A neural approach to extended logic programs*

Jesús Medina, Enrique Mérida-Casermeiro, and Manuel Ojeda-Aciego

Dept. Matemática Aplicada. Univ. Málaga, Spain.
{jmedina,merida,aciego}@ctima.uma.es

Abstract. A neural net based development of multi-adjoint logic programming is presented. Transformation rules carry programs into neural networks, where truth-values of rules relate to output of neurons, truth-values of facts represent input, and network functions are determined by a set of general operators; the output of the net being the values of propositional variables under its minimal model. Some experimental results are reported.

1 Introduction

One of the advantages of the use of neural networks is its massively parallel architecture-based dynamics which are inspired by the structure of human brain, adaptation capabilities, and fault tolerance. The latter provides the ability of dealing with modeling and control aspects of complex processes, as well as with uncertain, incomplete and/or inconsistent information, being fuzzy logic a powerful mathematical tool for its study.

Fuzzy logic systems are capable to express nonlinear input/output relationships by a set of qualitative if-then rules, and to handle both numerical data and linguistic knowledge, especially the latter, which is extremely difficult to quantify by means of traditional mathematics. Neural networks, on the other hand, has an inherent learning capability, which enables the networks to adaptively improve their performance. In this work, we introduce a hybrid approach to handling uncertainty, which is expressed in the rich language of multi-adjoint logic but is *implemented* by using ideas borrowed from the world of neural networks.

Multi-adjoint logic programming, which was introduced in [6] as a refinement of residuated logic programming, allows for very general connectives in the body of the rules; moreover, sufficient conditions for the continuity of its semantics are known. The handling of uncertainty inside our logic model is based on the use of a generalised set of truth-values, usually a (finite or infinite) subset of the real unit interval $[0, 1]$, instead of the Boolean constants $\{v, f\}$. Such an approach is interesting for applications, for instance, consider a situation in which connectives are built from the users preferences, it is likely that knowledge is described by a many-valued logic program where connectives have many-valued truth functions and aggregation operators (such as arithmetic mean or weighted sum) where different implications could be needed for different purposes, and different aggregators are defined for different users, depending on their preferences.

* Partially supported by Spanish DGI project BFM2000-1054-C02-02.

In this paper, following ideas in [7], we present a neural net based implementation of the fixpoint semantics of multi-adjoint logic programming, introduced in [6], with the advantage that, at least potentially, we can calculate in parallel the answer for any query. The implementation using neural networks needs some preprocessing of the initial program to transform it in a *homogeneous* program; the ideas under this definition are based on the results in [1].

2 Preliminary definitions

Multi-adjoint logic programming is a general theory of logic programming which allows the simultaneous use of different implications in the rules and rather general connectives in the bodies; a preliminar version was presented in [6], where models of these programs were proved to be post-fixpoints of the immediate consequences operator, which turned out to be monotonic under very general hypotheses. In addition, the continuity of the immediate consequences operator was studied, and some sufficient conditions for its continuity were obtained.

To make this paper as self-contained as possible, the necessary definitions about multi-adjoint structures are included in this section. For motivating comments on the multi-adjoint stuff the interested reader is referred to [6].

The first interesting feature of multi-adjoint logic programs is that a number of different implications are allowed in the bodies of the rules. Formally, the basic definition is given below:

Definition 1. *Let $\langle L, \preceq\rangle$ be a complete lattice. A* multi-adjoint lattice *$\mathcal{L}$ is a tuple $(L, \preceq, \leftarrow_1, \&_1, \ldots, \leftarrow_n, \&_n)$ satisfying the following items:*

1. *$\langle L, \preceq\rangle$ is bounded, i.e. it has bottom and top elements;*
2. *$\top \,\&_i\, \vartheta = \vartheta \,\&_i\, \top = \vartheta$ for all $\vartheta \in L$ for $i = 1, \ldots, n$;*
3. *$(\leftarrow_i, \&_i)$ is an* adjoint pair *in $\langle L, \preceq\rangle$ for $i = 1, \ldots, n$; i.e.*
   (a) Operation $\&_i$ is increasing in both arguments.
   (b) Operation $\leftarrow_i$ is increasing in the first argument and decreasing in the second.
   (c) For any $x, y, z \in P$, $x \preceq (y \leftarrow_i z)$ holds if and only if $(x \,\&_i\, z) \preceq y$ holds.

