Reasoning and Learning About Past Temporal Knowledge in Connectionist Models

Conference Paper · August 2007

DOI: 10.1109/IJCNN.2007.4371178 · Source: DBLP

3 authors:

Rafael Vergara Borges
Universidade Federal do Rio Grande do Sul
11 PUBLICATIONS **114** CITATIONS
SEE PROFILE

Luís C. Lamb
Universidade Federal do Rio Grande do Sul
158 PUBLICATIONS **1,203** CITATIONS
SEE PROFILE

Artur D'Avila Garcez
City, University of London
160 PUBLICATIONS **1,837** CITATIONS
SEE PROFILE

Reasoning and learning about past temporal knowledge in connectionist models

Rafael V. Borges[1], Luís C. Lamb[2] and Artur S. d'Avila Garcez[3]

Abstract— **The integration of logic-based inference systems and connectionist learning architectures may lead to the construction of semantically sound cognitive models in artificial intelligence. The use of hybrid systems has shown promising results as regards the computation and learning of classical reasoning within neural networks. However, there still remains a number of open research issues on the integration of non-classical logics and neural networks. We present a new model for integrating symbolic reasoning about past temporal information and neural learning systems. We propose algorithms that translate background knowledge into a neural network and analyse the effectiveness of learning algorithms when subject to symbolic temporal knowledge. This opens several interesting research paths with possible applications to agents' decision making, cognitive modelling and knowledge-based systems.**

I. Introduction

The effective integration of connectionist and symbolic models in artificial intelligence has long been an open research issue (see e.g. [1], [2], [3], [4]). The use of such hybrid systems has shown promising results as regards the computation and learning of classical reasoning and logic programs by artificial neural networks [3], [5], [6]. In addition, it has been pointed out in [2] that human-inspired inference models may lead to more effective reasoning systems, as it is known that neural networks are fault-tolerant, learn and generalize robustly. In spite of such results, there still remains a number of open research issues on integrating logic and connectionist systems. For instance, the representation of symbolic notions of time, knowledge and uncertainty, which are fundamental in intelligent and multi-agent reasoning systems, has only recently been studied. Temporal and modal logics have been successfully applied to and studied in computer science [7], [8]. More recently, such logical systems have been intensively investigated as multi-agent systems (MAS) have become a successful technology. Such logics lie at the foundation of MAS and have been shown an adequate foundation for studies in coordination, cooperation, interaction, specification and reasoning in distributed intelligence [8].

Aiming at responding to these challenges, a new approach has been proposed to integrate other branches of symbolic reasoning and connectionist models, namely *Connectionist Modal and Non-classical Logics* [4], [9], [10], [11]. We believe that fully-fledged intelligent systems will unavoidably cater for adaptation, reasoning and learning in a composite fashion. Thus, there is a need for rich models of semantically well-defined cognitive behaviour, as defended by Valiant [12]. Hybrid systems, incorporating neural networks's learning capabilities and sound, logic-based reasoning offer themselves as an effective alternative to construct such robust and epistemically rich computational cognitive modelling systems. Temporal logics, therefore, are an important component of cognitive behaviour, in particular of any reasoning or learning system [8]. In the sequel, we shall propose a cognitive model which considers reasoning and learning with respect to past information. Reasoning with respect to the past is of relevance to several applications, e.g. agents' decision making [8], temporal pattern recognition [13] and knowledge-based systems [7].

Section II summarises neural-symbolic models as referred to in the paper. Section III shows how to compute a temporal operator in a connectionist model. It uses the representation machinery of temporal logic programs and neural networks' massive parallelism to learn and reason about temporal knowledge. It also illustrates the effectiveness and simplicity of the model through efficient empirical learning carried out by different model architectures. Section IV extends the model to tackle an enriched temporal language containing additional temporal operators such as *always* and *sometimes* in the past. An algorithm to translate such logic knowledge into neural networks is then introduced in Section V. We then conclude and propose directions for future research.

II. Preliminaries

A. Neural-Symbolic Systems

There are several methods for representing time and symbolic knowledge in multi-layer perceptrons (MLP). [4] considers a parallel representation of time, using an ensemble of MLP neural networks, where each network represents a specific time point. [13] describes the use of recurrent links and delay units to propagate values through time. NARX (Nonlinear Auto Regressive with eXogenous inputs) networks [14], are based on a recurrent multi-layer architecture where recurrent links are allowed only from output to input neurons. In NARX models, each time point is considered as the application of an input pattern and the subsequent propagation of values through the network. Each recurrent link implies in a delay on the value propagated, i.e. the activation value of an output neuron N in time t is applied to an input neuron N in time $t+1$. Also, delay units can be inserted before the input neurons in order to allow a greater delay for both input and recurrent values. Fig. 1 illustrates a NARX model. Note that, except for the resources for

[1]Institute of Informatics, Federal University of Rio Grande do Sul, Porto Alegre RS, Brazil: rvborges@inf.ufrgs.br

[2]Institute of Informatics, Federal University of Rio Grande do Sul, Porto Alegre RS, Brazil: LuisLamb@acm.org

[3]Department of Computing, City University London, EC1V 0HB, UK: aag@soi.city.ac.uk

temporal propagation, the core of the architecture consists of a traditional MLP network. [14] have proved that the NARX model simulates any fully-connected neural network.

