Available online at www.sciencedirect.com
ScienceDirect
Theoretical Computer Science 371 (2007) 34–53
Theoretical Computer Science
www.elsevier.com/locate/tcs

Connectionist modal logic: Representing modalities in neural networks

Artur S. d'Avila Garcez[a,*], Luís C. Lamb[b], Dov M. Gabbay[c]

[a] *Department of Computing, City University, London EC1V 0HB, UK*
[b] *Institute of Informatics, Federal University of Rio Grande do Sul, Porto Alegre, RS 91501-970, Brazil*
[c] *Department of Computer Science, King's College, London WC2R 2LS, UK*

Abstract

Modal logics are amongst the most successful applied logical systems. Neural networks were proved to be effective learning systems. In this paper, we propose to combine the strengths of modal logics and neural networks by introducing *Connectionist Modal Logics* (CML). CML belongs to the domain of neural-symbolic integration, which concerns the application of problem-specific symbolic knowledge within the neurocomputing paradigm. In CML, one may represent, reason or learn modal logics using a neural network. This is achieved by a *Modalities Algorithm* that translates modal logic programs into neural network ensembles. We show that the translation is sound, i.e. the network ensemble computes a fixed-point meaning of the original modal program, acting as a distributed computational model for modal logic. We also show that the fixed-point computation terminates whenever the modal program is well-behaved. Finally, we validate CML as a computational model for integrated knowledge representation and learning by applying it to a well-known testbed for distributed knowledge representation. This paves the way for a range of applications on integrated knowledge representation and learning, from practical reasoning to evolving multi-agent systems.

Keywords: Models of computation; Neural-symbolic learning systems; Knowledge representation; Modal logics; Artificial neural networks

1. Introduction

Neural-Symbolic integration concerns the application of problem-specific symbolic knowledge within the neurocomputing paradigm. In contrast with symbolic learning systems, neural networks encode patterns and their generalisations implicitly in a set of weights, so reflecting the statistical properties of the training data [3]. The merging of theory (background knowledge) and data learning (learning by examples) in neural networks has been indicated as providing learning systems that are more effective than purely symbolic or purely connectionist systems, especially when data are noisy [39,40]. In order to merge theory and data learning, one first translates the background knowledge into the initial architecture of a neural network, and then trains the network with examples (e.g., using

* Corresponding address: Department of Computing, Northampton Square, School of Informatics, City University London, London, EC1V 0HB, UK. Tel.: + 44 20 7040 8344; fax: +44 20 7040 8587.
E-mail addresses: aag@soi.city.ac.uk (A.S. d'Avila Garcez), LuisLamb@acm.org (L.C. Lamb), dov.gabbay@kcl.ac.uk (D.M. Gabbay).

0304-3975/$ - see front matter © 2006 Elsevier B.V. All rights reserved.
doi:10.1016/j.tcs.2006.10.023

backpropagation—the neural learning algorithm most successfully applied to real-world problems such as DNA sequence analysis and pattern recognition problems [26,34]).

Neural-symbolic learning systems [15] can exploit the massively parallel architecture of neural networks and are capable of learning from examples and background knowledge (incomplete symbolic descriptions of the problem). However, most of the efforts so far have been directed towards the representation of classical logic and logic programming in connectionist settings [1,36–38]. In particular, neural systems have not been shown able to fully represent and learn expressive languages such as modal and predicate logics [10].

In this paper, we propose a new approach for the representation and learning of propositional modal logics in neural networks, namely, *Connectionist Modal Logic* (CML). The approach allows for the solution of distributed knowledge representation problems in neural networks by exploiting any available background knowledge specified as a modal logic program. We use the language of modal logic programming [33,35] extended to allow modalities such as *necessity* and *possibility* in the head of clauses. We then present an algorithm that sets up an ensemble of *Connectionist Inductive Learning and Logic Programming* (C-ILP) networks [15] representing the modal clauses. A theorem then shows that the resulting network ensemble computes a fixed-point semantics of the original modal program. In other words, the network ensemble can be used as a massively parallel system for modal logic program representation and reasoning. We validate the system by applying it to the muddy children puzzle, a well-known problem in the domain of distributed knowledge representation [19,29].

Learning in the CML system is achieved by using backpropagation for training each individual network in the ensemble, which in turn corresponds to the current knowledge of an agent within a possible world. This will be exemplified in this paper with the use of the muddy children puzzle [19].

We argue that CML renders neural-symbolic learning systems with the ability to provide a better balance between expressive power and computational feasibility, due to the use of a more expressive, yet computationally tractable, knowledge representation language. The well-established translation between propositional modal logic and the two-variable fragment of first order logic [44] indicates that neural-symbolic learning systems may go beyond propositional logic.[1]

As argued in [13,17], we believe that the combination of non-classical logics and neural networks may provide the way forward towards the provision of an integrated system of expressive reasoning and robust learning. The provision of such a system, integrating the two most fundamental phenomena of intelligent cognitive behaviour (i.e. the ability to learn from experience and the ability to reason from what has been learned) has been identified by Valiant as a key challenge for Computer Science [41]. Ultimately, our goal is to produce biologically plausible models with integrated reasoning and learning capabilities, in which neural networks provide the inspiration and the machinery necessary for cognitive computation and learning, while non-classical logics provide practical reasoning and explanation capabilities to the models, facilitating the interaction between them and the outside world.

In Section 2, we briefly present the basic concepts of modal logic and artificial neural networks used throughout the paper. In Section 3, we present the Modalities Algorithm that translates extended modal programs into artificial neural networks. The networks obtained are ensembles of C-ILP networks, each representing a (learnable) possible world. We then show that the networks compute a fixed-point semantics of the given modal theory, thus proving soundness of the Modalities Algorithm. We also prove termination of the Modalities Algorithm. In Section 4, we apply the system to the muddy children puzzle and report on the effectiveness of CML w.r.t. reasoning and learning. In Section 5, we conclude and discuss directions for future work.

2. Preliminaries

In this section, we present some basic concepts of Modal Logic and Artificial Neural Networks that will be used throughout the paper.

[1] In [44], p. 2, Vardi states that "(propositional) modal logic, in spite of its apparent propositional syntax, is essentially a first-order logic, since the necessity and possibility modalities quantify over the set of possible worlds"; and in [44], p. 7, "the states in a Kripke structure correspond to domain elements in a relational structure, and modalities are nothing but a limited form of quantifiers". A comprehensive treatment of this subject, including the study of correspondences between propositional modal logics and (fragments of) first order logic, can be found in [42,43].

2.1. Modal logic and extended modal programs

Modal logic began with the analysis of concepts such as necessity and possibility from a philosophical perspective [28,30]. A main feature of modal logic is the use of *possible world semantics* (proposed by Kripke and Hintikka), which has significantly contributed to the development of new models for non-classical logics, many of which have had a great impact in computer science. In modal logic, a proposition is necessary in a world if it is true in all worlds which are possible in relation to that world, whereas it is possible in a world if it is true in at least one world which is possible in relation to that same world. This is expressed in the semantics formalisation by a (binary) relation between possible worlds.

Modal logic was found to be appropriate in the study of mathematical necessity (in the logic of provability), time, knowledge, belief, obligation and other concepts and modalities [8]. In artificial intelligence and computing, modal logics are among the most employed formalisms to analyse and represent reasoning in multi-agent systems and concurrency properties [19]. The basic modal logic definitions that we use in this paper are as follows. As usual, the language of propositional modal logic extends the language of propositional logic with the $\Box$ (*necessity*) and $\Diamond$ (*possibility*) operators.

Definition 1. A *modal atom* is of the form MA where $M \in \{\Box, \Diamond\}$ and A is an atom. A *modal literal* is of the form ML where L is a literal.

Definition 2. A *modal program* is a finite set of clauses of the form $\alpha_1 \wedge \cdots \wedge \alpha_n \rightarrow \alpha_{n+1}$, where α_i $(1 \leq i \leq n)$ is either an atom or a modal atom, and α_{n+1} is an atom.

We define *extended modal programs* as modal programs extended to allow single modalities $\Box$ and $\Diamond$ in the head of clauses, thus extending Sakakibara's *modal logic programming* [33,35]. In addition, each clause is labelled by the possible world in which it holds, similarly to Gabbay's *Labelled Deductive Systems* [22].

Definition 3. An *extended modal program* is a finite set of clauses C of the form $\omega_i : \beta_1 \wedge \cdots \wedge \beta_n \rightarrow \beta_{n+1}$, where ω_i is a label representing a world in which the associated clause holds, β_i $(1 \leq i \leq n)$ is either a literal or a modal literal, and β_{n+1} is either an atom or a modal atom, and a finite set of relations $\mathcal{R}(\omega_i, \omega_j)$ between worlds ω_i and ω_j.

For example: $\mathcal{P} = \{\omega_1 : r \rightarrow \Box q;\ \omega_1 : \Diamond s \rightarrow r;\ \omega_2 : s;\ \omega_3 : q \rightarrow \Diamond p;\ \mathcal{R}(\omega_1, \omega_2),\ \mathcal{R}(\omega_1, \omega_3)\}$ is an extended modal program.[2]

Formulas in modal logic programming will be interpreted in Kripke models, which are defined as follows.

Definition 4 (*Kripke Models*). Let $\mathcal{L}$ be a modal language. A *Kripke model* for $\mathcal{L}$ is a tuple $\mathcal{M} = \langle \Omega, \mathcal{R}, v \rangle$ where Ω is a set of possible worlds, v is a mapping that assigns to each propositional letter of $\mathcal{L}$ a subset of Ω, and $\mathcal{R}$ is a binary relation over Ω.

A modal formula φ is said to be true at a possible world ω of a model $\mathcal{M}$, written $(\mathcal{M}, \omega) \models \varphi$ if the following satisfiability condition holds.

Definition 5 (*Satisfiability of Modal Formulas*). Let $\mathcal{L}$ be a modal language, and let $\mathcal{M} = \langle \Omega, \mathcal{R}, v \rangle$ be a Kripke Model. The satisfiability relation $\models$ is uniquely defined as follows:

(i) $(\mathcal{M}, \omega) \models p$ iff $\omega \in v(p)$ for a propositional letter p
(ii) $(\mathcal{M}, \omega) \models \neg\varphi$ iff $(\mathcal{M}, \omega) \not\models \varphi$
(iii) $(\mathcal{M}, \omega) \models \varphi \wedge \psi$ iff $(\mathcal{M}, \omega) \models \varphi$ and $(\mathcal{M}, \omega) \models \psi$
(iv) $(\mathcal{M}, \omega) \models \varphi \vee \psi$ iff $(\mathcal{M}, \omega) \models \varphi$ or $(\mathcal{M}, \omega) \models \psi$
(v) $(\mathcal{M}, \omega) \models \varphi \rightarrow \psi$ iff $(\mathcal{M}, \omega) \not\models \varphi$ or $(\mathcal{M}, \omega) \models \psi$
(vi) $(\mathcal{M}, \omega) \models \Box\varphi$ iff for all $\omega_i \in \Omega$, if $\mathcal{R}(\omega, \omega_i)$ then $(\mathcal{M}, \omega_i) \models \varphi$
(vii) $(\mathcal{M}, \omega) \models \Diamond\varphi$ iff there exists a ω_i such that $\mathcal{R}(\omega, \omega_i)$ and $(\mathcal{M}, \omega_i) \models \varphi$.

[2] In extended logic programming, one may also allow the use of explicit (classical) negation [24]. We use $\sim$ to refer to default negation [9], and $\neg$ to refer to explicit negation. Following [24], any atom A preceded by $\neg$ is renamed as a new atom A^* not present in the language. For example, $p \wedge \sim \neg q \rightarrow \neg r$ becomes $p \wedge \sim q^* \rightarrow r^*$.

A variety of proof methods for modal logics have been developed over the years, e.g. [6,20]. In some of these, formulas are labelled by the worlds in which they hold, thus facilitating the reasoning process (see [6] for a discussion on this topic). In the natural deduction-style rules for modal reasoning shown in Table 1, the notation $\omega : \varphi$ means that the formula φ holds at the possible world ω. Moreover, the explicit reference to the accessibility relation helps in deriving what formulas hold in the worlds which are related by $\mathcal{R}$. The rules we shall represent in CML are similar to the ones reproduced in Table 1, obtained from [6].