The need of the monotonicity of operators $\leftarrow_i$ and $\&_i$ is clear, if they are to be interpreted as generalised implications and conjunctions. The third property in the definition can be adequately interpreted as a generalised modus-ponens rule.

Originally, the multi-adjoint paradigm was developed for multi-adjoint lattices, however, for the sake of simplicity, in this specific implementation we will restrict our attention to $[0, 1]$. As example of adjoint pairs in this lattice, we have the *product* $(\leftarrow_P, \&_P)$, *Gödel* $(\leftarrow_G, \&_G)$ and *Łukasiewicz* $(\leftarrow_L, \&_L)$ adjoint pairs; which are defined as

$$x \leftarrow_P y = \min(1, x/y) \qquad x \,\&_P\, y = x \cdot y$$

$$x \leftarrow_G y = \begin{cases} 1 & \text{if } y \leq x \\ x & otherwise \end{cases} \qquad x \,\&_G\, y = \min(x, y)$$

$$x \leftarrow_L y = \min(1 - x + y, 1) \qquad x \,\&_L\, y = \max(0, x + y - 1)$$

Definition 2. *A* multi-adjoint program *is a set of weighted rules* $\langle F, \vartheta \rangle$ *satisfying the following conditions:*

1. F *is a formula of the form* $A \leftarrow_i \mathcal{B}$ *where* A *is a propositional symbol called the* head *of the rule, and* $\mathcal{B}$ *is a well-formed formula built from propositional symbols* $B_1, \ldots, B_n$ *(*$n \geq 0$*) by the use of monotone operators in the multi-adjoint* Ω*-algebra, which is called the* body *formula.*
2. *The* weight ϑ *is an element (a truth-value) of* $[0, 1]$.

Facts *are rules with body* $\top$ *(which usually will not be written),*[1] *and a* query *(or* goal*) is a propositional symbol intended as a question* $?A$ *prompting the system.*

Regarding the implementation as a neural network, it will be useful to give a name to a specially simple type of rule: the *homogeneous rules*.

Definition 3. *A weighted formula is* homogeneous *if it has one of the following forms:*

$$\langle A \leftarrow_i \&_i(B_1, \ldots, B_n), \vartheta \rangle \qquad \langle A \leftarrow_i @(B_1, \ldots B_n), \top \rangle \qquad \langle A \leftarrow_i B_1, \vartheta \rangle$$

where the B_i *are propositional symbols.*

The homogeneous rules represent exactly the simplest type of (proper) rules we can have in our program. In some sense, homogeneous rules allow a straightforward generalization of the standard logic programming framework, in that no operators other than $\leftarrow_i$ and $\&_i$ are used.

Definition 4. *An* interpretation *is a mapping* I *from the set of propositional symbols* Π *to the lattice* $\langle L, \preceq \rangle$.

The fixed point semantics provided by the immediate consequences operator, given by van Emden and Kowalski [8], is generalised to the framework of multi-adjoint logic programs, as shown below:

Definition 5. *Let* $\mathbb{P}$ *be a multi-adjoint program. The* immediate consequences operator*,* $T^{\mathfrak{L}}_{\mathbb{P}} : \mathcal{I}_{\mathfrak{L}} \to \mathcal{I}_{\mathfrak{L}}$*, maps interpretations to interpretations, and for* $I \in \mathcal{I}_{\mathfrak{L}}$ *and* $A \in \Pi$ *is defined by*

$$T^{\mathfrak{L}}_{\mathbb{P}}(I)(A) = \sup \Big\{ \vartheta \,\&_i\, \hat{I}(\mathcal{B}) \mid \langle A \leftarrow_i \mathcal{B}, \vartheta \rangle \in \mathbb{P} \Big\}$$

