A Connectionist Cognitive Model for Temporal Synchronisation and Learning*

Luís C. Lamb and **Rafael V. Borges**
Institute of Informatics
Federal University of Rio Grande do Sul
Porto Alegre, RS, 91501-970, Brazil
LuisLamb@acm.org; rvborges@inf.ufrgs.br

Artur S. d'Avila Garcez
Department of Computing
City University London
London EC1V 0HB, UK
aag@soi.city.ac.uk

Abstract

The importance of the efforts towards integrating the symbolic and connectionist paradigms of artificial intelligence has been widely recognised. Integration may lead to more effective and richer cognitive computational models, and to a better understanding of the processes of artificial intelligence across the field. This paper presents a new model for the representation, computation, and learning of temporal logic in connectionist systems. The model allows for the encoding of past and future temporal logic operators in neural networks, through a neural-symbolic translation algorithms introduced in the paper. The networks are relatively simple and can be used for reasoning about time and for learning by examples with the use of standard neural learning algorithms. We validate the model in a well-known application dealing with temporal synchronisation in distributed knowledge systems. This opens several interesting research paths in cognitive modelling, with potential applications in agent technology, learning and reasoning.

Introduction

The construction of rich computational cognitive models has recently been pointed out as a key research question for computer science and cognitive computation (Valiant 2003). To cope with the requirements of constructing a rich intelligent behaviour model one should integrate expressive reasoning and robust learning in a sound way. However, learning, which has been studied typically under experimental, statistical approaches would then have to be integrated with the reasoning component of intelligent systems, which has mostly been studied using logic-based formalisms. In order to respond to this, we seek to incorporate in a single model the two fundamental aspects of intelligent behaviour, namely reasoning and learning. Although challenging, the construction of such computational cognitive models would meet the requirements for a long standing problem in artificial intelligence: the integration of the connectionist and the symbolic paradigms of artificial intelligence, which has long been recognised as a standing research issue in the field (Page 2000; Smolensky & Legendre 2006; Sun 1995; Touretzky & Hinton 1985; Valiant 2000). Integration may lead to more effective and richer cognitive computational models, and to a better understanding of the processes of artificial intelligence across the field.

Several efforts have been made in this direction. However, most of them deal with knowledge expressed as production rules or logic programming (d'Avila Garcez, Broda, & Gabbay 2002; Shastri 1999; Towell & Shavlik 1994). This work deals with dynamic knowledge, which evolves in time. We present a model for representing, computing, and learning temporal logic in connectionist systems. The model allows for the encoding of past and future temporal logic operators in neural networks, through a translation algorithm introduced in the paper. The networks are relatively simple and can be used for reasoning about time and learning by examples with the use of standard neural learning algorithms. We apply the model in a number of experiments, dealing with learning, reasoning and synchronisation in a distributed knowledge environment.

Temporal logic has been amply successful in computer science. It has been used in the formalisation of several computational properties and concepts including verification, specification and derivation of computing systems. More recently, such techniques have been successfully used in artificial intelligence, in particular, for modelling several dimensions of multi-agent systems, including model checking, coordination, evolution and cooperation (Fisher, Gabbay, & Vila 2005). This work contributes towards the representation of such expressive, highly successful logical languages in a connectionist system.

We will show that, as pointed out in (Smolensky & Legendre 2006), cognitive models based on neural-symbolic integration can benefit from their complementary nature. Human-inspired inference models may lead to more effective reasoning systems, as it is known that neural networks are fault-tolerant and generalize robustly (Browne & Sun 2001). We will take advantage of a connectionist architecture to learn symbolic temporal knowledge based on inference mechanisms from logic, which is also used as background knowledge in the learning process. Our experiments suggest that the proposed model is rich enough to deal with temporal reasoning and learning in distributed environments, meeting two requirements put forward in (Valiant 2003): learning and reasoning are integrated in the same model and are tractable.

*Research supported by the Brazilian Research Council CNPq.

Next, we introduce the basics of connectionist and temporal models used in the paper. We then present a representation of temporal formalisms in connectionist systems. An algorithm that translates temporal theories including past and future operators into neural networks is introduced and we prove that the translation and the computation of temporal knowledge in our model is sound. We then validate the approach with experiments using well-known testbeds for temporal knowledge synchronisation in distributed systems, and show that empirical learning benefits from using temporal background knowledge. Finally, we conclude and point out directions for future research.