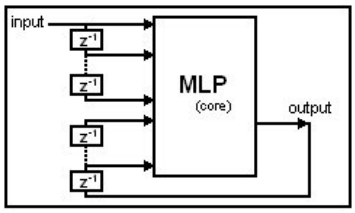

Fig. 1. NARX architecture

Knowledge Based Artificial Neural Networks (KBANN) [15] are neural networks composed of unipolar sigmoid neurons, representing knowledge and allowing the application of empirical learning. [16] proposed a model to represent logic programs in MLP networks with only one hidden layer using neurons with threshold activation functions. [17] incorporated a learning dimension through the use of (bipolar) sigmoid activation functions leading to the Connectionist Inductive Learning and Logic Programming (CILP) system. CILP correctly computes the semantics of any acceptable logic program, solving some limitations of KBANN and presents good learning performance in real-world applications. Next, several definitions used in the paper are introduced.

Definition 1 *An atom A is a propositional variable; a literal L is an atom A or a negation of an atom ($\sim A$). A clause is an implication of the form $A \leftarrow L_1, L_2, ..., L_n$ with $n \geq 0$, where A is an atom and L_i, $1 \leq i \leq n$, are literals. A program $\mathcal{P}$ is a set of clauses. An interpretation of a program $\mathcal{P}$ is a mapping from each atom of a program to a truth value (true or false). The Immediate Consequence Operator $T_\mathcal{P}$ of a program $\mathcal{P}$ is a mapping from an interpretation $I_\mathcal{P}$ of P to another interpretation, and is defined as: $T_\mathcal{P}(I_\mathcal{P})(A)$ is true if and only if, there is a clause in $\mathcal{P}$ of the form $A \leftarrow L_1, L_2, ..., L_n$ and $\bigwedge_{i=1}^{n} I_\mathcal{P}(L_i)$ is true.*

To compute the fixed point semantics of a logic program, following [16], CILP uses a feedforward network to compute the $T_\mathcal{P}$ operator of a program, and connects recurrent links from output to input to allow the recursive computation of such operator into a stable state. In order to compute the $T_\mathcal{P}$ operator, each input neuron is used to represent an atom of the program $\mathcal{P}$, and a hidden neuron is created for representing each clause c in $\mathcal{P}$, being connected with all the input neurons representing an atom A that occurs in the body of c (with weight W if A occurs positively, and $-W$ if it is negated). Also, an output neuron is created to represent each atom A being connected (with weight W) to each hidden neuron representing a clause where A occurs as head. The value of W and of the thresholds of the neurons are set in order to make each hidden neuron compute a conjunction of the inputs, and each output neuron compute a disjunction of the values of the hidden neurons. Recurrent links connect output and input neurons representing the same atom. Lemma 2 formalizes the correct computation of logic programs in the CILP system.

Lemma 2 *For each logic program $\mathcal{P}$, there exists an artificial neural network $\mathcal{N}$, generated through the application of the CILP translation algorithm over $\mathcal{P}$, such that $\mathcal{N}$ can be trained by backpropagation and $\mathcal{N}$ computes the fixed point semantics of $\mathcal{P}$ [17].*

B. The Missing Links Issue

In CILP, a neural network computes a logic program by mapping an input (numerical) vector, received from an external environment, to an output vector. These vectors encode the interpretation of each atom of the logic programs. However, in a logic program all input information is encoded in the clauses, and the fixed point operator should be the same independently of the initial interpretation(received as input). In order to integrate the network's and the program's behaviour, one treats the values in the input vector (external input to the network) as representing *facts* (i.e. rules without a body). Facts are assigned truth-value *true* in every computation of the $T_\mathcal{P}$ operator. If an atom A is assigned to false by this input vector, this information corresponds to a *default negation* of A (represented as $\sim A$), then $\sim A$ remains *true* if there is no positive assignment to A. The networks will have a connected output neuron to represent an atom if and only if it appears as head of a clause in the program. Thus, if we need to verify the interpretation of an atom A that is not head of any clause, but receives an external assignment, the network will not output this value (such cases are investigated in the next section). This is known as the *missing links issue* [18]. Further, typical neural-symbolic systems do not deal with input neurons used to receive both external values and recurrent links. To solve these issues, the solution of [18] consists of inserting a clause of the form $A \leftarrow A*$ for each atom A whose value is needed as output of the system and which also receives an external assignment. After the translation to a network, such insertion will ensure the existence of two different input neurons to represent the same atom A in the original program: a neuron in_{A*}, that only receives the external input related to atom A, and propagates this value to the output neuron representing A (out_A), and a neuron in_A that only receives the recurrent value from out_A.