As usual, it is possible to characterise the semantics of a multi-adjoint logic program by the post-fixpoints of $T_{\mathbb{P}}$; that is, an interpretation I is a model of a multi-adjoint logic program $\mathbb{P}$ iff $T_{\mathbb{P}}(I) \sqsubseteq I$. The $T_{\mathbb{P}}$ operator is proved to be monotonic and continuous under very general hypotheses, see [6].

Once we know that $T_{\mathbb{P}}$ can be continuous under very general hypotheses, then the least model can be reached in at most countably many iterations beginning with the least interpretation denoted $\triangle$, that is, the least model is $T^{\omega}_{\mathbb{P}}(\triangle)$.

In the next section we present a model of neural network which allows to evaluate the $T_{\mathbb{P}}$ operator and, therefore, by iteration will be able to approximate the actual values of the least model up to any prescribed precision.

[1] We will consider one *designated implication* to be used for the representation of facts, which is denoted $\leftarrow$. This designated implication will be also used in the procedure of translation of a program into a homogeneous one.

3 Obtaining a homogeneous program

In this section we present a procedure for transforming a given multi-adjoint logic program into a homogeneous one.

Handling rules. We will state a procedure for transforming a given program in another (equivalent) one containing only facts and homogeneous rules. It is based on two types of transformations:

$T1$. A weighted formula $\langle A \leftarrow_i \&_j(\mathcal{B}_1, \ldots, \mathcal{B}_n), \vartheta\rangle$ is substituted by the formulas:

$$\langle A \leftarrow_i A_1, \vartheta\rangle \qquad \langle A_1 \leftarrow_j \&_j(\mathcal{B}_1, \ldots, \mathcal{B}_n), \top\rangle$$

where A_1 is a fresh propositional symbol, and $\langle \leftarrow_j, \&_j\rangle$ is an adjoint pair.
For the case $\langle A \leftarrow_i @(\mathcal{B}_1, \ldots, \mathcal{B}_n), \vartheta\rangle$ in which the main connective of the body of the rule happens to be an aggregator, the transformation is similar:

$$\langle A \leftarrow_i A_1, \vartheta\rangle \qquad \langle A_1 \leftarrow @(\mathcal{B}_1, \ldots, \mathcal{B}_n), \top\rangle$$

where A_1 is a fresh propositional symbol, and $\leftarrow$ is a designated implication.

$T2$. A weighted formula $\langle A \leftarrow_i \Theta(\mathcal{B}_1, \ldots, \mathcal{B}_n), \vartheta\rangle$, where Θ is either $\&_i$ or an aggregator, and a component $\mathcal{B}_k$ is assumed to be either of the form $\&_j(\mathcal{C}_1, \ldots, \mathcal{C}_l)$ or $@(\mathcal{C}_1, \ldots, \mathcal{C}_l)$ is substituted by the following pair of formulas, respectively:

- $\langle A \leftarrow_i \Theta(\mathcal{B}_1, \ldots, \mathcal{B}_{k-1}, A_1, \mathcal{B}_{k+1}, \ldots, \mathcal{B}_n), \vartheta\rangle$ and $\langle A_1 \leftarrow_j \&_j(\mathcal{C}_1, \ldots, \mathcal{C}_l), \top\rangle$
- $\langle A \leftarrow_i \Theta(\mathcal{B}_1, \ldots, \mathcal{B}_{k-1}, A_1, \mathcal{B}_{k+1}, \ldots, \mathcal{B}_n), \vartheta\rangle$ and $\langle A_1 \leftarrow @(\mathcal{C}_1, \ldots, \mathcal{C}_l), \top\rangle$

The procedure to transform the rules of a program so that all the resulting rules are homogeneous, is based in the two previous transformations as follows:

1. Apply T1 to rules $\langle A \leftarrow_i \Theta(\mathcal{B}_1, \ldots, \mathcal{B}_n), \vartheta\rangle$ such that either $\Theta = \&_j$ with $i \neq j$, or $\Theta = @$ and $\vartheta \neq \top$.
2. Apply T2 to rules $\langle A \leftarrow_i \Theta(\mathcal{B}_1, \ldots, \mathcal{B}_n), \vartheta\rangle$ such that either $\Theta = \&_i$, or $\Theta = @$ and $\vartheta = \top$.

