METALEVEL PRIORITIES AND NEURAL NETWORKS

A. S. D'AVILA GARCEZ[‡], K. BRODA[‡], D. M. GABBAY[§]

[‡]Department of Computing
Imperial College, London, SW7 2BZ
{aag,kb}@doc.ic.ac.uk

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

ABSTRACT. This paper shows to which extent metalevel priorities, in the sense of Nute's superiority relation, can be encoded into single hidden layer neural networks.

1. INTRODUCTION

Recently there has been increasing interest in logic programming-based default reasoning approaches which are not using negation as failure in their object language. Instead, default reasoning is modelled by rules with explicit negation [7] and a metalevel priority relation between rules [2, 10].

In this paper, we extend the Connectionist Inductive Learning and Logic Programming (*C-IL²P*) System [5] to deal with metalevel priorities in the sense of Nute's superiority relation [11]. Throughout, we use $r_j \succ r_i$ to indicate that rule r_j has higher priority than rule r_i.

We show that if $\succ$ is a linear order, single hidden layer networks can encode such a superiority relation. When $\succ$ is a partial order, the representation in the network is not straightforward, although some special cases can be dealt with.

In section 2, we recall *C-IL²P's* algorithm for inserting background knowledge into neural networks. In section 3, we show how linearly ordered theories can be encoded into a neural network. We also investigate to which extent partially ordered theories can be encoded. In section 4, we conclude and discuss directions for future work.

2. THE *C-IL²P* SYSTEM

C-IL²P's 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 [5]. Following [8] (see also [9]), a *Translation Algorithm* maps an extended logic program[1] $\mathcal{P}$ into a single hidden layer neural network $\mathcal{N}$ such that $\mathcal{N}$ computes the least fixpoint of $\mathcal{P}$. This provides a massively parallel model for computing the answer set semantics of $\mathcal{P}$ [7]. In addition, $\mathcal{N}$ can be trained with examples, using for instance backpropagation [13], having $\mathcal{P}$ as background knowledge. The knowledge acquired by training can then be extracted [4], closing the learning cycle (as in [14]). In what follows, we recall *C-IL²P's Translation Algorithm* by presenting an example of the insertion of knowledge into the network.

[1]An extended program is a general logic program extended with explicit negation. Throughout, we use $\sim$ for default negation and $\neg$ for explicit (classical) negation.

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: *(1)* 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 *(2)* 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 2.1. *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 1, setting weights (W's) and thresholds* (θ's) *in such a way that conditions (1) and (2) above are satisfied. Note that, if* $\mathcal{N}$ *ought to be fully-connected, any other link (not shown in Figure 1) should receive weight zero initially.*

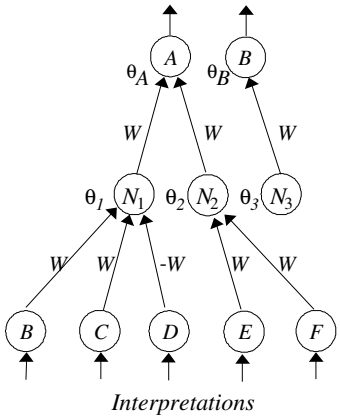

FIGURE 1. Sketch of a neural network for the above logic program $\mathcal{P}$.

Note that, in Example 2.1, 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 $\mathcal{P}$.[2] Note also that each hidden neuron N_l corresponds to a rule r_l of $\mathcal{P}$. In order to compute the answer set of $\mathcal{P}$, output neuron B should feed input neuron B such that $\mathcal{N}$ is used to iterate $T_{\mathcal{P}}$, the fixpoint operator of $\mathcal{P}$. $\mathcal{N}$ will eventually converge to a stable state which is identical to the answer set of $\mathcal{P}$ (see [5]). Let us now recall *C-IL*^{2}*P*'s *Translation Algorithm* from $\mathcal{P}$ to $\mathcal{N}$.

Notation: : Given a general logic program $\mathcal{P}$, let q denote the number of rules r_l ($1 \leq l \leq q$) occurring in $\mathcal{P}$; η, 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[3]; $g(x) = x$, the standard linear activation function; W (resp. -W), the weight of connections associated with positive (resp. negative) literals; θ_l, the threshold of hidden neuron N_l associated with 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,

[2] An *interpretation* is a function from propositional variables to $\{true, false\}$. A *model* for $\mathcal{P}$ is an interpretation that maps $\mathcal{P}$ to *true*.

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

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 use $\overrightarrow{k}$ as a short for $(k_1, ..., k_q)$, and $\overrightarrow{\mu}$ as a short for $(\mu_1, ..., \mu_q)$.

For instance, for the program $\mathcal{P}$ of Example 2.1, $q = 3$, $\eta = 6$, $k_1 = 3$, $k_2 = 2$, $k_3 = 0$, $p_1 = 2$, $p_2 = 2$, $p_3 = 0$, $n_1 = 1$, $n_2 = 0$, $n_3 = 0$, $\mu_1 = 2$, $\mu_2 = 2$, $\mu_3 = 1$, $MAX_{r_1}(k_1, \mu_1) = 3$, $MAX_{r_2}(k_2, \mu_2) = 2$, $MAX_{r_3}(k_3, \mu_3) = 1$, 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 *(1)* and *(2)* above are satisfied (the proof is provided in [5]).