Another issue to be considered is the number of feedforward executions a CILP network needs in order to compute the fixed point semantics of a program $\mathcal{P}$. To synchronize these feedforward executions with the presentation of new input patterns a general strategy consists of defining a constant $v_\mathcal{P}$ such that $v_\mathcal{P}$ executions of the $T_\mathcal{P}$ operator (and thus $v_\mathcal{P}$ feedforward executions of the network) are sufficient to compute the fixed point for any input [18]. Such value is defined as the greatest v_A among all atoms A in a program $\mathcal{P}$, where v_A is defined as 0 if A does not appear as head of any clause; or as $v_B + 1$ if A is head of at least one clause, and v_B is the greatest v_{Bi} among all atoms B_i (in positive or negated form) in the body of a clause with head A.

III. Combining Architectures for Temporal Reasoning

Several network architectures can be combined to efficiently compute temporal operators. For instance, NARX architectures can be used to represent some temporal logic constructions, such as the previous time operator ●. In order to do so, one extends logic programs providing a semantics for the new temporal operator. Delay units used in NARX models are used to represent the operator as shown in the sequel. An atom α in temporal logic programs is defined as a expression of the form $\bullet^n A$, where $\bullet^n$ is a chain of n previous time operators, $n \geq 0$, and A is a propositional variable. The concepts of literal, clause and program are analogous to Def. 1. The semantics of temporal programs can then be defined:

Definition 3 *The $\bullet T_\mathcal{P}$ immediate consequence operator of a program $\mathcal{P}$ maps an interpretation $I^t_\mathcal{P}$ of $\mathcal{P}$ at time t to an interpretation $\bullet T_\mathcal{P}(I^t_\mathcal{P})$, where $\bullet T_\mathcal{P}(I^t_\mathcal{P})(\alpha)$ is* true *(i) if there is a clause α as in Def. 1; or (ii) if α is of the form $\bullet^n\beta$ and $\mathcal{F}^{t-n}_\mathcal{P}(\beta)$ is* true*, where $\mathcal{F}^t_\mathcal{P}$ is the fixed point of $\mathcal{P}$ at time point t.*

Our method thus uses the temporal order to compute the semantics of a program at timepoint t with the values of $\mathcal{F}^{t-n}_\mathcal{P}$ fixed (i.e. stable) during the computation. In order to perform the propagation of values to the next timepoint, the NARX model uses delay units applied to two different kinds of values in the input: externally applied values and recurrent links. Both strategies can be used in order to represent the ● operator. The former can be used to propagate the values received as input to the network; the latter can be used to propagate values for the computation of the fixed point in a previous timepoint. We illustrate such representations with a testbed for temporal architectures [13]: the prediction of the next value of a temporal sequence composed by (sub)sequences of three bits, where the first two are random and the third one is the result of a binary XOR operation on previous bits, as in Table I.

TABLE I
Temporal XOR sequence

in(α)	0	0	**0**	1	0	**1**	1	1	**0**	0	1	**1**
out(β)	0	**0**	1	0	**1**	1	1	**0**	0	1	**1**	?

The program that represents this sequence has two clauses: $\beta \leftarrow \alpha, \sim \bullet\alpha$ and $\beta \leftarrow\sim \alpha, \bullet\alpha$, where α is the atom representing the input and β represents the output. To represent atom $\bullet\alpha$, we use a delay unit directly in the input, as shown in Fig. 2($Arch_1$). Also, if we consider that α is required as output, we create a delayed recurrent link from an output neuron representing α to the input neuron representing $\bullet\alpha$. In this case, the solution of the missing links issue requires the insertion of a clause $\alpha \leftarrow \alpha*$. This architecture is shown in Fig. 2($Arch_2$). Notice that the insertion of this new clause increases the value of $v_\mathcal{P}$ to 2.

In order to analyse the learning ability over these architectures, two experiments were considered. First, only the error of the second bit of each subsequence was applied to the network during backpropagation. For the remaining timepoints, the error considered at the output was zero with no influence on the weights. In the second experiment, all the error values were considered in the training phase, including the random bits of the sequence, which cause noise in the learning process. Different architectures ($Arch_1$ and $Arch_2$), following the model of Fig. 1 were used in the experiments. Connections with weight 0 were inserted in order to achieve a fully-connected network. Results are depicted in Figs. 3 and 4. The difference between $Arch_2A$ and $Arch_2B$ is that, in $Arch_2A$ the weights related to the neuron corresponding to $\alpha \leftarrow \alpha*$ were randomly initialized. Networks $Arch_3$ were built following [19], which presents a model similar to $Arch_2$, but uses $in_{\alpha*}$ directly to compute the XOR operator, and not in_α. Both architectures correctly compute the program.