Handling facts. After the exhaustive application of the previous procedure we can assume that all our rules are homogeneous. Regarding facts, it might happen that the program contained facts about the same propositional symbol but with different weights.

Assume all the facts about A are $\langle A \leftarrow \top, \vartheta_j\rangle$, con $j \in \{1, \ldots, l\}$, then the following fact is substituted for the previous ones: $\langle A \leftarrow \top, \vartheta\rangle$ where $\vartheta = \sup\{\vartheta_j \mid j \in \{1, \ldots, l\}\}$.

The new program obtained from $\mathbb{P}$ after the homogenization of rules and facts is denoted $\mathbb{P}^*$. Note that in this new program there are new propositional symbols.

Preservation of the semantics. It is necessary to check that the semantics of the initial program has not been changed by the transformation. The following results will show that every model of $\mathbb{P}^*$ is also a model of $\mathbb{P}$ and, in addition, the minimal model of $\mathbb{P}^*$ is also the minimal model of $\mathbb{P}$.

Theorem 1. *Every model of $\mathbb{P}^*$ is also a model of $\mathbb{P}$.*

Theorem 2. *The minimal model of $\mathbb{P}^*$ when restricted to the variables in Π is also the minimal model of $\mathbb{P}$.*

4 Model of neural network

Using neural networks in the context of logic programming is not a novel idea; for instance, in [1] it is shown how fuzzy logic programs can be transformed into neural nets. In [2] the fixed point of the $T_{\mathbb{P}}$ operator for aciclic logic programs is constructed. This result is later extended in [3] to deal with the first order case. Our approach in this paper is interesting since our logic is much richer than classical or the usual versions of fuzzy logic in the literature, although we only consider the propositional case.

The set of operators to be implemented will consist of the three most important adjoint pairs defined previously. Note that every continuous t-norm is expressible as an ordinal sum of them [4]. We will implement a family of weighted sums defined as:

$$@_{(n_1,\ldots,n_m)}(p_1,\ldots,p_m) = \frac{n_1 p_1 + \cdots + n_m p_m}{n_1 + \cdots + n_m}$$

Each process unit is associated to either a propositional symbol of the initial program or an homogeneous rule. The state of the i-th neuron at time t is expressed by its output $S_i(t)$ and the state of the network is expressed by the state vector $\boldsymbol{S}(t) = (S_1(t), S_2(t), \ldots, S_N(t))$. The initial state for neurons associated to propositional symbols is ϑ_A if there is a fact $\langle A, \vartheta_A \rangle$ in the program and zero for any other component.

Regarding the user interface, there are two types of neurons, visible or hidden, the output of the visible neurons is the output of the net, whereas the output of the hidden neurons is only used as input values for other neurons. The set of visible neurons is formed by those associated with propositional symbols of the initial program, the others are hidden neurons.

The connection between neurons is denoted by a matrix of weights $\mathbf{W}$, in which w_{ij} indicates the existence or absence of connection from unit i to j; if the neuron i is associated with a weighted sum then w_{ij} represents the weights associated to input j. In the internal register of neuron i are allocated the i-th row of the matrix $\mathbf{W}$, the initial truth-value v_i, together with a signal m_i that indicates whether the neuron is associated to either a fact or a rule. So the net is a distributed information system.

Therefore, we have two vectors: one storing the truth-values $\boldsymbol{v}$ of atoms and homogeneous rules, and another $\boldsymbol{m}$ storing the type of the neurons in the net.