Preliminaries

This section introduces the basics of connectionist models and symbolic temporal reasoning used in the paper. We assume familiarity with neural networks models and only summarise used concepts. A neural network can be seen as a massively parallel distributed processor that stores experiential knowledge (Haykin 1999). A multilayer perceptron (MLP) is composed of several layers of simple processing units, the artificial neurons. There are several methods for representing time and symbolic knowledge in MLPs. d'Avila Garcez & Lamb (2006) consider a parallel representation of time, using an ensemble of MLPs, where each network represents a specific timepoint. Elman (1990) describes the use of recurrent links and delay units to propagate values through time. Nonlinear Auto Regressive with eXogenous inputs (NARX) networks (Siegelmann, Horne, & Giles 1995) are based on a recurrent multi-layer architecture where recurrent links are allowed only from output to input neurons. In such models, each timepoint 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 propagated value i.e., the activation value of an output neuron N at time t is applied to an input neuron N at $t + 1$. Also, delay units can be inserted before the input neurons in order to allow a greater delay for both input and recurrent values.

In order to represent rich symbolic knowledge in connectionist models, such as modal and temporal knowledge (which have been shown adequate in modelling multi-agent cognition (Fisher, Gabbay, & Vila 2005)), one typically makes use of a hybrid approach, translating symbolic knowledge into a neural network, e.g. (d'Avila Garcez, Broda, & Gabbay 2002; d'Avila Garcez & Lamb 2006). The temporal knowledge representation language that we will use is based on an extension of logic programming clauses. Thus, the following logic definitions will be useful.

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 $\mathcal{P}$ 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.*

Temporal Reasoning

We start by defining a language that extends (propositional) logic programs with a unary temporal operator $\bullet$ that represent the immediately previous timepoint. $\bullet\alpha$ denotes that α is *true* at the previous timepoint. The syntax of $\bullet$-based programs can be defined as a set of clauses $\alpha \leftarrow \lambda_1, \lambda_2, ..., \lambda_n$, where α is an (temporal) atom and λ_i, for $1 \leq i \leq n$ and $n \geq 0$, are literals. An atom is defined as any expression $\bullet^m A$, where $\bullet^m$ is a chain of *m previous time* operators, with $m \geq 0$, and A is a propositional variable. A literal is an atom or the negation of an atom. We characterize the semantics of a $\bullet$-based program through the use of a fixed point definition. We define the *immediate consequence operator* $\bullet T_\mathcal{P}$ of a $\bullet$-based program $\mathcal{P}$ as a mapping between interpretations $I^t_\mathcal{P}$ at timepoint t.

Definition 2 *$\bullet T_\mathcal{P}(I^t_\mathcal{P})(\alpha)$ is* true *if and only if one of the following holds: (i) there is a clause in $\mathcal{P}$ of the form $\alpha \leftarrow \lambda_1, \lambda_2, ..., \lambda_n$ where $I^t_\mathcal{P}(\bigwedge_{i=1}^{n} \lambda_i)$ is* true*; (ii) α is an atom of the form $\bullet\beta$, and $\mathcal{F}^{t-1}_\mathcal{P}(\beta)$ is* true*, where $\mathcal{F}^t_\mathcal{P}$ is the fixed point of $\mathcal{P}$ at time t, i.e., $T_\mathcal{P}(\mathcal{F}^t_\mathcal{P})(\alpha) = \mathcal{F}^t_\mathcal{P}(\alpha)$.*

Following (Gelfond & Lifschitz 1988), we can show that the $\bullet T_\mathcal{P}$ operator converges to a unique stable state for a large class of propositional logic programs. Such stable state represents the *fixed point semantics* of the program. The approach used in (d'Avila Garcez, Broda, & Gabbay 2002) to compute the semantics of a logic program $\mathcal{P}$ consists in generating an input neuron to represent each atom in $\mathcal{P}$, a hidden neuron for each clause C of $\mathcal{P}$ (computing the conjunction of the body literals in C), and an output neuron for each atom α, computing the disjunction of all the clauses where α is the head. To recursively compute the $T_\mathcal{P}$ operator, recurrent links are set up from the output to the input neuron representing the same atom, in such a way that the resulting interpretation of one computation of $T_\mathcal{P}$ is applied as input for the next one.

