Garcez, A. d'Avila, Lamb, L. C. & Gabbay, D. M. (2007). Connectionist modal logic: Representing modalities in neural networks. Theoretical Computer Science, 371(1-2), 34 - 53. doi: 10.1016/j.tcs.2006.10.023 <http://dx.doi.org/10.1016/j.tcs.2006.10.023 >

Connectionist Modal Logics for Distributed Knowledge Representation

Artur S. d'Avila Garcez[δ], Luís C. Lamb[λ],
Dov M. Gabbay[γ] and Krysia B. Broda[β]

[δ]Dept. of Computing, City University, London EC1V 0HB
aag@soi.city.ac.uk

[λ]Instituto de Informática-PPGC, UFRGS, Porto Alegre, 91501-970, Brazil
lamb@inf.ufrgs.br

[γ]Dept. of Computer Science, King's College, London WC2R 2LS, UK
dg@dcs.kcl.ac.uk

[β]Dept. of Computing, Imperial College, London, SW72BZ, UK
kb@doc.ic.ac.uk

Abstract

Neural-Symbolic integration concerns the application of problem-specific symbolic knowledge within the neurocomputing paradigm. In this paper, we propose a new approach for representing, reasoning and learning modal logics in a connectionist framework. We do so by extending the *C-IL²P* system [15, 16] to allow the representation of modal theories in neural networks. This is achieved by a *Modalities Algorithm* that translates any modal logic program into a neural networks ensemble. In addition, we prove that the ensemble computes a fixed-point semantics of the original modal theory. An immediate result of our approach is the ability to perform learning from examples efficiently by applying Backpropagation to each network of the ensemble, where each network learns a possible world representation. We also show the effectiveness of our approach as a distributed knowledge representation and learning mechanism by applying it to the well-known muddy children puzzle. Our approach paves the way for the integrated representation and learning of distributed knowledge, with a broad range of applications from practical reasoning to evolving multi-agent systems.

Keywords: 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' learning implicitly encodes patterns and their generalizations in the networks' weights, so reflecting the statistical properties of the trained data [6]. It has been indicated that neural networks can outperform symbolic learning systems, especially when data are noisy [33, 46]. This result, due also to the massively parallel architecture of neural networks, contributed decisively to the growing interest in developing *Neural-Symbolic Learning Systems*, i.e., hybrid systems based on neural networks that are capable of learning from examples and background knowledge [15].

Nevertheless, most of the efforts so far have been directed towards the representation of classical logic in a connectionist setting [1, 30, 43, 44, 45]. In particular, neural systems have not been shown to fully represent and learn expressive languages such as modal and predicate logics [11]. Our aim is not only to represent such logics in neural networks, but also to allow the derived networks to be trained with examples.

In this paper, we propose a new approach for the representation and learning of propositional modal logics in a connectionist framework. We use the language of Modal Logic Programming [42] extended to allow modalities such as necessity and possibility in the head of clauses [38][1]. We then present an algorithm to set up an ensemble of *Connectionist Inductive Learning and Logic Programming (C-IL^{2}P)* networks [15, 16], each network being an extension of Holldobler and Kalinke's parallel model for logic programming [29]. A theorem then shows that the resulting ensemble computes a fixed-point semantics of the original modal theory. In other words, the network ensemble can be seen as a massively parallel system for modal logic 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 and reasoning [17, 32].

In addition, the merging of theory (background knowledge) and data learning (learning from examples) in neural networks has been indicated to provide a learning system that is more effective than purely symbolic and purely connectionist systems [20, 46, 47]. In order to do so, one might first translate the background knowledge into a neural network initial architecture, and then train it with examples using, for example, *Backpropagation*, the neural learning algorithm most successfully applied in real-world problems such as DNA sequence analysis and pattern recognition [40, 27]. Learning in the connectionist modal logic system is achieved by training each individual *C-IL^{2}P* network, which in turn corresponds to the current knowledge of an agent within a possible world, using *Backpropagation*. This is exemplified by learning the knowledge of an agent in the muddy children puzzle using *C-IL^{2}P* networks.

The connectionist modal logic framework presented here renders Neural-Symbolic Learning Systems with the ability to provide a more expressive rep-

[1] Notice that Sakakibara's Modal Logic Programming [42] is referred to as Modal Prolog in the well-known temporal and modal logic programming survey of Orgun and Ma [38].

resentation language. Moreover, as modal logics have been the subject of intense investigation in knowledge representation [17], this work is a contribution towards the learning of modally defined aspects of reasoning using neural networks. It contributes to the integration of both research programmes - neural networks and modal logics - into a unified foundation.

In Section 2, we briefly present the basic concepts of modal logic and artificial neural networks used throughout this paper. In Section 3, we present a Modalities Algorithm that translates extended modal programs into artificial neural networks. The network obtained is an ensemble of simple *C-IL*^{2}*P* networks, each representing a (learnable) possible world. We then show that the network computes a fixed-point semantics of the given modal theory, thus proving the correctness of the Modalities Algorithm. In Section 4, we apply the system to the muddy children puzzle, a well-known problem of distributed reasoning in multi-agent environments. 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. We also extend Sakakibara's Modal Logic Programming [38, 42] to allow modalities $\Box$ and $\Diamond$ in the head of the clauses and default negation $\sim$ in the body of the clauses[2] [10]. Finally, we define a fixed-point semantics for such extension.

2.1 Modal Logic and Extended Modal Programs

Modal logic began with the analysis of concepts such as necessity and possibility under a philosophical perspective [31, 34]. A main feature of modal logics is the use of *possible world semantics* (proposed by Kripke and Hintikka) which has significantly contributed to the development of new non-classical logics, many of which have had a great impact in computing science [2, 37]. 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 to study mathematical necessity (in the logic of provability), time, knowledge, belief, obligation and other concepts and modalities [9]. 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 [17]. The basic definitions about modal logics that we shall use in this paper are as follows. As usual, the language of propositional modal logic extends the language of propostional logic

[2] The use of classical negation $\neg$ in the body of clauses is allowed as done in [25] by renaming each literal of the form $\neg A$ by a new literal A^* not present in the language.

with the $\Box$ and $\Diamond$ operators. Moreover, we assume that any clause is ground over a finite domain (i.e. they contain no variables).

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 $MA_1, ..., MA_n \rightarrow A$.*

We define *extended modal programs* as modal programs extended with modalities $\Box$ and $\Diamond$ in the head of clauses, and default negation $\sim$ in the body of clauses. In addition, each clause is labelled by the possible world in which they hold, similarly to Gabbay's Labelled Deductive Systems [21].

Definition 3 *An* extended modal program *is a finite set of clauses C of the form $\omega_i : ML_1, ..., ML_n \rightarrow MA$, where ω_i is a label representing a world in which the associated clause holds, and a finite set of relations $\mathcal{R}(\omega_i, \omega_j)$ between worlds ω_i and ω_j in C.*

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. Formulas in modal logic 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 α is said to be true at a possible world ω of a model $\mathcal{M}$, written $(\mathcal{M}, \omega) \models \alpha$ if one of the following satisfiability conditions 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) $(\mathcal{M}, \omega) \models \neg\alpha$ *iff* $(\mathcal{M}, \omega) \not\models \alpha$
(iii) $(\mathcal{M}, \omega) \models \alpha \wedge \beta$ *iff* $(\mathcal{M}, \omega) \models \alpha$ *and* $(\mathcal{M}, \omega) \models \beta$
(iv) $(\mathcal{M}, \omega) \models \alpha \vee \beta$ *iff* $(\mathcal{M}, \omega) \models \alpha$ *or* $(\mathcal{M}, \omega) \models \beta$
(v) $(\mathcal{M}, \omega) \models \alpha \rightarrow \beta$ *iff* $(\mathcal{M}, \omega) \not\models \alpha$ *or* $(\mathcal{M}, \omega) \models \beta$
(vi) $(\mathcal{M}, \omega) \models \Box\alpha$ *iff for all* $\omega_1 \in \Omega$, *if* $\mathcal{R}(\omega, \omega_1)$ *then* $(\mathcal{M}, \omega_1) \models \alpha$
(vii) $(\mathcal{M}, \omega) \models \Diamond\alpha$ *iff there exists a* ω_1 *such that* $\mathcal{R}(\omega, \omega_1)$ *and* $(\mathcal{M}, \omega_1) \models \alpha$.