The signal m_i indicates the functioning mode of the neuron. If $m_i = 1$, then the neuron is associated to a propositional symbol (visible neuron). Its next state is the maximum value among all the operators involved in its input and the initial truth-values v_i. More precisely:

$$S_i(t+1) = \max\left\{ v_i, \max_{k/w_{ik}>0\{S_k(t)\}} \right\}$$

When a neuron is associated to the product, Gödel, or Łukasiewicz implication, respectively, then the signal m_i is set to 2, 3, and 4, respectively.

The output of the neuron mimics the behaviour of the implication in terms of the adjoint property when a rule of type m_i has been used; specifically, the output in the next instant will be:

- Product implication, $m_i = 2$: $S_i(t+1) = v_i \prod_{k/w_{ik}>0} S_k(t)$

– Gödel implication, $m_i = 3$: $S_i(t+1) = \min\left\{v_i, \min_{k/w_{ik}>0}\{S_k(t)\}\right\}$

– Łukasiewicz implication, $m_i = 4$: $S_i(t+1) = \max\{v_i + \sum_{k/w_{ik}>0}(S_k(t)-1), 0\}$

Neurons associated to aggregation operators have signal $t_i = 5$. Its output is:

$$S_i(t+1) = \sum_k w'_{ik} S_k(t) \qquad \text{where} \qquad w'_{ik} = \frac{w_{ik}}{\sum_r w_{ir}}$$

Example 1. The non homogeneous rule $\langle p \leftarrow_P @_{(3,7)}(q,r), 0.5\rangle$ is decomposed into:

$$\alpha = \langle h \leftarrow @_{(3,7)}(q,r), 1\rangle \qquad \beta = \langle p \leftarrow_P h, 0.5\rangle$$

The neurons corresponding to the new propositional symbol h and to homogeneous rules α and β are hidden neurons.

Note that in the n-th iteration, the output of neuron of the rule α is $S_\alpha(n) = @_{(3,7)}(S_q(n-1), S_r(n-1))$, and this output is used by the rule β in the next iteration to obtain $S_\beta(n+1) = 0.5 \cdot S_\alpha(n)$. □

The following result relates the behavior of the components of the state vector with the immediate consequence operator.

Theorem 3. *Given a homogeneous program $\mathbb{P}$, we have that $T^n_{\mathbb{P}}(\triangle)(A) = S_A(2n-2)$ for all propositional symbol A and $n \geq 1$.*

5 Representing a homogenous program by a neural net

Each neuron in the net represents either a symbol of the initial program $\mathbb{P}$ or a new rules of the homogeneous program $\mathbb{P}^*$. The different types of neurons are described below:

1. *A propositional symbol:* Its type is $m_i = 1$. The initial truth-value v_i is set either to 0 (by default) or to the truth-value if A is a fact.
   The i-th row of the matrix of weights has all components set to 0 but those corresponding to rules whose head is the given propositional symbol, in which case has value 1.
2. *Product implication:* These neurons correspond to a homogeneous product rule. Its internal registers are $m_i = 2$, v_i uses the truth-value of the rule in v_i, and the corresponding row in the matrix of weights is fixed with all components 0, except those assigned to propositional symbols involved in the body, which are set to 1.
3. *Gödel implication:* Similar to the previous case with $m_i = 3$.
4. *Łukasiewicz implication:* Similar to the previous case with $m_i = 4$.
5. *Weighted sums:* These neurons are related to rules of the type:

   $$\langle p \leftarrow @_{(n_1,n_2,\ldots,n_k)}(q_1, q_2, \ldots, q_k), 1\rangle$$

   So the truth-value is always one and it is unimportant which type of implication is used since all of them assign the same value to the head.
   The register of this type of neurons is set with truth-value $v_i = 1$, its type is $m_i = 5$, and the vector $w_{i\bullet}$ indicates the weights ($w_{ij} \geq 0$) of the rest of neurons on the output of the weighted sum.