- **Translation Algorithm:**

Given a general logic program $\mathcal{P}$, consider that the literals of $\mathcal{P}$ are numbered from 1 to η such that the input and output layers of $\mathcal{N}$ are vectors of maximum length η, where the i-th neuron represents the i-th literal of $\mathcal{P}$. We assume, for mathematical convenience and without loss of generality, that $A_{max} = -A_{min}$.

1. Calculate $MAX_{\mathcal{P}}(\overrightarrow{k}, \overrightarrow{\mu})$ of $\mathcal{P}$;
2. Calculate the values of A_{min} and W such that the following is satisfied:

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

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

3. 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:

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

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

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

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

3. Adding Metalevel Priorities

In [5], we have seen that a single hidden layer network can represent either a general or an extended logic program. In both cases, the network does not contain negative connections from

the hidden layer to the output. What would then be the meaning of negative weights from the hidden to the output layer of the network? In this paper, we are interested in answering this question. We start with an example.

Example 3.1. *Let* $\mathcal{P} = \{r_1 : fingertips,\ r_2 : alibi,\ r_3 : fingertips \rightarrow guilty,\ r_4 : alibi \rightarrow \neg guilty\}$ *and* $r_3 \succ r_4$ *(stating that* $fingertips \rightarrow guilty$ *is a stronger evidence than* $alibi \rightarrow \neg guilty$*). A neural network that encodes* $\mathcal{P}$ *but not* $r_3 \succ r_4$ *will compute the inconsistent answer set* $\{fingertips, alibi, guilty, \neg guilty\}$*. Alternatively,* $r_3 \succ r_4$ *could be incorporated in the object-level by changing the rule "*$alibi \rightarrow \neg guilty$*" to "*$alibi, \sim fingertips \rightarrow \neg guilty$*". The new program would compute the answer set* $\{fingertips, alibi, guilty\}$*, which contains the intended answers for* $\mathcal{P}$ *complemented by* $r_3 \succ r_4$*.*

Notwithstanding, how could we represent the above priority explicitly in the neural network? In the same way that negative weights from input to hidden neurons are interpreted as negation by default because they contribute to block the activation of the hidden neurons, negative weights from hidden to output neurons could be seen as the implementation of metalevel priorities. Figure 2 illustrates the idea. A negative weight from hidden neuron r_3 *to output neuron* $\neg guilty$ *could implement* $r_3 \succ r_4$*, provided that whenever* r_3 *is activated, it* blocks *(or* inhibits*) the activation of* $\neg guilty$*, which is the conclusion of* r_4*.*

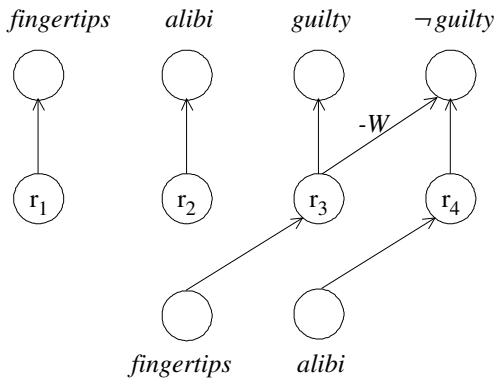

FIGURE 2. Adding Metalevel Priorities.

In the above example, $r_3 \succ r_4$ means that, whenever the conclusion of r_3 holds, the conclusion of r_4 does not hold. Hence, $r_i \succ r_j$ defines priorities among rules that can override the conclusions one of another. It is, thus, similar to the *superiority relation* defined by Nute in [11] and later investigated in [1].

In the sequel, we recall some of the basic definitions of Nute's superiority relation. A superiority (binary) relation $\succ$ is a strict partial order, i.e., an irreflexive and transitive relation on a set. Rule r_1 is said to be superior to rule r_2 if $r_1 \succ r_2$. When the antecedents of two rules are derivable, the superiority relation is used to adjudicate the conflict. If either rule is superior to the other, then the superior rule is applied. In other words, $\succ$ provides information about the relative strength of the rules. Following [1], we will define superiority relations over rules with contradictory conclusions only. More precisely, for all r_i, r_j, if $r_i \succ r_j$ then r_i and r_j have complementary literals (x and $\neg x$) as consequents. A cycle in the superiority relation (e.g., $r_1 \succ r_2$ and $r_2 \succ r_1$) is counter-intuitive from the knowledge representation perspective, and thus $\succ$ is also required to be an acyclic relation, i.e., we assume that the transitive closure of $\succ$ is irreflexive.