A variety of proof procedures for modal logics has been developed over the years, e.g., [18, 41]. In some of these, formulas are labelled by the worlds in which they hold, thus facilitating the modal reasoning process (see [41] for a discussion on this topic). In the natural deduction-style rules below, the notation $\omega : \alpha$ represents that the formula α holds at the possible world ω. Moreover, the

Table 1: Rules for modality operators

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

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

explicit reference to the accessibility relation also helps in deriving what formula holds in the worlds which are related by $\mathcal{R}$. The rules we shall represent using *C-IL*^{2}*P* are similar to the ones presented in [41], which we reproduce below.

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

Semantics for Extended Modal Logic Programs

In what follows, we define a model-theoretic semantics for extended modal programs. When computing the semantics of the program, we have to consider both the fixed-point of a particular world, and the fixed-point of the program as a whole. When computing the fixed-point in each world, 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 of 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 [7] to it. When considering interacting worlds, there are two cases to be addressed, according to the $\Box I$ and $\Diamond I$ rules in Table 1, together with the accessibility relation $\mathcal{R}$, which might render additional consequences in each world.

Definition 6 *(*Herbrand Universe for Extended Modal Logic Programs*) Let $\mathcal{P}$ be an extended modal logic program. The* Herbrand universe *$U_\mathcal{P}$ of $\mathcal{P}$ is the set of all ground atoms which can be formed out of the constants and function symbols appearing in $\mathcal{P}$. Each term has the same interpretation in every world.*

Definition 7 *(*Herbrand Base of Extended Modal Logic Programs*) Let $\mathcal{P} = \{\mathcal{P}_1, ..., \mathcal{P}_k\}$ be an extended modal program, where each $\mathcal{P}_i$ is the set of modal*

clauses that hold in a world ω_i *(*$1 \leq i \leq k$*). The* Herbrand base $B_{\mathcal{P}}$ *of an extended modal program is the set of all ground predicates constructed by using predicate symbols from* $\mathcal{P}$ *with ground atoms from the Herbrand universe* $U_{\mathcal{P}}$ *as arguments.*

Definition 8 (Modal Immediate Consequence Operator) *Let* $\mathcal{P} = \{\mathcal{P}_1, ..., \mathcal{P}_k\}$ *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* $B_{\mathcal{P}}$ *be the Herbrand base of* $\mathcal{P}$ *and* I *be a Herbrand interpretation for* $\mathcal{P}_i$*. The mapping* $MT_{\mathcal{P}_i} : 2^{B_{\mathcal{P}}} \rightarrow 2^{B_{\mathcal{P}}}$ *in* ω_i *is defined as follows:* $MT_{\mathcal{P}_i}(I) = \{MA \in B_{\mathcal{P}} \mid$ *either (i) or (ii) or (iii) below holds*$\}$.

(i) $ML_1, ..., ML_n \rightarrow MA$ *is a clause in* $\mathcal{P}_i$ *and* $\{ML_1, ..., ML_n\} \subseteq I$*;*

(ii) $M = \Diamond$ *and there exists a world* ω_j *such that* $\mathcal{R}(\omega_i, \omega_j)$*,* $ML_1, ..., ML_m \rightarrow A$ *is a clause in* $\mathcal{P}_j$ *and* $\{ML_1, ..., ML_m\} \subseteq J$*, where* J *is a Herbrand interpretation for* $\mathcal{P}_j$*;*

(iii) $M = \Box$ *and for each world* ω_j *such that* $\mathcal{R}(\omega_i, \omega_j)$*,* $ML_1, ..., ML_o \rightarrow A$ *is a clause in* $\mathcal{P}_j$ *and* $\{ML_1, ..., ML_o\} \subseteq K$*, where* K *is a Herbrand interpretation for* $\mathcal{P}_j$*.*

Definition 9 (Global Modal Immediate Consequence Operator) *Let* $\mathcal{P} = \{\mathcal{P}_1, ..., \mathcal{P}_k\}$ *be an extended modal program. Let* $B_{\mathcal{P}}$ *be the Herbrand base of* $\mathcal{P}$ *and* I_i *be a Herbrand interpretation for* $\mathcal{P}_i$ *(*$1 \leq i \leq k$*). The mapping* $MT_{\mathcal{P}} : 2^{B_{\mathcal{P}}} \rightarrow 2^{B_{\mathcal{P}}}$ *is defined as follows:* $MT_{\mathcal{P}}(I_1, ..., I_k) = \bigcup_{j=1}^{k} \{MT_{\mathcal{P}_j}\}$.

In the case of definite extended modal programs, by renaming each modal atom MA_i by a new atom A_j, we can apply the following result of Ramanujam [39], regarding the fixed-point semantics of distributed definite logic programs[3].

Theorem 1 *(*Minimal Model of Distributed Programs *[39]) For each distributed definite logic program* $\mathcal{P}$*, the function* $MT_{\mathcal{P}}$ *has a unique fixed-point. The sequence of all* $MT^m_{\mathcal{P}}(I_1, ..., I_k), m \in \mathbb{N}$*, converges to this fixed-point* $MT^{\varpi}_{\mathcal{P}}(I_1, ..., I_k)$*, for each* $I_i \subseteq 2^{B_{\mathcal{P}}}$*.*

In order to provide a fixed-point semantics for extended modal programs, we have to extend the definition of *acceptable* programs [3, 4].

Definition 10 *(*Level Mapping*) Let* $\mathcal{P}$ *be a general logic program.* A level mapping *for* $\mathcal{P}$ *is a function* $|\ | : B_{\mathcal{P}} \rightarrow \mathbb{N}$ *from ground atoms to natural numbers. For* $A \in B_{\mathcal{P}}$*,* $|A|$ *is called the* level *of* A *and* $|\sim A| = |A|$*.*

Definition 11 *(*Acceptable Programs*) Let* $\mathcal{P}$ *be a program,* $|\ |$ *a level mapping for* $\mathcal{P}$*, and* $\mathbf{I}$ *a model of* $\mathcal{P}$*.* $\mathcal{P}$ *is called* acceptable *w. r. t* $|\ |$ *and* $\mathbf{I}$ *if for every clause* $L_1, ..., L_k \rightarrow A$ *in* $\mathcal{P}$ *the following implication holds. If* $\mathbf{I} \vDash \bigwedge_{j=1}^{i-1} L_j$ *then* $|A| > |L_j|$ *for* $1 \leq i \leq k$*.*

[3] Definite distributed logic programs are simply tuples $< P_1, ..., P_n >$ where each P_i is the set of clauses (program) associated with each site i. Each P_i is called a *component program* of the *composite program* [39]. Mutatis mutandis, there is a correspondence between this notion and that of an extended modal program.

Theorem 2 *(*Minimal Model of Acceptable Programs *[19]) For each acceptable program $\mathcal{P}$, the function $T_\mathcal{P}$ has a unique fixed-point*[4]*. The sequence of all $T^m_\mathcal{P}(I), m \in \mathbb{N}$, converges to this fixed-point $T^\varpi_\mathcal{P}(I)$ (which is identical to the* stable model *of $\mathcal{P}$ [24])*[5]*, for each $I \subseteq B_\mathcal{P}$.*

Definition 12 *(*Acceptable Extended Modal Programs*) An extended modal program $\mathcal{P}$ is acceptable iff the program $\mathcal{P}'$, obtained by renaming each modal literal ML_i in $\mathcal{P}$ by a new literal L_j not in the language $\mathcal{L}$ is acceptable.*

Following [5], one can construct the semantics of extended modal programs by considering extended modal ground formulas in order to compute a fixed-point. As a result, one can associate with every extended modal program a modal ground program (the modal closure of the program) so that both programs have the same models. Hence, the classical results about the fixed-point semantics of logic programming can be applied directly to the modal ground closure of a program. Thus, Theorem 3, below, follows directly from Theorems 2 and 1.

Theorem 3 *(*Minimal Model of Acceptable Extended Modal Programs*) For each acceptable extended modal program $\mathcal{P}$, the function $MT_\mathcal{P}$ has a unique fixed-point. The sequence of all $MT^m_\mathcal{P}(I_1, ..., I_k), m \in \mathbb{N}$, converges to this fixed-point $MT^\varpi_\mathcal{P}(I_1, ..., I_k)$, for each $I_i \subseteq 2^{B_\mathcal{P}}$.*

Finally, note that in the above semantics, the choice of an arbitrary world for $\Diamond$ elimination (made before the computation of $MT_\mathcal{P}$) may lead to different fixed-points of a given extended modal program. Such a choice is similar to the approach adopted by Gabbay in [22], in which one chooses a point in the future for execution and then backtracks if judged necessary (and at all possible).

