Neural-Symbolic Intuitionistic Reasoning

Artur S. d'AVILA GARCEZ[α], Luis C. LAMB[β] and Dov M. GABBAY[γ]

[α]Dept. of Computing, City University, London, EC1V 0HB, UK. aag@soi.city.ac.uk
[β]Instituto de Informática, UFRGS, Porto Alegre, RS, 91501-970, Brazil. lamb@inf.ufrgs.br
[γ]Dept. of Computer Science, King's College, London WC2R2LS, UK. dg@dcs.kcl.ac.uk

Abstract

In this paper, we present a new computational model for intuitionistic logic. We use an ensemble of *Connectionist Inductive Learning and Logic Programming* (C-ILP) neural networks to represent intuitionistic clauses, and show that for each intuitionistic program there exists a corresponding C-ILP ensemble such that the ensemble computes the fixed point of the program. This provides a massively parallel model for intuitionistic reasoning. In addition, C-ILP ensembles can be trained to adapt from examples using standard neural networks learning algorithms.
Keywords: Neural-Symbolic Integration, Intuitionistic Logic, Artificial Neural Networks.

1 Introduction

Neural-Symbolic integration is about the application of problem-specific symbolic knowledge within the connectionist paradigm [10]. Until recently, neural-symbolic systems were not able to fully represent, reason and learn expressive languages other than propositional and fragments of first-order logic [6]. However, in [12, 11] a new approach to knowledge representation and reasoning within the neural-symbolic paradigm was proposed. In [12], it was shown that Modal Logics could be effectively represented in Neural Networks; in [11], it was shown that they could be trained to encode possible world representations and temporal information. In this paper, we follow the same line of research to deal with intuitionistic logic, which is very important to the logical foundations of computation [1, 8]. We do so by setting up an ensemble of *Connectionist Inductive Learning and Logic Programming* (C-ILP) networks [13, 9, 10], each network being an extension of Holldobler and Kalinke's parallel model for Logic Programming [16], to compute a fixed point semantics of an intuitionistic theory. The networks are set up by an *Intuitionistic Algorithm* introduced in this paper. The proof that the networks compute a fixed point semantics of the associated intuitionistic theory is then given. This provides a unified computational foundation for the above areas, i.e. neural networks and intuitionistic reasoning.

In Section 2, we briefly present the basic concepts of intuitionistic logics and artificial neural networks used throughout this paper. In Section 3, we introduce the *Intuitionistic Algorithm* and prove that the network computes a fixed point semantics of the given theory, thus proving the correctness of the algorithm. Section 4 concludes and discusses directions for future work.

2 Preliminaries

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

[0]Artur Garcez is partly supported by the Nuffield Foundation. Luis Lamb is partly supported by CNPq. The authors would like to thank the referees for their comments.

2.1 Intuitionistic Logic and Programs

The language of intuitionistic logic includes propositional letters denoted by $A, B, C...$, the connectives $\neg$, $\wedge$ (sometimes abbreviated in clauses by ","), and an intuitionistic implication $\Rightarrow$. Formulas are denoted by $\alpha, \beta, \gamma...$ We define *labelled intuitionistic programs* as sets of clauses where each clause is labelled by the point in which they hold, similarly to Gabbay's Labelled Deductive Systems [14]. Note that, in what follows, each point could be seen as a possible world.

Definition 1 *(*Labelled Intuitionistic Program*) A Labelled Intuitionistic Program is a finite set of clauses C of the form $\omega_i : A_1, ..., A_n \Rightarrow A_0$, where A_k $(0 \leq k \leq n)$ are atoms and ω_i is a label representing a point in which the associated clause holds, and a finite set of relations $\mathcal{R}$ between points ω_i $(1 \leq i \leq m)$ in C.*

We shall interpret intuitionistic logics using Kripke semantics [4], which we define as follows.