It has been proved in [1] that the above superiority relation does not add to the expressive power of Defeasible Logic [11]. In fact, (object level) default negation and (metalevel) superiority relations are interchangeable (see [10]). The superiority relation adds, however, to the epistemic power of Defeasible Logic because it obviously allows one to represent information in a more natural way. For example, the Nixon diamond problem is more easily expressed as follows: $r_1 : quaker(Nixon) \rightsquigarrow pacifist(Nixon)$, $r_2 : republican(Nixon) \rightsquigarrow \neg pacifist(Nixon)$, where $\rightsquigarrow$ should be read as *normally implies.* The definition of an adequate priority relation between r_1 and r_2 would solve the inconsistency regarding the pacifism of Nixon, when both $quaker(Nixon)$ and $republican(Nixon)$ are *true.* Also for epistemological reasons, in many cases it is useful to have both default negation and metalevel priorities. This facilitates the expression of (object-level) priorities in the sense of default reasoning and (metalevel) priorities in the sense that a given conclusion should be overridden by another with higher priority.[4]

Hence, the superiority relation we discuss here essentially makes explicit the priorities encoded in the object level. As a result, a network $\mathcal{N}_{\succ}$ encoding a program $\mathcal{P}_{\succ}$ with explicit priorities is expected to behave exactly as the network $\mathcal{N}$ that encodes the equivalent extended program $\mathcal{P}$ without priorities. The following definition clarifies what we mean by the equivalence between $\mathcal{P}_{\succ}$ and $\mathcal{P}$ in terms of $T_{\mathcal{P}}$.

Definition 3.1. *Let $\mathcal{P}_{\succ} = \{r_1, r_2, ..., r_q\}$ be an extended program with an explicit superiority relation $\succ$. Let $\mathcal{P}$ be the same extended program $\mathcal{P}_{\succ}$ without the superiority relation $\succ$. For any two rules r_i, r_j in $\mathcal{P}_{\succ}$ such that $r_j \succ r_i$, let $\mathcal{P}' = \mathcal{P} - r_i$, i.e., $\mathcal{P}'$ is the program $\mathcal{P}$ without rule r_i. We define $T_{\mathcal{P}_{\succ}} = T_{\mathcal{P}'}$ if r_j fires, and $T_{\mathcal{P}_{\succ}} = T_{\mathcal{P}}$ otherwise.*

3.1. **Linearly Ordered Theories.** Firstly, let us consider the case of linearly ordered sets. This includes the case of exceptions of exceptions, i.e., when a conclusion derived based on a given preference relation is overridden by another preference relation. Take, for instance, the following example: Let $\mathcal{P} = \{r_1 : birds \rightarrow fly$, $r_2 : penguins \rightarrow bird$, $r_3 : penguins \rightarrow \neg fly$, $r_4 : superpenguins \rightarrow fly\}$ and $r_4 \succ r_3 \succ r_1$. In this example, r_3 should block r_1, and r_4 should block r_3. Intuitively, we should have $W_{fly,r_1}, W_{fly,r_4} > 0$ and $W_{fly,r_3} < 0$, where W_{fly,r_i} denotes the weight from hidden neuron r_i to output neuron fly. In fact, $W_{fly,r_1} = 1.94, W_{fly,r_3} = -1.93, W_{fly,r_4} = 3.97$ and $\theta_{fly} = -1.93$ implements $r_4 \succ r_3 \succ r_1$. These values were obtained by actually training a network and using fixed weights from the input to the hidden layer (given by *C-IL^{2}P*'s *Translation Algorithm*).

Following [6], we will consider the case of *finite* linearly ordered sets. Assume $\mathcal{P} = \{r_1, ..., r_q\}$ and $r_q \succ ... \succ r_2 \succ r_1$. As a result, any subset of $\mathcal{P}$ is also linearly ordered. We are interested in the subset of $\mathcal{P}$ with complementary conclusions x and $\neg x$. We represent it as a list $(r_1, r_2, ..., r_j)$ where $r_j \succ ... \succ r_2 \succ r_1$. Assume r_1 has consequent x. We need to assign values to weights $W_{xr_1}, W_{xr_2}, W_{xr_3}, ..., W_{xr_j}, \theta_x$ and $W_{\neg xr_2}, W_{\neg xr_3}, ..., W_{\neg xr_j}, \theta_{\neg x}$ such that $r_j \succ ... \succ r_2 \succ r_1$ holds. Let:

$$\begin{cases} W_{xr_1} = W, \\ W_{xr_2} = -W + \varepsilon, \\ W_{xr_n} = W_{xr_{n-2}} - W_{xr_{n-1}} + \varepsilon, \text{ if } n \in \{3, 5, 7...\} \text{ or} \\ W_{xr_n} = W_{xr_{n-2}} - W_{xr_{n-1}} - \varepsilon, \text{ if } n \in \{4, 6, 8...\}. \end{cases}$$

[4] In a trained neural network, both representations might be encoded simultaneously, so that the network is more robust.

where $W > 0$ and ε is a small positive number such that $W >> \varepsilon$ (typically $\varepsilon = 0.01$). Similarly, let $W_{\neg x r_2} = W, W_{\neg x r_3} = -W + \varepsilon$, and $W_{\neg x r_n} = W_{\neg x r_{n-2}} - W_{\neg x r_{n-1}} + \varepsilon$ for $n = 5, 7, 9...$, while $W_{\neg x r_n} = W_{\neg x r_{n-2}} - W_{\neg x r_{n-1}} - \varepsilon$ for $n = 4, 6, 8...$[5].