In order to build a connectionist computational architecture for representing $\bullet$-based programs, we will add recurrent links from output units representing an atom α to the input neuron representing $\bullet^n\alpha$, with a chain of n delay units. These units simulate *short term memory*, holding the activation value of a neuron, relative to a time $t - n$, during the computation relative to the time t. If an atom α does not appear as the head of any clause, we insert a chain of delay units directly on the input connection, and therefore the input value applied to the neuron will present the required delay. Algorithm 1 computes the translation of $\bullet$-based programs into a neural network. A $\bullet$-based logic program $\mathcal{P}$ is input to the algorithm and it outputs a neural network architecture that computes the (fixed point) semantics of $\mathcal{P}$. In Algorithm 1 the following notation is used: $max_\mathcal{P}(k, \mu)$ 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 $\mathcal{P}$. $AddLink(\mathcal{N}, source, target, W)$ denotes the insertion of a link from a neuron *source* to a neuron *target* in a network $\mathcal{N}$, with weight W. Algorithm 1 is used together with Algorithm 2 (introduced in the sequel) so as to render temporal reasoning in our model.

Algorithm 1: ●-based connectionist computation

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

A Cognitive Model for Temporal Reasoning

In this section we define the temporal language we use for knowledge representation and the algorithm that will allow integrated connectionist temporal reasoning and learning in the cognitive model. The language used here represents linear temporal knowledge dealing with both past and future. Therefore, we need to extend the syntax of the ●-based formalisation and characterise a fixed point semantics for these extended programs. The unary past operators ●, ■ and ♦ are respectively defined as *previous time*, *always in the past* and *sometime in the past*. Future time operators ○, □ and ◇ are also defined. The binary $\mathbb{S}$ and $\mathbb{Z}$ operators (*since* and "*zince*") denote that a proposition has been *true* since the occurrence of another, but $\alpha\mathbb{Z}\beta$ also allows the case where α has always occurred. The $\mathbb{U}$ (*until*) and $\mathbb{W}$ (*unless*) operators are defined mirroring $\mathbb{S}$ and $\mathbb{Z}$, in the future time.

Definition 3 *(Extended Temporal Formulas) An atom is inductively defined as follows: (i) If p is a propositional variable, then p is an atom; (ii) If α and β are atoms, then $\bullet\alpha$, $\blacksquare\alpha$, $\blacklozenge\alpha$, $\alpha\mathbb{S}\beta$ and $\alpha\mathbb{Z}\beta$ are also atoms; (iii) If α and β are atoms, then $\bigcirc\alpha$, $\Box\alpha$, $\Diamond\alpha$, $\alpha\mathbb{U}\beta$ and $\alpha\mathbb{W}\beta$ are also atoms.*

Our model makes use of a declarative sequential approach, based on temporal sequence of events, i.e. the consequence relations are of the form *past time antecedent* $\rightarrow$ *future time consequent* (see e.g. (Fisher, Gabbay, & Vila 2005)). However, this is an imperative approach where the antecedent is used to infer the actions that an agent must perform in the future. We will then define a declarative approach where past operators are used to represent information that has been propagated through time, and the future operators denote commitments of an agent, abstracting away imperative steps needed to compute such commitments, as suggested in intentional models of agency (*cf.*, (Georgeff & Rao 1995)). Each past time operator is defined recursively with respect to the present and the immediately previous timepoint.

Definition 4 *The application of the immediate consequence operator $T_\mathcal{P}$ for an interpretation $I^t_\mathcal{P}$ of a program $\mathcal{P}$ at time t with respect to past operators is defined as:*
(i) $T_\mathcal{P}(I^t_\mathcal{P})(\bullet\alpha)$ is true if $\mathcal{F}^{t-1}_\mathcal{P}(\alpha)$ is true;
(ii) $T_\mathcal{P}(I^t_\mathcal{P})(\blacksquare\alpha)$ is true if $\mathcal{F}^{t-1}_\mathcal{P}(\blacksquare\alpha)$ is true and $I^t_\mathcal{P}(\alpha)$ is true;
(iii) $T_\mathcal{P}(I^t_\mathcal{P})(\blacklozenge\alpha)$ is true if $\mathcal{F}^{t-1}_\mathcal{P}(\blacklozenge\alpha)$ is true or $I^t_\mathcal{P}(\alpha)$ is true;
(iv) $T_\mathcal{P}(I^t_\mathcal{P})(\alpha\mathbb{S}\beta)$ (resp. $T_\mathcal{P}(I^t_\mathcal{P})(\alpha\mathbb{Z}\beta)$) is true if $I^t_\mathcal{P}(\beta)$ is true or both *$\mathcal{F}^{t-1}_\mathcal{P}(\alpha\mathbb{S}\beta)$ (resp. $\mathcal{F}^{t-1}_\mathcal{P}(\alpha\mathbb{Z}\beta)$) and $I^t_\mathcal{P}(\alpha)$ are* true.