Definition 2 *(*Kripke Models for Intuitionistic Propositional Logic*) Let $\mathcal{L}$ be an intuitionistic language. A* model *for $\mathcal{L}$ is a tuple $\mathcal{M} = \langle \Omega, \mathcal{R}, v \rangle$ where Ω is a set of points, v is a mapping that assigns to each $\omega \in \Omega$ a subset of $\mathcal{L}$, and $\mathcal{R}$ is a reflexive, transitive binary relation over Ω, such that:*

1. $(\mathcal{M}, \omega) \models p$ *iff* $p \in v(\omega)$ *(for proposition p)*
2. $(\mathcal{M}, \omega) \models \neg\alpha$ *iff for all ω' such that $\mathcal{R}(\omega, \omega')$, $(\mathcal{M}, \omega') \not\models \alpha$*
3. $(\mathcal{M}, \omega) \models \alpha \wedge \beta$ *iff* $(\mathcal{M}, \omega) \models \alpha$ *and* $(\mathcal{M}, \omega) \models \beta$
4. $(\mathcal{M}, \omega) \models \alpha \Rightarrow \beta$ *iff for all ω' with $\mathcal{R}(\omega, \omega')$ we have $(\mathcal{M}, \omega') \models \beta$ whenever we have $(\mathcal{M}, \omega') \models \alpha$*

2.2 Fixed Point Semantics of Intuitionistic Programs

In what follows, we define a model-theoretic semantics for labelled intuitionistic programs. When computing the semantics of the program, we have to consider both the fixed point of a particular point where a clause holds and the fixed point of the program as a whole. When computing the fixed point in each point, we have to consider the consequences derived locally and the consequences derived from the interaction between points. Locally, fixed points are computed as in a fixed point semantics for Horn clauses à la van Emden and Kowalski [17]. When considering interaction between points in a Kripke structure, one has to take into account the meaning of intuitionistic implication as in Definition 2.

We treat negation as follows. We use atom A' to denote a negative literal A. This form of renaming, as used to represent *explicit negation* [5], allows the use of the standard fixed point operator $T_{\mathcal{P}}$ [17]. For example, given $A_1, ..., A'_k, ..., A_n \Rightarrow A_0$ with A'_k renaming $\neg A_k$, an interpretation that assings true to A' represents that $\neg A_k$ is true; it does not represent that A_k is false. The atom A' is called the *positive form* of the negative literal $\neg A_k$. Following Definition 2 of intuitionistic negation, A' shall be true in a point ω_i if and only if A does not hold in every point ω_j such that $\mathcal{R}(\omega_i, \omega_j)$.

Definition 3 (Local Consequence Operator) *Let $\mathcal{P} = \{\mathcal{P}_1, ..., \mathcal{P}_k\}$ be a labelled intuitionistic program, where $\mathcal{P}_i$ is a set of clauses that hold in a point 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 $IT_{\mathcal{P}_i} : 2^{B_{\mathcal{P}}} \rightarrow 2^{B_{\mathcal{P}}}$ in ω_i is defined as follows: $IT_{\mathcal{P}_i}(I) = \{A_0 \in B_{\mathcal{P}} \mid A_1, ..., A_n \Rightarrow A_0$ is a clause in $\mathcal{P}_i$ and $\{A_1, ..., A_n\} \subseteq I$ or, if A_0 is A'_0, for all ω_j such that $\mathcal{R}(\omega_i, \omega_j)$, $A_0 \notin IT_{\mathcal{P}_j}(J)\}$, where $IT_{\mathcal{P}_j}(J)$ is defined as $IT_{\mathcal{P}_i}(I)$ and J is a Herbrand interpretation for $\mathcal{P}_j$.*

Definition 4 (Global Consequence Operator) *Let $\mathcal{P} = \{\mathcal{P}_1, ..., \mathcal{P}_k\}$ be a labelled intuitionistic 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 $IT_\mathcal{P} : 2^{B_\mathcal{P}} \rightarrow 2^{B_\mathcal{P}}$ is defined as follows: $IT_\mathcal{P}(I_1, ..., I_k) = \bigcup_{l=1}^{k}\{IT_{\mathcal{P}_l}\}$.*

The following theorem by Ramanujam [19], regarding the fixed point semantics of distributed definite logic programs will be useful.