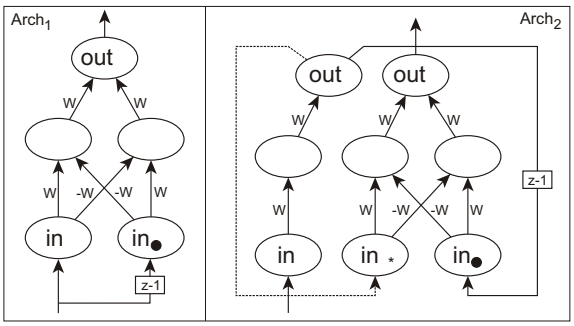

Fig. 2. Architectures for the temporal sequence XOR problem

For each architecture, three combinations of clauses were used to define the initial weights of the connections for the computation of the XOR operator. In Fig. 3, the left column represents the result for a network generated without knowledge, i.e. with all weights randomly initialized. The middle bar chart represents a network that uses a neuron to represent $\beta \leftarrow\sim \alpha, \bullet\alpha$ and one with random weights. The right column represents the experiments with full knowledge. We have run the same process for all networks. The learning process was run using backpropagation, for 500 epochs and learning rate of 0.3. All networks were submitted twice to a process of 10-fold cross-validation over a dataset with 3000 patterns, i.e. each network was trained 20 times with 2700 patterns, and each trained network was tested over a set of 300 patterns. To calculate the error we used the RMSE (root mean square error) averaged over 20 validations. We also repeated the process for Elman networks as described in [13], and the error of such experiments are depicted as the horizontal lines in the charts. The upper region of each column, represented in light grey, show the error after 500 epochs of the training phase, and the dark region represents the smallest error obtained during the process. We used these different values to indicate convergence of the training process.

We can observe in the charts the effects of the insertion of background knowledge in the networks. The correct propagation of the value of α at a time point $t-1$ to the

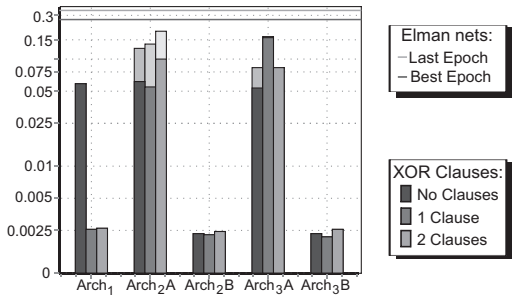

Fig. 3. RMSE in experiments without noise

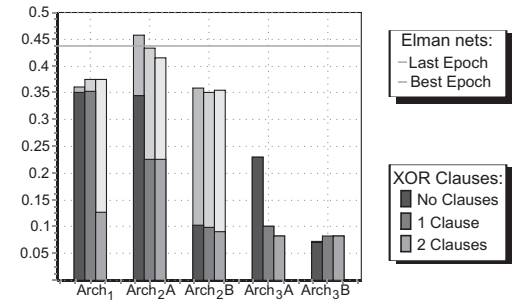

Fig. 4. RMSE in experiments with noise

neuron representing $\bullet l$ at time point t ($Arch_1$, $Arch_2B$ and $Arch_3B$) causes a considerable reduction of the error in the recognition of temporal sequence, improving convergence and noise tolerance. Also, in certain cases, the insertion of clauses helped in the performance of the network, especially when using the $Arch_1$ architecture.

It is also important to highlight that the use of input delay units ($Arch_1$), whenever possible, reduces the complexity of the network. Every time an atom α does not appear as head of any clause, the input neuron representing $\bullet\alpha$ can receive its input value using this approach. In the other cases, there is the need of a recurrent link, in order to propagate the correct value of the atom. In Section V, we introduce an algorithm to realize the translation of logic programs to neural networks, exploring the use of such delay units.

IV. Representing additional past time operators

In this section, we describe a method to represent the semantics of useful past time temporal operators. We start by defining the dual operators *always* ($\blacksquare$) and *sometimes in the past* ($\blacklozenge$). A formula $\blacksquare\alpha$ at a timepoint t denotes that α is *true* at every timepoint $t' \leq t$; the formula $\blacklozenge\alpha$ at t indicates that there exists a timepoint $t' \leq t$ such that α is *true* at t'. We consider a discrete time line, where a formula $\blacksquare\alpha$ or $\blacklozenge\alpha$ can be inferred from past assignments to α. Each operator is inductively defined w.r.t. the time line. At $t = 1$ (initial time point), both $\blacksquare\alpha$ and $\blacklozenge\alpha$ are *true* if and only if α is *true*. For $t \geq 1$, the operators are defined based on the previous and present time points: (i) $\blacksquare\alpha$ is true at t iff α is true at t and $\blacksquare\alpha$ is true at $t-1$; (ii) $\blacklozenge\alpha$ is true at t iff α is true at t or $\blacklozenge\alpha$ is true at $t-1$.