For a discrete linear time flow beginning at $t = 1$, we arbitrarily define the values of $\mathcal{F}^0_\mathcal{P}(\bullet\alpha)$, $\mathcal{F}^0_\mathcal{P}(\blacksquare\alpha)$ and $\mathcal{F}^0_\mathcal{P}(\alpha\mathbb{S}\beta)$ as *false*, and the values of $\mathcal{F}^0_\mathcal{P}(\blacklozenge\alpha)$ and $\mathcal{F}^0_\mathcal{P}(\alpha\mathbb{Z}\beta)$ as *true*. Commitments are newly inferred formulas based on the memory of present and past intentions. For an atom $\bigcirc\alpha$ at t, the model must guarantee that α will be *true* at $t+1$. The remaining (future) operators can also be defined with respect to the present and the next timepoint. The formula $\Box\alpha$ can be written as $\alpha \wedge \bigcirc\Box\alpha$. Therefore, we must ensure that, if $\Box\alpha$ is *true* at t, α must also be *true* at t and $\Box\alpha$ must be *true* at time $t+1$. The remaining operators are defined as follows:
(i) $\Diamond\alpha \equiv \alpha \vee \bigcirc\Diamond\alpha$;
(ii) $\alpha\mathbb{U}\beta \equiv \beta \vee (\alpha \wedge \bigcirc(\alpha\mathbb{U}\beta)) \equiv (\beta \vee \alpha) \wedge (\beta \vee \bigcirc(\alpha\mathbb{U}\beta))$;
(iii) $\alpha\mathbb{W}\beta \equiv \beta \vee (\alpha \wedge \bigcirc(\alpha\mathbb{W}\beta)) \equiv (\beta \vee \alpha) \wedge (\beta \vee \bigcirc(\alpha\mathbb{W}\beta))$.
To define $\Diamond$, $\mathbb{U}$ and $\mathbb{W}$, the use of disjunctions is necessary, so an individual analysis of each case (i, ii, iii) must be done in order to define how to assign values to a specific disjunct. We avoid the case where both disjuncts are *false* for formulas $(i) - (iii)$ above by using $\rightarrow$ and $\neg$ instead of $\vee$, e.g. using $\neg p \rightarrow q$ with q *true* by default (i.e. unless p is *true*), instead of $p \vee q$. In order to define the antecedent in the above implication, information about previous timepoints is used to infer subsequent ones. For instance, $\Diamond\alpha$ can be represented

as $\neg\alpha \rightarrow \bigcirc\Diamond\alpha$. The only case where the choice is somewhat involved is for the disjunction $\alpha \vee \beta$ used to represent $\alpha\mathbb{U}\beta$ and $\alpha\mathbb{W}\beta$. In such cases, for $\alpha\mathbb{U}\beta$, β is considered as the default ($\neg\alpha \rightarrow \beta$); for $\alpha\mathbb{W}\beta$, α is the default ($\neg\beta \rightarrow \alpha$). This choice is due to the definition of the operators, since $\alpha\mathbb{U}\beta$ requires that a sequence of α must be ended by β, and $\alpha\mathbb{W}\beta$ holds for infinite sequences of α. Based on these definitions, we can now consider the following rules to define the immediate consequence operator, thus extending the semantics of the programs to allow the representation of commitments.

Definition 5 *The immediate consequence operator $T_{\mathcal{P}}$, for an interpretation $I^t_{\mathcal{P}}$ of a program $\mathcal{P}$ at a time t with respect to future temporal operators is defined as:*
(i) $T_{\mathcal{P}}(I^t_{\mathcal{P}})(\alpha)$ is true if $\mathcal{F}^{t-1}_{\mathcal{P}}(\bigcirc\alpha)$ is true;
(ii) $T_{\mathcal{P}}(I^t_{\mathcal{P}})(\alpha)$ is true if $I^t_{\mathcal{P}}(\Box\alpha)$ is true;
(iii) $T_{\mathcal{P}}(I^t_{\mathcal{P}})(\Box\alpha)$ is true if $\mathcal{F}^{t-1}_{\mathcal{P}}(\Box\alpha)$ is true;
(iv) $T_{\mathcal{P}}(I^t_{\mathcal{P}})(\Diamond\alpha)$ is true if $\mathcal{F}^{t-1}_{\mathcal{P}}(\Diamond\alpha)$ is true and $\mathcal{F}^{t-1}_{\mathcal{P}}(\alpha)$ is false;
(v) $T_{\mathcal{P}}(I^t_{\mathcal{P}})(\beta)$ is true if $I^t_{\mathcal{P}}(\alpha\mathbb{U}\beta)$ is true and $I^t_{\mathcal{P}}(\alpha)$ is false;
(vi) $T_{\mathcal{P}}(I^t_{\mathcal{P}})(\alpha\mathbb{U}\beta)$ is true if $\mathcal{F}^{t-1}_{\mathcal{P}}(\alpha\mathbb{U}\beta)$ is true and $\mathcal{F}^{t-1}_{\mathcal{P}}(\beta)$ is false;
(vii) $T_{\mathcal{P}}(I^t_{\mathcal{P}})(\alpha)$ is true if $I^t_{\mathcal{P}}(\alpha\mathbb{W}\beta)$ is true and $I^t_{\mathcal{P}}(\beta)$ is false;
(viii) $T_{\mathcal{P}}(I^t_{\mathcal{P}})(\alpha\mathbb{W}\beta)$ is true if $\mathcal{F}^{t-1}_{\mathcal{P}}(\alpha\mathbb{W}\beta)$ is true and $\mathcal{F}^{t-1}_{\mathcal{P}}(\beta)$ is false.