Definition 5 *(*Distributed Programs*) Definite distributed logic programs are tuples $< \mathcal{P}_1, ..., \mathcal{P}_n >$ where each $\mathcal{P}_i$ is the set of clauses (program) associated with each point i. Each $\mathcal{P}_i$ is called a* component program *of the* composite program *[19].*[1]

Theorem 1 *(*Fixed Point Model of Distributed Programs *[19]) For each distributed definite logic program $\mathcal{P}$, the function $T_\mathcal{P}$ has a unique fixed point. The sequence of all $T^m_\mathcal{P}(I_1, ..., I_k), m \in \aleph$, converges to this fixed point $T^\varpi_\mathcal{P}(I_1, ..., I_k)$, for each $I_i \subseteq 2^{B_\mathcal{P}}$.*

Following [3], one can construct the semantics of labelled intuitionistic programs by considering the ground instances of definite logic programs in order to compute the fixed point. In this way, one associates with every labelled intuitionistic program a ground program (the closure of the program) so that both programs have the same models. As a result, Theorem 2, below, follows directly from Theorem 1.

Theorem 2 *(*Fixed Point Model of Labelled Intuitionistic Programs*) For each labelled intuitionistic program P, the function $IT_\mathcal{P}$ has a unique fixed point. The sequence of all $IT^m_\mathcal{P}(I_1, ..., I_k), m \in \aleph$, converges to this fixed point $IT^\varpi_\mathcal{P}(I_1, ..., I_k)$, for each $I_i \subseteq 2^{B_\mathcal{P}}$.*

2.3 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*. 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 $U_i(t) = g_i(I_i(t), W_i)$, 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 $O_i(t + \Delta t) = f_i(A_i(t + \Delta t))$.

Usually, 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 feedforward 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*.

[1] Clearly, there is a correpondence between distributed programs and labelled programs in the sense that each P_i corresponds to a set of clauses labelled w_i.

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 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).

In this paper, we concentrate on single hidden layer networks, since they are universal approximators [7]. We also use *bipolar* semi-linear activation functions $h(x) = \frac{2}{1+e^{-\beta x}} - 1$ with inputs in $\{-1, 1\}$.

3 Connectionist Intuitionistic Logic

In this section, we introduce a new connectionist model for intuitionistic reasoning. We do so by mapping intuitionistic semantics into an ensemble of C-ILP networks[2]. Let us start with an example.

Example 1 *(*Connectionist Intuitionistic Implication and Negation*)*

(a) Connectionist Intuitionistic Implication*: Let $\mathcal{P} = \{\omega_1 : A \Rightarrow B,\ \mathcal{R}(\omega_1, \omega_2)\}$ be a labelled intuitionistic program. Figure 1 shows an ensemble (w_1,w_2) that implements $\mathcal{P}$. According to the semantics of the above intuitionistic implication, $\omega_1 : A \Rightarrow B$ and $\mathcal{R}(\omega_1, \omega_2)$ imply $\omega_2 : A \Rightarrow B$. This can be implemented by copying the neural representation of $A \Rightarrow B$ in ω_1 to ω_2. In Figure 1, $A \Rightarrow B$ is implemented through hidden neuron h such that output neuron B is active if input neuron A is active. $\mathcal{P}$ is implemented by copying the implementation of $A \Rightarrow B$ via h to ω_2. We will see exactly how this is done in Section 3.2.*

(b) Connectionist Intuitionistic Negation*: In addition to the intuitionistic implication, we need to implement the intuitionistic negation of Definition 2. Suppose $\mathcal{P} = \{\omega_1 : \neg A \Rightarrow B,\ \mathcal{R}(\omega_1, \omega_2),\ \mathcal{R}(\omega_1, \omega_3)\}$. We rename $\neg A$ as A' and implement the implication as before. However, we must also make sure that A' will be derived in ω_1 if A is not derived in ω_2 and ω_3. This can be implemented in the ensemble by connecting the occurencies of A in ω_2 and ω_3 to A' in ω_1 (see hidden neuron n in Figure 2) such that if A is not activated in ω_2 and A is not activated in ω_3 then A' is activated in ω_1.[3] Differently from the case of the implication, the implementation of negation requires the use of negative weights (to denote the "not activation" of a neuron). We shall use dashed arrows to represent negative weights.*

In what follows, we describe in detail how each C-ILP network is built to represent definite logic programs (Section 3.1), and how the ensemble of C-ILP networks can be connected to represent labelled intuitionistic programs (Section 3.2).

3.1 The C-ILP System

C-ILP [10, 13] is a massively parallel computational model based on an artificial neural network that integrates inductive learning from examples and background knowledge with deductive learning from logic programming. Following [15] (see also [16]), a *Translation Algorithm* maps a 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 $\mathcal{P}$ [17]. In addition, $\mathcal{N}$ can be trained with examples using *Backpropagation*

[2]We will recall how each C-ILP network is built in Section 3.1.