- **Metalevel Priorities Algorithm 1 (Linear Ordering)**[6]

1. Consider a Logic Program $\mathcal{P} = \{r_1, ..., r_q\}$ and its equivalent neural network $\mathcal{N}$, obtained by applying the *Translation Algorithm* of *C-IL*^{2}*P*. For each subset $(r_1, r_2, ..., r_j)$ of rules in $\mathcal{P}$ with consequents x and $\neg x$ such that $r_j \succ ... \succ r_2 \succ r_1$, do:
   (a) Calculate θ_1 such that:

$$(1 - A_{\min})W + \varepsilon A_{\min} - A_{\min}(\sum_{k=3,5...}^{j} W_{x r_k}) - (\sum_{k=4,6...}^{j-1} W_{x r_k}) - h^{-1}(-A_{\min}) < \theta_1$$

$$\theta_1 < A_{\min} W - A_{\min}(\sum_{k=2,4...}^{j-1} W_{x r_k}) - (\sum_{k=3,5...}^{j} W_{x r_k}) - h^{-1}(A_{\min})$$

   (b) Calculate θ_2 such that:

$$(1 - A_{\min})W + \varepsilon A_{\min} - A_{\min}(\sum_{k=4,6...}^{j-1} W_{\neg x r_k}) - (\sum_{k=5,7...}^{j} W_{\neg x r_k}) - h^{-1}(-A_{\min}) < \theta_2$$

$$\theta_2 < A_{\min} W - A_{\min}(\sum_{k=3,5...}^{j} W_{\neg x r_k}) - (\sum_{k=4,6...}^{j-1} W_{\neg x r_k}) - h^{-1}(A_{\min})$$

   (c) Add a connection from each hidden neuron r_i $(1 \leq i \leq j)$ to the output neuron x, and set the connection weight to $W_{x r_i}$;
   (d) Add a connection from each hidden neuron r_i $(2 \leq i \leq j)$ to the output neuron $\neg x$, and set the connection weight to $W_{\neg x r_i}$;
   (e) If j is an *odd* number:
      (i) Set the threshold θ_x of output neuron x to θ_1,
      (ii) Set the threshold $\theta_{\neg x}$ of output neuron $\neg x$ to θ_2.
   (f) If j is an *even* number:
      (i) Set the threshold θ_x of output neuron x to θ_2,
      (ii) Set the threshold $\theta_{\neg x}$ of output neuron $\neg x$ to θ_1.

Theorem 3.1. *Let* $\mathcal{P} = \{r_1, r_2, ..., r_q\}$ *and* $r_q \succ ... \succ r_2 \succ r_1$. *Let* $\mathcal{N}$ *be the network obtained by using the* Translation Algorithm *over* $\mathcal{P}$. *If* $\mathcal{N}$ *is modified by using the* Metalevel Priorities *Algorithm 1 then* $\mathcal{N}$ *computes* $T_{\mathcal{P}_\succ}$, *where* $\mathcal{P}_\succ$ *is the program* $\mathcal{P}$ *together with the preference relation* $r_q \succ ... \succ r_2 \succ r_1$.

Proof. By induction.

---

[5] Note that $W_n = W_{n-2} - W_{n-1} \pm \varepsilon$ is responsible for assigning the weights such that: if j is an even number, $W_{x r_1}, W_{x r_3}, ..., W_{x r_{j-1}} > 0$ and $W_{x r_2}, W_{x r_4}, ..., W_{x r_j} < 0$; if j is an odd number, $W_{x r_1}, W_{x r_3}, ..., W_{x r_j} > 0$ and $W_{x r_2}, W_{x r_4}, ..., W_{x r_{j-1}} < 0$.

[6] The values of θ_x and $\theta_{\neg x}$ are obtained from the proof of Theorem 3.1.

We consider the subset $R = (r_1, r_2, ..., r_j)$ of $\mathcal{P}$ containing the rules with conclusions x and $\neg x$. Let r_i $(1 \leq i \leq j)$ be the last neuron from left to right in $(r_1, r_2, ..., r_j)$ to be activated in $\mathcal{N}$ for a given input vector $\mathbf{i}$. Assume r_1 has consequent x.

We distinguish several cases:

1. If j is an *odd* number and i is an *odd* number, show that x is activated:

   Basis: "If $i = 1$ then x is activated". In the worst case, $r_3, r_5, ..., r_j = -1$ and $r_2, r_4, ..., r_{j-1} = -A_{min}$, while $r_1 = A_{min}$. We need to satisfy:

   $$A_{\min} W - A_{\min} \sum_{k=2,4...}^{j-1} W_{xr_k} - \sum_{k=3,5...}^{j} W_{xr_k} - \theta_x > h^{-1}(A_{\min}) \tag{1}$$

   which yields:

   $$\theta_x < A_{\min} W - A_{\min} \sum_{k=2,4...}^{j-1} W_{xr_k} - \sum_{k=3,5...}^{j} W_{xr_k} - h^{-1}(A_{\min}) \tag{2}$$

   and, from the *Metalevel Priorities* Algorithm 1 *(steps 1(a) and 1(e))*, Equation 2 is clearly satisfied.