Table 1
Rules for modality operators

$$\frac{\begin{array}{c}[R(\omega, g_\varphi(\omega))] \\ \vdots \\ g_\varphi(\omega) : \varphi\end{array}}{\omega : \Box\varphi}\,\Box I \qquad \frac{\omega_1 : \Box\varphi,\ R(\omega_1, \omega_2)}{\omega_2 : \varphi}\,\Box E$$

$$\frac{\omega : \Diamond\varphi}{f_\varphi(\omega) : \varphi,\ R(\omega, f_\varphi(\omega))}\,\Diamond E \qquad \frac{\omega_2 : \varphi,\ R(\omega_1, \omega_2)}{\omega_1 : \Diamond\varphi}\,\Diamond I$$

The $\Diamond E$ rule can be seen (informally) as a *skolemisation* of the existential quantifier over possible worlds, which is semantically implied by the formula $\Diamond\varphi$ in the premise. The term $f_\varphi(\omega)$ defines a particular possible world uniquely associated with the formula φ, and inferred to be accessible from the possible world ω (i.e. $\mathcal{R}(\omega, f_\varphi(\omega))$). In the $\Box I$ rule, the (temporary) assumption $[\mathcal{R}(\omega, g_\varphi(\omega))]$ should be read as: given an arbitrary accessible world $g_\varphi(\omega)$, if one can derive $g_\varphi(\omega) : \varphi$ then it is possible to show that $\Box\varphi$ holds at ω. The rule for $\Diamond I$ represents that if we have a relation $\mathcal{R}(\omega_1, \omega_2)$, and if φ holds in ω_2 then it must be the case that $\Diamond\varphi$ holds in ω_1. The rule $\Box E$ represents that if $\Box\varphi$ holds in a world ω_1, and ω_1 is related to ω_2, then we can infer that φ holds in ω_2.

Semantics for extended modal logic programs

In what follows, we define a model-theoretic semantics for extended modal programs. According to the rules for modalities given above, we will deal with $\Diamond$ by making a choice of world ω_j in which to have A when $\Diamond A$ is true in ω_i and $\mathcal{R}(\omega_i, \omega_j)$. In this paper, we choose an arbitrary world (i.e. one that is uniquely associated with A). In practice, one may opt to manage several neural networks, one for each choice, in the same way that one may opt to manage several graphs as in the (modal) tableaux prover LoTREC [7]. Under any choice, if the program is well-behaved (e.g., in the sense of Fitting's metric methods [21]), we should be able to prove that the computation terminates with our neural network converging to a fixed-point of the meaning operator.

When computing the fixed-point, we have to consider the consequences derived locally and the consequences derived from the interaction between worlds. Locally, fixed-points are computed as in the stable model semantics for logic programming, by simply renaming each modal literal ML_i by a new literal L_j not in the language $\mathcal{L}$, and applying the Gelfond–Lifschitz transformation [4]. When considering interacting worlds, there are four more cases to be addressed, according to the rules in Table 1.

Hence, we proceed as follows. Given an extended modal program, for each literal of the form $\Diamond L$ in the head of a clause, we choose a world and *connect* (in a sense that will become clear soon) $\Diamond L$ to literal L in this world. For each literal of the form $\Box L$, we connect $\Box L$ to literals L in every world related to that of $\Box L$, and similarly for the other rules. The definition of modal consequence operator below captures this.

Definition 6 (*Modal Immediate Consequence Operator*). Let $\mathcal{P} = \{\mathcal{P}_1, \ldots, \mathcal{P}_k\}$ be an extended modal program, where each $\mathcal{P}_i$ is the set of modal clauses that hold in a world ω_i $(1 \leq i \leq k)$. Let $B_\mathcal{P}$ denote the set of (modal) atoms occurring in $\mathcal{P}$ (i.e. the Herbrand base of $\mathcal{P}$), and I a Herbrand interpretation for $\mathcal{P}$. Let $\omega_i : \alpha$ denote an atom or a modal atom α holding in world ω_i. The mapping $MT_\mathcal{P} : 2^{B_\mathcal{P}} \to 2^{B_\mathcal{P}}$ is defined as follows: $MT_\mathcal{P}(I) = \{\omega_i : \alpha \in B_\mathcal{P} \mid$ either (a), (b), (c), (d) or (e) below holds$\}$.
(a) $\omega_i : \beta_1, \ldots, \beta_n \to \alpha$ is a clause in $\mathcal{P}$ and $\{\beta_1, \ldots, \beta_n\} \subseteq I$;
(b) $\omega_i : \alpha$ is of the form $\omega_i : A$, ω_i is of the type $f_A(\omega_k)$ (i.e. ω_i is a particular possible world uniquely associated with A), and there exists a world ω_k such that $\mathcal{R}(\omega_k, \omega_i)$, and $\omega_k : \beta_1, \ldots, \beta_m \to \Diamond A$ is a clause in $\mathcal{P}$ with $\{\beta_1, \ldots, \beta_m\} \subseteq I$;

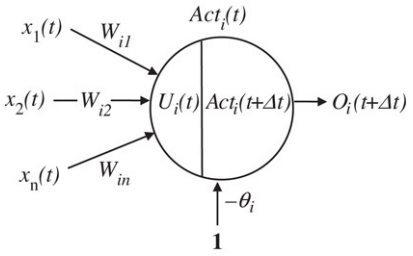

Fig. 1. The processing unit or neuron.

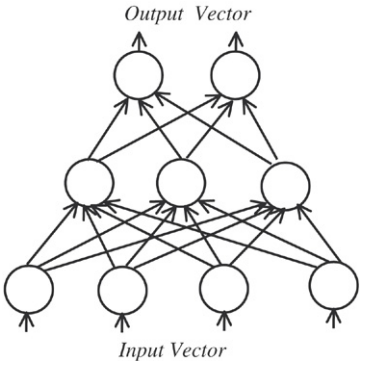

Fig. 2. A typical feedforward neural network.

(c) $\omega_i : \alpha$ is of the form $\omega_i : \Diamond A$ and there exists a world ω_j such that $\mathcal{R}(\omega_i, \omega_j)$, and $\omega_j : \beta_1, \ldots, \beta_m \rightarrow A$ is a clause in $\mathcal{P}$ with $\{\beta_1, \ldots, \beta_m\} \subseteq I$;
(d) $\omega_i : \alpha$ is of the form $\omega_i : \Box A$ and for each world ω_j such that $\mathcal{R}(\omega_i, \omega_j)$, $\omega_j : \beta_1, \ldots, \beta_o \rightarrow A$ is a clause in $\mathcal{P}$ with $\{\beta_1, \ldots, \beta_o\} \subseteq I$;
(e) $\omega_i : \alpha$ is of the form $\omega_i : A$ and there exists a world ω_k such that $\mathcal{R}(\omega_k, \omega_i)$, and $\omega_k : \beta_1, \ldots, \beta_o \rightarrow \Box A$ is a clause in $\mathcal{P}$ with $\{\beta_1, \ldots, \beta_o\} \subseteq I$.

2.2. Artificial neural networks

An artificial neural network is a directed graph. A unit in this graph is characterised, at time t, by its input vector $I_i(t)$, its input potential $U_i(t)$, its activation state $\mathrm{Act}_i(t)$, and its output $O_i(t)$. The units (neurons) of the network are interconnected via a set of directed and weighted connections. If there is a connection from unit i to unit j, then $W_{ji} \in \mathbb{R}$ denotes the *weight* associated with such a connection.

We start by characterising the neuron's *functionality* (see Fig. 1). The *activation state* of a neuron i at time t ($\mathrm{Act}_i(t)$) is a bounded real or integer number. The output of neuron i at time t ($O_i(t)$) is given by an *output rule* f_i such that $O_i(t) = f_i(\mathrm{Act}_i(t))$. The input potential of neuron i at time t ($U_i(t)$) is obtained by applying a *propagation rule* (g_i) such that $U_i(t) = g_i(I_i(t), W_i)$, where $I_i(t)$ contains the input signals $(x_1(t), x_2(t), \ldots, x_n(t))$ to neuron i at time t, and W_i denotes the weight vector $(W_{i1}, W_{i2}, \ldots, W_{in})$ to neuron i. In addition, θ_i (an extra weight with input always fixed at 1) is known as the *threshold* of neuron i. Finally, the neuron's new activation state $\mathrm{Act}_i(t + \Delta t)$ is given by its *activation rule* h_i, which is a function of the neuron's current activation state and input potential, i.e. $\mathrm{Act}_i(t + \Delta t) = h_i(\mathrm{Act}_i(t), U_i(t))$, and the neuron's new output value $O_i(t + \Delta t) = f_i(\mathrm{Act}_i(t + \Delta t))$.

In general, h_i does not depend on the previous activation state of the unit, that is, $\mathrm{Act}_i(t + \Delta t) = h_i(U_i(t))$, the propagation rule g_i is a weighted sum, such that $U_i(t) = \sum_j W_{ij} x_j(t)$, and the output rule f_i is given by the identity function, i.e. $O_i(t) = \mathrm{Act}_i(t)$.

The units of a neural network can be organised in layers. A *n-layer feedforward network* N is an acyclic graph. It consists of a sequence of layers and connections between consecutive layers, containing one input layer, $n - 2$ hidden layers and one output layer, where $n \geq 2$. When $n = 3$, we say that N is a *single hidden layer network*. When each unit occurring in the i-th layer is connected to each unit occurring in the $i + 1$-th layer, we say that N is a *fully-connected network* (see Fig. 2).

The most interesting properties of a neural network do not arise from the functionality of each neuron, but from the collective effect resulting from the interconnection of units. Let r and s be the number of units occurring, respectively, in the input and output layers of a multilayer feedforward network. The network computes a function $f : \mathbb{R}^r \to \mathbb{R}^s$ as follows. The input vector is presented to the input layer at time t_1 and propagated through the hidden layers to the output layer. At each time point, all units update their input potential and activation state synchronously. At time t_n the output vector is read off the output layer. In addition, most neural models have a *learning rule*, responsible for changing the weights of the network so that it learns to approximate f given a number of *training examples* (input vectors and their respective target output vectors). The idea is to *minimise the error* associated with the set of examples by performing small changes to the network's weights. In the case of *backpropagation* [34], the learning process occurs as follows: given an input vector $\mathbf{i}$ and corresponding target vector $\mathbf{t}$, the network's output $\mathbf{o} = f(\mathbf{i})$ may be compared with the target, and an error such as

$$\mathrm{Err}(\mathbf{W}) = \frac{1}{2}\sum_i (\mathbf{o} - \mathbf{t})^2 \tag{1}$$

over a number i of examples ($(\mathbf{i}, \mathbf{t})$ pairs) can be computed. This error may be minimised by *gradient descent*, i.e. by the iterative application of changes

$$\Delta\mathbf{W} = -\eta \cdot \nabla_{\mathbf{W}} \cdot \mathrm{Err}(\mathbf{W}) \tag{2}$$

to the weight vector $\mathbf{W}$, where $\eta > 0$ is called the network's *learning rate* and

$$\nabla_{\mathbf{W}} = \left(\frac{\partial \mathrm{Err}(\mathbf{W})}{\partial W_{11}}, \frac{\partial \mathrm{Err}(\mathbf{W})}{\partial W_{12}}, \ldots, \frac{\partial \mathrm{Err}(\mathbf{W})}{\partial W_{ij}}\right). \tag{3}$$

Backpropagation training may lead to a local rather than a global error minimum. In an attempt to ameliorate this problem and also improve training time, a *term of momentum* can be added to the learning process. The term of momentum allows a network to respond not only to the local gradient, but also to recent trends in the error surface, acting as a low pass filter.

Momentum is added to backpropagation learning by making weight changes equal to the sum of a fraction of the last weight change and the new change suggested by the backpropagation rule. Eq. (4) shows how backpropagation with momentum is expressed mathematically

$$\Delta\mathbf{W}(i) = -\eta \cdot \nabla_{\mathbf{W}(i)} \cdot \mathrm{Err}(\mathbf{W}(i)) + \xi\,\Delta\mathbf{W}(i-1) \tag{4}$$

where $\xi\,\Delta W(i-1)$ is the term of momentum and $0 < \xi < 1$ is the momentum constant. Typically, $\xi = 0.9$.

The ultimate measure of the success of a neural network should not be how closely the network approximates the training data, but how well it accounts for yet unseen cases, i.e. how well the network generalises to new data. In order to evaluate the network's *generalisation*, the set of examples is commonly partitioned into a *training set* and a *testing set*, as detailed in Section 4.2 for the muddy children puzzle example.

In this paper, we concentrate on single hidden layer feedforward networks, which are universal approximators [11] and have typically been used in a number of practical applications. We use a *bipolar* semi-linear activation function $h(x) = \frac{2}{1+e^{-\beta x}} - 1$ with inputs in $\{-1, 1\}$, and the *backpropagation* learning algorithm to perform training from examples.