In the sequel we define an algorithm that translates temporal logic programs (containing future and past operators) into semantically equivalent $\bullet$-based ones. For instance, in order to represent the rule for the $\blacksquare$ operator in Def. 4, we must consider the relation from $\mathcal{F}^{t-1}_{\mathcal{P}}(\blacksquare\alpha)$ and $I^t_{\mathcal{P}}(\alpha)$ to $T_{\mathcal{P}}(I^t_{\mathcal{P}})(\blacksquare\alpha)$. By definition, if $\mathcal{F}^{t-1}_{\mathcal{P}}(\blacksquare\alpha)$ holds, then $\bullet\blacksquare\alpha$ holds at t, so we can represent such rule as a clause $\blacksquare\alpha \leftarrow \alpha, \bullet\blacksquare\alpha$. The remaining rules can be translated similarly (Fisher, Gabbay, & Vila 2005). This is shown in Algorithm 2. The following results guarantee that the algorithms are sound, in the sense that the connectionist model correctly represents and computes temporal knowledge.

Lemma 6 *Let Q be a program obtained by Algorithm 2 from an input program $\mathcal{P}$. For every atom α in $\mathcal{P}$, and every interpretation $I^t_{\mathcal{P}}$, $\bullet T_Q(I^t_{\mathcal{P}})(\alpha)$ holds.*

Proof: Note that the translation computed by the algorithm only outputs clauses representing exactly the semantic rules of temporal operators added to the program. ($\leftarrow$) Assume that $T_{\mathcal{P}}(I^t_{\mathcal{P}})(\alpha)$ holds. Therefore, either $\bullet T_{\mathcal{P}}(I^t_{\mathcal{P}})(\alpha)$ is true, and the addition of clauses does not change its value, or α is defined with respect to the semantic rules of the operators. In this case, the algorithm computes a clause for each rule, such that the body of the clause is defined such that α holds. Thus, by definition of $\bullet T_{\mathcal{P}}$, we have that $\bullet T_Q(I^t_{\mathcal{P}})(\alpha)$ holds. ($\rightarrow$) If $T_{\mathcal{P}}(I^t_{\mathcal{P}})(\alpha)$ is *false*, then newly inserted clauses do not change either the value of α, or of $\bullet T_{\mathcal{P}}$. Since all rules for the temporal operators are represented by new clauses, none of such rules assigns *true* to α and then to $\bullet T_Q(I^t_{\mathcal{P}})(\alpha)$. □

Theorem 7 *Let $\mathcal{N}$ be a network obtained by the application of algorithms 1 and 2 on a temporal logic program $\mathcal{P}$. Therefore, $\mathcal{N}$ computes the fixed point semantics of $\mathcal{P}$.*

Proof: By Lemma 6, the computation of $\bullet T_{\mathcal{P}}$ with respect to the program extended with the temporal operators is sufficient for the computation of $T_{\mathcal{P}}$. Inserting delay units as done in Algorithm 1 is sufficient to provide the network with information about the past, since a chain of n delay units inserted before an input neuron will provide the neuron with information from timepoint $t-n$. Given the soundness of the translation of logic programs into neural networks (d'Avila Garcez & Lamb 2006), the network extended with delay units will correctly represent the semantics of formulas of type $\bullet^n\alpha$, and hence of program $\mathcal{P}$. □

