Interval-valued Neural Multi-Adjoint Logic Programs

J. Medina, E. Mérida-Casermeiro, and M. Ojeda-Aciego

Dept. Matemática Aplicada. Universidad de Málaga
{jmedina,merida,aciego}@ctima.uma.es

Abstract. The framework of multi-adjoint logic programming has shown to cover a number of approaches to reason under uncertainty, imprecise data or incomplete information. In previous works, we have presented a neural implementation of its fix-point semantics for a signature in which conjunctors are built as an ordinal sum of a finite family of basic conjunctors (Gödel and Łukasiewicz t-norms). Taking into account that a number of approaches to reasoning under uncertainty consider the set of subintervals of the unit interval as the underlying lattice of truth-values, in this paper we pursue an extension of the previous approach in order to accomodate calculation with truth-intervals.

1 Introduction

A number of different approaches have been proposed with the aim of better explaining observed facts, specifying statements, reasoning and/or executing programs under some type of uncertainty whatever it might be. The frameworks of *fuzzy logic programming* [10] and *residuated logic programming* [1] abstract the details of several well-known approaches to generalized logic programming.

Multi-adjoint logic programs were introduced as a common umbrella to cover a number of approaches to reason under uncertainty, imprecise data or incomplete information and, in particular, can be instantiated as both fuzzy logic programming and residuated logic programming. The handling of uncertainty in the multi-adjoint approach is based on the use of a generalised set of truth-values as an extension of fuzzy logic programming. On the other hand, multi-adjoint logic programming generalizes residuated logic programming in that several different implications are allowed in the same program, as a means to facilitate the task of the specification.

The recent paradigm of soft computing promotes the use and integration of different approaches for problem solving. The approach presented in [7, 9] introduced a hybrid framework to handling uncertainty, expressed in the language of multi-adjoint logic but implemented by using ideas from the world of neural networks.

Several semantics have been proposed for multi-adjoint logic programs but, regarding the implementation, the fix-point semantics was the chosen one: given a multi-adjoint logic program $\mathbb{P}$ its meaning (the minimal model) is obtained

by iterating the $T_{\mathbb{P}}$ operator. At least theoretically, by computing the sequence of iterations of $T_{\mathbb{P}}$ one could answer in parallel all the possible queries to $\mathbb{P}$; in order to take advantage of this potential parallelism, a recurrent neural network implementation of $T_{\mathbb{P}}$ was introduced in [7], where the truth values belonged to the unit interval, the connectives were the usual t-norms Gödel, product and Łukasiewicz, together with any weighted sum. Later, this neural net was improved in order to be able to use any finite ordinal sum of Gödel, product and Łukasiewicz t-norms in [8].

The machinery underlying multi-adjoint programs is that of adjoint pairs, which abstracts out the behaviour of classical conjunction and implication and provides a convenient version of the modus-ponens rule to be used on sets of truth-values more general than $\{0, 1\}$. A possible extension of the previous approach consists in implementing an interval-based semantics. This generalization is not unnatural, since when defining a fuzzy set sometimes it is not easy to associate a value in the unit interval to any element in the set, but we'd rather associate an interval instead; this generalization of fuzzy set is called an interval-valued fuzzy set.

The method of using intervals, either symbolic $(\inf, \sup)$ or numerical $[a, b]$, to describe uncertain information has been adopted in several mechanisms, and they are useful in applications such as decision and risk analysis, engineering design, and scheduling. Intervals are also used in some frameworks for generalized logic programming such as the *hybrid probabilistic logic programs* [2] and the *probabilistic deductive databases* [6]. Just note that, in the latter framework, one could write rules like:

$$paper\_accepted \xleftarrow{\langle [0.7,0.95],[0.03,0.2] \rangle} good\_work, good\_referees$$

where we have a complex confidence value containing two probability intervals, one for the case where *paper_accepted* is true and other for the case where *paper_accepted* is false (there can exist some lack of information, or undefinedness, in these intervals).

The purpose of this paper is to present a refined version of the neural implementation of [7, 8] in order to cope with interval-valued data. The main difference with the previous version is that the type of conjunctors considered in the neuron are pairs of conjunctors on the unit interval, the first (second) one intended to make the calculations on the initial (final) point of the truth-intervals; in addition, the procedure to handle the aggregators, usually weighted sums, has been conveniently adapted to work with truth-intervals.