3. Connectionist modal logic

In this section, we introduce CML. We shall use ensembles of C-ILP networks (described in detail in Section 3.1) as the underlying architecture to represent modal theories. We then present an efficient translation algorithm from extended modal programs to neural network ensembles.

Let us start with a simple example. It briefly illustrates how an ensemble of C-ILP networks can be used for modelling non-classical reasoning with modal logic. Input and output neurons may represent $\Box L$, $\Diamond L$ or L, where L is a literal.

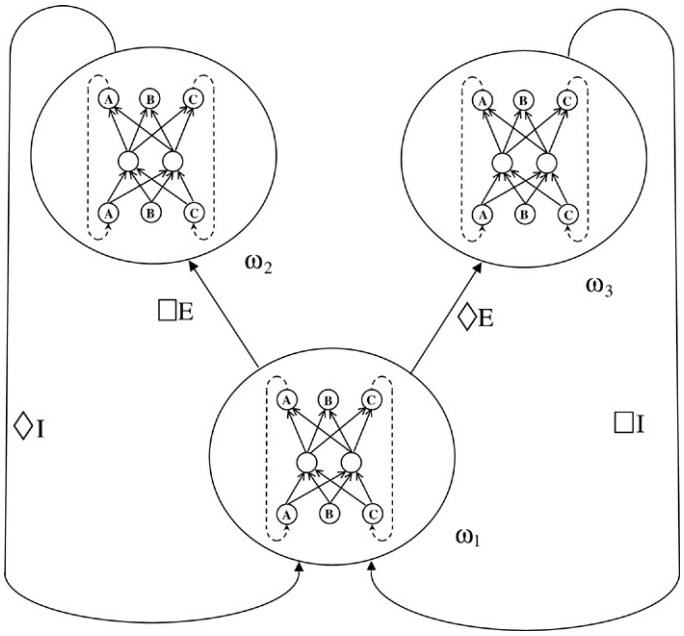

Fig. 3. An ensemble of networks representing modalities.

Example 7. Fig. 3 shows an ensemble of three C-ILP networks ($\omega_1, \omega_2, \omega_3$), which might communicate in different ways. The idea is to see ω_1, ω_2 and ω_3 as *possible worlds,* and to incorporate modalities into the language of C-ILP by connecting the neurons in the different networks according to the rules of Table 1. For example, a rule (i) "If $\omega_1 : \Box A$ then $\omega_2 : A$" could be implemented by connecting neuron $\Box A$ in ω_1 to neuron A in ω_2 such that, whenever $\Box A$ is activated in ω_1, A is activated in ω_2. Similarly, (ii) "If $(\omega_2 : A)$ or $(\omega_3 : A)$ then $\omega_1 : \Diamond A$" could be implemented by connecting neurons A of ω_2 and ω_3 to neuron $\Diamond A$ of ω_1 (through a hidden neuron in ω_1) such that, whenever A is activated in either ω_2 or ω_3, $\Diamond A$ is activated in ω_1. Examples (i) and (ii) simulate, in a finite universe, the rules of $\Box$ *Elimination* and $\Diamond$ *Introduction* (see Table 1). The representation of such modalities in neural networks will be described in detail in Section 3.2.

Due to the simplicity of each C-ILP network, e.g. ω_1 in Fig. 3, one may perform inductive learning within each *possible world* using standard backpropagation. As a result, the main problem to be tackled when it comes to learning is how to set up the connections that establish the necessary communication between networks, e.g. ω_1 and ω_2. As mentioned above, in the case of modal logic, such connections may be defined by the modal rules for natural deduction given in Table 1. The *Modalities Algorithm* presented in Section 3.2 will implement those rules.

3.1. The C-ILP system

C-ILP [15,18] is a massively parallel computational model based on a feedforward artificial neural network. It integrates inductive learning by examples and background knowledge with deductive learning using logic programming. A *Translation Algorithm* maps a general logic program[3] $\mathcal{P}$ into a single hidden layer neural network $\mathcal{N}$ such that $\mathcal{N}$ computes the least fixed-point of $\mathcal{P}$ (see also [27]). In addition, $\mathcal{N}$ can be trained by examples using *backpropagation* [34], and having $\mathcal{P}$ as background knowledge. The knowledge acquired by training can then be extracted [14], closing the learning cycle (as in [40]).

Let us exemplify how the C-ILP *Translation Algorithm* works. Each clause (r_l) of $\mathcal{P}$ is mapped from the input layer to the output layer of $\mathcal{N}$ through one neuron (N_l) in the single hidden layer of $\mathcal{N}$. Intuitively, the *Translation Algorithm* from $\mathcal{P}$ to $\mathcal{N}$ has to implement the following conditions: (C1) The input potential of a hidden neuron (N_l)

[3] Recall that a general clause is a rule of the form $L_1, \ldots, L_k \rightarrow A$, where A is an atom and L_i $(1 \leq i \leq k)$ is a literal (an atom or the negation of an atom). A general logic program is a finite set of general clauses [31].

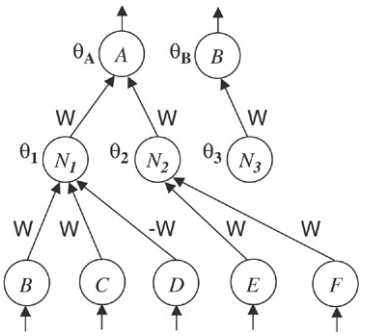

Fig. 4. Sketch of neural network $\mathcal{N}$ for logic program $\mathcal{P}$.

can only exceed N_l's threshold (θ_l), activating N_l, when all the positive antecedents of r_l are assigned the truth-value *true* while all the negative antecedents of r_l are assigned *false*; and (C2) The input potential of an output neuron (A) can only exceed A's threshold (θ_A), activating A, when at least one hidden neuron N_l that is connected to A is activated.

Example 8. Consider the logic program $\mathcal{P} = \{B;\ B \wedge C \wedge \sim D \rightarrow A;\ E \wedge F \rightarrow A\}$. The *Translation Algorithm* derives the network $\mathcal{N}$ of Fig. 4, setting weights (W) and thresholds (θ) in such a way that conditions (C1) and (C2) above are satisfied. Note that, if $\mathcal{N}$ is to be fully-connected, any other link (not shown in Fig. 4) should receive weight zero initially.

Note that, in Example 8, each input and output neuron of $\mathcal{N}$ is associated with an atom of $\mathcal{P}$. As a result, each input and output vector of $\mathcal{N}$ can be associated with an interpretation for $\mathcal{P}$. Note also that each hidden neuron N_l corresponds to a clause r_l of $\mathcal{P}$. In order to compute a fixed-point semantics of $\mathcal{P}$, output neuron B should feed input neuron B such that $\mathcal{N}$ is used to iterate $T_{\mathcal{P}}$, the fixed-point operator[4] of $\mathcal{P}$ [27]. $\mathcal{N}$ will eventually converge to a stable state which is identical to the stable model of $\mathcal{P}$ [18]. Let us recall the C-ILP translation algorithm.

Notation. Given a general logic program $\mathcal{P}$, let q denote the number of clauses r_l ($1 \leq l \leq q$) occurring in $\mathcal{P}$; υ, the number of literals occurring in $\mathcal{P}$; $A_{\min}$, the minimum activation value for a neuron to be *active* (or, analogously, for its associated literal to be assigned truth-value *true*), $A_{\min} \in (0, 1)$; $A_{\max}$, the maximum activation value when a neuron is not active (or when its associated literal is *false*), $A_{\max} \in (-1, 0)$; $h(x) = \frac{2}{1+e^{-\beta x}} - 1$, the bipolar semi-linear activation function[5]; $g(x) = x$, the standard linear activation function; $s(x) = y$, the standard nonlinear activation function ($y = 1$ if $x > 0$; and $y = 0$ otherwise), also known as the step function; W (resp. $-W$), the weight of connections associated with positive (resp. negative) literals; θ_l, the threshold of hidden neuron N_l associated with clause r_l; θ_A, the threshold of output neuron A, where A is the head of clause r_l; k_l, the number of literals in the body of clause r_l; p_l, the number of positive literals in the body of clause r_l; n_l, the number of negative literals in the body of clause r_l; μ_l, the number of clauses in $\mathcal{P}$ with the same atom in the head, for each clause r_l; $\text{MAX}_{r_l}(k_l, \mu_l)$, the greater element between k_l and μ_l for clause r_l; and $\text{MAX}_{\mathcal{P}}(k_1, \ldots, k_q, \mu_1, \ldots, \mu_q)$, the greatest element among all k's and μ's for $\mathcal{P}$. We also use $\overrightarrow{k}$ as a shorthand for $(k_1, \ldots, k_q)$, and $\overrightarrow{\mu}$ as a shorthand for $(\mu_1, \ldots, \mu_q)$.

For instance, for the program $\mathcal{P}$ of Example 8, $q = 3$, $\upsilon = 6$, $k_1 = 3$, $k_2 = 2$, $k_3 = 0$, $p_1 = 2$, $p_2 = 2$, $p_3 = 0$, $n_1 = 1$, $n_2 = 0$, $n_3 = 0$, $\mu_1 = 2$, $\mu_2 = 2$, $\mu_3 = 1$, $\text{MAX}_{r_1}(k_1, \mu_1) = 3$, $\text{MAX}_{r_2}(k_2, \mu_2) = 2$, $\text{MAX}_{r_3}(k_3, \mu_3) = 1$, and $\text{MAX}_{\mathcal{P}}(k_1, k_2, k_3, \mu_1, \mu_2, \mu_3) = 3$.

In the *Translation Algorithm* below, we define $A_{\min}$, W, θ_l, and θ_A such that conditions (C1) and (C2) above are satisfied. Eqs. (5)–(8) below are obtained from the proof of Theorem 9 [18]. We assume, for mathematical convenience and without loss of generality, that $A_{\max} = -A_{\min}$. In this way, we associate truth-value *true* with values in the interval $(A_{\min}, 1)$, and truth-value *false* with values in the interval $(-1, -A_{\min})$. Theorem 9 guarantees that values in

[4] The mapping $T_{\mathcal{P}}$ is defined as follows: Let I be a Herbrand *interpretation,* then $T_{\mathcal{P}}(I) = \{A_0 \mid L_1, \ldots, L_n \rightarrow A_0$ is a ground clause in $\mathcal{P}$ and $\{L_1, \ldots, L_n\} \subseteq I\}$.

[5] We use the bipolar semi-linear activation function for convenience. Any monotonically increasing activation function could have been used here.

the interval $[-A_{\min}, A_{\min}]$ do not occur in the network with weights W and thresholds θ, but informally this interval may be associated with a third truth-value *unknown*.[6]

We start by calculating $\mathrm{MAX}_{\mathcal{P}}(\overrightarrow{k}, \overrightarrow{\mu})$ of $\mathcal{P}$ and $A_{\min}$ such that:

$$A_{\min} > \frac{\mathrm{MAX}_{\mathcal{P}}(\overrightarrow{k}, \overrightarrow{\mu}) - 1}{\mathrm{MAX}_{\mathcal{P}}(\overrightarrow{k}, \overrightarrow{\mu}) + 1} \tag{5}$$

Translation algorithm

(1) Calculate the value of W such that the following is satisfied:

$$W \geq \frac{2}{\beta} \cdot \frac{\ln(1 + A_{\min}) - \ln(1 - A_{\min})}{\mathrm{MAX}_{\mathcal{P}}(\overrightarrow{k}, \overrightarrow{\mu})(A_{\min} - 1) + A_{\min} + 1}; \tag{6}$$

(2) For each clause r_l of $\mathcal{P}$ of the form $L_1, \ldots, L_k \rightarrow A (k \geq 0)$:
   (a) Create input neurons $L_1, \ldots, L_k$ and output neuron A in $\mathcal{N}$ (if they do not exist yet);
   (b) Add a neuron N_l to the hidden layer of $\mathcal{N}$;
   (c) Connect each neuron L_i $(1 \leq i \leq k)$ in the input layer to the neuron N_l in the hidden layer. If L_i is a positive literal then set the connection weight to W; otherwise, set the connection weight to $-W$;
   (d) Connect the neuron N_l in the hidden layer to the neuron A in the output layer and set the connection weight to W;
   (e) Define the threshold (θ_l) of the neuron N_l in the hidden layer as:

$$\theta_l = \frac{(1 + A_{\min})(k_l - 1)}{2} W. \tag{7}$$

   (f) Define the threshold (θ_A) of the neuron A in the output layer as:

$$\theta_A = \frac{(1 + A_{\min})(1 - \mu_l)}{2} W. \tag{8}$$

(3) Set $g(x)$ as the activation function of the neurons in the input layer of $\mathcal{N}$. In this way, the activation of the neurons in the input layer, given by each input vector $\mathbf{i}$, will represent an interpretation for $\mathcal{P}$.
(4) Set $h(x)$ as the activation function of the neurons in the hidden and output layers of $\mathcal{N}$. In this way, a gradient descent learning algorithm, such as *backpropagation*, can be applied to $\mathcal{N}$ efficiently.