   Inductive Step: "if x is activated for $i = n$ then x is activated for $i = n+2$". For $i = n$, in the worst case, $r_1, r_3, ..., r_{n-2} = -1$ and $r_2, r_4, ..., r_{n-1} = 1$, $r_n = A_{min}$ and $r_{n+1}, ..., r_j$ are not activated, i.e., $r_{n+1}, r_{n+3}, ..., r_{j-1} = -A_{min}$ and $r_{n+2}, r_{n+4}, ..., r_j = -1$. If x is activated then

   $$- \sum_{k=1,3...}^{n-2} W_{xr_k} + \sum_{k=2,4...}^{n-1} W_{xr_k} + A_{\min} W_{xr_n} - A_{\min} \sum_{k=n+1,n+3...}^{j-1} W_{xr_k} -$$

   $$\sum_{k=n+2,n+4...}^{j} W_{xr_k} > h^{-1}(A_{\min}) + \theta_x \tag{3}$$

   holds.

   For $i = n + 2$ we have $r_1, r_3, ..., r_n = -1$ and $r_2, r_4, ..., r_{n+1} = 1$, $r_{n+2} = A_{min}$ and $r_{n+3}, ..., r_j$ not activated, i.e., $r_{n+3}, r_{n+5}, ..., r_{j-1} = -A_{min}$ and $r_{n+4}, r_{n+6}, ..., r_j = -1$. We need to show that

   $$- \sum_{k=1,3...}^{n} W_{xr_k} + \sum_{k=2,4...}^{n+1} W_{xr_k} + A_{\min} W_{xr_{n+2}} - A_{\min} \sum_{k=n+3,n+5...}^{j-1} W_{xr_k} -$$

   $$\sum_{k=n+4,n+6...}^{j} W_{xr_k} > h^{-1}(A_{\min}) + \theta_x \tag{4}$$

   also holds.

   It is sufficient to show that:

   $$- \sum_{k=1,3...}^{n} W_{xr_k} + \sum_{k=2,4...}^{n+1} W_{xr_k} + A_{\min} W_{xr_{n+2}} - A_{\min} \sum_{k=n+3,n+5...}^{j-1} W_{xr_k} -$$

   $$\sum_{k=n+4,n+6...}^{j} W_{xr_k} > - \sum_{k=1,3...}^{n-2} W_{xr_k} +$$

   $$\sum_{k=2,4...}^{n-1} W_{xr_k} + A_{\min} W_{xr_n} - A_{\min} \sum_{k=n+1,n+3...}^{j-1} W_{xr_k} - \sum_{k=n+2,n+4...}^{j} W_{xr_k} \tag{5}$$

Simplifying Equation 5, we obtain:

$$(1 + A_{\min})W_{xr_{n+2}} + (1 + A_{\min})W_{xr_{n+1}} > (1 + A_{\min})W_{xr_n} \quad (6)$$

Thus, if $W_{xr_{n+2}} + W_{xr_{n+1}} > W_{xr_n}$ then x is activated for $i = n + 2$. Since $W_{xr_{n+2}} = W_{xr_n} - W_{xr_{n+1}} + \varepsilon$ and $\varepsilon > 0$, we have $W_{xr_{n+2}} + W_{xr_{n+1}} = W_{xr_n} + \varepsilon > W_{xr_n}$. This completes the proof of the inductive step.

The remaining cases are:

2. If j is an *odd* number and i is an *even* number, show that x is not activated,
3. If j is an *even* number and i is an *odd* number, show that x is activated,
4. If j is an *even* number and i is an *even* number, show that x is not activated,
5. If no neuron in R is activated, show that x is not activated.

The proofs for Cases 2,3,4 and 5 are analogous to Case 1. ∎

Example 3.2. *Let $r_5 \succ ... \succ r_2 \succ r_1$. Taking $A_{min} = 0.9$, $W = 20$ and $\varepsilon = 0.1$, we calculate $-61.06 < \theta_1 < -53.13$ and $29.03 < \theta_2 < 46.97$. $W_{xr_1} = W_{\neg xr_2} = 20$, $W_{xr_2} = W_{\neg xr_3} = -19.9$, $W_{xr_3} = W_{\neg xr_4} = 40$, $W_{xr_4} = W_{\neg xr_5} = -60$, and $W_{xr_5} = 100.1$. Taking $\theta_x = -55$ and $\theta_{\neg x} = 45$, the network of Figure 3 will implement $r_5 \succ ... \succ r_2 \succ r_1$. Note that, if a smaller value for A_{min} is desired, a larger value for W may be necessary in order to satisfy the constraints on θ_x and $\theta_{\neg x}$.*

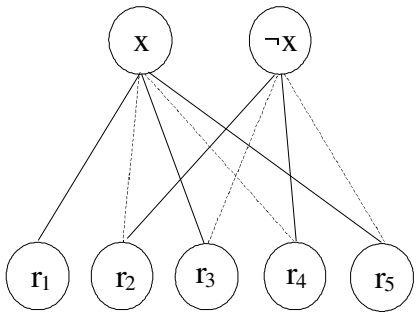

FIGURE 3. Linear ordering on $\{r_1, ..., r_5\}$. Dotted lines indicate negative weights. r_1 activates x, r_2 activates $\neg x$ and blocks x, r_3 activates x and blocks $\neg x$, and so on.

3.2. **Partially Ordered Theories.** Let us now consider the case of partially ordered sets. We start with an example.