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 $A_i(t)$, 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 $W_{ji} \in \Re$ denotes the *weight* associated with such a connection.

We start by characterising the neuron's *functionality* (see Figure 1). The *activation state* of a neuron i at time t ($A_i(t)$) is a bounded real or integer number. The output of neuron i at time t ($O_i(t)$) is given by the *output rule* f_i, such that $O_i(t) = f_i(A_i(t))$. The input potential of neuron i at time t ($U_i(t)$) is obtained by applying the *propagation rule* of neuron i (g_i) such that $U_i(t) =$

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

[5] A Herbrand interpretation I of a general logic program $\mathcal{P}$ is called stable iff $T_{\mathcal{P}_I}(I) = I$, where $\mathcal{P}_I$ is the definite program obtained from applying the Gelfond-Lifschitz Transformation over $\mathcal{P}$.

$g_i(I_i(t), W_i)$, where $I_i(t)$ contains the input signals $(x_1(t), x_2(t), ..., x_n(t))$ to neuron i at time t, and W_i denotes the weight vector $(W_{i1}, W_{i2}, ..., W_{in})$ 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 $A_i(t + \Delta t)$ is given by its *activation rule* h_i, which is a function of the neuron's current activation state and input potential, i.e. $A_i(t + \Delta t) = h_i(A_i(t), U_i(t))$, and the neuron's new output value $O_i(t + \Delta t) = f_i(A_i(t + \Delta t))$.

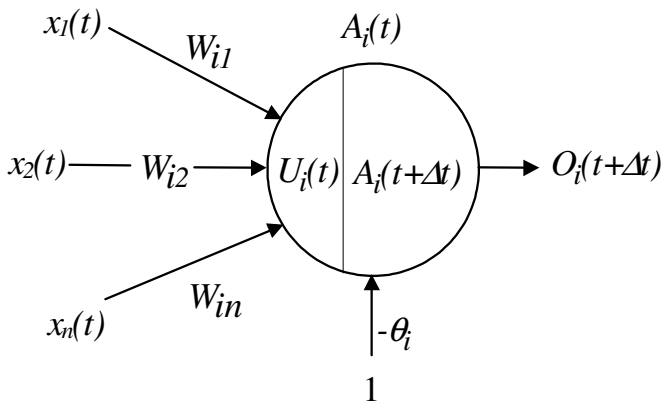

Figure 1: The Processing Unit or Neuron.

In general, h_i does not depend on the previous activation state of the unit, that is, $A_i(t + \Delta t) = h_i(U_i(t))$, 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. $O_i(t) = A_i(t)$.

The units of a neural network can be organised in layers. A *n-layer feed-forward network* N is an acyclic graph. N consists of a sequence of layers and connections between successive 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 $i + 1$-*st* layer, we say that N is a *fully-connected network* (see Figure 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 in the input and output layer, respectively. A multilayer feedforward network N computes a function $f : \Re^r \rightarrow \Re^s$ 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

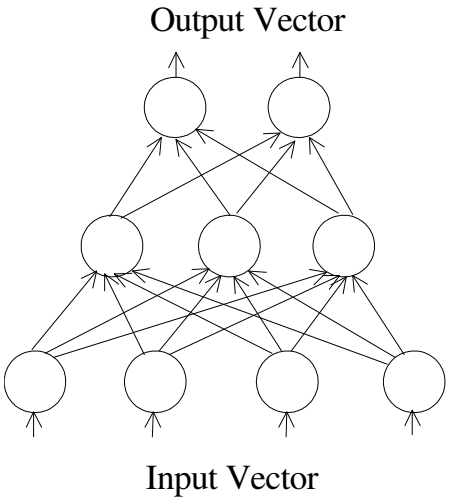

Figure 2: A typical feedforward Neural Network.

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* [40], the learning process occurs as follows: given a set of input patterns $\mathbf{i}^i$ and corresponding target vectors $\mathbf{t}^i$, the network's outputs $\mathbf{o}^i = f(\mathbf{i}^i)$ may be compared with the targets $\mathbf{t}^i$, and an error such as

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

can be computed. This error depends on the set of example ($(\mathbf{i}, \mathbf{t})$ pairs) and may be minimised by *gradient descent*, i.e. by the iterative application of changes

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

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

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

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, the *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. Equation 4 shows how Backpropagation with momentum is expressed mathematically.

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

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

The ultimate measure of success of neural networks learning 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.

In this paper, we concentrate on single hidden layer feedforward networks, since they can approximate any (Borel) measurable function arbitrarily well, regardless of the dimension of the input space [12]. In this sense, single hidden layer networks are *approximators* of virtually any function of interest. We also use *bipolar* semi-linear activation functions $h(x) = \frac{2}{1+e^{-\beta x}} - 1$ with inputs in $\{-1, 1\}$, and the *Backpropagation* learning algorithm to perform training from examples.

3 Connectionist Modal Logic

In this section, we extend the *C-IL^{2}P* system to represent modal theories by using ensembles of *C-IL^{2}P* networks. In [29], it is shown that the semantics of any first order acyclic logic program [35] can be approximated by a single hidden layer recurrent neural network (although no translation algorithm is presented). Since this is precisely the kind of network used by *C-IL^{2}P,* ensembles of *C-IL^{2}P* networks can enhance the expressive power of the system, yet maintaining the simplicity needed for performing inductive learning efficiently. In this section, we also present an efficient translation algorithm from extended modal programs to artificial neural networks.

We start with a simple example. It briefly illustrates how an ensemble of *C-IL^{2}P* 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.

Example 1 *Figure 3 shows an ensemble of three* C-IL^{2}P *networks* $(\omega_1, \omega_2, \omega_3)$*, which might "communicate" in many different ways. If we look at* ω_1*,* ω_2 *and* ω_3 *as* possible worlds*, we might be able to incorporate modalities in the language of* C-IL^{2}P*. For example, (i) "If* $\omega_1 : \Box A$ *then* $\omega_2 : A$*" could be communicated from* ω_1 *to* ω_2 *by connecting* $\Box A$ *in the output layer of* ω_1 *to* A *in the output layer of* ω_2 *such that, whenever* $\Box A$ *is activated in* ω_1*,* A *is activated in* ω_2*. In addition, analogously to the feedback of* C-IL^{2}P *networks, we could have feedback between*

ensembles of C-IL^{2}P *networks. For example,* (ii) *"If* $(\omega_2 : A) \vee (\omega_3 : A)$ *then* $\omega_1 : \Diamond A$*" could be implemented by connecting output neurons* A *of* ω_2 *and* ω_3 *into output neuron* $\Diamond A$ *of* ω_1*, through two hidden neurons (say,* h_1 *and* h_2*) in* ω_1 *such that* $\omega_1 : h_1 \vee h_2 \rightarrow \Diamond A$*. Examples* (i) *and* (ii) *simulate, in a finite universe, the rules of* $\Box$ Elimination *and* $\Diamond$ Introduction *(see Table 1). The representation of modalities in neural networks will be described in detail in Section 3.2.*

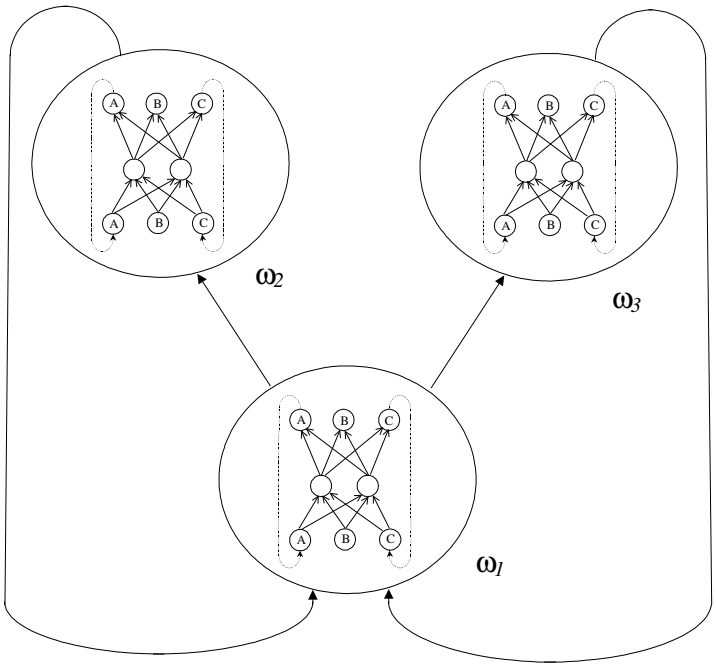

Figure 3: An ensemble of *C-IL^{2}P* networks for modeling uncertainty by using modalities and possible worlds.