Theorem 9 (*[18]*). *For each propositional general logic program $\mathcal{P}$, there exists a feedforward artificial neural network $\mathcal{N}$ with exactly one hidden layer and semi-linear neurons such that $\mathcal{N}$ computes the fixed-point operator $T_{\mathcal{P}}$ of $\mathcal{P}$.*

Now, let $T^n_{\mathcal{P}} \stackrel{\text{def}}{=} T_{\mathcal{P}}(T^{n-1}_{\mathcal{P}})$ with $T^0_{\mathcal{P}} \stackrel{\text{def}}{=} T_{\mathcal{P}}(\{\emptyset\})$. Say that $\mathcal{P}$ is *well-behaved* if, after a finite number m of iterations, $T^m_{\mathcal{P}} = T^{m-1}_{\mathcal{P}}$. It is not difficult to see that if $\mathcal{P}$ is well-behaved and we use $\mathcal{N}$ to iterate $T_{\mathcal{P}}$ then $\mathcal{N}$ will converge to $T^m_{\mathcal{P}}$, as follows. Consider a feedforward neural network $\mathcal{N}$ with p input neurons $(i_1, \ldots, i_p)$ and q output neurons $(o_1, \ldots, o_q)$. Assume that each input and output neuron in $\mathcal{N}$ is labelled by an atom A_k associated with it. Let us use $\mathrm{name}(i_i) = \mathrm{name}(o_j)$ to denote the fact that the literal associated with neuron i_i is the same as the literal associated with neuron o_j.

Let:

$$\mathrm{valuation}(\mathrm{Act}(x)) = \begin{cases} 1, & \text{if } \mathrm{Act}(x) > A_{\min}, \\ -1, & \text{otherwise} \end{cases}$$

where $\mathrm{Act}(x)$ is the activation state of neuron x.

We say that the computation of $\mathcal{P}$ by $\mathcal{N}$ *terminates* when $\mathrm{valuation}(\mathrm{Act}(i_i)) = \mathrm{valuation}((\mathrm{Act}(o_j))$ for every pair of neurons (i_i, o_j) in $\mathcal{N}$ such that $\mathrm{name}(i_i) = \mathrm{name}(o_j)$.

[6] If a network obtained by the Translation Algorithm is then trained by examples with the use of a learning algorithm that does not impose any constraints on the weights, values in the interval $[-A_{\min}, A_{\min}]$ may occur and should be interpreted as *unknown* by following a three-valued interpretation.

From Theorem 9 and the definition of $T_{\mathcal{P}}^{n}$ above, it is clear that, starting from $\{\emptyset\}$ (i.e. $\mathbf{i} = (i_1, \ldots, i_p) = [-1, -1, \ldots, -1]$), if $\mathcal{P}$ is well-behaved then the computation of $\mathcal{P}$ by $\mathcal{N}$ terminates. The computation is as follows (below, we use $\mathbf{o} = \mathcal{N}(\mathbf{i})$ to denote the output vector $\mathbf{o} = (o_1, \ldots, o_q)$ obtained by presenting input vector $\mathbf{i}$ to network $\mathcal{N}$):

(1) `Let` $\mathbf{i} = [-1, -1, \ldots, -1]$;
(2) `Repeat:`
  (a) `Calculate` $\mathbf{o} = \mathcal{N}(\mathbf{i})$;
  (b) `For each` o_j `in` $\mathbf{o}$, `do:`
    (i) `If` name(o_j) = name(i_i) `Then replace the value of` i_i `in` $\mathbf{i}$ `by` valuation(Act(o_j));
(3) `Until` valuation(Act(o_j)) = valuation(Act(i_i)) `for all` (i_i, o_j) `s.t.` name(i_i) = name(o_j).

The set $\bigcup \text{name}(x) \subseteq B_{\mathcal{P}}$ of input and output neurons x in $\mathcal{N}$ for which valuation(Act(x)) $= 1$ will denote $T_{\mathcal{P}}^{m}$. When it is clear from the context, we may write *neuron* A_k to indicate the neuron in $\mathcal{N}$ associated with atom A_k in $\mathcal{P}$.

Example 10 (*Example* 8 *continued*). To construct the network of Fig. 4, firstly we calculate $\text{MAX}_{\mathcal{P}}(\overrightarrow{k}, \overrightarrow{\mu}) = 3$ and $A_{\min} > 0.5$. Then, $\theta_1 = (1 + A_{\min})W$, $\theta_2 = (1 + A_{\min})W/2$, $\theta_3 = -(1 + A_{\min})W/2$, $\theta_A = -(1 + A_{\min})W/2$ and $\theta_B = 0$. Now, suppose $A_{\min} = 0.6$, we obtain $W \geq 6.931/\beta$. Alternatively, suppose $A_{\min} = 0.7$, then $W \geq 4.336/\beta$. Let us take $A_{\min} = 0.7$ and $h(x)$ as the standard bipolar semi-linear activation function ($\beta = 1$). Then, if $W = 4.5$,[7] $\mathcal{N}$ will compute the operator $T_{\mathcal{P}}$ of $\mathcal{P}$. The computation of $\mathcal{P}$ by $\mathcal{N}$ terminates when $m = 2$ with $T_{\mathcal{P}}^{m} = \{B\}$.

3.2. Computing modalities in neural networks

In this section, we present the computational machinery of CML. We use the C-ILP *Translation Algorithm* presented in Section 3.1 to create each network of the ensemble, and the following *Modalities Algorithm* to interconnect the different networks and perform reasoning. The *Modalities Algorithm* translates, in a finite universe, natural deduction modal rules into the networks. Intuitively, the accessibility relation is represented by connections between networks. As depicted in Fig. 3 where $\mathcal{R}(\omega_1, \omega_2)$ and $\mathcal{R}(\omega_1, \omega_3)$, connections from ω_1 to ω_2 and ω_3 represent either $\Box E$ or $\Diamond E$; connections from ω_2 and ω_3 to ω_1 represent either $\Box I$ or $\Diamond I$.

Let $\mathcal{P}$ be an extended modal program with clauses of the form $\omega_i : ML_1, \ldots, ML_k \rightarrow MA$, where each L_j is a literal, A is an atom and $M \in \{\Box, \Diamond\}$, $1 \leq i \leq n$, $0 \leq j \leq k$. As in the case of individual C-ILP networks, we start by calculating $\text{MAX}_{\mathcal{P}}(\overrightarrow{k}, \overrightarrow{\mu}, n)$ of $\mathcal{P}$ and $A_{\min}$ such that:

$$A_{\min} > \frac{\text{MAX}_{\mathcal{P}}(\overrightarrow{k}, \overrightarrow{\mu}, n) - 1}{\text{MAX}_{\mathcal{P}}(\overrightarrow{k}, \overrightarrow{\mu}, n) + 1} \tag{9}$$

but now we also need to take into account the number n of networks (i.e. possible worlds) in the ensemble, and thus we use $\text{MAX}_{\mathcal{P}}(\overrightarrow{k}, \overrightarrow{\mu}, n)$ instead of simply $\text{MAX}_{\mathcal{P}}(\overrightarrow{k}, \overrightarrow{\mu})$.

Modalities algorithm

(1) Let $\mathcal{P}_i \subseteq \mathcal{P}$ be the set of clauses labelled by ω_i in $\mathcal{P}$. Let $W^M \in \mathbb{R}$.
(2) For each $\mathcal{P}_i$ do:
  (a) Rename each ML_j in $\mathcal{P}_i$ by a new literal not occurring in $\mathcal{P}$ of the form $L_j^{\Box}$ if $M = \Box$, or $L_j^{\Diamond}$ if $M = \Diamond$[8];
  (b) Call *Translation Algorithm;*
  (c) Let $\mathcal{N}_i$ be the neural network that denotes $\mathcal{P}_i$.
(3) For each output neuron $L_j^{\Diamond}$ in $\mathcal{N}_i$, do:
  (a) Add a hidden neuron L_j^{M} to an arbitrary $\mathcal{N}_k$ $(0 \leq k \leq n)$ such that $\mathcal{R}(\omega_i, \omega_k)$;
  (b) Set the step function $s(x)$ as the activation function of L_j^{M};

[7] Note that a sound translation from $\mathcal{P}$ to $\mathcal{N}$ does not require all the weights in $\mathcal{N}$ to have the same absolute value. We unify the weights ($|W|$) for the sake of simplicity of the translation algorithm and to comply with previous work.

[8] This allows us to treat each ML_j as a literal and apply the *Translation Algorithm* directly to $\mathcal{P}_i$ by labelling neurons as $\Box L_j$, $\Diamond L_j$, or L_j.

(c) Connect $L_j^{\Diamond}$ in $\mathcal{N}_i$ to L_j^M and set the connection weight to 1;
(d) Set the threshold θ^M of L_j^M such that $-1 < \theta^M < A_{\min}$;
(e) Create an output neuron L_j with threshold $\theta_{L_j} = (1 + A_{\min}) \cdot (1 - \mu_{L_j}) \cdot W/2$ in $\mathcal{N}_k$, if it does not exist yet;
(f) Connect L_j^M to L_j in $\mathcal{N}_k$, and set the connection weight to $W_{L_j}^M > h^{-1}(A_{\min}) + \mu_{L_j} W + \theta_{L_j}$.[9]

(4) For each output neuron $L_j^{\Box}$ in $\mathcal{N}_i$, do:
(a) Add a hidden neuron L_j^M to each $\mathcal{N}_k$ $(0 \leq k \leq n)$ such that $\mathcal{R}(\omega_i, \omega_k)$;
(b) Set the step function $s(x)$ as the activation function of L_j^M;
(c) Connect $L_j^{\Box}$ in $\mathcal{N}_i$ to L_j^M and set the connection weight to 1;
(d) Set the threshold θ^M of L_j^M such that $-1 < \theta^M < A_{\min}$;
(e) Create output neurons L_j with thresholds $\theta_{L_j} = (1 + A_{\min}) \cdot (1 - \mu_{L_j}) \cdot W/2$ in each $\mathcal{N}_k$, if they do not exist yet;
(f) Connect L_j^M to L_j in $\mathcal{N}_k$, and set the connection weight to $W_{L_j}^M > h^{-1}(A_{\min}) + \mu_{L_j} W + \theta_{L_j}$.

(5) For each output neuron L_j in $\mathcal{N}_k$ such that $\mathcal{R}(\omega_i, \omega_k)$ $(0 \leq i \leq n)$, do:
(a) Add a hidden neuron $L_j^{\vee}$ to $\mathcal{N}_i$ if it does not exist yet;
(b) Set the step function $s(x)$ as the activation function of $L_j^{\vee}$;
(c) For each ω_i such that $\mathcal{R}(\omega_i, \omega_k)$, do:
(i) Connect L_j in $\mathcal{N}_k$ to $L_j^{\vee}$ and set the connection weight to 1;
(ii) Set the threshold $\theta^{\vee}$ of $L_j^{\vee}$ such that $-nA_{\min} < \theta^{\vee} < A_{\min} - (n-1)$;
(iii) Create an output neuron $L_j^{\Diamond}$ with threshold $\theta_{L_j^{\Diamond}} = (1 + A_{\min}) \cdot (1 - \mu_{L_j^{\Diamond}}) \cdot W/2$ in $\mathcal{N}_i$ if it does not exist yet;
(iv) Connect $L_j^{\vee}$ to $L_j^{\Diamond}$ in $\mathcal{N}_i$ and set the connection weight to $W_{L_j^{\Diamond}}^M > h^{-1}(A_{\min}) + \mu_{L_j^{\Diamond}} W + \theta_{L_j^{\Diamond}}$.