In order to show the power of the neural implementation of the $T_{\mathbb{P}}$-operator, we present a more complex example.

Example 2. Consider the program with the following rules

$$\begin{array}{lll}
\langle s_1 \leftarrow_P t_1, 0.6\rangle & \langle p_1 \leftarrow_G r_1 \&_L s_1, 0.9\rangle & \langle x_1 \leftarrow_P @_{(5,2,1)}(p_1, x_2, x_3), 0.8\rangle \\
\langle r_1 \leftarrow_G q_1, 0.7\rangle & \langle p_2 \leftarrow_P r_2 \&_P s_2 \&_P t_2, 0.8\rangle & \langle x_2 \leftarrow_P @_{(5,3,1)}(p_2, x_1, x_3), 0.9\rangle \\
\langle r_2 \leftarrow_P q_2, 0.6\rangle & \langle p_3 \leftarrow_P r_3 \&_P s_3, 0.8\rangle & \langle x_3 \leftarrow_P @_{(5,3,3)}(p_3, x_1, x_2), 0.9\rangle \\
\langle r_3 \leftarrow_P q_3, 0.7\rangle & & \langle x \leftarrow_P @_{(4,3,2)}(x_1, x_2, x_3), 0.9\rangle
\end{array}$$

and facts $\langle t_1, 0.4\rangle$, $\langle q_1, 0.5\rangle$, $\langle s_2, 0.5\rangle$, $\langle q_2, 0.5\rangle$, $\langle t_2, 0.7\rangle$, $\langle q_3, 0.5\rangle$, $\langle s_3, 0.6\rangle$.

In this example the propositional symbols p_1, p_2 and p_3 can be considered as the same economic characteristic in each country when only are considered internal market information and x_1, x_2 and x_3 can be considered as that economic characteristic when relationship among countries are being considered. Finally x can be shown as a global economic one.

Since there exist non homogeneous rules the program is transformed into:

$$\begin{array}{rr}
\langle p_1 \leftarrow_L r_1 \&_L s_1, 0.8\rangle & \langle r_1 \leftarrow_L q_1, 0.7\rangle \\
\langle s_1 \leftarrow_P t_1, 0.6\rangle & \langle h_1 \leftarrow_P @_{(5,2,1)}(p_1, x_2, x_3), 1\rangle \\
\langle x_1 \leftarrow_P h_1, 0.8\rangle & \langle p_2 \leftarrow_P r_2 \&_P s_2 \&_P t_2, 0.8\rangle \\
\langle r_2 \leftarrow_P q_2, 0.6\rangle & \langle h_2 \leftarrow_P @_{(5,3,1)}(p_2, x_1, x_3), 1\rangle \\
\langle x_2 \leftarrow_P h_2, 0.9\rangle & \langle p_3 \leftarrow_P r_3 \&_P s_3, 0.8\rangle \\
\langle r_3 \leftarrow_P q_3, 0.7\rangle & \langle h_3 \leftarrow_P @_{(5,3,3)}(p_3, x_1, x_2), 1\rangle \\
\langle x_3 \leftarrow_P h_3, 0.9\rangle & \langle h \leftarrow_P @_{(4,3,2)}(x_1, x_2, x_3), 1\rangle \\
\langle x \leftarrow_P h, 0.9\rangle &
\end{array}$$

So, the network will have thirty-three neurons. The first eighteen are associated to propositional symbols: p_1, q_1, r_1, s_1, t_1, x_1, p_2, q_2, r_2, s_2, t_2, x_2, p_1, q_3, r_3, s_3, x_3, x, and the remaining ones are associated to the homogeneous rules.