[3]The activation of A' in w_1 would then trigger the activation of B in w_1 (since $\neg A \Rightarrow B$) using C-ILP's feedback (i.e. the recurrent connection from output neuron A' to input neuron A' inside w_1).

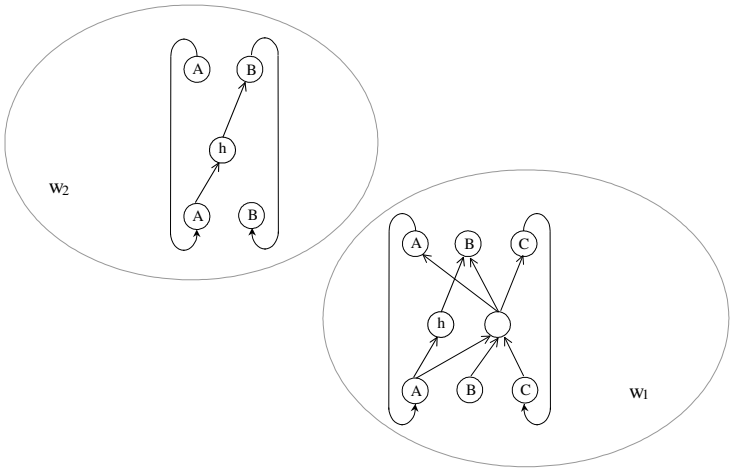

Figure 1: Representing intuitionistic implication

[20], having $\mathcal{P}$ as background knowledge. The knowledge acquired by training can then be extracted [9], closing the learning cycle (as in [21]).

Let us exemplify how C-ILP*'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 *(*C-ILP*) Consider the logic program* $\mathcal{P} = \{BC \sim D \rightarrow A; EF \rightarrow A; \rightarrow B\}$ *where* $\sim$ *stands for* default negation *(e.g., Prolog's negation by failure) [17]. The* Translation Algorithm *derives the network* $\mathcal{N}$ *of Figure 3, 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 3) 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 the fixed point operator of $\mathcal{P}$ [13]. $\mathcal{N}$ will eventually converge to a stable state which is identical to the stable model of $\mathcal{P}$ provided that $\mathcal{P}$ is an acceptable program (see [2] for the definition of acceptable programs).

Since in this paper we do not allow the use of default negation ($\sim$), and we rename any negative literal ($\neg A$) appearing in the program by what is called its positive form (A'), we only need to worry about the part of C-ILP's *Translation Algorithm* that deals with definite logic programs. Note that, in this case, the network will contain only positive weights (W). The algorithm works as follows.

Notation: Given a definite logic program $\mathcal{P}$, let q denote the number of rules r_l ($1 \leq l \leq q$) occurring in $\mathcal{P}$; η, the number of atoms 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 *non active* (or $false$), $A_{max} \in (-1, 0)$; $h(x) = \frac{2}{1+e^{-\beta x}} - 1$, the bipolar semi-linear activation function with inputs in $\{-1, 1\}$[4]; $g(x) = x$, the standard linear

[4]We use the bipolar semi-linear activation function for convenience. Any monotonically increasing activation