Due to the simplicity of each *C-IL^{2}P* network, e.g. ω_1, performing inductive learning within each *possible world* is straightforward. The main problem to be tackled when it comes to learning in the new neural model, therefore, is how to set up the connections that establish the necessary communication between networks, e.g., ω_1 and ω_2. In the case of modal logic, such connections are defined analogously to the modal rules of natural deduction (Table 1). The *Modalities Algorithm* presented in Section 3.2 implements those rules, but first we recall how the *C-IL^{2}P* system works.

3.1 The *C-IL^{2}P* System

C-IL^{2}P [15, 16] is a massively parallel computational model based on a feedforward artificial neural network that integrates inductive learning from examples and background knowledge with deductive learning from logic programming. Following [28] (see also [29]), a *Translation Algorithm* maps a general logic program $\mathcal{P}$ into a single hidden layer neural network $\mathcal{N}$ such that $\mathcal{N}$ computes the least fixed-point of $\mathcal{P}$. This provides a massively parallel model for computing the stable model semantics of $\mathcal{P}$ [35]. In addition, $\mathcal{N}$ can be trained with examples using *Backpropagation* [40], having $\mathcal{P}$ as background knowledge. The knowledge acquired by training can then be extracted [14], closing the learning cycle (as in [47]).

Let us exemplify how *C-IL^{2}P's Translation Algorithm* works. Each rule (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) 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 2 *Consider the logic program* $\mathcal{P} = \{BC \sim D \rightarrow A; EF \rightarrow A; \rightarrow B\}$. *The* Translation Algorithm *derives the network* $\mathcal{N}$ *of Figure 4, setting weights* ($W's$) *and thresholds* ($\theta's$) *in such a way that conditions* (***C1***) *and* (***C2***) *above are satisfied. Note that, if* $\mathcal{N}$ *ought to be fully-connected, any other link (not shown in Figure 4) should receive weight zero initially.*

Note that, in Example 2, 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 rule r_l of $\mathcal{P}$. In order to compute the stable models 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 of $\mathcal{P}$. $\mathcal{N}$ will eventually converge to a stable state which is identical to the answer set of $\mathcal{P}$ [16].

Notation : Given a general logic program $\mathcal{P}$, let q denote the number of rules r_l ($1 \leq l \leq q$) occurring in $\mathcal{P}$; v, the number of literals occurring in $\mathcal{P}$; A_{min}, the minimum activation for a neuron to be considered "active" (or $true$), $A_{min} \in (0, 1)$; A_{max}, the maximum activation for a neuron to be considered "not active" (or $false$), $A_{max} \in (-1, 0)$; $h(x) = \frac{2}{1+e^{-\beta x}} - 1$, the bipolar semi-linear activation function[6]; $g(x) = x$, 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;

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

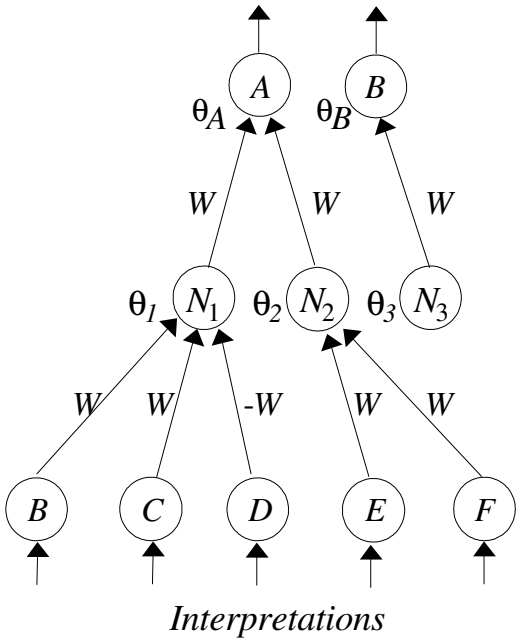

Figure 4: Sketch of a neural network for the above logic program $\mathcal{P}$.

W (resp. -W), the weight of connections associated with positive (resp. negative) literals; θ_l, the threshold of hidden neuron N_l associated with rule r_l; θ_A, the threshold of output neuron A, where A is the head of rule r_l; k_l, the number of literals in the body of rule r_l; p_l, the number of positive literals in the body of rule r_l; n_l, the number of negative literals in the body of rule r_l; μ_l, the number of rules in $\mathcal{P}$ with the same atom in the head, for each rule r_l; $MAX_{r_l}(k_l, \mu_l)$, the greater element among k_l and μ_l for rule r_l; and $MAX_{\mathcal{P}}(k_1, ..., k_q, \mu_1, ..., \mu_q)$, the greatest element among all k's and μ's of $\mathcal{P}$. We also use $\overrightarrow{k}$ as a shorthand for $(k_1, ..., k_q)$, and $\overrightarrow{\mu}$ as a shorthand for $(\mu_1, ..., \mu_q)$.

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

In the *Translation Algorithm* below, we define A_{min}, W, θ_l, and θ_A such that conditions (**C1**) and (**C2**) above are satisfied. Given a general logic program $\mathcal{P}$, consider that the literals of $\mathcal{P}$ are numbered from 1 to υ such that the input and output layers of $\mathcal{N}$ are vectors of length υ, where the i-th neuron represents the i-th literal of $\mathcal{P}$. We assume, for mathematical convenience and without loss of generality, that $A_{max} = -A_{min}$. We start by calculating $MAX_{\mathcal{P}}(\overrightarrow{k}, \overrightarrow{\mu})$ of

$\mathcal{P}$ and A_{min} such that:

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

The Translation Algorithm

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

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

2. For each rule r_l of $\mathcal{P}$ of the form $L_1, ..., L_k \rightarrow A$ $(k \geq 0)$:

   (a) Add a neuron N_l to the hidden layer of $\mathcal{N}$;

   (b) 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 $-W$;

   (c) Connect the neuron N_l in the hidden layer to the neuron A in the output layer and set the connection weight to W;

   (d) Define the threshold (θ_l) of the neuron N_l in the hidden layer as:

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

   (e) 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 of $\mathcal{N}$, 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 on $\mathcal{N}$ efficiently.

5. If $\mathcal{N}$ ought to be fully-connected, set all other connections to zero.

Theorem 4 *[15, 16] 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 $\mathrm{T}_{\mathcal{P}}$ of $\mathcal{P}$.*

3.2 The Modal *C-IL²P* System

In this section, we extend the *C-IL²P* system to deal with modalities. We use the *Translation Algorithm* presented in Section 3.1 for creating each network of the ensemble, and the following *Modalities Algorithm* for interconnecting the different networks. The *Modalities Algorithm* translates natural deduction rules into the networks. Intuitively, the accessibility relation is represented in the metalevel by connections between (sub)networks, as depicted in Figure 5, 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 (Table 1). Connections from ω_2 and ω_3 to ω_1 represent either $\Box$I or $\Diamond$I.

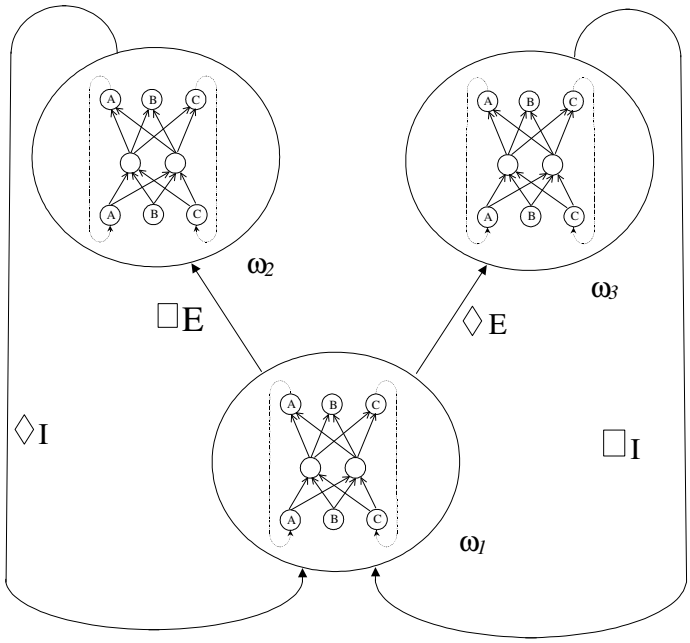

Figure 5: An ensemble of networks that represents modalities.