We arbitrarily define the values of $\blacksquare\alpha$ as *true* and $\blacklozenge\alpha$ as *false* at a virtual time point $t = 0$ so that the definition of the operators is the same for every time point. Since we are using a linear time line (and we derive the present from the past) we can represent the operators as implications ($\bullet\blacksquare\alpha \wedge \alpha \rightarrow \blacksquare\alpha$ and $\bullet\blacklozenge\alpha \vee \alpha \rightarrow \blacklozenge\alpha$) and use them as clauses in a logic program (see Fig 5). The architectures of Section III can be used to encode prior domain knowledge, and the architecture of Fig. 5 to represent the temporal clauses.

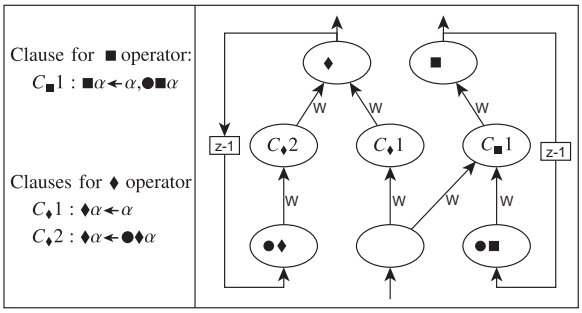

Fig. 5. Representation of ■ and ♦ operators

In order to assess the effectiveness of this proposed model w.r.t. learning and the use of background knowledge we have conducted several experiments. The network was subject to sets of 10 input vectors. For each new application, the value of the recurrent link was reset. The data set consists of all 1024 possible sets, and a process of 8-fold cross-validation was applied twice to each network configuration. The remaining configurations were the same as in the XOR experiments. Initially, two individual networks were used, one to learn the ■ operator and one to learn the ♦ operator. Then, a single network was used to try and learn both operators in a combined way. Results in Table II show that the networks are capable of learning the operators. In most experiments the insertion of background knowledge (clauses) improved the performance of the network, but the insertion of the ♦ rule seemed to be making the learning with examples more difficult. A possible reason is that there is a number of connections with weight zero in the network with prior knowledge, as opposed to the network without prior knowledge, which has randomly initialized weights. Zero weights are probably a more acute problem in the case of the ♦ rule, which requires two clauses, than in the case of ■. We are currently investigating whether the use of small random numbers instead of zeros would change the learning results.

Our model also represents the linear semantics of the binary operator *since* ($\mathbb{S}$). A formula $\alpha\mathbb{S}\beta$ is *true* at t iff β is *true* at $t' \leq t$ and α has been true at every time point u such that $t' < u \leq t$. We define at $t = 0$, $\alpha\mathbb{S}\beta$ as *false*, and for every time point $t \geq 1$, $\alpha\mathbb{S}\beta$ is true if and only if β is true at t, or both α is true at t and $\alpha\mathbb{S}\beta$ is true at $t-1$. The *since* operator can also be described as a pair of clauses in a logic program: $(\alpha\mathbb{S}\beta) \leftarrow \beta$; $(\alpha\mathbb{S}\beta) \leftarrow \alpha, \bullet\alpha\mathbb{S}\beta$.

TABLE II

RESULTS OF ■ AND ♦ EXPERIMENTS

Individual learning		Combined learning	
Experiment	RMSE	Experim.	RMSE
■(no rules)	$3.07 \cdot 10^{-2}$	No rules	5.5×10^{-2}
■(with rules)	$2.03 \cdot 10^{-3}$	■ rules	$2.38 \cdot 10^{-2}$
♦(no rules)	$3.06 \cdot 10^{-3}$	♦ rules	$1.46 \cdot 10^{-1}$
♦(with rules)	$3.84 \cdot 10^{-3}$	All rules	$4.78 \cdot 10^{-4}$

We have run several experiments to assess the performance of networks generated from different sets of clauses, and to analyse the influence of background knowledge. Two processes of 10-fold cross-validation were run, with networks configured with values as in the other experiments. Data sets were composed by 3000 vectors with three values each: the first two (α and β) were randomly generated, and the third one was computed by the network. Table III shows an example of the testbed, and Table IV presents different rules used as background knowledge and the RMSE for each configuration. Notice that the insertion of background knowledge causes a negligible variation in the networks' performance in this instance. We are currently investigating whether certain networks converge faster than others by measuring the number of epochs required to achieve a predefined RMSE.