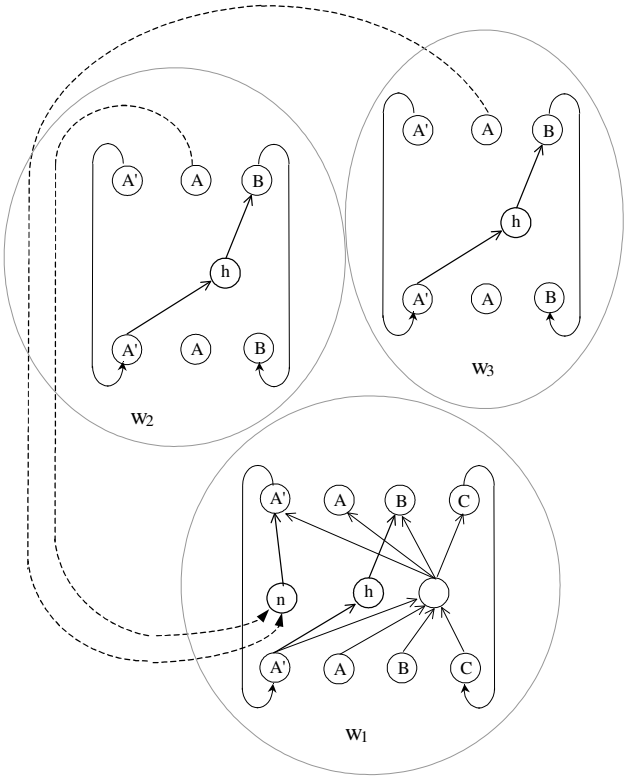

Figure 2: Representing intuitionistic negation

activation function; W, the weight of a network connection; θ_l, the threshold of hidden neuron N_l associated with rule r_l; θ_{A_0}, the threshold of output neuron A_0, where A_0 is the head of rule r_l; k_l, the number of atoms 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 $(\mu_1, ..., \mu_q)$.

In the *Translation Algorithm* (positive programs) below, we define A_{min}, W, θ_l, and θ_{A_0} such that conditions (**C1**) and (**C2**) above are satisfied. Given a definite logic program $\mathcal{P}$, consider that the atoms of $\mathcal{P}$ are numbered from 1 to η such that the input and output layers of $\mathcal{N}$ are vectors of length η, where the i-th neuron represents the i-th atom 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: $A_{min} > ((MAX_{\mathcal{P}}(\overrightarrow{k}, \overrightarrow{\mu}) - 1)/(MAX_{\mathcal{P}}(\overrightarrow{k}, \overrightarrow{\mu}) + 1))$.

- **Translation Algorithm (positive programs):**

1. Calculate the value of W such that the following is satisfied: $W \geq (2/\beta) \cdot (ln\,(1 + A_{\min}) - ln\,(1 - A_{\min}))/(MAX_{\mathcal{P}}(\overrightarrow{k}, \overrightarrow{\mu}) \cdot (A_{\min} - 1) + A_{\min} + 1))$
2. For each rule r_l of $\mathcal{P}$ of the form $A_1, ..., A_k \rightarrow A_0$ $(k \geq 0)$:
   (a) Add a neuron N_l to the hidden layer of $\mathcal{N}$;
   (b) Connect each neuron A_i $(1 \leq i \leq k)$ in the input layer to the neuron N_l in the hidden layer and set the connection weight to W;

function could have been used here.

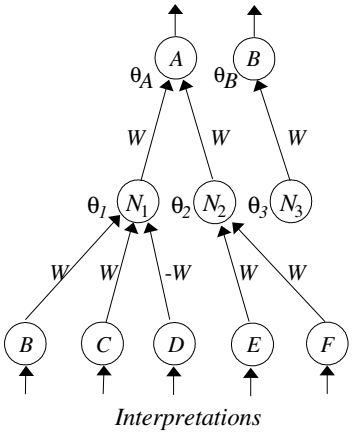

Figure 3: Sketch of a neural network for program $\mathcal{P}$ of Example 2.

(c) Connect the neuron N_l in the hidden layer to the neuron A_0 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: $\theta_l = ((1 + A_{\min}) \cdot (k_l - 1)/2)W$

(e) Define the threshold (θ_{A_0}) of the neuron A_0 in the output layer as: $\theta_{A_0} = ((1 + A_{\min}) \cdot (1 - \mu_l)/2)W$

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-based learning algorithm, such as *Backpropagation*, can be applied on $\mathcal{N}$.

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

Theorem 3 *(*Correctness of Translation Algorithm *[10, 13]) For each definite logic program $\mathcal{P}$, there exists a feedforward 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 $\mathcal{P}$.*

Corollary 4 *(*Connectionist Fixed Point Computation *[10, 13]) Let $\mathcal{P}$ be a definite program. There exists a recurrent neural network $\mathcal{N}_r$ 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 $(T^{\varpi}_{\mathcal{P}}(\mathbf{i}))$ of $T_\mathcal{P}$.*

Hence, in order to use the network $\mathcal{N}$ of Figure 3 as a massively parallel model for Logic Programming, we just have to follow two steps: (i) add neurons to the input and output layers of $\mathcal{N}$, allowing it to be recurrently connected; and (ii) add the correspondent recurrent connections with fixed weight $W_r = 1$, so that the activation of output neuron A feeds back into the activation of input neuron A, the activation of output neuron B feeds back into the activation of input neuron B, and so on (as in, e.g., Figure 2). For instance, given any initial activation in the input layer of $\mathcal{N}_r$ (network of Figure 3 recurrently connected), it always converges to the following stable state: $A = \mathit{false}$, $B = \mathit{true}$, $C = \mathit{false}$, $D = \mathit{false}$, $E = \mathit{false}$, and $F = \mathit{false}$, that represents the unique fixed point of $\mathcal{P}$.