Example 3.3. *Let $\mathcal{P} = \{r_1, r_2, r_3\}, r_1 \succ r_3, r_2 \succ r_3$. Let the consequents of r_1 and r_2 be x and the consequent of r_3 be $\neg x$. We want to add negative weights $-\delta_1$ from r_1 to $\neg x$, and $-\delta_2$ from r_2 to $\neg x$ such that (1) the activation of $\neg x$ is greater than A_{min} when r_3 is activated, provided r_1 and r_2 are not activated, and (2) the activation of $\neg x$ is smaller than $-A_{min}$ when either r_1 or r_2 are activated, regardless of the activation of r_3. Assume that the weight from r_3 to $\neg x$ is W, the threshold of $\neg x$, $\theta_{\neg x} = \delta' W$, where $\delta' \in \Re$, and take, for the sake of simplicity, $\delta_1 = \delta_2 = W$.*

In Case (1), the minimum activation of r_3 is A_{min} while, in the worst case, the activations of r_1 and r_2 are both $-A_{min}$. Thus, we have:

$$A_{min}W + A_{min}W + A_{min}W - \delta' W > h^{-1}(A_{min}) \quad (7)$$

In Case (2), either r_1 presents activation A_{min} while, in the worst case, r_2 is at -1 and r_3 is at 1; or r_2 presents activation A_{min} while, again in the worst case, r_1 is at -1 and r_3 is at

1. *Since we have taken* $-\delta_1 = -\delta_2 = -W$, *both cases yield the same inequality, and we have:*

$$W + W - A_{min}W - \delta' W < h^{-1}(-A_{min}) \tag{8}$$

From Equations 7 and 8 above we obtain, respectively, $W > \frac{h^{-1}(A_{min})}{3A_{min}-\delta'}$ *and the constraint* $3A_{min} - \delta' > 0$, *and* $W > \frac{h^{-1}(-A_{min})}{2-A_{min}-\delta'}$ *and the constraint* $2 - A_{min} - \delta' < 0$. *Therefore, we need to satisfy* $2 - A_{min} < \delta' < 3A_{min}$ *or* $A_{min} > 1/2$. *Taking* $A_{min} = 0.6$ *and* $\delta' = 1.5$, *if* $W = 15$, *and therefore* $\theta_{\neg x} = 22.5$, *a network* $\mathcal{N}$ *that encodes* $\mathcal{P}$ *will also encode* $r_1 \succ r_3$ *and* $r_2 \succ r_3$. *Note that, one might need to change the original value of* W *in* $\mathcal{N}$, *obtained from the* Translation Algorithm (step 2), *in order to satisfy* $W > \frac{h^{-1}(A_{min})}{3A_{min}-\delta'}$ *and* $W > \frac{h^{-1}(-A_{min})}{2-A_{min}-\delta'}$.

The following algorithm is used for partially ordered theories.

- **Metalevel Priorities Algorithm 2 (Partial Ordering)**[7]

1. Consider a Logic Program $\mathcal{P} = \{r_1, ..., r_q\}$ and its equivalent neural network $\mathcal{N}$, obtained by applying the *Translation Algorithm* of *C-IL*^{2}*P*. Let $r_1, ..., r_n$ be the rules in $\mathcal{P}$ with consequent x, and $r_{n+1}, ..., r_o$ be the rules in $\mathcal{P}$ with consequent $\neg x$. Let $m = o - n$. If $r_i \succ r_j$ for $1 \leq j \leq n$ and $n+1 \leq i \leq o$, do:
   (a) Calculate:

$$A_{\min} > MAX\left(\frac{MAX_{\mathcal{P}}(\overrightarrow{k}, \overrightarrow{\mu}) - 1}{MAX_{\mathcal{P}}(\overrightarrow{k}, \overrightarrow{\mu}) + 1}, \frac{n(m+1)-1}{n(m+1)+1}\right),$$

   where $MAX(f_1, ..., f_s)$ returns the greatest number among $f_1, ..., f_s$;
   (b) Add a connection c_i from each hidden neuron r_i to the output neuron x;
   (c) Calculate:

$$\begin{aligned} \frac{1 - 2n + A_{\min}}{m - 1 - (m+1)A_{\min}} &< \delta < \frac{1 - n + (n+1)A_{\min}}{m(1 - A_{\min})} \text{ and} \\ MAX\left(n + \delta(m - 1 - A_{\min}), -nA_{\min} + m\delta\right) &< \delta' < 1 - n + A_{\min} + mA_{\min}\delta; \end{aligned}$$

   (d) Calculate:

$$W' > MAX\left(\frac{h^{-1}(-A_{\min})}{n + \delta(m - 1 - A_{\min}) - \delta'}, \frac{h^{-1}(A_{\min})}{1 - n + A_{\min} + mA_{\min}\delta - \delta'}, \frac{h^{-1}(-A_{\min})}{-nA_{\min} + m\delta - \delta'}, W\right);$$

   (e) Set the weights of connections c_j from each hidden neuron r_j to the output neuron x as W';
   (f) Set the weights of connections c_i from each hidden neuron r_i to the output neuron x as $-\delta W'$;
   (g) Set the threshold θ_x of output neuron x as $\delta' W'$.