The structure of the paper is as follows: In Section 2, the syntax and semantics of multi-adjoint logic programs are introduced; in Section 3, the new proposed neural model for homogeneous multi-adjoint programs is presented in order to work with truth-intervals, a high level implementation is introduced and proven to be sound. The paper finishes with some conclusions and future work.

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. To make this paper as self-contained as possible, the necessary definitions about multi-adjoint structures are included in this section. The basic definition is the generalization of residuated lattice given below:

Definition 1. *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 a bounded lattice, 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. $(\&_i, \leftarrow_i)$ *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 argument,*
   (c) *For any* $x, y, z \in P$*, we have that* $x \preceq (y \leftarrow_i z)$ *holds if and only if* $(x \,\&_i\, z) \preceq y$ *holds.*

2.1 Syntax and semantics

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, which is called the* body*, built from propositional symbols* $B_1, \ldots, B_n$ *(*$n \geq 0$*) by the use of monotone operators.*
2. *The* weight ϑ *is an element of the underlying truth-values lattice* L*.*

Facts *are rules with body*[1] $\top$ *and a* query *(or* goal*) is a propositional symbol intended as a question* $?A$ *prompting the system.*

Once presented the syntax of multi-adjoint programs, the semantics is given below.

Definition 3. *An* interpretation *is a mapping* I *from the set of propositional symbols* Π *to the lattice* $\langle L, \preceq \rangle$*.*

Note that each of these interpretations can be uniquely extended to the whole set of formulas, and this extension is denoted as $\hat{I}$. The set of all the interpretations is denoted $\mathcal{I}_{\mathfrak{L}}$.

The ordering $\preceq$ of the truth-values L can be easily extended to $\mathcal{I}_{\mathfrak{L}}$, which also inherits the structure of complete lattice and is denoted $\sqsubseteq$. The minimum element of the lattice $\mathcal{I}_{\mathfrak{L}}$, which assigns $\bot$ to any propositional symbol, will be denoted $\triangle$.

[1] It is also customary not to write any body.

Definition 4.

1. *An interpretation* $I \in \mathcal{I}_{\mathfrak{L}}$ satisfies $\langle A \leftarrow_i \mathcal{B}, \vartheta \rangle$ *if and only if* $\vartheta \preceq \hat{I}(A \leftarrow_i \mathcal{B})$.
2. *An interpretation* $I \in \mathcal{I}_{\mathfrak{L}}$ *is a* model *of a multi-adjoint logic program* $\mathbb{P}$ *iff all weighted rules in* $\mathbb{P}$ *are satisfied by* I.

The operational approach to multi-adjoint logic programs used in this paper will be based on the fixpoint semantics provided by the immediate consequences operator, given in the classical case by van Emden and Kowalski, which can be generalised to the multi-adjoint framework by means of the adjoint property, as shown below:

Definition 5. *Let* $\mathbb{P}$ *be a multi-adjoint program; the* immediate consequences operator, $T_{\mathbb{P}} \colon \mathcal{I}_{\mathfrak{L}} \to \mathcal{I}_{\mathfrak{L}}$, *maps interpretations to interpretations, and for* $I \in \mathcal{I}_{\mathfrak{L}}$ *and* $A \in \Pi$ *is given by*

$$T_{\mathbb{P}}(I)(A) = \sup \left\{ \vartheta \,\&_i\, \hat{I}(\mathcal{B}) \mid \langle A \leftarrow_i \mathcal{B}, \vartheta \rangle \in \mathbb{P} \right\}$$

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.

Once one knows 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, that is, the least model is $T_{\mathbb{P}} \uparrow \omega(\triangle)$.

3 An interval-valued network for multi-adjoint logic programming

Regarding the implementation as a neural network, the introduction of the so-called *homogeneous rules* given in [7], provided a simpler and standard representation for any multi-adjoint program.

Definition 6. *A weighted formula is said to be* homogeneous *if it has one of the following forms:*

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

where $A, B_1, \ldots, B_n$ *are propositional symbols.*

The homogeneous rules represent exactly the simplest type of (proper) rules one can have in a 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. The way in which a general multi-adjoint program is homogenized is irrelevant for the purposes of this paper;

anyway, it is worth mentioning that it is a model-preserving procedure with linear complexity.