Algorithm 2: Temporal Logic Knowledge Translation

```
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 = β𝕊γ then
            AddClause(β𝕊γ ← γ);
            AddClause(β𝕊γ ← β, ●(β𝕊γ));
        if α = β𝕑γ then
            AddClause(β𝕑γ ← γ);
            AddClause(β𝕑γ ← β, ●(β𝕑γ));
        if α = ○β then AddClause(β ← ● ○ β);
        if α = □β then
            AddClause(β ← □β);
            AddClause(□β ← ●□β);
        if α = ◇β then AddClause(◇β ← ●◇β, ~ ●β);
        if α = β𝕌γ then
            AddClause(γ ← β𝕌γ, ~ β);
            AddClause(β𝕌γ ← ●(β𝕌γ), ~ ●γ);
        if α = β𝕎γ then
            AddClause(β ← β𝕎γ, ~ γ);
            AddClause(β𝕎γ ← ●(β𝕎γ), ~ ●γ);
    end
end
```

Empirical Learning and Synchronisation

In this section we validate our approach. We apply the model to a classical problem of synchronisation in distributed environments, namely, the *Dining Philosophers Problem*, originally from (Dijkstra 1971): *n philosophers sit at a table, spending their time thinking and eating. In the centre of the table there is a plate of noodles, and a philosopher needs two forks to eat it. The number of forks on the table is the same as the number of philosophers. One fork is placed between each pair of philosophers and they will only use the forks to their immediate right and left. They never talk to each other, which creates the possibility of deadlock and starvation.*

We represent the knowledge of each philosopher (agent) using temporal logic programs, and compute their behaviour in our model. An agent's policy will model the following behaviour: from the moment that information $hungry_i$ is known to agent i, she must start trying to get forks (say, from the left) until all forks are in use. When an agent has two forks, she may eat until she is sated (i.e. an external input $sated_i$ is applied). An agent can communicate with the environment through five distinct actions: eat_i, $dropL_i$ and $dropRi$, representing that the agent is returning a fork (left or right) to the table, and $pickL_i$, $pickR_i$, in which the agent tries to allocate the left and the right forks. Since a fork may not be available when an agent tries to pick it, the en-

$pickL_1 \mathbb{W} gotL_1 \leftarrow hungry_1$; $pickR_1 \mathbb{W} gotR_1 \leftarrow gotL_1$
$eat_1 \mathbb{W} sated_1 \leftarrow gotR_1$; $dropL_1 \leftarrow sated_1$
$dropR_1 \leftarrow fulfill_1$; $sated_1 \leftarrow sated_1^*$
$GotL_1 \leftarrow GotL_1^*$
$GotR_1 \leftarrow GotR_1^*$
$pickL_1 \mathbb{W} gotL_1 \leftarrow \bullet(pickL_1 \mathbb{W} got_{1,A}), \sim \bullet gotL_1$
$pickL_1 \leftarrow pickL_1 \mathbb{W} gotL_1, \sim gotL_1$
$pickR_1 \mathbb{W} gotR_1 \leftarrow \bullet(pickR_1 \mathbb{W} gotR_1), \sim \bullet gotR_1$
$pickR_1 \leftarrow pickR_1 \mathbb{W} gotR_1, \sim gotR_1$
$eat_1 \mathbb{W} sated_1 \leftarrow \bullet(eat_1 \mathbb{W} sated_1), \sim \bullet sated_1$
$eat_1 \leftarrow eat_1 \mathbb{W} sated_1, \sim sated_1$

Table 1: An agent's temporal knowledge representation

vironment responds to agent i through the information $gotL_i$ and $gotR_i$, denoting that agent i was successfully allocated a fork. The environment randomly sends signals *hungry* and *sated* to the agents, and responds to actions performed by the agents, allowing only one agent to be allocated a particular fork at each time. Agents do not receive any information about their state (being hungry, holding forks, etc); they only receive information about individual events and internally represent their states with respect to these events.

Table 1 illustrates the logic program that represents an agent's behaviour. The upper half of the table describes the original knowledge and the lower half describes knowledge translated by Algorithm 2. In order to analyse the learning capacity of the networks representing each agent, we extend our environment to give each agent the necessary information so that a supervised learning algorithm can be used. Such information is the action the agent executes at each timepoint, according to the default policy, and the agent's state of affairs (such state is stored in the environment). Three different configurations are used in our experiments. The behaviour of fully knowledgeable (FK) agents is represented in a network generated by the translation of all rules in Table 1. This generates a network with layers containing, respectively, thirteen, fourteen, and eleven neurons. Partial knowledge (PK) agents are represented by networks generated from the lower part of Table 1, and inserting eight additional hidden neurons to allow for learning of the other rules with the same number of neurons in the hidden layer (fourteen). All the connections to and from these new neurons are randomly initialized. Finally, the no knowledge (NK) agents have all connections randomly set.