Theorem 3.2. *Let* $r_1, ..., r_n, r_{n+1}, ..., r_o \in \mathcal{P}$ *and* $r_i \succ r_j$ *for* $1 \leq j \leq n$ *and* $n+1 \leq i \leq o$. *Let* $m = o - n$. *Let* $\mathcal{N}$ *be the network obtained by using the* Translation Algorithm *over* $\mathcal{P}$. *If* $\mathcal{N}$ *is modified by using the* Metalevel Priorities Algorithm 2 *then* $\mathcal{N}$ *computes* $T_{\mathcal{P}_\succ}$, *where* $\mathcal{P}_\succ$ *is the program* $\mathcal{P}$ *together with the preference relation* $r_i \succ r_j$.

Proof. We need to show that when any hidden neuron r_i is activated in $\mathcal{N}$, the consequent of rules r_j (x) is not activated in $\mathcal{N}$ (*Case 1*). We also need to guarantee, since we are changing

[7] The values of A_{min}, W, δ and δ' result from the proof of Theorem 3.2 below.

the threshold of x, that when any hidden neuron r_j is activated in $\mathcal{N}$, provided no hidden neuron r_i is activated, x is also activated in $\mathcal{N}$ (*Case 2i*), and that when no hidden neuron r_j is activated in $\mathcal{N}$, provided no hidden neuron r_i is activated, x is not activated in $\mathcal{N}$ (*Case 2ii*).

Case 1: In the worst case, n neurons r_j present activation 1, one neuron r_i is at A_{min} and $m-1$ neurons r_i are at -1. Let $W > 0$ be the weight from neurons r_j to x. Let $-\delta W$ ($\delta \in \Re^+$) be the weight from neurons r_i to x, and $\delta' W$ ($\delta' \in \Re$) the threshold of x. Thus, Equation 9 has to be satisfied.

$$nW - A_{\min}\delta W + (m-1)\delta W - \delta' W < h^{-1}(-A_{\min}) \tag{9}$$

Solving Equation 9 for the connection weight W yields Equations 10 and 11.

$$W > \frac{h^{-1}(-A_{\min})}{n + \delta(m - 1 - A_{\min}) - \delta'} \tag{10}$$

$$n + \delta(m - 1 - A_{\min}) - \delta' < 0 \tag{11}$$

Case 2i: Again in the worst case scenario, when one neuron r_j presents activation A_{min} and $n-1$ neurons r_j are at -1, provided the m neurons r_i are at $-A_{min}$, x should be activated. Equation 12 has to be satisfied.

$$A_{\min}W - (n-1)W + mA_{\min}\delta W - \delta' W > h^{-1}(A_{\min}) \tag{12}$$

Solving Equation 12 for the connection weight W yields Equations 13 and 14.

$$W > \frac{h^{-1}(A_{\min})}{1 - n + A_{\min} + mA_{\min}\delta - \delta'} \tag{13}$$

$$1 - n + A_{\min} + mA_{\min}\delta - \delta' > 0 \tag{14}$$

Case 2ii: Finally, when all neurons r_j are at $-A_{min}$ and all neurons r_i are at -1, the output neuron x should not be activated.

$$-nA_{\min}W + m\delta W - \delta' W < h^{-1}(-A_{\min}) \tag{15}$$

Solving Equation 15 yields Equations 16 and 17 below.

$$W > \frac{h^{-1}(-A_{\min})}{-nA_{\min} + m\delta - \delta'} \tag{16}$$

$$-nA_{\min} + m\delta - \delta' < 0 \tag{17}$$

From Equations 11, 14 and 17, we derive the following constraints on δ' :

$$n + \delta(m - 1 - A_{\min}) < \delta' < 1 - n + A_{\min} + mA_{\min}\delta \tag{18}$$

$$-nA_{\min} + m\delta < \delta' < 1 - n + A_{\min} + mA_{\min}\delta \tag{19}$$

From Equations 18 and 19, and assuming $m - 1 - (m+1)A_{\min} < 0$, we obtain the following constraint on δ :

$$\frac{1 - 2n + A_{\min}}{m - 1 - (m+1)A_{\min}} < \delta < \frac{1 - n + (n+1)A_{\min}}{m(1 - A_{\min})} \tag{20}$$

From Equation 20 we derive Equation 21, and solving Equation 21 for A_{min}, we obtain Equation 22.

$$(n(m+1)+1)\ {A_{\min}}^2 + 2A_{\min} - n(m+1) + 1 > 0 \tag{21}$$

(22) $$A_{\min} > \frac{n(m+1)-1}{n(m+1)+1}$$

Since we have assumed that $m - 1 - (m+1)A_{\min} < 0$ then $A_{\min} > \frac{m-1}{m+1}$ must hold. However, $\frac{n(m+1)-1}{n(m+1)+1} > \frac{m-1}{m+1}$, for $m \geq 1$ and $n \geq 1$, and thus Equation 22 suffices. Since *Metalevel Priorities Algorithm 2* complies with the above constraints, this completes the proof. ∎

The above theorem shows that single hidden layer networks can encode superiority relations between rules when $\succ$ defines a preference between *models*. This is so because, when r_j are the rules in $\mathcal{P}$ with consequent x, and r_i are the rules in $\mathcal{P}$ with consequent $\neg x$, $r_i \succ r_j$ actually states that any model of $\mathcal{P}$ containing $\neg x$ is preferred over any model of $\mathcal{P}$ containing x. However, the theorem does not show that when $\succ$ is any partial order, the intended meaning of $\mathcal{P}$ will be computed by $\mathcal{N}$. The following example illustrates this problem.