Let $\mathcal{P}$ be an extended modal program with clauses of the form $\omega_i : ML_1, ..., ML_k \rightarrow MA$, 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-IL²P* networks, we start by calculating $MAX_{\mathcal{P}}(\overrightarrow{k}, \overrightarrow{\mu}, n)$ of $\mathcal{P}$ and A_{min} such that:

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

which, now, also considers the number n of networks (possible worlds) in the ensemble.

The Modalities Algorithm

1. Let $\mathcal{P}_i \subseteq \mathcal{P}$ be the set of clauses labelled by ω_i in $\mathcal{P}$. Let $\mathcal{N}_i$ be the neural network that denotes $\mathcal{P}_i$. Let $W^M \in \Re$ be such that $W^M > h^{-1}(A_{\min}) + \mu_l W + \theta_A$, where μ_l, W and θ_A are obtained from *C-IL^{2}P's Translation Algorithm*[7].

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 *C-IL^{2}P's Translation Algorithm;*

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;

   (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) Connect L_j^M to L_j in $\mathcal{N}_k$ and set the connection weight to W^M.

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) Connect L_j^M to L_j in $\mathcal{N}_k$ and set the connection weight to W^M.

5. For each output neuron L_j in $\mathcal{N}_k$ such that $\mathcal{R}(\omega_i, \omega_k)$ $(0 \leq i \leq m)$, do:

   (a) Add a hidden neuron $L_j^{\vee}$ to $\mathcal{N}_i$;

   (b) Set the step function $s(x)$ as the activation function of $L_j^{\vee}$;

   (c) For each output neuron $L_j^{\Diamond}$ in $\mathcal{N}_i$, 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. Connect $L_j^{\vee}$ to $L_j^{\Diamond}$ in $\mathcal{N}_i$ and set the connection weight to W^M.

[7] Recall that μ_l is the number of connections to output neuron l.

[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.

6. For each output neuron L_j in $\mathcal{N}_k$ such that $\mathcal{R}(\omega_i, \omega_k)$ $(0 \leq i \leq m)$, do:
   (a) Add a hidden neuron $L_j^{\wedge}$ to $\mathcal{N}_i$;
   (b) Set the step function $s(x)$ as the activation function of $L_j^{\wedge}$;
   (c) For each output neuron $L_j^{\Box}$ in $\mathcal{N}_i$, 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. Connect $L_j^{\wedge}$ to $L_j^{\Box}$ in $\mathcal{N}_i$ and set the connection weight to W^M.

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

Example 3 *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* C-IL^{2}P's Translation Algorithm, *which creates three neural networks to represent the worlds* ω_1, ω_2, *and* ω_3 *(see Figure 6). 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 Figure 6.*

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 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 world* $\mathcal{N}_j$ *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 Figure 6, 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* $\wedge$. *The algorithm terminates when all output neurons have been connected.*

We are now in a 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 8).

Theorem 5 *For any extended modal program* $\mathcal{P}$ *there exists an ensemble of feedforward neural networks* $\mathcal{N}$ *with a single hidden layer and semi-linear neurons, 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* $\mathcal{N}$, *obtained by the above* Modalities Algorithm, *computes* $MT_{\mathcal{P}}$. *Throughout, we assume that* $\mathcal{N}_i$ *and* $\mathcal{N}_j$ *are two arbitrary sub-networks of* $\mathcal{N}$, *representing possible worlds* ω_i *and* ω_j, *respectively, such that* $\mathcal{R}(\omega_i, \omega_j)$. *We distinguish two*

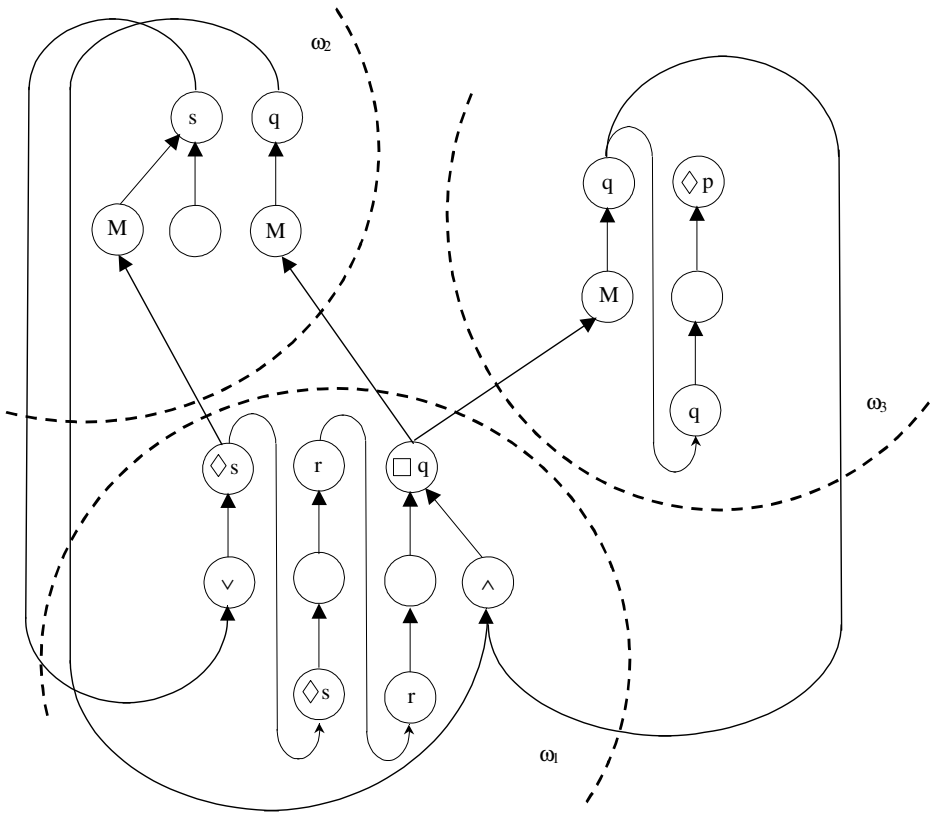

Figure 6: The ensemble of networks $\{\mathcal{N}_1, \mathcal{N}_2, \mathcal{N}_3\}$ that represents $\mathcal{P}$.

cases: (a) rules with modalities $\Box$ *and* $\Diamond$ *in the head, and (b) rules with no modalities in the head.*

(a) Firstly, note that rules with $\Box$ *in the head must satisfy* $\Box E$*, while rules 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* $ML_1, ..., ML_k \rightarrow A$ *s.t.* $ML_1, ..., ML_k$ *are satisfied by interpretation* $\mathbf{j}$*, or (ii) there exists a clause of* $\mathcal{P}_i$ *of the form* $ML_1, ..., ML_k \rightarrow \Box A$ *s.t.* $ML_1, ..., ML_k$ *are satisfied by interpretation* $\mathbf{i}$*, or even (iii) there exists a clause of* $\mathcal{P}_i$ *of the form* $ML_1, ..., ML_k \rightarrow \Diamond A$ *s.t.* $ML_1, ..., ML_k$ *are satisfied by interpretation* $\mathbf{i}$*, and the* Modalities Algorithm *(step 3a) has selected* $\mathcal{N}_j$ *as the arbitrary network* $\mathcal{N}_k$*.*

(←) (i) results directly from Theorem 4. (ii) and (iii) share the same proof, as follows: from Theorem 4, we know that if $ML_1, ..., ML_k$ *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 ($MA > A_{min}$) then A^M presents activation* 1*. As a result, the minimum activation of A is $h(W^M - \mu_A W - \theta_A)$. Now, since $W^M > h^{-1}(A_{min}) + \mu_A W + \theta_A$, we have $h(W^M - \mu_A W - \theta_A) > A_{min}$ and, therefore, A is active ($A > A_{min}$). ($\rightarrow$) 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 4, $\mathcal{N}_j$ computes the fixed-point operator $T_{\mathcal{P}_j}$ of $\mathcal{P}_j$. Thus, if $ML_1, ..., ML_k$ is not satisfied by $\mathbf{j}$ then A is not active in $\mathcal{N}_j$.*

(b) Rules 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 ($\Box A > A_{min}$) if and only if: (i) there exists a clause of $\mathcal{P}_i$ of the form $ML_1, ..., ML_k \rightarrow \Box A$ s.t. $ML_1, ..., ML_k$ are satisfied by interpretation $\mathbf{i}$, or (ii) for all $\mathcal{N}_j$, there exists a clause of $\mathcal{P}_j$ of the form $ML_1, ..., ML_k \rightarrow A$ s.t. $ML_1, ..., ML_k$ are satisfied by interpretation $\mathbf{j}$. Each neuron $\Diamond A$ in the output layer of $\mathcal{N}_i$ is active ($\Diamond A > A_{min}$) if and only if: (iii) there exists a clause of $\mathcal{P}_i$ of the form $ML_1, ..., ML_k \rightarrow \Diamond A$ s.t. $ML_1, ..., ML_k$ are satisfied by interpretation $\mathbf{i}$, or (iv) there exists a clause of $\mathcal{P}_j$ of the form $ML_1, ..., ML_k \rightarrow A$ s.t. $ML_1, ..., ML_k$ are satisfied by interpretation $\mathbf{j}$.