Two learning approaches were considered. First, offline learning was implemented, where the agent only receives information from the environment, and her actions do not change the environment. Figure 1 depicts the evolution of error for the three agents (FK, PK and NK) in time, using backpropagation (Rumelhart, Hinton, & Williams 1986). It shows the Root Mean Squared Error (RMSE) based on the difference between the networks' output values and the expected values calculated by the environment. For these experiments, we have used 500 epochs, each epoch consisting of 200 consecutive patterns for training, and the next 200 patterns for testing. In Table 2, the first two lines indicate how many epochs were needed for an agent to achieve RMSE below 0.2 and 0.1, respectively, averaged over eight runs of the offline learning process. The next two lines show, respectively, the averaged smallest error obtained during learning, and the averaged error after 500 epochs.

	FK	HK	NK
RMSE ≤ 0.2	0	73	137.88
RMSE ≤ 0.1	0	80	155.63
Smallest Error	0.032	0.016	0.082
Final Error	0.032	0.07	0.79

Table 2: Offline Learning Results

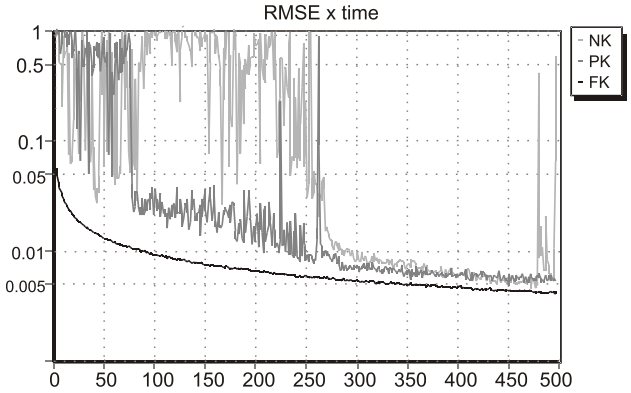

Figure 1: Offline Learning Error in Time

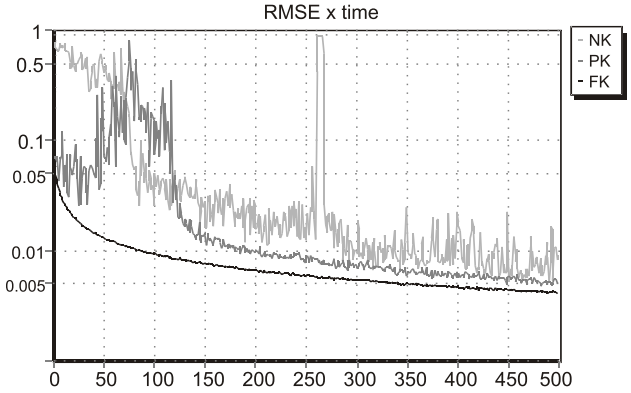

Figure 2: Online Error in Time

Next, we have carried out online learning, with an agent acting over the environment during the learning experiments. We have used an environment with three agents, where two of them are fully knowledgeable. We have run three different experiments, varying the knowledge level of the remaining agent. We have run the experiments for 100,000 timepoints. Figure 2 shows the averaged error of each agent during the first 500 epochs. It illustrates how the different networks converge to the desired behaviour. In this experiment, we have also analysed the behaviour of the system as a whole, measuring the allocation of forks to agents as the relation between the number of agents eating and the number of agents