The internal registers of the network are fixed as follows:

$$\begin{aligned}
\boldsymbol{m} &= (1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,5,5,5,5,3,3,2,2,2,2,2,2,4,2,2) \\
\boldsymbol{v} &= (0, 0.75, 0, 0, 0.8, 0, 0, 0.8, 0, 0.8, 0.7, 0, 0, 0.7, 0, 0.8, \\
&\qquad 0, 0, 1, 1, 1, 1, 0.9, 0.7, 0.8, 0.9, 0.8, 0.75, 0.9, 0.8, 0.95, 0.9, 0.9)
\end{aligned}$$

W is a 33×33 matrix with all components zeroes except those below:

$$\begin{array}{lllllll}
w_{1,23} = 1 & w_{3,24} = 1 & w_{4,25} = 1 & w_{6,26} = 1 & w_{7,27} = 1 & w_{9,28} = 1 & w_{12,29} = 1 \\
w_{13,30} = 1 & w_{15,31} = 1 & w_{17,32} = 1 & w_{18,33} = 1 & w_{19,1} = 5 & w_{19,12} = 2 & w_{19,17} = 1 \\
w_{20,6} = 3 & w_{20,7} = 5 & w_{20,17} = 1 & w_{21,6} = 3 & w_{21,12} = 3 & w_{21,13} = 5 & w_{22,6} = 4 \\
w_{22,12} = 3 & w_{22,17} = 2 & w_{23,3} = 1 & w_{23,4} = 1 & w_{24,2} = 1 & w_{25,5} = 1 & w_{26,19} = 1 \\
w_{27,9} = 1 & w_{27,10} = 1 & w_{27,11} = 1 & w_{28,8} = 1 & w_{29,20} = 1 & w_{30,15} = 1 & w_{30,16} = 1 \\
w_{31,14} = 1 & w_{32,21} = 1 & w_{33,22} = 1 & & & &
\end{array}$$

The network gets stabilized in 130 steps, giving the values of the minimal model $T_{\mathbb{P}}^{\omega}(\triangle)$ for each propositional symbol of the initial program.

6 Concluding remarks and future work

A new neural model has been introduced, which implements the fixpoint semantics of the multi-adjoint logic programming paradigm, which is a new approach to the treatment of reasoning under fuzzy data and/or uncertainty. As a result, it is possible to obtain the truth-values of all propositional symbols involved in the program in a parallel way. Due to space limitations, only a subset of connectives are implemented, but the framework can easily be modified to deal with other types of fuzzy rules and/or connectives.

As future work, we will extend the framework by adding learning capabilities to the net, so that it will be able to adapt the truth-values of the rules in a given program to fit a number of observations. Following this idea, a neural net implementation for abductive multi-adjoint logic programming [5] is planned.

References

1. P. Eklund and F. Klawonn. Neural fuzzy logic programming. *IEEE Trans. on Neural Networks*, 3(5):815–818, 1992.
2. S. Hölldobler and Y. Kalinke. Towards a new massively parallel computational model for logic programming. In *ECAI'94 workshop on Combining Symbolic and Connectioninst Processing*, pages 68–77, 1994.
3. S. Hölldobler, Y. Kalinke, and H.-P. Störr. Approximating the semantics of logic programs by recurrent neural networks. *Applied Intelligence*, 11(1):45–58, 1999.
4. E.P. Klement, R. Mesiar, and E. Pap. *Triangular norms*. Kluwer academic, 2000.
5. J. Medina, M. Ojeda-Aciego, and P. Vojtáš. A multi-adjoint logic approach to abductive reasoning. . Lect. Notes in Computer Science 2237, pages 269–283, 2001.
6. J. Medina, M. Ojeda-Aciego, and P. Vojtáš. Multi-adjoint logic programming with continuous semantics. Lect. Notes in Artificial Intelligence 2173, pages 351–364, 2001.
7. E. Mérida-Casermeiro, G. Galán, and J. Muñoz Pérez. An efficient multivalued Hopfield network for the traveling salesman problem. *Neural Processing Letters*, 14:203–216, 2001.
8. M. H. van Emden and R. Kowalski. The semantics of predicate logic as a programming language. *Journal of the ACM*, 23(4):733–742, 1976.