TABLE III

EXAMPLE OF 'SINCE' OPERATOR SEQUENCE

Time:	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
α :	F	F	T	T	F	T	T	T	T	F
β :	F	T	F	F	F	F	F	T	F	T
$\alpha \mathbb{S} \beta$:	F	T	T	T	F	F	F	T	T	T

TABLE IV

RESULTS OF 'SINCE' EXPERIMENTS

Set of rules	RMSE
$(\alpha \mathbb{S} \beta) \leftarrow \beta$; $(\alpha \mathbb{S} \beta) \leftarrow \alpha, \bullet \alpha \mathbb{S} \beta$	7.22×10^{-3}
$(\alpha \mathbb{S} \beta) \leftarrow \beta$	7.21×10^{-3}
$(\alpha \mathbb{S} \beta) \leftarrow \alpha, \bullet \alpha \mathbb{S} \beta$	7.07×10^{-3}
No rules	7.09×10^{-3}

V. THE TEMPORAL-CONNECTIONIST ALGORITHM

We now introduce an algorithm that translates temporal logic programs with past operators into neural networks. Next, we define the algorithm's input language. Definitions 4 and 5 present the input language and its corresponding semantics, by means of the $pastT_{\mathcal{P}}$ immediate consequence operator.

Definition 4 *A past atom can be inductively defined as follows: (i) a propositional variable A is a past atom; (ii) if α and β are past atoms, then $\bullet\alpha$, $\blacksquare\alpha$, $\blacklozenge\alpha$, $\alpha \mathbb{S} \beta$ are also past atoms. A past temporal logic program is a logic program containing past atoms.*

Definition 5 *The immediate consequence operator $pastT_{\mathcal{P}}$ of an interpretation $I^t_{\mathcal{P}}$ of a program $\mathcal{P}$ at a time t is defined as follows: (i) $pastT_{\mathcal{P}}(I^t_{\mathcal{P}})(\alpha)$ is true if there is a clause with α in the head as in Def. 1; (ii) $pastT_{\mathcal{P}}(I^t_{\mathcal{P}})(\bullet\alpha)$ is true if $\mathcal{F}^{t-1}_{\mathcal{P}}(\alpha)$ is true; (iii) $pastT_{\mathcal{P}}(I^t_{\mathcal{P}})(\blacksquare\alpha)$ is true if $\mathcal{F}^{t-1}_{\mathcal{P}}(\blacksquare\alpha)$ is true and $I^t_{\mathcal{P}}(\alpha)$ is true; (iv) $pastT_{\mathcal{P}}(I^t_{\mathcal{P}})(\blacklozenge\alpha)$is true if $\mathcal{F}^{t-1}_{\mathcal{P}}(\blacklozenge\alpha)$ is true or $I^t_{\mathcal{P}}(\alpha)$ is true; (v) $pastT_{\mathcal{P}}(I^t_{\mathcal{P}})(\alpha \mathbb{S} \beta)$ is true if $I^t_{\mathcal{P}}(\beta)$ is true or both $\mathcal{F}^{t-1}_{\mathcal{P}}(\alpha \mathbb{S} \beta)$ and $I^t_{\mathcal{P}}(\alpha)$ are true;*

The first step (Algorithm 1) is to produce a new logic program, representing the semantics of ■, ♦ and $\mathbb{S}$ as new clauses. This new program enjoys an important property: $pastT_{\mathcal{P}} = \bullet T_{\mathcal{P}}$.

Algorithm 1: Logic Conversion

```
Logic_Conversion(P)
    foreach α ∈ atoms(P) do
        if α = ■β then
            AddClause(■β ← β, ●■β);
        if α = ♦β then
            AddClause(♦β ← β);
            AddClause(♦β ← ●♦β);
        if α = β S γ then
            AddClause(β S γ ← γ);
            AddClause(β S γ ← β, ●(β S γ));
    end
end
```

In order to compute the $\bullet T_{\mathcal{P}}$ operator, we make use of a NARX architecture to build a reduced network (*cf.* [18]). We then introduce Algorithm 2 that translates a program $\mathcal{P}$ into a network $\mathcal{N}$ considering the case where $\mathcal{P}$ only requires an atom as output if it appears as head of a clause; also $\mathcal{P}$ does not present any atom simultaneously receiving input information and appearing as head of a clause, therefore overcoming the *missing links issue*. To provide the correct information for an input neuron representing a formula of type $\bullet^n\alpha$, the algorithm checks if there is an atom of the form $\bullet^i\alpha$, $0 \leq i < n$, as output of a clause. In this case, the algorithm generates a recurrent link from the output neuron representing the atom $\bullet^j\alpha$, where j is the greatest value of i, and the input neuron representing $\bullet^n\alpha$. Such recurrent link presents a chain of $n - i$ delayed units, i.e. the information of $out_{\bullet^j\alpha}$ at time t will be applied to $in_{\bullet^n\alpha}$ at time $t + n - j$. If no atom $\bullet^i\alpha$ appear as head of any clause in $\mathcal{P}$, the input neuron $\bullet^n\alpha$ is connected directly to the input, with n delay units. Lines 2 to 26 of Algorithm 2 execute the CILP translation to allow the computation of the $T_{\mathcal{P}}$ operator; lines 27 to 31 insert recurrent links for the computation of the fixed point and lines 32 to 38 insert the temporal mechanisms described above. [1]

Theorem 6 below shows the translation is sound, i.e. the computation of past temporal programs by neural networks

[1] In the algorithm, $max_{\mathcal{P}}(k, \mu)$ is the largest between the number of literals in a clause and the number of clauses with the same head in the program $\mathcal{P}$; k is the number of literals in the body of a clause, μ is the number of clauses with the same head; A_{min} is the minimum activation value for a neuron to be active (or true). Neurons in the input layer are labelled in_α; neurons in the output layer are labelled out_α where α is the atom represented by these neurons. h_i are hidden neurons representing each clause of the program. $AddLink(\mathcal{N}, source, target, W)$ denotes the insertion of a link from a neuron *source* to a neuron *target* in a network $\mathcal{N}$, with weight W.