3.2 The Intuitionistic C-ILP System

In this section, we extend the C-ILP system to deal with intuitionistic implication and negation. Given a distributed (or labelled) definite program $\mathcal{P} = \{\mathcal{P}_1, ..., \mathcal{P}_n\}$, we apply the Translation Algorithm of Section 3.1 n times to produce its neural counterpart as an ensemble of C-ILP networks $N_1, ..., N_n$. Now, in the case of labelled intuitionistic programs, if $\mathcal{R}(\omega_i, \omega_j)$ and $\omega_i : A_1, ..., A_k \Rightarrow A_0$, we need to add a clause of the form $A_1, ..., A_k \Rightarrow A_0$ to $\mathcal{P}_j$ before we apply the Translation Algorithm to it. We say that $\{\omega_i : A_1, ..., A_k \Rightarrow A_0, \mathcal{R}(\omega_i, \omega_j)\}$ can be written as $\{\omega_i : A_1, ..., A_k \Rightarrow A_0, \omega_j : A_1, ..., A_k \Rightarrow A_0\}$. As for the intuitionistic negation, once the network ensemble is derived, each network $\mathcal{N}_i$ containing neurons A' (corresponding to program $\mathcal{P}_i$) needs to be connected to each network $\mathcal{N}_j$ in the ensemble whenever $\mathcal{R}(\omega_i, \omega_j) \in \mathcal{P}$. More precisely, output neuron A in $\mathcal{N}_j$ needs to be connected to output neuron A' in $\mathcal{N}_i$ through a new hidden neuron created in $\mathcal{N}_i$ (see Figure 2) such that A' is active in $\mathcal{N}_i$ if A is not active in $\mathcal{N}_j$. Note that neuron A' could also become active in $\mathcal{N}_i$ if atom A' belongs to the head of a clause in $\mathcal{P}_i$ and its body is satisfied by an interpretation (network input). Any hidden neuron created to encode negation shall use activation function $s(x) = y$, where $y = 1$ if $x > 0$, and $y = 0$ otherwise; $s(x)$ is known as the standard nonlinear activation function (also called the step function). This is so because, these particular hidden neurons encode (meta-level) knowledge about negation, while the other hidden neurons encode (object-level) knowledge about the problem. The former are not expected to be trained from examples and, as a result, the use of the step function will simplify the intuitionistic algorithm. The latter are trained using Backpropagation, and therefore require a derivable, semi-linear activation function.

Let $\mathcal{P} = \{\mathcal{P}_1, ..., \mathcal{P}_n\}$ be a labellel intuitionistic program. As in the case of individual C-ILP networks, we start by calculating $MAX_{\mathcal{P}}(\overrightarrow{k}, \overrightarrow{\mu}, n)$ of $\mathcal{P}$ and A_{min} such that: $A_{min} > (MAX_{\mathcal{P}}(\overrightarrow{k}, \overrightarrow{\mu}, n) - 1)/(MAX_{\mathcal{P}}(\overrightarrow{k}, \overrightarrow{\mu}, n) + 1)$, which now also considers the number n of networks (points) in the ensemble.

- **Intuitionistic Algorithm**

1. For each clause c_l of the form $A_1, ..., A_k \Rightarrow A_0$ in $\mathcal{P}_i$ $(1 \leq i \leq n)$ such that $\mathcal{R}(\omega_i, \omega_j) \in \mathcal{P}$, do:

   (a) add a clause $A_1, ..., A_k \Rightarrow A_0$ to $\mathcal{P}_j$ $(1 \leq j \leq n)$.