(6) For each output neuron L_j in $\mathcal{N}_k$ such that $\mathcal{R}(\omega_i, \omega_k)$ $(0 \leq i \leq n)$, do:
(a) Add a hidden neuron $L_j^{\wedge}$ to $\mathcal{N}_i$ if it does not exist yet;
(b) Set the step function $s(x)$ as the activation function of $L_j^{\wedge}$;
(c) For each ω_i such that $\mathcal{R}(\omega_i, \omega_k)$, do:
(i) Connect L_j in $\mathcal{N}_k$ to $L_j^{\wedge}$ and set the connection weight to 1;
(ii) Set the threshold $\theta^{\wedge}$ of $L_j^{\wedge}$ such that $n - (1 + A_{\min}) < \theta^{\wedge} < nA_{\min}$;
(iii) Create an output neuron $L_j^{\Box}$ with threshold $\theta_{L_j^{\Box}} = (1 + A_{\min}) \cdot (1 - \mu_{L_j^{\Box}}) \cdot W/2$ in $\mathcal{N}_i$ if it does not exist yet;
(iv) Connect $L_j^{\wedge}$ to $L_j^{\Box}$ in $\mathcal{N}_i$ and set the connection weight to $W_{L_j^{\Box}}^M > h^{-1}(A_{\min}) + \mu_{L_j^{\Box}} W + \theta_{L_j^{\Box}}$.[10]

(7) For each $\mathcal{P}_i$, recurrently connect each output neuron L_j (resp. $L_j^{\Diamond}$, $L_j^{\Box}$) in $\mathcal{N}_i$ to its corresponding input neuron L_j (resp. $L_j^{\Diamond}$, $L_j^{\Box}$) in $\mathcal{N}_i$ with weight $W_r = 1$ (this essentially allows one to iterate $MT_{\mathcal{P}}$, thus using the ensemble to compute the extended modal program in parallel, as exemplified below).

Let us now illustrate the use of the *Modalities Algorithm* with the following example.

Example 11. Let $\mathcal{P} = \{\omega_1 : r \rightarrow \Box q;\ \omega_1 : \Diamond s \rightarrow r;\ \omega_2 : s;\ \omega_3 : q \rightarrow \Diamond p;\ \mathcal{R}(\omega_1,\omega_2),\ \mathcal{R}(\omega_1,\omega_3)\}$. We start by applying the *Translation Algorithm,* which creates three neural networks to represent the worlds ω_1, ω_2, and ω_3 (see Fig. 5). Then, we apply the *Modalities Algorithm*. Hidden neurons labelled by $\{M, \vee, \wedge\}$ are created using the *Modalities Algorithm.* The remaining neurons are all created using the *Translation Algorithm*. For the sake of clarity, unconnected input and output neurons are not shown in Fig. 5. Taking $\mathcal{N}_1$ (which represents ω_1), output neurons $L_j^{\Diamond}$ should be connected to output neurons L_j in an arbitrary network $\mathcal{N}_i$ (which represents ω_i) to which $\mathcal{N}_1$ is related. For example, taking $\mathcal{N}_i = \mathcal{N}_2$, $\Diamond s$ in $\mathcal{N}_1$ is connected to s in $\mathcal{N}_2$. Then, output neurons $L_j^{\Box}$ should be connected to

[9] Recall that μ_L is the number of connections to output neuron L, and that θ_L is the threshold of output neuron L. Note also that μ_L, W and θ_L are all obtained from the *Translation Algorithm.*

[10] W^M values are derived from the proof of Theorem 12 below.

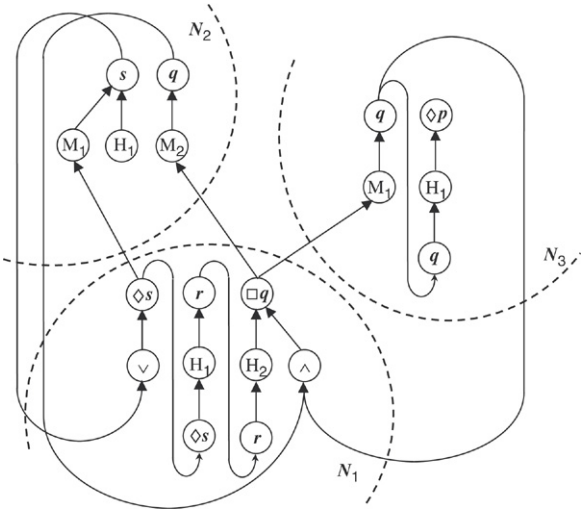

Fig. 5. The ensemble of networks $\{\mathcal{N}_1, \mathcal{N}_2, \mathcal{N}_3\}$ that represents $\mathcal{P}$.

output neurons L_j in every network $\mathcal{N}_i$ to which $\mathcal{N}_1$ is related. For example, $\Box q$ in $\mathcal{N}_1$ is connected to q in both $\mathcal{N}_2$ and $\mathcal{N}_3$. Now, taking $\mathcal{N}_2$, output neurons L_j need to be connected to output neurons $L_j^{\Diamond}$ and $L_j^{\Box}$ in every network $\mathcal{N}_j$ related to $\mathcal{N}_2$. For example, s in $\mathcal{N}_2$ is connected to $\Diamond s$ in $\mathcal{N}_1$ via the hidden neuron denoted by $\vee$ in Fig. 5, while q in $\mathcal{N}_2$ is connected to $\Box q$ in $\mathcal{N}_1$ via the hidden neuron denoted by $\wedge$. Similarly, q in $\mathcal{N}_3$ is connected to $\Box q$ in $\mathcal{N}_1$ via $\wedge$. Finally, output neurons $\Diamond s$ and r in $\mathcal{N}_1$ are connected to input neurons $\Diamond s$ and r, respectively, in $\mathcal{N}_1$, and output neuron q in $\mathcal{N}_3$ is connected to input neuron q in $\mathcal{N}_3$, all connections with weight 1. The algorithm terminates when all output neurons have been connected.

Table 2 contains a valid set of weights for the connections shown in Fig. 5, obtained from the *Modalities Algorithm* and the *Translation Algorithm*. We use (X_{N_i}, Y_{N_j}) to denote the weight from neuron X in network N_i to neuron Y in network N_j, and (X_{N_i}) to denote the threshold of neuron X in network N_i, following Fig. 5. The calculations are as follows. From Eq. (9), $A_{\min} > (\text{MAX}_{\mathcal{P}}(1,2,3)-1)/(\text{MAX}_{\mathcal{P}}(1,2,3)+1)$. Let $A_{\min} = 0.6$. From Eq. (6), taking $\beta = 1$, $W \geq 2(\ln(1.6) - \ln(0.4))/(2(-0.4)+1.6) = 1.1552$. Let $W = 2$. Thus, all feedforward connections internal to a network will receive weight 2. Recall that all feedback connections internal to a network will receive weight 1 (see Table 2). Then, the thresholds of hidden neurons H are calculated according to Eq. (7), and the thresholds of all the output neurons are calculated according to Equation 8. For example, $(H1_{N_1}) = 2((1+0.6)\cdot(1-1))/2 = 0$, $(\Box q_{N_1}) = 2((1+0.6)\cdot(1-1))/2 = 0$, $(H1_{N_2}) = 2((1+0.6)\cdot(0-1))/2 = -1.6$ and $(\Diamond s_{N_1}) = 2((1+0.6)\cdot(1-0))/2 = 1.6$. Now, thresholds and weights for neurons M, $\wedge$ and $\vee$ need to be calculated. From the Modalities Algorithm, connections between networks, e.g. $(\Diamond s_{N_1}, M1_{N_2})$, will receive weight 1; the thresholds θ^M of neurons M must satisfy $-1 < \theta^M < A_{\min}$ (e.g.: $(M1_{N_2}) = 0.5$); the thresholds $\theta^{\vee}$ of neurons $\vee$ must satisfy $-nA_{\min} < \theta^{\vee} < A_{\min} - (n-1)$ (e.g.: $(\vee_{N_1}) = -1.6$); and the thresholds $\theta^{\wedge}$ of neurons $\wedge$ must satisfy $n - (1 + A_{\min}) < \theta^{\wedge} < nA_{\min}$ (e.g.: $(\wedge_{N_1}) = 1.6$).[11] Finally, weights $W_L^M > h^{-1}(0.6) + 2\mu_L + \theta_L$ must be calculated.[12] For example, output neuron $\Diamond s$ in N_1 has $\mu = 0$ and $\theta = 1.6$, and thus $W^M > 2.986$. Similarly, output neuron s in N_2 has $\mu = 1$ and $\theta = 0$, and thus $W^M > 3.386$. Although not necessary, let us unify $W^M = 4$ for all of the five remaining weights (see Table 2).

Soundness of the modal computation

We are now in the position to show that the ensemble of neural networks $\mathcal{N}$ obtained from the above *Modalities Algorithm* is equivalent to the original extended modal program $\mathcal{P}$, in the sense that $\mathcal{N}$ computes the *modal immediate consequence operator* $MT_{\mathcal{P}}$ of $\mathcal{P}$ (see Definition 6). In other words, the theorem below guarantees that the computational process carried out by our connectionist model is meaningful; it shows from a technical viewpoint that

[11] Recall that $n = 3$ in this example.

[12] Recall that $W = 2$ in this example, $h^{-1}(A_{\min}) = -\frac{1}{\beta}\ln\left(\frac{1-A_{\min}}{1+A_{\min}}\right)$.

Table 2
A valid set of weights and thresholds for the network of Fig. 5

$(\vee_{N_1}, \Diamond s_{N_1}) = 4$	$(M1_{N_2}, s_{N_2}) = 4$	$(M1_{N_3}, q_{N_3}) = 4$
$(\Diamond s_{N_1}, \Diamond s_{N_1}) = 1$	$(H1_{N_2}, s_{N_2}) = 2$	$(q_{N_3}, q_{N_3}) = 1$
$(\Diamond s_{N_1}, H1_{N_1}) = 2$	$(M2_{N_2}, q_{N_2}) = 4$	$(q_{N_3}, H1_{N_3}) = 2$
$(H1_{N_1}, r_{N_1}) = 2$		$(H1_{N_3}, \Diamond p_{N_3}) = 2$
$(r_{N_1}, r_{N_1}) = 1$		
$(r_{N_1}, H2_{N_1}) = 2$		
$(H2_{N_1}, \Box q_{N_1}) = 2$		
$(\wedge_{N_1}, \Box q_{N_1}) = 4$		
$(\Diamond s_{N_1}, M1_{N_2}) = 1$	$(s_{N_2}, \vee_{N_1}) = 1$	$(q_{N_3}, \wedge_{N_1}) = 1$
$(\Box q_{N_1}, M2_{N_2}) = 1$	$(q_{N_2}, \wedge_{N_1}) = 1$	
$(\Box q_{N_1}, M1_{N_3}) = 1$		
$(\vee_{N_1}) = -1.6$	$(M1_{N_2}) = 0.5$	$(M1_{N_3}) = 0.5$
$(\Diamond s_{N_1}) = 1.6$	$(H1_{N_2}) = -1.6$	$(q_{N_3}) = 1.6$
$(H1_{N_1}) = 0$	$(M2_{N_2}) = 0.5$	$(H1_{N_3}) = 0$
$(r_{N_1}) = 0$	$(s_{N_2}) = 0$	$(\Diamond p_{N_3}) = 0$
$(H2_{N_1}) = 0$	$(q_{N_2}) = 1.6$	
$(\wedge_{N_1}) = 1.6$		
$(\Box q_{N_1}) = 0$		

the approach presented here is correct, as the model in use is capable of translating modal symbolic formalisms into artificial neural networks.

Theorem 12. *For any extended modal program $\mathcal{P}$ there exists an ensemble of feedforward neural networks $\mathcal{N}$ such that $\mathcal{N}$ computes the modal fixed-point operator $MT_{\mathcal{P}}$ of $\mathcal{P}$.*

Proof. We have to show that there exists $W > 0$ such that the network ensemble $\mathcal{N}$, obtained by the above *Modalities Algorithm*, computes $MT_{\mathcal{P}}$. Throughout, we assume that $\mathcal{N}_i$ and $\mathcal{N}_j$ are two arbitrary networks of $\mathcal{N}$, representing possible worlds ω_i and ω_j, respectively, such that $\mathcal{R}(\omega_i, \omega_j)$. We distinguish two cases: (a) clauses with modalities $\Box$ and $\Diamond$ in the head, and (b) clauses with no modalities in the head.

(a) Firstly, note that clauses with $\Box$ in the head must satisfy $\Box E$, while clauses with $\Diamond$ in the head must satisfy $\Diamond E$ in Table 1. Given input vectors $\mathbf{i}$ and $\mathbf{j}$ to $\mathcal{N}_i$ and $\mathcal{N}_j$, respectively, each neuron A in the output layer of $\mathcal{N}_j$ is active ($A > A_{\min}$) if and only if: (i) there exists a clause of $\mathcal{P}_j$ of the form $ML_1, \ldots, ML_k \rightarrow A$ s.t. $ML_1, \ldots, ML_k$ are satisfied by interpretation $\mathbf{j}$, or (ii) there exists a clause of $\mathcal{P}_i$ of the form $ML_1, \ldots, ML_k \rightarrow \Box A$ s.t. $ML_1, \ldots, ML_k$ are satisfied by interpretation $\mathbf{i}$, or even (iii) there exists a clause of $\mathcal{P}_i$ of the form $ML_1, \ldots, ML_k \rightarrow \Diamond A$ s.t. $ML_1, \ldots, ML_k$ are satisfied by interpretation $\mathbf{i}$, and the *Modalities Algorithm* (Step (3)(a)) has selected $\mathcal{N}_j$ as the arbitrary network $\mathcal{N}_k$.