($\leftarrow$) (i) and (iii) result directly from Theorem 4. (ii) and (iv) are proved in what follows: from Theorem 4, we know that if $ML_1, ..., ML_k$ are satisfied by interpretation $\mathbf{j}$ 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 $\mathcal{N}_i$ relates to, $\Box A$ 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 ($A > A_{min}$) 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* 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 $\mathcal{N}_i$ relates to, $\Diamond A$ 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 ($A > A_{min}$) 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 $h(W^M - \mu_{\Box A} W - \theta_{\Box A})$, 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 $h(W^M - \mu_{\Diamond A} W - \theta_{\Diamond A})$, and, exactly as in item (a) above, $\Diamond A$ is active in $\mathcal{N}_i$.*

($\rightarrow$) Again, (i) and (iii) result directly from Theorem 4. (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 $\mathcal{N}_i$ relates to $(0 \leq j \leq n)$. 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 $\mathcal{N}_i$ relates to* $(0 \leq j \leq n)$. *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 4, if $A^\wedge$ and $A^\vee$ present activation* 0, *$\mathcal{N}_i$ computes the fixed-point operator $MT_{\mathcal{P}_i}$ of $\mathcal{P}_i$. This completes the proof.* ■

Now, if each network $\mathcal{N}_i$ in the ensemble is transformed into a *partially recurrent network* $\mathcal{N}_i^r$ by linking the neurons in the output layer to the corresponding neurons in the input layer, the ensemble can be used to compute the extended modal program in parallel. For example, in Figure 6, if we connect output neurons $\Diamond s$ and r to input neurons $\Diamond s$ and r, respectively, in $\mathcal{N}_1$, and output neuron q to input neuron q in $\mathcal{N}_3$, 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 some of the logical consequences of the original program $\mathcal{P}$ given in Example 3. Although the computation is done in parallel in $\mathcal{N}$, following it by starting from facts (such as s in ω_2) would help in verifying this.

Corollary 6 *Let $\mathcal{P}$ be an acceptable extended modal program. There exists an ensemble of recurrent neural networks $\mathcal{N}^r$ with semi-linear neurons such that, starting from an arbitrary initial input, $\mathcal{N}^r$ converges to a stable state and yields the unique fixed-point $(MT_{\mathcal{P}}^{\varpi}(I))$ of $MT_{\mathcal{P}}$.*
Proof. *Assume that $\mathcal{P}$ is an acceptable program. By Theorem 5, $\mathcal{N}^r$ computes $MT_{\mathcal{P}}$. Recurrently connected, $\mathcal{N}^r$ computes the upwards powers $(T_{\mathcal{P}}^m(I))$ of $T_{\mathcal{P}}$. Finally, by Theorem 3, $\mathcal{N}^r$ computes the unique fixed-point $(MT_{\mathcal{P}}^{\varpi}(I))$ of $MT_{\mathcal{P}}$.* ■

Hence, in order to use $\mathcal{N}$ as a massively parallel model for modal logic, we simply have to recurrently connect $\mathcal{N}$ with fixed-weight links $W_r = 1$.

4 Case Study: The Connectionist Muddy Children Puzzle

In this section, we apply the Modal *C-IL^{2}P* System to the muddy children puzzle, a classic example of reasoning in multi-agent environments. In contrast with the also well-known wise men puzzle [17, 32], in which the reasoning process is sequential, here it is clear that a distributed (simultaneous) reasoning process happens, 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 other are muddy, but not themselves. Now, consider the following situation[9].

[9] We follow the muddy children problem description presented in [17]. We must also assume that all the agents involved in the situation are truthful and intelligent.

A caretaker announces that at least one child is muddy ($k \geq 1$) and asks "does any of you know if you have mud on your own face?"[10] 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, 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 answers *yes* the second time round.

If $k = 3$, suppose children 1, 2 and 3 are muddy. Every children can only answer *no* the first two times. Again, this allows 1 to reason as follows: "if 2 or 3 had said *yes* the second time, 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 know that they had mud in 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.

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/think 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 used analogously to 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. For example, $\mathbf{K}_j \mathrm{p}_i$ 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 ($k \leq n$). Let us consider the case in which three

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

children are playing in the garden ($n = 3$). Rule 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, rule 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 rules for children 2 and 3 are interpreted analogously.

Rules for Agent(child) 1:
r_1^1: $\mathbf{K}_1\mathrm{q}_1 \wedge \mathbf{K}_1\neg\mathrm{p}_2 \wedge \mathbf{K}_1\neg\mathrm{p}_3 \rightarrow \mathbf{K}_1\mathrm{p}_1$
r_2^1: $\mathbf{K}_1\mathrm{q}_2 \wedge \mathbf{K}_1\neg\mathrm{p}_2 \rightarrow \mathbf{K}_1\mathrm{p}_1$
r_3^1: $\mathbf{K}_1\mathrm{q}_2 \wedge \mathbf{K}_1\neg\mathrm{p}_3 \rightarrow \mathbf{K}_1\mathrm{p}_1$
r_4^1: $\mathbf{K}_1\mathrm{q}_3 \rightarrow \mathbf{K}_1\mathrm{p}_1$

Rules for Agent(child) 2:
r_1^2: $\mathbf{K}_2\mathrm{q}_1 \wedge \mathbf{K}_2\neg\mathrm{p}_1 \wedge \mathbf{K}_2\neg\mathrm{p}_3 \rightarrow \mathbf{K}_2\mathrm{p}_2$
r_2^2: $\mathbf{K}_2\mathrm{q}_2 \wedge \mathbf{K}_2\neg\mathrm{p}_1 \rightarrow \mathbf{K}_2\mathrm{p}_2$
r_3^2: $\mathbf{K}_2\mathrm{q}_2 \wedge \mathbf{K}_2\neg\mathrm{p}_3 \rightarrow \mathbf{K}_2\mathrm{p}_2$
r_4^2: $\mathbf{K}_2\mathrm{q}_3 \rightarrow \mathbf{K}_2\mathrm{p}_2$

Rules for Agent(child) 3:
r_1^3: $\mathbf{K}_3\mathrm{q}_1 \wedge \mathbf{K}_3\neg\mathrm{p}_1 \wedge \mathbf{K}_3\neg\mathrm{p}_2 \rightarrow \mathbf{K}_3\mathrm{p}_3$
r_2^3: $\mathbf{K}_3\mathrm{q}_2 \wedge \mathbf{K}_3\neg\mathrm{p}_1 \rightarrow \mathbf{K}_3\mathrm{p}_3$
r_3^3: $\mathbf{K}_3\mathrm{q}_2 \wedge \mathbf{K}_3\neg\mathrm{p}_2 \rightarrow \mathbf{K}_3\mathrm{p}_3$
r_4^3: $\mathbf{K}_3\mathrm{q}_3 \rightarrow \mathbf{K}_3\mathrm{p}_3$

Each set of rules r_m^l ($1 \leq l \leq n$, $m \in \mathbb{N}^+$) is implemented in a *C-IL*^{2}*P* network. Figure 7 shows the implementation of rules r_1^1 to r_4^1 (for agent 1)[11]. In addition, it contains p_1[12] and $\mathbf{K}q_1$, $\mathbf{K}q_2$ and $\mathbf{K}q_3$, all represented as facts. This is highlighted in grey in Figure 7. This setting complies with the presentation of the puzzle given in [32], 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. In contrast with p_1 and $\mathbf{K}q_k$ ($1 \leq k \leq 3$), $\mathbf{K}\neg\mathrm{p}_2$ and $\mathbf{K}\neg\mathrm{p}_3$ must be obtained from agents 2 and 3, respectively, whenever agent 1 does not see mud on their foreheads.