2. For each program $\mathcal{P}_i$ $(1 \leq i \leq n)$ in $\mathcal{P}$, do:

   (a) Call the **Translation Algorithm**

3. For each atom of the form A' in a clause c_l of $\mathcal{P}_i$, do:

   (a) Add a hidden neuron $N_{A'}$ to $\mathcal{N}_i$;

   (b) Set the step function $s(x)$ as the activation function of $N_{A'}$;

   (c) Set the threshold $\theta_{A'}$ of $N_{A'}$ such that $n - (1 + A_{min}) < \theta_{A'} < nA_{min}$;

   (d) For each network $\mathcal{N}_j$ corresponding to program $\mathcal{P}_j$ $(1 \leq j \leq n)$ in $\mathcal{P}$ such that $\mathcal{R}(\omega_i, \omega_j) \in \mathcal{P}$, do:

   (i) Connect the output neuron A of $\mathcal{N}_j$ to the hidden neuron $N_{A'}$ of $\mathcal{N}_i$ and set the connection weight to -1;

   (ii) Connect the hidden neuron $N_{A'}$ of $\mathcal{N}_i$ to the output neuron A' of $\mathcal{N}_i$ and set the connection weight to W^I such that $W^I > h^{-1}(A_{min}) + \mu_{A'}.W + \theta_{A'}$, where $\mu_{A'}$, W and $\theta_{A'}$ are obtained from C-ILP *'s Translation Algorithm*[5].

[5]Recall that $\mu_{A'}$ is the number of connections to output neuron A'.

Theorem 5 *(*Correctness of Intuitionistic Algorithm*) For each labelled intuitionistic program P, there exists an ensemble of neural networks $\mathcal{N}$ such that $\mathcal{N}$ computes the fixed point operator $IT_{\mathcal{P}}$ of $\mathcal{P}$.*

Proof. *We need to show that A' is active in $\mathcal{N}_i$ if and only if (i) there exists a clause of $\mathcal{P}_i$ of the form $A_1, ..., A_k \Rightarrow A'$ s.t. $A_1, ..., A_k$ are satisfied by an interpretation (input vector of $\mathcal{N}_i$) $\mathbf{i}$, or (ii) for all $\mathcal{P}_j \in \mathcal{P}$ such that $\mathcal{R}(\omega_i, \omega_j)$, there exists a clause of $\mathcal{P}_j$ of the form $A_1, ..., A_k \Rightarrow A$ such that A is not satisfied by an interpretation (input vector of $\mathcal{N}_j$) $\mathbf{j}$. Case (i) follows directly from Theorem 3. Case (ii) (if A is not active in any network $\mathcal{N}_j$ $(0 \leq j \leq n)$ to which $\mathcal{N}_i$ is related to, A' is active in $\mathcal{N}_i$): From the* Intuitionistic Algorithm*, $N_{A'}$ is a non-linear hidden neuron in $\mathcal{N}_i$. If A is not active ($A < -A_{min}$) in $\mathcal{N}_j$, the minimum input potential of $N_{A'}$ is $nA_{min} - \theta_{A'}$. Since $\theta_{A'} < nA_{min}$ (*Intuitionistic Algorithm*, step 3c), the minimum input potential of $N_{A'}$ is greater than zero and, therefore, $N_{A'}$ presents activation* 1*. As a result, the minimum activation of A' in $\mathcal{N}_i$ is $h(W^I - \mu_{A'}.W - \theta_{A'})$. Since $W^I > h^{-1}(A_{min}) + \mu_{A'}.W + \theta_{A'}$, we have $h(W^I - \mu_{A'}.W - \theta_{A'}) > A_{min}$ and, therefore, A' is active ($A' > A_{min}$). (if A is active in some network $\mathcal{N}_j$ $(0 \leq j \leq n)$ to which $\mathcal{N}_i$ is related to and for all clauses of the form $A_1, ..., A_k \Rightarrow A'$ in $\mathcal{P}_i$, $A_1, ..., A_k$ are not satisfied by $\mathbf{i}$ (input vector of $\mathcal{N}_i$) then A' is not active in $\mathcal{N}_i$): In the worst case, A is not active in $n-1$ networks with activation -1, and active in a single network with activation A_{min}. In this case, the input potential of $N_{A'}$ is $n - 1 - A_{min} - \theta_{A'}$ (recall that the weights to $N_{A'}$ are all set to -1). Since $\theta_{A'} > n - (1 + A_{min})$ (*Intuitionistic Algorithm*, step 3c), the maximum input potential of $N_{A'}$ is zero and, since $s(x)$ is the activation function of $N_{A'}$, $N_{A'}$ presents activation* 0*. From Theorem 3, if $A_1, ..., A_k$ are not satisfied by $\mathbf{i}$ then A' is not active. Finally, since the activation of $N_{A'}$ is zero, A' cannot be activated by $N_{A'}$, so A' is not active.* ■

