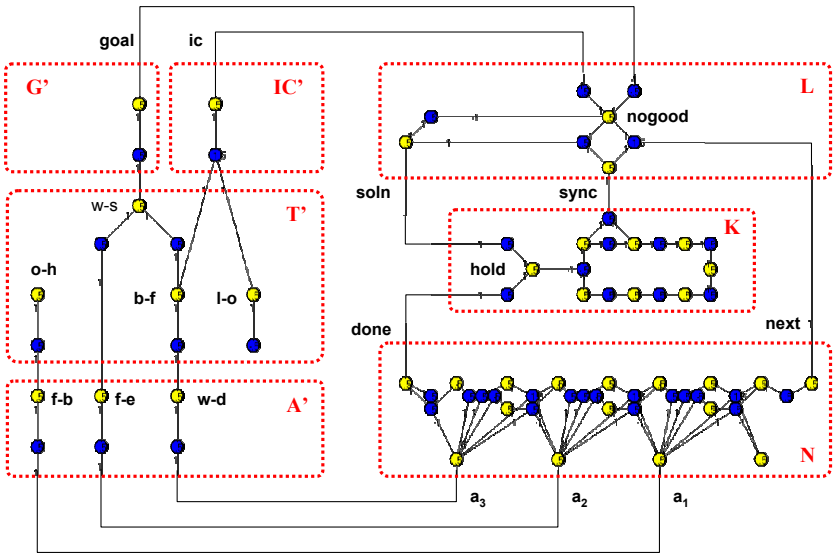


Figure 1(a) Simple Case

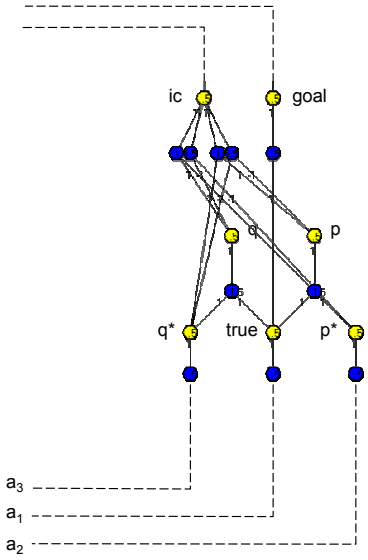


Figure 1(b) General Case

Example 4.1. *Consider the context obtained by extending Example 2.1 as described above: with one clause* $fan\_broke \leftarrow over\_heat$ *stating that the fan will break if the car overheats; with two clauses* $door\_open \leftarrow \neg door\_closed$ *and* $door\_closed \leftarrow \neg door\_open$ *stating that the car door is open if it is not closed and vive versa; with one goal* $door\_open$*; and with one constraint* $\bot \leftarrow over\_heat$*. The theories* T''*,* G''*,* IC'' *and* A'' *obtained by applying Definition 4.1 to this extended context are shown below.*

$$T'' = \left\{ \begin{array}{l} wont\_start \leftarrow true, battery\_flat \\ wont\_start \leftarrow true, fuel\_empty \\ battery\_flat \leftarrow true, wet\_day \\ overheat \leftarrow true, fan\_broke \\ fan\_broke \leftarrow true, over\_heat \\ door\_open \leftarrow true, door\_closed^* \\ door\_closed \leftarrow true, door\_open^* \\ lights\_on \leftarrow true \end{array} \right\}$$

$$G'' = \{\ wont\_start, door\_open, true\ \}$$

$$IC'' = \left\{ \begin{array}{l} \bot \leftarrow battery\_flat, lights\_on \\ \bot \leftarrow over\_heat \\ \bot \leftarrow door\_open, door\_open^* \\ \bot \leftarrow door\_closed, door\_closed^* \\ \bot \leftarrow \neg door\_open, \neg door\_open^* \\ \bot \leftarrow \neg door\_closed, \neg door\_closed^* \end{array} \right\}$$

$$A'' = \left\{ \begin{array}{l} fan\_broke, fuel\_empty, wet\_day, \\ door\_closed^*, door\_open^*, true \end{array} \right\}$$