In this section, we introduce a neural network that implements an extension of the previous approaches [7, 8] with the enhancements stated in the introduction: namely, the calculation with truth-intervals and the possibility of considering ordinal sums. This extension generates an overloaded use of intervals, on the one hand, as truth-values and, on the other hand, as the data needed in the definition of an ordinal sum which, aiming at self-contention, is recalled below:

Definition 7. *Let $(\&_i)_{i\in A}$ be a family of t-norms and a family of non-empty pairwise disjoint subintervals $[x_i, y_i]$ of $[0, 1]$. The* ordinal sum *of the summands $(x_i, y_i, \&_i)$, $i \in A$ is the t-norm $\&$ defined as*

$$\&(x,y) = \begin{cases} x_i + (y_i - x_i)\,\&_i(\frac{x-x_i}{y_i-x_i}, \frac{y-x_i}{y_i-x_i}) & \text{if } x, y \in [x_i, y_i] \\ \min(x, y) & \text{otherwise} \end{cases}$$

For the handling of truth-intervals we will consider the lattice $\langle \mathcal{I}([0, 1]), \leq\rangle$, where $\mathcal{I}([0, 1])$ is the set of all subintervals of $[0, 1]$, and given $[a, b], [c, d] \in \mathcal{I}([0, 1])$, we have

$$[a, b] \leq [c, d] \text{ if and only if } a \leq c \text{ and } b \leq d$$

$$\sup\{[a, b], [c, d]\} = [\sup\{a, c\}, \sup\{b, d\}]$$

$$\inf\{[a, b], [c, d]\} = [\inf\{a, c\}, \inf\{b, d\}]$$

3.1 Description of the net

Each process unit is associated either to a propositional symbol of the initial program or to an homogeneous rule of the transformed program $\mathbb{P}$. The state of the i-th neuron in the instant t is expressed by its output vector, $\boldsymbol{S_i(t)} = (S_i^1(t), S_i^2(t))$, which denotes an interval. Thus, the state of the network can be expressed by means of a state matrix $S(t)$, whose rows are the output of the neurons forming the network. In the initial state matrix, $S(0)$, all the rows are null, that is $(0, 0)$, but those corresponding to neurons associated to a propositional symbol A, in which case $\boldsymbol{S_A(0)}$ is defined to be:

$$\boldsymbol{S_A(0)} = (S_A^1(0), S_A^2(0)) = \begin{cases} (\vartheta_A^1, \vartheta_A^2) & \text{if } \langle A \leftarrow \top, [\vartheta_A^1, \vartheta_A^2]\rangle \in \mathbb{P}, \\ (0, 0) & \text{otherwise.} \end{cases}$$

The connection between neurons is denoted by a matrix of weights W, in which w_{ij} indicates the existence (value 1) or absence (value 0) of connection from unit j to unit i; if the neuron represents a weighted sum, then the weights are also represented in the entries associated to any of its inputs. The weights of the connections related to neuron i (that is, the i-th row of the matrix W) are represented by a vector $\boldsymbol{w_i}$, and are allocated in an internal vector register of the neuron.

Four more internal registers $\boldsymbol{v_i}, \boldsymbol{x_i}, \boldsymbol{y_i}, \boldsymbol{m_i}$ are defined in any neuron:

– The initial truth-interval $[v_i^1, v_i^2]$ of a propositional symbol or homogeneous rule is loaded in the internal register $\boldsymbol{v_i}$.
– The registers $\boldsymbol{x_i}, \boldsymbol{y_i}$ are used to restrict the domain of primitive conjunctors in order to enable the possibility of defining conjunctors as *ordinal sums* by using the technique presented in [8]. Note that if $\boldsymbol{x_i} = (x_i^1, x_i^2)$, then x_i^1 denotes the initial point of the domain of the first conjunctor, whereas x_i^2 denotes the initial point of the domain of the second conjunctor.
– Vector $\boldsymbol{m_i} = (m_i^1, m_i^2)$ indicates the functioning mode of the neuron, the possible pairs are $(1, 1)$ to denote a propositional symbol, or $(5, 5)$ to denote an aggregator rule, or (x, y) with $2 \leq x, y \leq 4$, to denote a rule where x (resp. y) denotes the t-norm acting in the beginning (resp. end) of the intervals.