Corollary 6 *(*Connectionist Intuitionistic Fixed Point Computation*) Let $\mathcal{P}$ be a labelled intuitionistic program. There exists an ensemble of recurrent neural networks $\mathcal{N}^r$ such that, starting from an arbitrary initial input, $\mathcal{N}^r$ converges to a stable state and yields the unique fixed-point ($IT^{\omega}_{\mathcal{P}}(\mathbf{i})$) of $IT_{\mathcal{P}}$.*

Proof. *By Theorem 5, $\mathcal{N}$ computes $IT_{\mathcal{P}}$. Recurrently connected, $\mathcal{N}^r$ computes the upward powers ($IT^m_{\mathcal{P}}(I)$) of $IT_{\mathcal{P}}$. Finally, by Theorem 2, $\mathcal{N}^r$ converges to the unique fixed point ($IT^{\omega}_{\mathcal{P}}(I)$) of $IT_{\mathcal{P}}$.* ■

Example 3 *(*Connectionist Intuitionistic Fixed Point Computation*) Consider again the ensemble of Figure 2. For any initial set of input vectors (interpretations $\mathbf{i}$, $\mathbf{j}$,...) to networks $\mathcal{N}_1$, $\mathcal{N}_2$, $\mathcal{N}_3$, (corresponding to points $\omega_1, \omega_2, \omega_3$), output neuron A will not be activated neither in $\mathcal{N}_2$ nor in $\mathcal{N}_3$. As a result, output neuron A' will eventually be activated (and remain activated) in $\mathcal{N}_1$. After that, a single step through $\mathcal{N}_1$'s recursive connection will activate output neuron B. As a result, when $\mathcal{N}_1$ converges to a stable state (i.e. same input and output vectors), A' and B will belong to such a state and, therefore, to the fixed point of $\mathcal{P}_1$.*

4 Concluding Remarks

In this paper, we have presented a new massively parallel model for intuitionistic logics. We have defined a class of labelled intuitionistic programs, and then presented an algorithm to translate the intuitionistic theory into an ensemble of C-ILP neural networks, and showed 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 the computation of intuitionistic logic. In addition, since each C-ILP network can be trained efficiently using the Backpropagation learning algorithm [20], one can adapt the C-ILP ensemble by training possible world representations from examples in each network. Extensions of the work presented in this paper include the study of how to represent properties of other non-classical logics (such as temporal logics [18], relevance and linear logics [1]), and of inference and learning of fragments of first order logic. In addition, as the Curry-Howard isomorphism (see e.g. [1]) establishes a relationship

between intuitionism and typed $\lambda - calculus$ (i.e. typed functional programs), it would be interesting to exploit this relationship w.r.t the connectionist model presented here, so that one could present such concepts under a connectionist computational foundation.

References

[1] S. Abramsky. Computational interpretations of linear logic. *Theoretical Computer Science*, 11(1–2):3–57, 1993.

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

[3] 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.

[4] P. Blackburn, M. de Rijke, and Y. Venema. *Modal Logic*. Cambridge University Press, 2001.

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

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

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

[8] R. Davies and F. Pfenning. A modal analysis of staged computation. *Journal of the ACM*, 48(3):555–604, 2001.

[9] 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.

[10] 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.

[11] A. S. d'Avila Garcez, L. C. Lamb, K. Broda, and D. M. Gabbay. Distributed knowledge representation in neural-symbolic learning systems: A case study. In *Proceedings of AAAI International FLAIRS Conference*, Florida, USA, 2003.

[12] A. S. d'Avila Garcez, L. C. Lamb, and D. M. Gabbay. A connectionist inductive learning system for modal logic programming. In *Proceedings of IEEE International Conference on Neural Information Processing ICONIP'02*, Singapore, 2002.

[13] 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.

[14] D. M. Gabbay. *Labelled Deductive Systems*, volume 1. Clarendom Press, Oxford, 1996.

[15] 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.

[16] 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.

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

[18] M. Maidl. The common fragment of CTL and LTL. In *Proceedings of 41st IEEE Symposium on Foundations of Computer Science*, pages 643–652, 2000.

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

[20] 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.

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