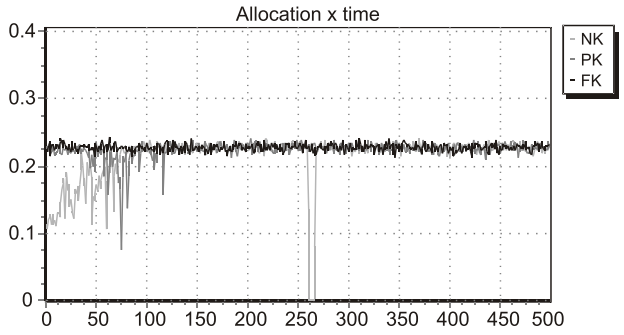

Figure 3: Resource allocation in time

wishing to eat at each timepoint, as depicted in Fig. 3. When compared with Fig. 2, it is clear that the decrease in the error is directly related to the agents' synchronisation, and therefore the proper use of resources. An indication of this behaviour is seen in the region between timepoints 260 and 265 for the NK agent. In such interval, the NK agent got hold of resources, preventing other agents from eating and therefore reducing the resource allocation rate. These results indicate that the use of symbolic background knowledge in the form of temporal logic rules may lead to more effective learning performance. Other temporal sequence experiments we have performed using the XOR problem as well as synthetic data (Borges, Lamb, & d'Avila Garcez 2007) corroborate the importance of exploiting any available background knowledge. We argue this is relevant in the case of temporal reasoning and dynamic memory models.

Conclusions and Future Work

This work has presented a cognitive computational model for integrated temporal learning and reasoning. It has shown that temporal knowledge can be successfully represented, computed and learned by connectionist models, and it has illustrated the capabilities of the model's learning dimension. In particular, the learning experiments corroborate the benefits of using symbolic background knowledge in the form of temporal logic rules. Further, the model has been shown effective in representing an agent's internal states in dynamic environments, incorporating the robust learning capabilities of neural networks and sound reasoning from temporal logic. It was shown that such integrated model can be used both to model an external agent (offline learning) and an agent immersed in the environment (online learning). In summary, this work contributes to the development of rich cognitive models, as seen as a challenge for computer science in (Valiant 2003), building upon the hypothesis defended in (Smolensky & Legendre 2006) that artificial intelligence may benefit from integrating the strengths of its symbolic and connectionist paradigms. A number of research paths remain open such as the integration of other symbolic logic systems within the neural computation paradigm, including branching time temporal logics and intentional cognitive models, with possible applications in distributed multi-agent systems.

References

Borges, R.; Lamb, L.; and d'Avila Garcez, A. 2007. Reasoning and learning about past temporal knowledge in connectionist models. In *Proc. of IJCNN 2007*. to appear.

Browne, A., and Sun, R. 2001. Connectionist inference models. *Neural Networks* 14:1331–1355.

d'Avila Garcez, A., and Lamb, L. 2006. A connectionist computational model for epistemic and temporal reasoning. *Neural Computation* 18(7):1711–1738.

d'Avila Garcez, A.; Broda, K.; and Gabbay, D. 2002. *Neural-Symbolic Learning Systems: Foundations and Applications*. Springer.

Dijkstra, E. 1971. Hierarchical ordering of sequential processes. *Acta Inf.* 1:115–138.

Elman, J. 1990. Finding structure in time. *Cognitive Science* 14(2):179–211.

Fisher, M.; Gabbay, D.; and Vila, L., eds. 2005. *Handbook of Temporal Reasoning in Artificial Intelligence*. Elsevier.

Gelfond, M., and Lifschitz, V. 1988. The stable model semantics for logic programming. In *Proc. of the Fifth Logic Programming Symposium*, 1070–1080. MIT Press.

Georgeff, M., and Rao, A. 1995. The semantics of intention maintenance for rational agents. In *Proc. of IJCAI-95*, 704–710.

Haykin, S. 1999. *Neural Networks: A Comprehensive Foundation*. Prentice Hall.

Page, M. 2000. Connectionist modelling in psychology: A localist manifesto. *Behavioral and Brain Sciences* 23:443–467.

Rumelhart, D.; Hinton, G.; and Williams, R. 1986. Learning internal representations by error propagation. In *Parallel Distributed Processing*, volume 1. MIT Press. 318–362.

Shastri, L. 1999. Advances in SHRUTI: a neurally motivated model of relational knowledge representation and rapid inference using temporal synchrony. *Applied Intelligence* 11:79–108.

Siegelmann, H.; Horne, B.; and Giles, C. L. 1995. Computational capabilities of recurrent NARX neural networks. Technical report, U. of Maryland, UMIACS-TR-95-12.

Smolensky, P., and Legendre, G. 2006. *The Harmonic Mind: From Neural Computation to Optimality-Theoretic Grammar*. MIT Press.

Sun, R. 1995. Robust reasoning: integrating rule-based and similarity-based reasoning. *Artificial Intelligence* 75(2):241–296.

Touretzky, D., and Hinton, G. 1985. Symbols among neurons. In *Proc. of IJCAI-85*, 238–243.

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

Valiant, L. 2000. A neuroidal architecture for cognitive computation. *Journal of the ACM* 47(5):854–882.

Valiant, L. 2003. Three problems in computer science. *Journal of the ACM* 50(1):96–99.