($\leftarrow$) (i) results directly from Theorem 9. (ii) and (iii) share the same proof, as follows: from Theorem 9, we know that if $ML_1, \ldots, ML_k$ are satisfied by interpretation $\mathbf{i}$ then MA is active in $\mathcal{N}_i$ (recall, $M \in \{\Box, \Diamond\}$). Hence, we only need to show that MA in $\mathcal{N}_i$ activates A in $\mathcal{N}_j$. From the *Modalities Algorithm*, A^M is a non-linear hidden neuron in $\mathcal{N}_j$. Thus, if MA is active ($\mathrm{Act}(MA) > A_{\min}$) then A^M presents activation 1. As a result, the minimum activation of A is $h(W_A^M - \mu_A W - \theta_A)$. Now, since $W_A^M > h^{-1}(A_{\min}) + \mu_A W + \theta_A$, we have $h(W_A^M - \mu_A W - \theta_A) > A_{\min}$ and, therefore, A is active ($\mathrm{Act}(A) > A_{\min}$).

($\rightarrow$) Directly from the *Modalities Algorithm*, since A^M is a non-linear neuron, it contributes with *zero* to the input potential of A in $\mathcal{N}_j$ when MA is not active in $\mathcal{N}_i$. In this case, the behaviour of A in $\mathcal{N}_j$ is not affected by $\mathcal{N}_i$. Now, from Theorem 9, $\mathcal{N}_j$ computes the fixed-point operator $T_{\mathcal{P}_j}$ of $\mathcal{P}_j$. Thus, if $ML_1, \ldots, ML_k$ is not satisfied by $\mathbf{j}$ then A is not active in $\mathcal{N}_j$.

(b) clauses with no modalities must satisfy $\Box I$ and $\Diamond I$ in Table 1. Given input vectors $\mathbf{i}$ and $\mathbf{j}$ to $\mathcal{N}_i$ and $\mathcal{N}_j$, respectively, each neuron $\Box A$ in the output layer of $\mathcal{N}_i$ is active ($\mathrm{Act}(\Box A) > A_{\min}$) if and only if: (i) there exists a clause of $\mathcal{P}_i$ of the form $ML_1, \ldots, ML_k \rightarrow \Box A$ s.t. $ML_1, \ldots, ML_k$ are satisfied by interpretation $\mathbf{i}$, or (ii) for all $\mathcal{N}_j$, there exists a clause of $\mathcal{P}_j$ of the form $ML_1, \ldots, ML_k \rightarrow A$ s.t. $ML_1, \ldots, ML_k$ are satisfied by interpretation $\mathbf{j}$. Each neuron $\Diamond A$ in the output layer of $\mathcal{N}_i$ is active ($\mathrm{Act}(\Diamond A) > A_{\min}$) if and only if: (iii) there exists a clause of $\mathcal{P}_i$ of

the form $ML_1, \ldots, ML_k \rightarrow \Diamond A$ s.t. $ML_1, \ldots, ML_k$ are satisfied by interpretation **i**, or (iv) there exists a clause of $\mathcal{P}_j$ of the form $ML_1, \ldots, ML_k \rightarrow A$ s.t. $ML_1, \ldots, ML_k$ are satisfied by interpretation **j**.

($\leftarrow$) (i) and (iii) result directly from Theorem 9. (ii) and (iv) are proved in what follows: from Theorem 9, we know that if $ML_1, \ldots, ML_k$ are satisfied by interpretation **j** then A is active in $\mathcal{N}_j$. (ii) We need to show that if A is active in every network $\mathcal{N}_j$ ($0 \leq j \leq n$) to which $\mathcal{N}_i$ is related, $\Box A$ is active in $\mathcal{N}_i$. From the *Modalities Algorithm*, $A^\wedge$ is a non-linear hidden neuron in $\mathcal{N}_i$. If A is active ($\text{Act}(A) > A_{\min}$) in $\mathcal{N}_j$, the minimum input potential of $A^\wedge$ is $nA_{\min} - \theta^\wedge$. Now, since $\theta^\wedge < nA_{\min}$ (*Modalities Algorithm*, Step (6)(c)(ii)), the minimum input potential of $A^\wedge$ is greater than *zero* and, therefore, $A^\wedge$ presents activation value 1. (iv) We need to show that if A is active in at least one network $\mathcal{N}_j$ ($0 \leq j \leq n$) to which $\mathcal{N}_i$ is related, $\Diamond A$ is active in $\mathcal{N}_i$. From the *Modalities Algorithm*, $A^\vee$ is a non-linear hidden neuron in $\mathcal{N}_i$. If A is active ($\text{Act}(A) > A_{\min}$) in $\mathcal{N}_j$, the minimum input potential of $A^\vee$ is $A_{\min} - \theta^\vee$. Now, since $\theta^\vee < A_{\min} - (n-1)$ (*Modalities Algorithm*, Step (5)(c)(ii)), and $n \geqslant 1$, the minimum input potential of $A^\vee$ is greater than *zero* and, therefore, $A^\vee$ presents activation 1. Finally, if $A^\wedge$ presents activation 1, the minimum activation of $\Box A$ is $h(W^M_{\Box A} - \mu_{\Box A} W - \theta_{\Box A})$, and, exactly as in item (a) above, $\Box A$ is active in $\mathcal{N}_i$. Similarly, if $A^\vee$ presents activation 1, the minimum activation of $\Diamond A$ is $h(W^M_{\Diamond A} - \mu_{\Diamond A} W - \theta_{\Diamond A})$, and, exactly as in item (a) above, $\Diamond A$ is active in $\mathcal{N}_i$.

($\rightarrow$) Again, (i) and (iii) result directly from Theorem 9. (ii) and (iv) are proved below: (ii) We need to show that if $\Box A$ is not active in $\mathcal{N}_i$ then at least one A is not active in $\mathcal{N}_j$ to which $\mathcal{N}_i$ is related ($0 \leq j \leq n$). If $\Box A$ is not active, $A^\wedge$ presents activation 0. In the worst case, A is active in $n-1$ networks with maximum activation (1.0), and not active in a single network with minimum activation ($-A_{\min}$). In this case, the input potential of $A^\wedge$ is $n - 1 - A_{\min} - \theta^\wedge$. Now, since $\theta^\wedge > n - (1 + A_{\min})$ (*Modalities Algorithm*, step (6)(c)(ii)), the maximum input potential of $A^\wedge$ is smaller than *zero* and, therefore, $A^\wedge$ presents activation 0. (iv) We need to show that if $\Diamond A$ is not active in $\mathcal{N}_i$ then A is not active in any network $\mathcal{N}_j$ to which $\mathcal{N}_i$ is related ($0 \leq j \leq n$). If $\Diamond A$ is not active, $A^\vee$ presents activation 0. In the worst case, A presents activation $-A_{\min}$ in all $\mathcal{N}_j$ networks. In this case, the input potential of $A^\vee$ is $-nA_{\min} - \theta^\vee$. Now, since $\theta^\vee > -nA_{\min}$ (*Modalities Algorithm*, step (5)(c)(ii)), the maximum input potential of $A^\vee$ is smaller than *zero* and, therefore, $A^\vee$ presents activation 0. Finally, from Theorem 9, if $A^\wedge$ and $A^\vee$ have activation 0, $\mathcal{N}_i$ computes the fixed-point operator $T_{\mathcal{P}_i}$ of $\mathcal{P}_i$. ■

Termination of the modal computation

A network ensemble can be used to compute extended modal programs in parallel in the same way that C-ILP networks are used to compute logic programs. Take a network ensemble $\{\mathcal{N}_1, \ldots, \mathcal{N}_n\}$ obtained from the Modalities Algorithm, and rename each input and output neuron $L_k^{\{\Box,\Diamond\}}$ in $\mathcal{N}_i$ ($1 \leq i \leq n$) as $\omega_i : L_k$, where L_k can be either a literal or a modal literal. This basically allows us to have copies of literal L_k in different possible worlds $(\omega_i, \omega_j, \ldots)$, and to treat the occurrence of L_k in $\mathcal{N}_i$ ($\omega_i : L_k$) as different from the occurrence of L_k in $\mathcal{N}_j$ ($\omega_j : L_k$). It is not difficult to see that we are left with a large single-hidden layer neural network $\mathcal{N}$, in which each input and output neuron is now labelled. This *flattened* network is a recurrent network containing feedback connections from the output layer to the input layer, and sometimes from the output to the hidden layer. Any feedback connection from output neurons $(o_j, o_k, \ldots)$ to a hidden neuron (h_i) may be replaced equivalently by feedback from the output to the input layer only, if we create new input neurons $o_j, o_k, \ldots$ and connect output o_j to input o_j, output o_k to input o_k, and so on, and then inputs $o_j, o_k, \ldots$ to hidden neuron h_i. As a result, as in the case of C-ILP networks, if $\mathcal{P}$ is well-behaved, the computation of $\mathcal{P}$ by $\mathcal{N}$ should terminate.

For example, in Fig. 5, since $\Diamond s$ and r in $\mathcal{N}_1$ and q in $\mathcal{N}_3$ are recursively connected, the ensemble computes $\{\Diamond s, r, \Box q\}$ in ω_1, $\{s, q\}$ in ω_2, and $\{q, \Diamond s\}$ in ω_3. As expected, these are logical consequences of the original program $\mathcal{P}$ given in Example 11. Although the computation is done in parallel in $\mathcal{N}$, following it by starting from facts (such as s in ω_2) may help verifying this.

Notice how the idea of labelling the neurons, allowing copies of neurons L_j to occur in the neural network simultaneously, allows us to give a modal interpretation to C-ILP networks as a corollary (below) of Theorem 12. Let $MT^n_{\mathcal{P}} \stackrel{\text{def}}{=} MT_{\mathcal{P}}(MT^{n\text{-}1}_{\mathcal{P}})$ with $MT^0_{\mathcal{P}} \stackrel{\text{def}}{=} MT_{\mathcal{P}}(\{\emptyset\})$. We say that an extended modal program $\mathcal{P}$ is *well-behaved* if, after a finite number m of iterations, $MT^m_{\mathcal{P}} = MT^{m\text{-}1}_{\mathcal{P}}$.

Corollary 13. *Let $\mathcal{P}$ be an extended modal program. There exists an ensemble of neural networks $\mathcal{N}$ such that, if $\mathcal{P}$ is well-behaved, the computation of $\mathcal{P}$ by $\mathcal{N}$ terminates. The set* $\bigcup \text{name}(x) \subseteq B_{\mathcal{P}}$ *of input and output neurons x in $\mathcal{N}$ for which* $\text{valuation}(\text{Act}(x)) = 1$ *will denote* $MT^{\text{m}}_{\mathcal{P}}$.

4. The connectionist muddy children puzzle

In this section, we apply CML to the muddy children puzzle, a classic example of reasoning in multi-agent environments. In contrast with the also well-known wise men puzzle [19,29], in which the reasoning process is sequential, here it is clear that a distributed (simultaneous) reasoning process occurs, as follows: There is a group of n children playing in a garden. A certain number of children k ($k \leq n$) has mud on their faces. Each child can see if the others are muddy, but cannot see if they themselves are muddy. Now, consider the following situation.[13]

A caretaker announces that at least one child is muddy ($k \geq 1$) and asks *do you know if you have mud on your faces*?[14] To help understanding the puzzle, let us consider the cases in which $k = 1$, $k = 2$ and $k = 3$.

If $k = 1$ (only one child is muddy), the muddy child answers *yes* at the first instance since she cannot see any other muddy child. All the other children answer *no* at the first instance.

If $k = 2$, suppose children 1 and 2 are muddy. In the first instance, all children can only answer *no*. This allows 1 to reason as follows: *if* 2 *had said yes the first time round, she would have been the only muddy child. Since* 2 *said no, she must be seeing someone else muddy; and since I cannot see anyone else muddy apart from* 2*, I myself must be muddy*! Child 2 can reason analogously, and also answer *yes* the second time round.