Figure 8 illustrates the interaction between three agents in the muddy children puzzle. The arrows connecting *C-IL*^{2}*P* networks implement the fact that when a child is muddy, the other children can see it. For the sake of clarity, the rules r_m^1, corresponding to neuron $\mathbf{K}_1\mathrm{p}_1$, are shown only in Figure 7. Analogously, the rules r_m^2 and r_m^3 for $\mathbf{K}_2\mathrm{p}_2$ and $\mathbf{K}_3\mathrm{p}_3$ would be represented in similar *C-IL*^{2}*P* networks. This is indicated in Figure 8 by neurons highlighted in black. In addition, Figure 8 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 Figure 7.

[11] Note that with the use *classical negation*, $\mathbf{K}p_i$ and $\mathbf{K}\neg p_i$ should be represented as two different input neurons [13]. Negative weights in the network would then represent *default negation*, allowing 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. This can be easily verified by renaming $\mathbf{K}\neg p_i$ by a new literal $\mathbf{K}p_i'$.

[12] Note the difference between p_1 (child 1 is muddy) and $\mathbf{K}p_1$ (child 1 knows she is muddy).

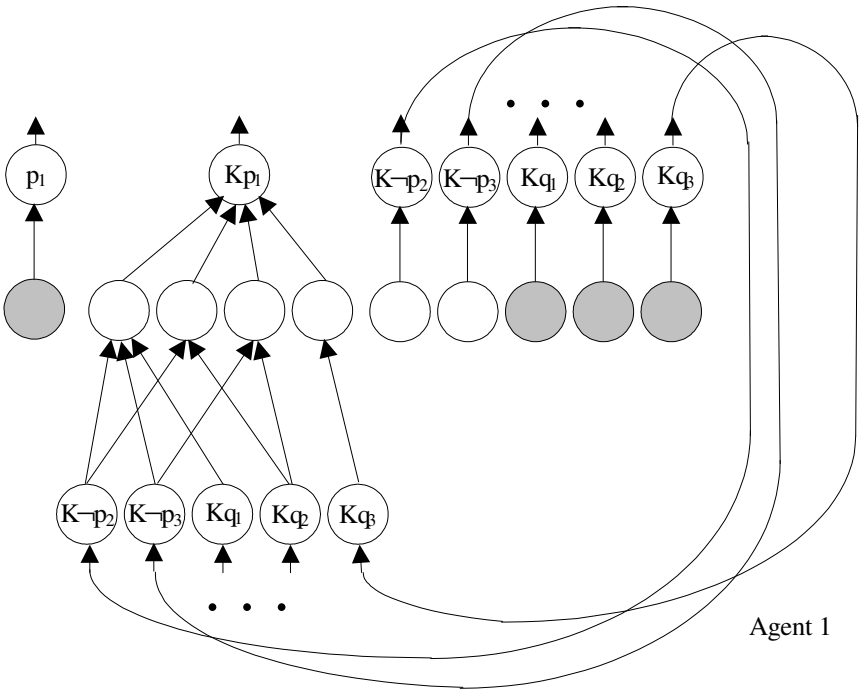

Figure 7: The implementation of rules $\{r_1^1, ..., r_4^1\}$.

4.2 Learning

As discussed in the Introduction, one of our main 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. As indicated in [20, 46], the merging of theory (background knowledge) and data learning (learning from examples) in neural networks may provide a learning system that is more effective than purely symbolic and purely connectionist systems. In order to implement such a hybrid system, one might first translate the background knowledge into a neural network initial architecture, and then train it with examples using some neural learning algorithm [47, 15].

In this section, 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 [13]. Our aim is to verify whether a particular agent i can learn from examples if he is muddy or not, i.e. learn rules r_1^i to r_4^i above.

We have performed two sets of experiments using the muddy children puz-

[13] Recall that each network in the ensemble is a *C-IL*^{2}*P* network and, therefore, can be trained with examples using standard Backpropagation.

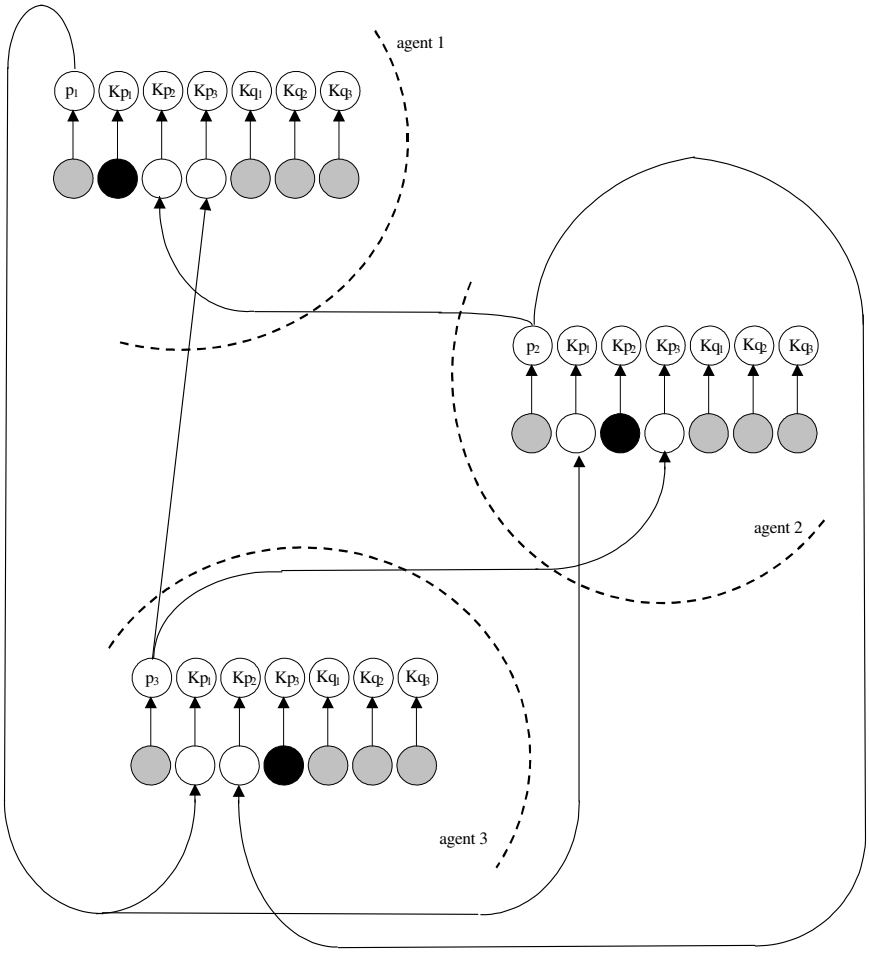

Figure 8: Interaction between agents in the muddy children puzzle.

zle in order 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 rule r_1^i : $\mathbf{K}_1 q_1 \wedge \mathbf{K}_1 \neg p_2 \wedge \mathbf{K}_1 \neg p_3 \rightarrow \mathbf{K}_1 p_1$ as background knowledge before training the networks with examples[14]. 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 [36]. One division is used for testing and the remaining $n-1$ divisions are used for

[14]Note that rule r_1^i works as a base rule for induction in the muddy children puzzle.

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 $\alpha = 0.1$, $h(x) = \frac{2}{1+e^{-\beta x}} - 1$ as activation function, and bipolar inputs in $\{-1, 1\}$.

The training sets were presented to the networks for $10,000$ epochs, and the sets of weights were updated every 16 epochs[15]. 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 $Err(\mathbf{W})$ (according to Equation 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 neural network's *generalisation* capability, the results corroborate the importance of exploiting any available background knowledge. In the first experiment, in which the connectionist modal system was trained with no background knowledge, the networks presented average test set accuracy of 84.37%. In the second experiment, in which rule 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 circumstances.

5 Conclusions and Future Work

In this paper, we have presented a new neural-symbolic learning system for modal logic. We have done so by presenting an algorithm to translate a modal theory into an ensemble of *C-IL^{2}P* neural networks [15, 16], and proved that the ensemble computes a fixed-point semantics of the theory. As a result, the ensemble can be seen as a new massively parallel model for modal logic. In addition, the ensemble can be used to learn possible world representations from examples using the standard Backpropagation learning algorithm as shown in Section 4.2. Finally, we have applied the Connectionist Modal Logic System to the muddy children puzzle, a well-known benchmark for distributed knowledge representation. We have both set up and trained networks to reason about this puzzle.