Due to space limitations, the network $\psi(T, G, IC, A)$ *is not shown. However, the reader can verify that the network converges to a least fixpoint in which exactly three abducibles* $fuel\_empty$*,* $door\_open^*$ *and* $true$ *are activated. This solution indicates that* G *and* IC *are satisfied in a stable model of* $T \cup \{fuel\_empty\}$ *where* $door\_open$ *is false. If a signal is applied to* $next$*, the network will reject all remaining hypotheses and converge to the done state, indicating that no other solutions exist for this context.*

The approach described above comprises a sound and complete method for solving ALPs in ANNs. It is interesting to distinguish two special cases of this problem which are of practical importance: first, given a context in which IC and A are both empty, ALP reduces to the problem of deciding whether G follows from T; second, given a context in which G, IC and A are all empty, ALP reduces to the problem of computing the stable models of T. It is instructive to consider a classic example of this latter problem.

Example 4.2. *Consider the following abductive context:*

$$\left(\left\{ \begin{array}{l} p \leftarrow \neg q \\ q \leftarrow \neg p \end{array} \right\}, \emptyset, \emptyset, \emptyset \right)$$

As remarked previously, solving this context amounts to computing the stable models of the following program:

$$P = \left\{ \begin{array}{l} p \leftarrow \neg q \\ q \leftarrow \neg p \end{array} \right\}$$

As observed in [8], this program is not easily handled by many other approaches as it has two stable models: $\{q\}$ *and* $\{p\}$*. Applying* ψ *to the this context results in the transformed context below and the sub-network shown in Figure 1(b) above.* [4]

$$\left(\left\{ \begin{array}{l} p \leftarrow q*, true \\ q \leftarrow p*, true \end{array} \right\}, \{\ true\ \}, \left\{ \begin{array}{l} \leftarrow p, p* \\ \leftarrow q, q* \\ \leftarrow \neg p, \neg p* \\ \leftarrow \neg q, \neg q* \end{array} \right\}, \left\{ \begin{array}{l} p* \\ q* \\ true \end{array} \right\} \right)$$

The reader can verify that the relaxed network converges into a stable state whereq, $p*$ *and* $true$ *alone are active – corresponding to the stable model* $\{q\}$*. Applying a signal to* $next$ *forces the network to converge to the next stable state where* p*,* $q*$ *and* $true$ *alone are active – corresponding to the stable model* $\{p\}$*. Applying another signal to* $next$ *forces the network to converge to the final* $done$ *state – indicating that these are the only two models of the program.*

[4] Note that the rest of the network is not shown because it is identical to that given in Figure 1(a).

5 Conclusion

This paper presented a novel method for realising abductive reasoning in neural networks. In particular, it proposed an algorithm for translating abductive logic programs into neural networks so that abductive inference can benefit from the massive-parallelism of the neural architecture. The methodology extends the original program with some additional control logic to ensure that the fixpoints of the network correspond to the stable models of the program. It also uses a well-known relationship between negation and abduction in order to correctly handle programs with positive and negative cycles. In contrast to earlier work, no restrictions are placed on the programs and, if required, the network can be made to enumerate all explanations. Moreover, because our methodology is a generalisation of existing neuro-symbolic techniques, we believe it can be more easily combined with standard learning approaches. In this way, we see our approach as a first tentative step towards the principled integration of abduction and induction at the sub-symbolic level – which could one day have applications in fields of cognitive modelling and scientific discovery.

At present we are still a long way from realising these goals. One problem with our current approach is that, although parallelism is exploited when checking each individual hypothesis, the number of hypotheses checked is exponential in the number of abducibles. Two complementary strategies should be explored in order to address this problem. The first is to use some form of pruning during the search as in symbolic ALP system such as [17]; and the second is to use some form of simplification when preprocessing the program as in answer set programing systems such as [19]. An important extension of the work involves exploiting the structure of the network to impose a preference on the order in which solutions are found. For example, it counter can be modified to output numbers in the order 0001, 0010, 0100, 1000, 0011, ... with the fewest number of bits high so that explanations will be discovered in order of minimality. In addition, the abducibles topologically far from the goal can be connected to the least significant bits of the counter, so that explanations will also be discovered in order of basicality.