If $k = 3$, suppose children 1, 2 and 3 are muddy. Every child can only answer *no* the first two time rounds. Again, this allows 1 to reason as follows: *if* 2 *or* 3 *had said yes the second time round, they would have been the only two muddy children. Thus, there must be a third person with mud. Since I can see only* 2 *and* 3 *with mud, this third person must be me*! Children 2 and 3 can reason analogously to conclude as well that *yes*, they are muddy.

The above cases clearly illustrate the need to distinguish between an agent's *individual knowledge* and *common knowledge* about the world in a particular situation. For example, when $k = 2$, after everybody says *no* in the first round, it becomes common knowledge that at least two children are muddy. Similarly, when $k = 3$, after everybody says *no* twice, it becomes common knowledge that at least three children are muddy, and so on. In other words, when it is common knowledge that there are at least $k - 1$ muddy children; after the announcement that nobody knows if they are muddy or not, then it becomes common knowledge that there are at least k muddy children, for if there were $k - 1$ muddy children all of them would have known that they had mud on their faces. Notice that this reasoning process can only start once it is common knowledge that at least one child is muddy, as announced by the caretaker.[15]

4.1. Distributed knowledge representation

Let us now formalise the muddy children puzzle in our connectionist modal logic framework. Typically, the way to represent the knowledge of a particular agent is to express the idea that an agent knows a fact α if the agent considers/thinks that α is true at every world the agent sees as possible. In such a formalisation, a $\mathbf{K}_j$ modality that represents the knowledge of an agent j is interpreted as a $\Box$ modality as defined in Section 2.1. In addition, we use p_i to denote that proposition p is *true* for agent i, so that $\mathbf{K}_j p_i$ means that agent j knows that p is *true* for agent i. We omit the subscript j of $\mathbf{K}$ whenever it is clear from the context. We use p_i to say that child i is muddy, and q_k to say that at least k children are muddy ($k \leq n$). Note, thus, the difference between p_1 (child 1 is muddy) and $\mathbf{K}p_1$ (child 1 knows she is muddy).

---

[13] We follow the muddy children problem description presented in [19]. We must also assume that all the agents involved in the situation are truthful and intelligent.

[14] Of course, if $k > 1$ they already know that there are muddy children amongst them.

[15] The question of how to represent common knowledge in neural networks is an interesting one. In this paper, we do this implicitly – as will become clearer in what follows – by connecting neurons appropriately as the reasoning progresses (for example, as we find out at round two that at least two children should be muddy). The representation of common knowledge in the object level would require the use of neurons that are activated when, e.g., "everybody knows" something (serving to implement in a finite domain the common knowledge axioms of [19]), but this would complicate the formalisation of the puzzle given in this paper. This explicit form of representation and its ramifications are worth investigating though, and should be treated in their own right in future work.

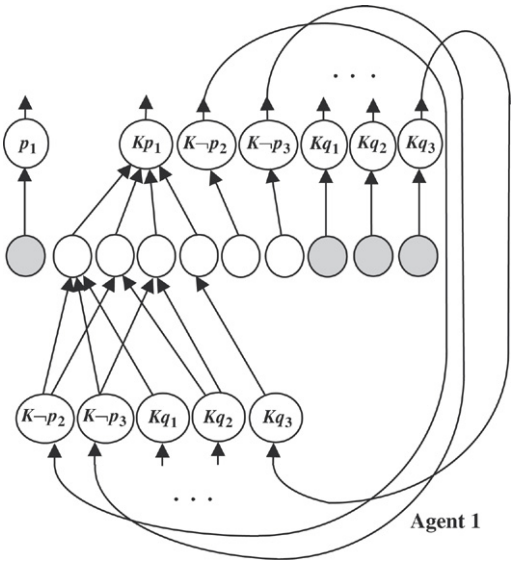

Fig. 6. The implementation of rules $\{r_1^1, \ldots, r_4^1\}$.

Let us consider the case in which three children are playing in the garden ($n = 3$). Clause r_1^1 below states that when child 1 knows that at least one child is muddy and that neither child 2 nor child 3 are muddy then child 1 knows that she herself is muddy. Similarly, clause r_2^1 states that if child 1 knows that there are at least two muddy children and she knows that child 2 is not muddy then she must also be able to know that she herself is muddy, and so on. The clauses for children 2 and 3 are interpreted analogously.

Clauses for agent(child) 1:

r_1^1: $\mathbf{K}_1 q_1 \wedge \mathbf{K}_1 \neg p_2 \wedge \mathbf{K}_1 \neg p_3 \rightarrow \mathbf{K}_1 p_1$

r_2^1: $\mathbf{K}_1 q_2 \wedge \mathbf{K}_1 \neg p_2 \rightarrow \mathbf{K}_1 p_1$

r_3^1: $\mathbf{K}_1 q_2 \wedge \mathbf{K}_1 \neg p_3 \rightarrow \mathbf{K}_1 p_1$

r_4^1: $\mathbf{K}_1 q_3 \rightarrow \mathbf{K}_1 p_1$

Clauses for agent(child) 2:

r_1^2: $\mathbf{K}_2 q_1 \wedge \mathbf{K}_2 \neg p_1 \wedge \mathbf{K}_2 \neg p_3 \rightarrow \mathbf{K}_2 p_2$

r_2^2: $\mathbf{K}_2 q_2 \wedge \mathbf{K}_2 \neg p_1 \rightarrow \mathbf{K}_2 p_2$

r_3^2: $\mathbf{K}_2 q_2 \wedge \mathbf{K}_2 \neg p_3 \rightarrow \mathbf{K}_2 p_2$

r_4^2: $\mathbf{K}_2 q_3 \rightarrow \mathbf{K}_2 p_2$

Clauses for agent(child) 3:

r_1^3: $\mathbf{K}_3 q_1 \wedge \mathbf{K}_3 \neg p_1 \wedge \mathbf{K}_3 \neg p_2 \rightarrow \mathbf{K}_3 p_3$

r_2^3: $\mathbf{K}_3 q_2 \wedge \mathbf{K}_3 \neg p_1 \rightarrow \mathbf{K}_3 p_3$

r_3^3: $\mathbf{K}_3 q_2 \wedge \mathbf{K}_3 \neg p_2 \rightarrow \mathbf{K}_3 p_3$

r_4^3: $\mathbf{K}_3 q_3 \rightarrow \mathbf{K}_3 p_3$

Each set of clauses r_m^l ($1 \leq l \leq n$, $m \in \mathbb{N}^+$) is implemented in a C-ILP network. Fig. 6 shows the implementation of clauses r_1^1 to r_4^1 (for agent 1).[16] In addition, it contains p_1 and $\mathbf{K}q_1$, $\mathbf{K}q_2$ and $\mathbf{K}q_3$, all represented as facts. This

[16] Note that $\mathbf{K}p_i$ and $\mathbf{K}\neg p_i$ should be represented by two different input neurons [12]. This can be done by renaming $\mathbf{K}\neg p_i$ by a new literal $\mathbf{K}p_i'$ before we call the Translation Algorithm. Negative weights in the network would then allow one to differentiate between $\mathbf{K}p_i$ and $\sim \mathbf{K}p_i$, and between $\mathbf{K}\neg p_i$ and $\sim\mathbf{K}\neg p_i$, respectively.

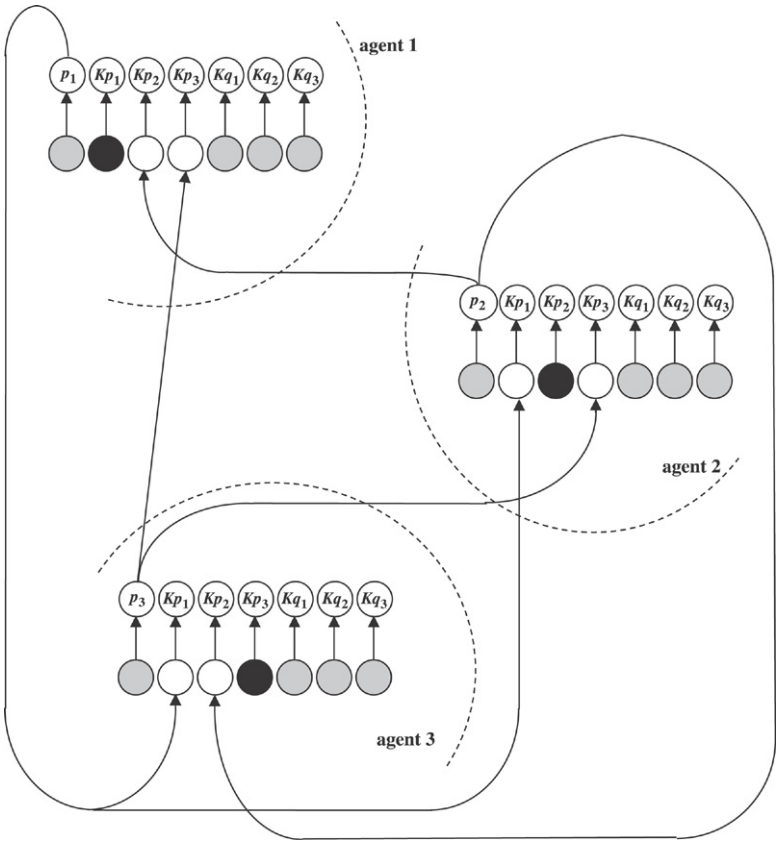

Fig. 7. Interaction between agents in the muddy children puzzle.

is highlighted in grey in Fig. 6. This setting complies with the presentation of the puzzle given in [29], in which *snapshots* of the knowledge evolution along time rounds are taken in order to logically deduce the solution of the problem without the addition of a time variable. Here, p_1 and $\mathbf{K}q_k$ ($1 \leq k \leq 3$) are obtained from the network's input, which denotes a snapshot in the computation (a particular round), while $\mathbf{K}\neg p_2$ and $\mathbf{K}\neg p_3$ are obtained from the other networks in the ensemble (representing agents 2 and 3, respectively, whenever agent 1 does not see mud on their foreheads). Notice that a complete solution to the puzzle would require the replication of the ensemble presented here across time points according to the different rounds of computation. This would produce a two-dimensional network ensemble, where in one dimension we have *agents* (as depicted here) and in the other we have *time*, so that we can represent the agents' knowledge evolution across time points explicitly [16].

Fig. 7 illustrates the interaction between three agents in the muddy children puzzle. The arrows connecting the networks implement the fact that when a child is muddy, the other children can see this. For the sake of clarity, the clauses r_m^1, corresponding to neuron $\mathbf{K}_1 p_1$, are shown only in Fig. 6. Analogously, the clauses r_m^2 and r_m^3 for $\mathbf{K}_2 p_2$ and $\mathbf{K}_3 p_3$ would be represented in similar networks. This is indicated in Fig. 7 by neurons highlighted in black. In addition, Fig. 7 only shows positive information about the problem. Recall that negative information such as $\neg p_1$, $\mathbf{K}\neg p_1$, $\mathbf{K}\neg p_2$ is to be added explicitly to the network, as shown in Fig. 6.

4.2. Learning in CML

As discussed in the Introduction, one of our objectives when developing neural-symbolic learning systems is to retain good learning capability while seeking to develop systems that can deal with more expressive languages such as modal logics. In order to implement such a system, one first translates the background knowledge into a neural network's initial architecture, and then trains it with examples using a neural learning algorithm [15,40]. In this section, we investigate this. We assume that not all of the rules as described in the previous section are known, and that such rules have to be learned by generalisations over examples (i.e. cases). We compare a situation in which each

agent knows one rule only, but not the other rules, and a situation in which no rule at all is known in advance. In both cases, the agents need to learn in order to be able to reason about the problem. We expect the first situation, in which some background knowledge is available, to offer a better performance than the latter.

We use the Modalities Algorithm given in Section 3.2 to perform the translation from a modal background knowledge to the initial ensemble architecture. We then use standard *backpropagation* to train each network of the ensemble with examples.[17] Our aim is to verify whether a particular agent i can learn from examples if he is muddy or not, i.e. learn clauses r_1^i to r_4^i above.

We have performed two sets of experiments to compare learning with background knowledge and without background knowledge. In the first set of experiments, we have created networks with random weights to which we then presented a number of training examples. In the second set of experiments, we have inserted clauses r_1^i: $\mathbf{K}_1 q_1 \wedge \mathbf{K}_1 \neg p_2 \wedge \mathbf{K}_1 \neg p_3 \rightarrow \mathbf{K}_1 p_1$ in the ensemble as background knowledge before training the networks with examples. Each training example states whether agent i is muddy or not, according to the truth-values of literals $\mathbf{K}_i q_1$, $\mathbf{K}_i q_2$, $\mathbf{K}_i q_3$, $\mathbf{K}_i p_1$, $\mathbf{K}_i \neg p_1$, $\mathbf{K}_i p_2$, $\mathbf{K}_i \neg p_2$, $\mathbf{K}_i p_3$, $\mathbf{K}_i \neg p_3$ (represented as input neurons).