Example 3.4. ***(Partial ordering)***
$laysEggs(platypus)$
$hasFur(platypus)$
$monotreme(platypus)$
$hasBill(platypus)$
$r_1 : monotreme(x) \rightarrow mammal(x)$
$r_2 : hasFur(x) \rightarrow mammal(x)$
$r_3 : layEggs(x) \rightarrow \neg mammal(x)$
$r_4 : hasBill(x) \rightarrow \neg mammal(x)$
$r_1 \succ r_3$, $r_2 \succ r_4$

Intuitively, we should be able to derive $mammal(platypus)$ *because for every reason against* $mammal(platypus)$*, i.e.,* r_3 *and* r_4*, there is a stronger reason for* $mammal(platypus)$*, respectively,* r_1 *and* r_2*. However, we can not encode* $r_1 \succ r_3$ *into a network* $\mathcal{N}$ *without defining whether* $r_4 \succ r_1$ *or vice-versa. Even if we translate the preference relations into object level negation as failure,* $\mathcal{N}$ *will not be able to decide whether* $mammal(platypus)$ *or* $\neg mammal(platypus)$*. In fact, this is also a problem in Logic Programming.*

On the other hand, if we explicitly define the relations between (previously incomparable) conflicting rules, such as r_1 *and* r_4*, then* $\mathcal{N}$ *will compute the intended meaning of* $\mathcal{P}$*. In this example, if* $r_1 \succ r_3, r_1 \succ r_4$ *and* $r_2 \succ r_3, r_2 \succ r_4$ *then* $mammal(platypus)$ *will be derived.*

We now present a final example.

Example 3.5. ***(No propagation of ambiguity)***
$quaker(nixon)$
$republican(nixon)$
$r_1 : quaker(x) \rightarrow pacifist(x)$
$r_2 : republican(x) \rightarrow \neg pacifist(x)$
$r_3 : republican(x) \rightarrow footballfan(x)$
$r_4 : pacifist(x) \rightarrow antimilitary(x)$
$r_5 : footballfan(x) \rightarrow \neg antimilitary(x)$
$r_5 \succ r_4$

Here we can prove $\neg antimilitary(nixon)$*, despite of the conflict regarding Nixon's pacifism. Note that neither propagation of ambiguity nor trivialization of theory occur in this example. If, however, the superiority relation were empty, an ambiguity about Nixon's militarism would exist.*

4. Conclusion

In this paper we have seen that negative weights from the hidden layer to the output of a single hidden layer neural network may be regarded as the implementation of metalevel priorities, which define a superiority relation between conflicting rules of an extended logic program. In this way, the implementation of such a superiority relation in the network is straightforward, due to a characteristic of the *Translation Algorithm* of *C-IL*^{2}*P*, namely, the association of each hidden neuron of $\mathcal{N}$ with a rule of $\mathcal{P}$. More complex and elaborated preference relations, such as in Brewka's preferred subtheories [3] and Prakken and Sartor's argumented-based extended logic programming with defeasible priorities [12], are not representable in the network as easily, and would require changes in the basic structure of the networks investigated here. Such, more sophisticated, preference relations are left as future work.

References

[1] G. Antoniou, D. Billington, and M. J. Maher. Sceptical logic programming based default reasoning - defeasible logic rehabilitated. In Rob Miller and Murray Shanahan, editors, *COMMON SENSE 98, the fourth symposium on logical formalizations of commonsense reasoning*, pages 1–20, London, 1998. Queen Mary and Westfield College, University of London.

[2] G. Antoniou, M. J. Mahar, and D. Billington. Defeasible logic versus logic programming without negation as failure. *The Journal of Logic Programming*, 42:47–57, 2000.

[3] G. Brewka. *Nonmonotonic Reasoning - Logical Foundations of Commonsense*, volume 12 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press, 1990.

[4] A. S. dAvila Garcez, K. Broda, and D. M. Gabbay. Symbolic knowledge extraction from trained neural networks: A new approach. Technical Report TR-98-014, Department of Computing, Imperial College, London, 1998. Revised and Extended version submitted.

[5] A. S. dAvila 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.

[6] D. M. Gabbay. Compromise revision: A position paper. To appear, 1997.

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

[8] 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*, 1994.

[9] 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.

[10] A. C. Kakas, P. Mancarella, and P. M. Dung. The acceptability semantics of logic programs. In *Proceedings of the eleventh International Conference on Logic Programming ICLP*, pages 504–519. MIT Press, 1994. Cambridge, MA.

[11] D. Nute. Defeasible logic. In D. M. Gabbay, C.J. Hogger, and J. A. Robinson, editors, *Handbook of Logic in Artificial Intelligence and Logic Programming*, volume 3, pages 353–396. Oxford Science Publications, 1994.

[12] H. Prakken and G. Sartor. Argument-based extended logic programming with defeasible priorities. *Journal of Applied Non-Classical Logic*, 7:25–75, 1997.

[13] 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. MIT Press, 1986.

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