This paper opens up a new area of research in which modalities 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 networks ensembles [14, 48]. In the case of connectionist modal logic, rule extraction methods would need to consider the more expressive knowledge representation language used here. In addition, extensions to the basic modal *C-IL^{2}P* system presented in this paper include the study of how

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

to represent other modal logics such as temporal [23], dynamic [26], and conditional logics of normality [8], as well as inference and learning of (fragments) of first-order modal logic.

The addition of a time variable to the approach presented here would allow the representation of knowledge evolution. This could be implemented using labelled transitions from one knowledge state to the next, with a linear timeflow, where each timepoint is associated with a state of knowledge, i.e. a network ensemble [23]. We have already initiated the investigations on how to represent linear temporal logic in the modal *C-IL*^{2}*P* framework.

One could also think of the modal *C-IL*^{2}*P* system presented here as a first step towards a model construction algorithm, which in turn allows for investigations in model checking of distributed reasoning systems in a connectionist framework. Alternatively, the modal *C-IL*^{2}*P* system can be seen as a connectionist theorem prover for modal logics, which can be implemented in hardware as a neural network. In summary, we believe that our connectionist modal logic framework addresses the need for integrated distributed knowledge representation and learning mechanisms in Artificial Intelligence.

References

[1] V. Ajjanagadde. *Rule-Based Reasoning in Connectionist Networks.* PhD thesis, University of Minnesota, 1997.

[2] G. Antoniou. *Nonmonotonic Reasoning.* MIT Press, Cambridge, MA, 1997.

[3] K. R. Apt and N. Bol. Logic programming and negation: a survey. *Journal of Logic Programming*, 19-20:9–71, 1994.

[4] K. R. Apt and D. Pedreschi. Reasoning about termination of pure prolog programs. *Information and Computation*, 106:109–157, 1993.

[5] M. Baudinet. Temporal logic programming is complete and expressive. In *Proceedings of ACM Symposium on Principles of Programming Languages*, pages 267–280, Austin, Texas, 1989.

[6] N. K. Bose and P. Liang. *Neural Networks Fundamentals with Graphs, Algorithms, and Applications.* McGraw-Hill, 1996.

[7] G. Brewka and T. Eiter. Preferred answer sets for extended logic programs. *Artificial Intelligence*, 109:297–356, 1999.

[8] K. Broda, D. M. Gabbay, L. C. Lamb, and A. Russo. Labelled natural deduction for conditional logics of normality. *Logic Journal of the IGPL*, 10(2):123–164, 2002.

[9] A. Chagrov and M. Zakharyaschev. *Modal Logic.* Clarendon Press, Oxford, 1997.

[10] K. L. Clark. Negation as failure. In H. Gallaire and J. Minker, editors, *Logic and Databases*, pages 293–322, Plenum Press, New York, 1978.

[11] I. Cloete and J. M. Zurada, editors. *Knowledge-Based Neurocomputing*. The MIT Press, 2000.

[12] G. Cybenco. Approximation by superposition of sigmoidal functions. In *Mathematics of Control, Signals and Systems 2*, pages 303–314. 1989.

[13] 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.

[14] A. S. d'Avila Garcez, K. Broda, and D. M. Gabbay. Symbolic knowledge extraction from trained neural networks: A sound approach. *Artificial Intelligence*, 125:155–207, 2001.

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

[16] A. S. d'Avila Garcez and G. Zaverucha. The connectionist inductive learning and logic programming system. *Applied Intelligence Journal, Special Issue on Neural Networks and Structured Knowledge*, 11(1):59–77, 1999.

[17] R. Fagin, J. Halpern, Y. Moses, and M. Vardi. *Reasoning about Knowledge*. MIT Press, 1995.

[18] M. Fitting. *Proof methods for modal and intuitionistic logics*. Reidel, Dordrecht, 1983.

[19] M. Fitting. Metric methods: Three examples and a theorem. *Journal of Logic Programming*, 21:113–127, 1994.

[20] L. M. Fu. *Neural Networks in Computer Intelligence*. McGraw Hill, 1994.

[21] D. M. Gabbay. *Labelled Deductive Systems*. Clarendom Press.

[22] D. M. Gabbay. The declarative past and imperative future. In H. Barringer, editor, *Proceedings of the Colloquium on Temporal Logic and Specifications*, LNCS 398, pages 409–448. Springer-Verlag, 1989.

[23] D. M. Gabbay, I. Hodkinson, and M. Reynolds. *Temporal logic: mathematical foundations and computational aspects*. Oxford University Press, 1994.

[24] M. Gelfond and V. Lifschitz. The stable model semantics for logic programming. In *Proceedings of the fifth Logic Programming Symposium*, pages 1070–1080, 1988.

[25] M. Gelfond and V. Lifschitz. Classical negation in logic programs and disjunctive databases. *New Generation Computing*, 9:365–385, 1991.

[26] D. Harel. Dynamic logic. In D. M. Gabbay and F. Guenthner, editors, *Handbook of Philosophical Logic*, volume 2, pages 497–604. D. Reidel, Boston, 1984.

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

[28] S. Holldobler and 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*, pages 68–77, 1994.

[29] S. Holldobler, Y. Kalinke, and H. P. Storr. Approximating the semantics of logic programs by recurrent neural networks. *Applied Intelligence Journal, Special Issue on Neural Networks and Structured Knowledge*, 11(1):45–58, 1999.

[30] S. Holldobler and F. Kurfess. CHCL: A connectionist inference system. In B. Fronhofer and G. Wrightson, editors, *Parallelization in Inference Systems*, pages 318–342. Springer, 1992.

[31] G. E. Hughes and M. J. Cresswell. *A new introduction to modal logic*. Routledge, London and New York, 1996.

[32] M. R. A. Huth and M. D. Ryan. *Logic in Computer Science: Modelling and Reasoning about Systems*. Cambridge University Press, 2000.

[33] F. J. Kurfess. Neural networks and structured knowledge. In C. Herrmann, F. Reine, and A. Strohmaier, editors, *Knowledge Representation in Neural Networks*, pages 5–22. Logos-Verlag, Berlin, 1997.

[34] C. Lewis. *A Survey of Symbolic Logic*. University of California Press, Berkeley, 1918.

[35] J. W. Lloyd. *Foundations of Logic Programming*. Springer-Verlag, 1987.

[36] T. M. Mitchell. *Machine Learning*. McGraw-Hill, 1997.

[37] R. Moore. Semantical considerations on nonmonotonic logic. *Artificial Intelligence*, 25(1):75–94, 1985.

[38] M. A. Orgun and W. Ma. An overview of temporal and modal logic programming. In *Proceedings of International Conference on Temporal Logic, ICTL'94*, LNAI 827, pages 445–479. Springer, 1994.

[39] R. Ramanujam. Semantics of distributed definite clause programs. *Theoretical Computer Science*, 68:203–220, 1989.

[40] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning internal representations by error propagation. In D. E. Rumelhart and J. L. McClelland, editors, *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, volume 1, pages 318–362. MIT Press, 1986.

[41] A. Russo. Generalising propositional modal logic using labelled deductive systems. In F. Baader and K. U. Schulz, editors, *Frontiers of Combining Systems, Applied Logic Series (APLS)*, volume 3, pages 57–74. Kluwer, 1996.

[42] Y. Sakakibara. Programming in modal logic: An extension of PROLOG based on modal logic. In *Logic Programming 86*, pages 81–91. Springer LNCS 264, 1986.

[43] L. Shastri. Advances in SHRUTI: a neurally motivated model of relational knowledge representation and rapid inference using temporal synchrony. *Applied Intelligence Journal, Special Issue on Neural Networks and Structured Knowledge*, 11:79–108, 1999.

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

[45] R. Sun and F. Alexandre. *Connectionist Symbolic Integration.* Lawrence Erlbaum Associates, Hillsdale, NJ, 1997.

[46] 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, and J. Zhang. The MONK's problems: A performance comparison of different learning algorithms. Technical Report CMU-CS-91-197, Carnegie Mellon University, 1991.

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

[48] Z. H. Zhou, Y. Jiang, and S. F. Chen. Extracting symbolic rules from trained neural network ensembles. *AI Communications*, in press, 2003.