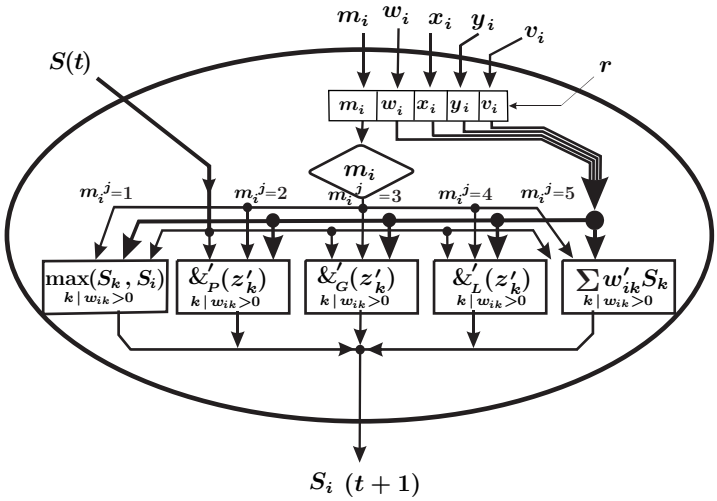

Fig. 1. The proposed generic neuron.

3.2 On the output of a neuron

Taking into account the values of the registers and the state matrix S at instant t the output of a given neuron is computed as follows:

If $\boldsymbol{m_i} = (1, 1)$, then its next state is the maximum value among all the operators involved in its input and its previous state in each component. More formally, let us denote $K_i = \{k \mid w_{ik} > 0\}$ then we have:[2]

$$\boldsymbol{S_i(t+1)} = \max\{\boldsymbol{S_i(t)}, \max\{\boldsymbol{S_k(t)} \mid k \in K_i\}\}$$

[2] Note that the computation involves the join operation in the lattice $\mathcal{I}([0, 1])$.

When $\boldsymbol{m_i} = (m_i^1, m_i^2)$, with $m_i^1, m_i^2 \in \{2, 3, 4\}$, the neuron i is associated to a homogenous rule where the connective is built from two basic conjunctors. The description of the output of the neuron will be given in terms of the vectors $\boldsymbol{z_k(t)} = (z_k^1(t), z_k^2(t))$, which accommodate the value of $S_k^j(t)$, originally in the subinterval $[x_i^j, y_i^j]$, to the unit interval and, for each $k \in K_i$ and $j \in \{1, 2\}$, are computed as follows:

$$z_k^j(t) = \begin{cases} 1 & \text{if } S_k^j(t) \geq y_i^j \\ \dfrac{S_k^j(t) - x_i^j}{y_i^j - x_i^j} & \text{if } x_i^j \leq S_k^j(t) < y_i^j \\ 0 & \text{if } S_k^j(t) < x_i^j \end{cases}$$

Once the vectors $\boldsymbol{z_k(t)}$ have been computed, then the corresponding conjunctor (product, Gödel or Łukasiewicz) is applied, and the result is again accommodate in the interval $[x_i^j, y_i^j]$. Thus, the output of the neuron is[3]

$$S_i^j(t+1) = \begin{cases} x_i^j + (y_i^j - x_i^j) \cdot (v_i^j \,\&_P\, z_1^j \,\&_P \cdots \&_P\, z_{N_i}^j) & \text{if } m_i^j = 2 \\ x_i^j + (y_i^j - x_i^j) \cdot (v_i^j \,\&_G\, z_1^j \,\&_G \cdots \&_G\, z_{N_i}^j) & \text{if } m_i^j = 3 \\ x_i^j + (y_i^j - x_i^j) \cdot (v_i^j \,\&_L\, z_1^j \,\&_L \cdots \&_L\, z_{N_i}^j) & \text{if } m_i^j = 4 \end{cases} \tag{1}$$

where N_i is the cardinal of $K_i = \{k \mid w_{ik} > 0\}$.

Note that, for instance, if $\boldsymbol{m_i} = (2, 3)$, the components $S_i^j(t)$ of the output mimic the behaviour of the product (for $j = 1$) and Gödel (for $j = 2$) implications, respectively, in terms of the adjoint property.

Finally, a neuron associated to an aggregator has $\boldsymbol{m_i} = (5, 5)$, and its output is

$$\boldsymbol{S_i(t+1)} = \sum_{k \in K_i} w'_{ik} \boldsymbol{S_k(t)} \qquad \text{where} \qquad w'_{ik} = \frac{w_{ik}}{\sum_{r \in K_i} w_{ir}}$$

3.3 Implementation

A number of simulations have been obtained through a MATLAB implementation in a conventional sequential computer. A high level description of the implementation is given below:

1. **Initialize** the network is with the appropriate values of the matrices V, X, Y, M, W and, in addition, a tolerance value *tol* to be used as a stop criterion. The output $\boldsymbol{S_i(t)}$ of the neurons associated to facts are initialized with its truth-value $\boldsymbol{v_i} = (v_i^1, v_i^2)$.
2. **Find** the neurons k (if any) which operate on the neuron i, that is, construct the set $K_i = \{k \mid w_{ik} > 0\}$ and calculate $N_i = \sum_{K_i} w_{ik}$.
   When $w_{ik} = 1$ for all $k \in K_i$ then N_i is the cardinal of K_i.