An key direction for future work is that of integrating abductive reasoning with inductive learning in order to realise the benefits suggested by recent symbolic machine learning systems. By providing a richer formalism for representing and reasoning about partial knowledge and integrity constraints, abduction could help to exercise a finer degree of control over which assumptions can and cannot be made in learning. In this context, it may be more appropriate to use a variation of the methodology presented in this paper, whereby the network is topology is projected onto a single layer recurrent network (computing the immediate consequence operator) and the threshold units are replaced by sigmoidal neurones. This should enable an experimental validation of the approach as well as a more detailed comparison with symbolic systems.

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] K. Eshghi and R.A. Kowalski, 'Abduction compared with negation by failure', in *Proceedings of the 6th International Conference on Logic Programming*, eds., G. Levi and M. Martelli, pp. 234–254. MIT Press, (1989).

[4] F. Esposito, G. Semeraro, N. Fanizzi, and S. Ferilli, 'Multistrategy Theory Revision: Induction and Abduction in INTHELEX', *Machine Learning*, **38(1/2)**, 133–156, (2000).

[5] N. Fraser. BrainBox Neural Network Simulator (v. 1.8), 2006. at http://neil.fraser.name/software/brainbox/.

[6] A. d'Avila Garcez, K. Broda, and D. Gabbay, *Neural-Symbolic Learning Systems: Foundations and Applications*, Perspectives in Neural Computing, Springer, 2002.

[7] 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).

[8] S. Höllbler and Y. Kalinke, 'Towards a massively parallel computational model for logic programming', in *Proceedings ECAI94 Workshop on Combining Symbolic and Connectionist Processing*, pp. 68–77, (1994).

[9] *Artificial Intelligence and Neural Networks: Steps Toward Principled Integration*, eds., V. Honavar and L. Uhr, Boston Academic Press, 1994.

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

[11] A.C. Kakas and P. Mancarella, 'Generalized Stable Models: a Semantics for Abduction', in *Proceedings of the 9th European Conference on Artificial Intelligence*, pp. 385–391. Pitman, (1990).

[12] A.C. Kakas and F. Riguzzi, 'Abductive concept learning', *New Generation Computing*, **18**(3), 243–294, (2000).

[13] P. Lima, 'Logical Abduction and Prediction of Unit Clauses in Symmetric Hopfield Networks', in *Artificial Neural Networks, 2*, eds., I. Aleksander and J. Taylor, volume 1, pp. 721–725. Elsevier, (1992).

[14] J. Medina, E. Mérida-Casermeiro, and M. Ojeda-Aciego. A neural approach to abductive multiadjoint reasoning, 2002.

[15] O. Ray, *Hybrid Abductive-Inductive Learning*, Ph.D. dissertation, Department of Computing, Imperial College London, UK, 2005.

[16] O. Ray. BrainBox Neural Network Abduction Demo Files, 2006. at http://www.doc.ic.ac.uk/˜or/neural/abduction/demo.

[17] O. Ray and A. Kakas, 'ProLogICA: a practical system for Abductive Logic Programming', in *Proceedings of the 11th International Workshop on Non-monotonic Reasoning*, (2006). to appear.

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

[19] Patrik Simons, Ilkka Niemelä, and Timo Soininen, 'Extending and implementing the stable model semantics.', *Artificial Intelligece*, **138**(1-2), 181–234, (2002).

[20] G. Towell and J. Shavlik, 'Knowledge-based artificial neural networks', *Artificial Intelligence*, **70**(1-2), 119–165, (1994).

[21] 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*, eds., L. Pereira and A. Nerode, pp. 60–81. MIT Press, (1993).

[22] 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).