Algorithm 2: Translation of ●-based programs

●-based_Translation($\mathcal{P}$)
Define $\frac{max_{\mathcal{P}}(k,\mu)-1}{max_{\mathcal{P}}(k,\mu)+1} \leq A_{min} < 1$;
Define $W \geq \frac{ln(1+A_{min})-ln(1-A_{min})}{max_{\mathcal{P}}(k,\mu)(A_{min}-1)+A_{min}+1} \cdot \frac{2}{\beta}$;
foreach $C_l \in Clauses(\mathcal{P})$ **do**
  $AddHiddenNeuron(\mathcal{N}, h_l)$;
  foreach $\alpha \in body(C_l)$ **do**
    if $in_\alpha \notin Neurons(\mathcal{N})$ **then**
      $AddInputNeuron(\mathcal{N}, in_\alpha)$;
      $ActivationFunction(in_\alpha) \leftarrow g(x)$;
    $AddLink(\mathcal{N}, in_\alpha, h_l, W)$;
  end
  foreach $\sim \alpha \in body(C_l)$ **do**
    if $in_\alpha \notin Neurons(\mathcal{N})$ **then**
      $AddInputNeuron(\mathcal{N}, in_\alpha)$;
      $ActivationFunction(in_\alpha) \leftarrow g(x)$;
    $AddLink(\mathcal{N}, in_\alpha, h_l, -W)$;
  end
  $\alpha \leftarrow head(C_l)$;
  if $out_\alpha \notin Neurons(\mathcal{N})$ **then**
    $AddOutputNeuron(\mathcal{N}, out_\alpha)$;
  $AddLink(\mathcal{N}, h_l, out_\alpha, W)$;
  $Threshold(h_l) \leftarrow \frac{(1+A_{min})(k_l-1)}{2} W$;
  $Threshold(out_\alpha) \leftarrow \frac{(1+A_{min})(1-\mu_l)}{2} W$;
  $ActivationFunction(h_l) \leftarrow h(x)$;
  $ActivationFunction(out_\alpha) \leftarrow h(x)$;
end
foreach $\alpha \in atoms(\mathcal{P})$ **do**
  if $(in_\alpha \in neurons(\mathcal{N})) \wedge (out_\alpha \in neurons(\mathcal{N}))$ **then**
    $AddLink(\mathcal{N}, out_\alpha, in_\alpha, 1)$
end
foreach $in_\alpha \in neurons(\mathcal{N})$ **do**
  if $(\alpha = \bullet^n \beta)$ **then**
    if $\exists i < n (out_{\bullet^i \beta} \in neurons(\mathcal{N}))$ **then**
      $j \leftarrow maximum(i)$;
      $AddDelayedLink(\mathcal{N}, n-j, out_{\bullet^j \beta}, in_\alpha)$;
    else
      $AddInputDelay(\mathcal{N}, n, in_\alpha)$
end
return $\mathcal{N}$;
end

is correct, offering a sound theoretical foundation for our methodology.

Theorem 6 *Let $\mathcal{N}$ be a neural network generated by the application of translation algorithms 1 and 2 on a past temporal logic program $\mathcal{P}$. Then $\mathcal{N}$ computes the fixed point semantics of $\mathcal{P}$ (i.e., it computes output neuron out_α for each atom $\alpha \in \mathcal{P}$ such that for each timepoint t, the activation value act_α of out_α after $v_\mathcal{P}$ feedforward steps in t is (i) $act_\alpha \geq A_{min}$, if $\mathcal{F}^t_\mathcal{P}(\alpha)$ is* true*; (ii) $act_\alpha \leq -A_{min}$, if $\mathcal{F}^t_\mathcal{P}(\alpha)$ is* false*.*

Proof(sketch): The insertion of new clauses in Algorithm 1 renders $past T_\mathcal{P}(I^t_\mathcal{P})(\alpha) = \bullet T_\mathcal{P}(I^t_\mathcal{P})(\alpha)$. Since CILP's translation algorithm correctly computes the classical $T_\mathcal{P}$ operator, all we need to do is make sure that information about the past is correctly propagated. Inserting delay units as done in Algorithm 2 is sufficient to provide such information about the past, so that the network computes the fixed point of $\mathcal{P}$. The chains of n delay units, inserted before the input neurons, will feed a neuron with information applied at timepoint $t-n$. Since the value of α is correctly applied to these delays, from both input and recurrent link, the input neurons will receive correct information about $\bullet^n \alpha$. □