[3] The functions corresponding to each case are represented in Fig. 1 as $\&'_P$, $\&'_G$, $\&'_L$, resp.

3. **Repeat** Update all the states $\boldsymbol{S_i(t)} = (S_i^1(t), S_i^2(t))$ of the neurons of the network:
   (a) If $\boldsymbol{m_i} = (1,1)$, then update the state of neuron i as follows, for $j \in \{1,2\}$

$$S_i^j(t+1) = \begin{cases} \max\{S_i^j(t), \max_{K_i} S_k^j(t)\} & \text{if } K_i \neq \varnothing \\ v_i^j & \text{otherwise} \end{cases}$$

   (b) If $m_i^j = 2$, 3 or 4, then update the state of neuron i to $S_i^j(t+1)$ as defined in Eq (1).
   (c) If $m_i^j = 5$, then the neuron corresponds to an aggregator, and is updated by:

$$S_i^j(t+1) = \frac{1}{N_i} \sum_{k \in K_i} w_{ik} \cdot S_k^j(t)$$

   Until the stop criterion $\|S(t+1) - S(t)\|_\infty < tol$ is fulfilled.

We introduce some toy examples below in order to show the behavior of the network.

Example 1. Consider the homogenous program with the fact $\langle r \leftarrow \top, [0.1, 0.2]\rangle$ and the two rules $\langle p \leftarrow @_{1,2}(r,q), [1,1]\rangle, \langle q \leftarrow_{PG} p, [0.6, 0.7]\rangle$.

The net for the program will consist of five neurons, three of which represent propositional symbols p, q, r, and the rest are needed to represent the rules (one for the aggregator and another for the product-Gödel rule).

Note that no ordinal sum occurs in the program, therefore the matrices X and Y are constantly 0 and 1, respectively. The initial values for the rest of matrices are:

$$V = \begin{pmatrix} 0.0 & 0.0 \\ 0.0 & 0.0 \\ 0.1 & 0.2 \\ 1.0 & 1.0 \\ 0.6 & 0.7 \end{pmatrix}, \qquad M = \begin{pmatrix} 1 & 1 \\ 1 & 1 \\ 1 & 1 \\ 5 & 5 \\ 2 & 3 \end{pmatrix}, \qquad W = \begin{pmatrix} \cdot & \cdot & \cdot & 1 & \cdot \\ \cdot & \cdot & \cdot & \cdot & 1 \\ \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & 1 & 2 & \cdot & \cdot \\ \cdot & 1 & \cdot & \cdot & \cdot \end{pmatrix}$$

After running the net, it gets stabilized after 328 iterations providing the following truth-intervals for p, q and r (only three decimal digits are given):

$$p = [0.055, 0.200], \qquad q = [0.033, 0.200], \qquad r = [0.100, 0.200] \qquad \square$$

Example 2. Consider the fact $\langle r, [0.3, 0.5]\rangle$ and the rules $\langle p \leftarrow @_{1,2}(r,q), [1,1]\rangle$ and $\langle q \leftarrow_{TG} p \,\&_{TG}\, r, [0.7, 0.8]\rangle$, where T is the ordinal sum given by the Łukasiewicz conjunction on $[0.2, 0.4]$ and product conjunction on $[0.6, 0.9]$.

The net for the program consists of eight neurons, three of which represent variables p, q, r, and the rest are needed to represent the rules (one for the aggregator and four for the TG one).