We have evaluated the networks using *cross-validation*, a testing methodology in which the set of examples is permuted and divided into n sets [32]. One division is used for testing and the remaining $n-1$ divisions are used for training. The testing division is never seen by the learning algorithm during the training process. The procedure is repeated n times so that every partition is used once for testing. In both experiments, we have used $n = 8$ over a set of 32 examples. In addition, we have used a learning rate $\eta = 0.2$, a term of momentum $\xi = 0.1$, $h(x) = \frac{2}{1+e^{-\beta x}} - 1$ as activation function, and bipolar inputs in $\{-1, 1\}$.

The training sets were presented to the networks for 10,000 epochs,[18] and the sets of weights were updated, as usual, after every epoch. For each experiment, this resulted in 8 networks being trained with 28 examples, with 4 examples reserved for testing. All 16 networks reached a training set error $\mathrm{Err}(\mathbf{W})$, according to Eq. (1), smaller than 0.01 before 10,000 epochs had elapsed. In other words, all the networks have been trained successfully. Recall that learning takes place locally in each network. Any connection between networks in the ensemble is defined by the rules of natural deduction for modalities presented in Section 2.1.

As for the networks' *generalisation* capability, the results corroborate the importance of exploiting any available background knowledge (assuming the background knowledge is correct, of course). In the first experiment, in which the connectionist modal system was trained with no background knowledge, the networks presented an average test set accuracy of 84.37%. In the second experiment, in which clauses r_1^i had been added to the networks prior to training, an average test set accuracy of 93.75% was obtained under exactly the same training conditions.

5. Conclusions and future work

In this paper, we have presented a new connectionist computational model, namely, *Connectionist Modal Logic* (CML). We introduced an algorithm that translates extended modal programs into ensembles of C-ILP neural networks [15,18], and proved that the ensembles compute a fixed-point semantics of the programs. The computation always terminates when the program is well-behaved, and thus the network ensembles can be used as a distributed computational model for modal logic. In addition, we have applied the CML system to the muddy children puzzle, a well-known testbed for distributed knowledge representation. We have both set-up and trained network ensembles to reason about this puzzle. The networks can learn possible world representations from examples by using standard neural learning algorithms such as backpropagation.

This paper opens up a new area of research in which modal reasoning can be represented and learned using artificial neural networks. There are several avenues of research to be pursued as a result. For instance, an important aspect of *neural-symbolic learning systems* – not dealt with in this paper – is rule extraction from neural network ensembles [14, 45]. In the case of CML, rule extraction methods would need to consider the more expressive knowledge representation language used here. Since we have shown that modalities can be represented in network ensembles, one should expect, when extracting rules from a given trained network ensemble, that rules with modalities would offer a better representation formalism for the ensemble either in terms of rule comprehensibility or rule expressiveness.

[17] Recall that each network in the ensemble is a C-ILP network and, therefore, can be trained with examples using standard backpropagation.

[18] An epoch is defined as one pass through the complete set of training examples.

Extensions of CML would include the study of how to represent other modal logics such as temporal [23], dynamic [25], and conditional logics of normality [5], as well as inference and learning of (fragments) of first-order modal logic [2]. The addition of a time variable to the approach presented here allows for the representation of knowledge evolution. This could be implemented using labelled transitions from one knowledge state to the next with a linear time flow, where each time point is associated with a state of knowledge, i.e. with a network ensemble, as hinted at in [16].

Finally, one could think of the system presented here as a first step towards a model construction algorithm, which in turn allows for investigations in model checking of distributed systems in a connectionist setting. CML can be seen as a starting point towards the construction of a connectionist theorem prover for modal logics, possibly to be implemented in hardware as a neural network. In summary, we see CML as a theoretical model addressing the need for integrated distributed knowledge representation, computation, and learning mechanisms in artificial intelligence and computer science.

Acknowledgements

We are grateful to S. Holldobler and the anonymous referees for their useful comments. Artur Garcez is partly funded by The Royal Society, UK. Luis Lamb is partly funded by the Brazilian Research Council CNPq and by the CAPES foundation.

References

[1] V. Ajjanagadde, Rule-Based Reasoning in Connectionist Networks, Ph.D. Thesis, University of Minnesota, 1997.
[2] M. Baldoni, L. Giordano, A. Martelli, A modal extension of logic programming: Modularity, beliefs and hypothetical reasoning, Journal of Logic and Computation 8 (5) (1998) 597–635.
[3] N.K. Bose, P. Liang, Neural Networks Fundamentals with Graphs, Algorithms, and Applications, McGraw-Hill, 1996.
[4] G. Brewka, T. Eiter, Preferred answer sets for extended logic programs, Artificial Intelligence 109 (1999) 297–356.
[5] K. Broda, D.M. Gabbay, L.C. Lamb, A. Russo, Labelled natural deduction for conditional logics of normality, Logic Journal of the IGPL 10 (2) (2002) 123–163.
[6] K. Broda, D.M. Gabbay, L.C. Lamb, A. Russo, Compiled Labelled Deductive Systems: A Uniform Presentation of Non-Classical Logics, Research Studies Press, Institute of Physics Publishing, 2004.
[7] M.A. Castilho, L. Farinas del Cerro, O. Gasquet, A. Herzig, Modal tableaux with propagation rules and structural rules, Fundamenta Informaticae 32 (1997) 281–297.
[8] A. Chagrov, M. Zakharyaschev, Modal Logic, Clarendon Press, Oxford, 1997.
[9] K.L. Clark, Negation as failure, in: H. Gallaire, J. Minker (Eds.), Logic and Databases, Plenum Press, New York, 1978, pp. 293–322.
[10] I. Cloete, J.M. Zurada (Eds.), Knowledge-Based Neurocomputing, The MIT Press, 2000.
[11] G. Cybenco, Approximation by superposition of sigmoidal functions, in: Mathematics of Control, Signals and Systems, vol. 2, 1989, pp. 303–314.
[12] A.S. d'Avila Garcez, Extended theory refinement in knowledge-based neural networks, in: Proceedings of IEEE International Joint Conference on Neural Networks, IJCNN'02, Honolulu, Hawaii, 2002.
[13] A.S. d'Avila Garcez, Fewer epistemological challenges for connectionism, in: S.B. Cooper, B. Lowe, L. Torenvliet (Eds.), Proceedings of Computability in Europe, CiE 2005, in: LNCS, vol. 3526, Springer-Verlag, Amsterdam, The Netherlands, June 2005, pp. 139–149.
[14] A.S. d'Avila Garcez, K. Broda, D.M. Gabbay, Symbolic knowledge extraction from trained neural networks: A sound approach, Artificial Intelligence 125 (2001) 155–207.
[15] A.S. d'Avila Garcez, K. Broda, D.M. Gabbay, Neural-Symbolic Learning Systems: Foundations and Applications, in: Perspectives in Neural Computing, Springer-Verlag, 2002.
[16] A.S. d'Avila Garcez, L.C. Lamb, Reasoning about time and knowledge in neural-symbolic learning systems, in: S. Thrun, L. Saul, B. Schoelkopf (Eds.), Proceedings of NIPS 2003, in: Advances in Neural Information Processing Systems, vol. 16, MIT Press, Vancouver, Canada, 2004, pp. 921–928.
[17] A.S. d'Avila Garcez, L.C. Lamb, Neural-symbolic systems and the case for non-classical reasoning, in: S. Artemov, H. Barringer, A.S. d'Avila Garcez, L.C. Lamb, J. Woods (Eds.), We Will Show Them! Essays in Honour of Dov Gabbay, International Federation for Computational Logic, College Publications, London, 2005, pp. 469–488.
[18] A.S. d'Avila Garcez, G. Zaverucha, The connectionist inductive learning and logic programming system, Applied Intelligence 11 (1) (1999) 59–77.
[19] R. Fagin, J. Halpern, Y. Moses, M. Vardi, Reasoning about Knowledge, MIT Press, 1995.
[20] M. Fitting, Proof Methods for Modal and Intuitionistic Logics, Reidel, Dordrecht, 1983.
[21] M. Fitting, Metric methods: Three examples and a theorem, Journal of Logic Programming 21 (1994) 113–127.
[22] D.M. Gabbay, Labelled Deductive Systems, vol. 1, Clarendom Press, Oxford, 1996.
[23] D.M. Gabbay, I. Hodkinson, M. Reynolds, Temporal Logic: Mathematical Foundations and Computational Aspects, Oxford University Press, 1994.

[24] M. Gelfond, V. Lifschitz, Classical negation in logic programs and disjunctive databases, New Generation Computing 9 (1991) 365–385.
[25] D. Harel, Dynamic logic, in: D.M. Gabbay, F. Guenthner (Eds.), Handbook of Philosophical Logic, vol. 2, D. Reidel, Boston, 1984, pp. 497–604.
[26] S. Haykin, Neural Networks: A Comprehensive Foundation, Prentice Hall, 1999.
[27] S. Holldobler, Y. Kalinke, Toward a new massively parallel computational model for logic programming, in: Proceedings of the Workshop on Combining Symbolic and Connectionist Processing, ECAI 94, 1994, pp. 68–77.
[28] G.E. Hughes, M.J. Cresswell, A New Introduction to Modal Logic, Routledge, London, New York, 1996.
[29] M.R.A. Huth, M.D. Ryan, Logic in Computer Science: Modelling and Reasoning about Systems, Cambridge Univ. Press, 2000.
[30] C. Lewis, A Survey of Symbolic Logic, University of California Press, Berkeley, 1918.
[31] J.W. Lloyd, Foundations of Logic Programming, Springer-Verlag, 1987.
[32] T.M. Mitchell, Machine Learning, McGraw-Hill, 1997.
[33] M.A. Orgun, W. Ma, An overview of temporal and modal logic programming, in: Proc. Inl. Conf. Temporal Logic, in: LNAI, vol. 827, Springer, 1994, pp. 445–479.
[34] D.E. Rumelhart, G.E. Hinton, R.J. Williams, Learning internal representations by error propagation, in: D.E. Rumelhart, J.L. McClelland (Eds.), Parallel Distributed Processing, vol. 1, MIT Press, 1986, pp. 318–362.
[35] Y. Sakakibara, Programming in modal logic: An extension of PROLOG based on modal logic, in: Logic Programming 86, in: LNCS, vol. 264, Springer, 1986, pp. 81–91.
[36] L. Shastri, Advances in SHRUTI: A neurally motivated model of relational knowledge representation and rapid inference using temporal synchrony, Applied Intelligence 11 (1999) 79–108.
[37] R. Sun, Robust reasoning: Integrating rule-based and similarity-based reasoning, Artificial Intelligence 75 (2) (1995) 241–296.
[38] R. Sun, F. Alexandre, Connectionist Symbolic Integration, Lawrence Erlbaum Associates, 1997.
[39] S.B. Thrun, J. Bala, E. Bloedorn, I. Bratko, B. Cestnik, J. Cheng, K. De Jong, S. Dzeroski, S.E. Fahlman, D. Fisher, R. Haumann, K. Kaufman, S. Keller, I. Kononenko, J. Kreuziger, R.S. Michalski, T. Mitchell, P. Pachowicz, Y. Reich, H. Vafaie, K. Van de Welde, W. Wenzel, J. Wnek, J. Zhang, The MONK's problems: A performance comparison of different learning algorithms, Technical Report CMU-CS-91-197, Carnegie Mellon University, 1991.
[40] G.G. Towell, J.W. Shavlik, Knowledge-based artificial neural networks, Artificial Intelligence 70 (1) (1994) 119–165.
[41] L.G. Valiant, Three problems in computer science, Journal of the ACM 50 (1) (2003) 96–99.
[42] J. van Benthem, Modal Logic and Classical Logic, Bibliopolis, Napoli, 1983.
[43] J. van Benthem, Correspondence theory, in: D.M. Gabbay, F. Guenthner (Eds.), Handbook of Philosophical Logic, D. Reidel Publishing Company, Dordrecht, 1984, pp. 167–247 (Chapter II.4).
[44] M.Y. Vardi, Why is modal logic so robustly decidable, in: N. Immerman, P. Kolaitis (Eds.), Descriptive Complexity and Finite Models, in: Discrete Mathematics and Theoretical Computer Science, vol. 31, DIMACS, 1997, pp. 149–184.
[45] Z.H. Zhou, Y. Jiang, S.F. Chen, Extracting symbolic rules from trained neural network ensembles, AI Communications 16 (1) (2003) 3–15.