VI. Conclusions and future work

We have analysed the representation of past temporal operators in neural-symbolic systems from two perspectives: by translation of temporal programs into a connectionist architecture and by learning past temporal information in such systems. We have illustrated advantages and disadvantages of this hybrid approach using testbeds, and also explored features of NARX models to reduce the networks complexity of [18], [19]. The extension of this system for dealing with future time operators is an important development in order to model a broader range of aspects of cognitive behaviour, with possible applications to agents cognitive modelling. The analysis of the proposed model under different empirical learning techniques, such as reinforcement learning can lead to the application of this model to real-life problems. Finally, the extraction of rules from temporal neural networks needs to be investigated to provide explanation capabilities to the proposed neural-symbolic models.

Acknowledgments: This work has been partly supported by CNPq (Brazilian National Research Council), FAPERGS Foundation and CAPES.

References

[1] D. Touretzky and G. Hinton, "Symbols among neurons," in *Proc. of the IJCAI-87*. Morgan Kaufman, 1987, pp. 238–243.
[2] A. Browne and R. Sun, "Connectionist inference models," *Neural Networks*, vol. 14, no. 10, pp. 1331–1355, 2001.
[3] A. S. d'Avila Garcez, K. Broda, and D. M. Gabbay, *Neural-Symbolic Learning Systems: Foundations and Applications*. Springer, 2002.
[4] A. S. d'Avila Garcez and L. C. Lamb, "A connectionist computational model for epistemic and temporal reasoning," *Neural Computation*, vol. 18, no. 7, pp. 1711–1738, 2006.
[5] L. Shastri, "Advances in SHRUTI: a neurally motivated model of relational knowledge representation and rapid inference using temporal synchrony," *Applied Intelligence*, vol. 11, pp. 79–108, 1999.
[6] P. Hitzler, S. Hölldobler, and A. K. Seda, "Logic programs and connectionist networks." *J. Applied Logic*, vol. 2, no. 3, pp. 245–272, 2004.
[7] R. Fagin, J. Halpern, Y. Moses, and M. Vardi, *Reasoning about Knowledge*. MIT Press, 1995.
[8] M. Fischer, D. Gabbay, and L. Vila, Eds., *Handbook of temporal reasoning in artificial intelligence*. Elsevier, 2005.
[9] A. S. d'Avila Garcez and L. C. Lamb, "Reasoning about time and knowledge in neural symbolic learning systems." in *NIPS*, S. Thrun, L. K. Saul, and B. Schölkopf, Eds. MIT Press, 2003.
[10] A. S. d'Avila Garcez, L. C. Lamb, and D. M. Gabbay, "Connectionist modal logic: Representing modalities in neural networks," *Theoretical Computer Science*, vol. doi:10.1016/j.tcs.2006.10.023, 2006.
[11] A. S. d'Avila Garcez, L. C. Lamb, and D. M. Gabbay, "Connectionist computations of intuitionistic reasoning," *Theoretical Computer Science*, vol. 358, no. 1, pp. 34–55, 2006.
[12] L. G. Valiant, "Three problems in computer science," *J. ACM*, vol. 50, no. 1, pp. 96–99, 2003.
[13] J. L. Elman, "Finding structure in time," *Cognitive Science*, vol. 14, no. 2, pp. 179–211, 1990.
[14] H. T. Siegelmann, B. G. Horne, and C. L. Giles, "Computational capabilities of recurrent narx neural networks," Univ. of Maryland, Report No. UMIACS-TR-95-12, College Park, Tech. Rep., 1995.
[15] G. G. Towell and J. W. Shavlik, "Knowledge-based artificial neural networks," *Artificial Intelligence*, vol. 70, no. 1-2, pp. 119–165, 1994.
[16] S. Holldobler and Y. Kalinke, "Toward a new massively parallel computational model for logic programming," in *Proc. of the Workshop on Combining Symbolic and Connectionist Processing, ECAI 94*, 1994, pp. 68–77.
[17] A. S. d'Avila Garcez and G. Zaverucha, "The connectionist inductive learning and logic programming system," *Applied Intelligence*, vol. 11, no. 1, pp. 59–77, 1999.
[18] R. V. Borges, L. C. Lamb, and A. d'Avila Garcez, "Towards reasoning about the past in neural symbolic systems," in *Proceedings of the 3rd International Workshop on Neural-Symbolic Learning and Reasoning IJCAI 2007*, 2007.
[19] R. V. Borges, L. C. Lamb, and A. S. d'Avila Garcez, "Combining architectures for temporal learning in neural symbolic systems," in *Proceedings of the 6th International Conference on Hybrid Intelligent Systems(HIS 06')*. IEEE, 2006.