The initial values of the net are the matrices:

$$V = \begin{pmatrix} 0.0 & 0.0 \\ 0.0 & 0.0 \\ 0.3 & 0.5 \\ 1.0 & 1.0 \\ 0.7 & 0.8 \\ 0.7 & 0.8 \\ 0.7 & 0.8 \\ 1.0 & 1.0 \end{pmatrix}, M = \begin{pmatrix} 1 & 1 \\ 1 & 1 \\ 1 & 1 \\ 5 & 5 \\ 4 & 3 \\ 2 & 3 \\ 3 & 3 \\ 3 & 3 \end{pmatrix}, X = \begin{pmatrix} 0.0 & 0.0 \\ 0.0 & 0.0 \\ 0.0 & 0.0 \\ 0.0 & 0.0 \\ 0.2 & 0.0 \\ 0.6 & 0.0 \\ 0.0 & 0.0 \\ 0.0 & 0.0 \end{pmatrix}, Y = \begin{pmatrix} 1.0 & 1.0 \\ 1.0 & 1.0 \\ 1.0 & 1.0 \\ 1.0 & 1.0 \\ 0.4 & 1.0 \\ 0.9 & 1.0 \\ 1.0 & 1.0 \\ 1.0 & 1.0 \end{pmatrix}$$

together with the following matrix of weights

$$W = \begin{pmatrix} \cdot & \cdot & \cdot & 1 & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & 2 & 1 & \cdot & \cdot & \cdot & \cdot & \cdot \\ 1 & \cdot & 1 & \cdot & \cdot & \cdot & \cdot & \cdot \\ 1 & \cdot & 1 & \cdot & \cdot & \cdot & \cdot & \cdot \\ 1 & \cdot & 1 & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & 1 & 1 & 1 & \cdot \end{pmatrix}$$

After running the net, it gets stabilized at 430 iterations giving the following output for p, q and r:

$$p = [0.233, 0.500], \qquad q = [0.355, 0.500], \qquad r = [0.400, 0.500] \qquad \square$$

Regarding the soundness of the implementation sketched above, the following theorem can be obtained, although space restrictions do not allow to include the proof.

Theorem 1. *Given a homogeneous program $\mathbb{P}$ and a propositional symbol A, then the sequence $\boldsymbol{S_A(n)}$ approximates the value of the least model of $\mathbb{P}$ in A up to any prescribed level of precision.*

4 Conclusions and future work

A new neural-like model has been proposed which extends that recently given to multi-adjoint logic programming in such a way that it is possible both to do calculations with truth-intervals and obtaining the computed truth-values of all propositional symbols involved in the program in a parallel way. This extended approach considers, in addition to the three most important adjoint pairs in the unit interval (product, Gödel, and Łukasiewicz) and weighted sums, the combinations as finite ordinal sums of the previous conjunctors.

The original model of neuron could have been extended by considering simpler units, each one dedicated to represent a different type of homogeneous rule or propositional symbol; however, we have decided to extend the original generic

model of neuron, capable of adapting to perform different functions according with its inputs. The advantage of this choice is related to the uniform (although more complex) description of the units, and the attainment of a clearer network than using simpler units. An analysis of the compromise between simplicity of the units and complexity of the network will be the subject of future work.

References

1. C.V. Damásio and L. Moniz Pereira. Monotonic and residuated logic programs. In *Symbolic and Quantitative Approaches to Reasoning with Uncertainty, ECSQARU'01*, pages 748–759. Lect. Notes in Artificial Intelligence, 2143, 2001.
2. A. Dekhtyar and V.S. Subrahmanian. Hybrid Probabilistic Programs, *Journal of Logic Programming* 43(3):187–250, 2000
3. P. Eklund and F. Klawonn. Neural fuzzy logic programming. *IEEE Tr. on Neural Networks*, 3(5):815–818, 1992.
4. 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.
5. 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.
6. L. V. S. Lakshmanan and F. Sadri. On a theory of probabilistic deductive databases. *Theory and Practice of Logic Progr.*, 1(1):5–42, 2001.
7. J. Medina, E. Mérida-Casermeiro, and M. Ojeda-Aciego. A neural implementation of multi-adjoint logic programming. *Journal of Applied Logic*, 2(3):301-324, 2004.
8. J. Medina, E. Mérida-Casermeiro, and M. Ojeda-Aciego. Decomposing Ordinal Sums in Neural Multi-Adjoint Logic Programs. *IBERAMIA 2004*, pages 717–726. Lect. Notes in Artificial Intelligence 3315, 2004.
9. J. Medina, E. Mérida-Casermeiro, and M. Ojeda-Aciego. A neural approach to extended logic programs. In *7th Intl Work Conference on Artificial and Natural Neural Networks, IWANN'03*, pages 654–661. Lect. Notes in Computer Science 2686, 2003.
10. P. Vojtáš. Fuzzy logic programming. *Fuzzy Sets and Systems*, 124(3):361–370, 2001.