A neural approach to abductive multi-adjoint reasoning

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

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

Abstract. A neural approach to propositional multi-adjoint logic programming was recently introduced. In this paper we extend the neural approach to multi-adjoint deduction and, furthermore, modify it to cope with abductive multi-adjoint reasoning, where adaptations of the uncertainty factor in a knowledge base are carried out automatically so that a number of given observations can be adequately explained.

Keywords. Knowledge Representation, Logic Programming, Neural Nets

1 Introduction

Uncertainty, incompleteness, and/or inconsistency have to be faced, sooner or later, when dealing with complex applications of knowledge representation. As a result, several frameworks for manipulating data and knowledge have been proposed in the form of extensions to classical logic programming and deductive databases. The underlying uncertainty formalism in the proposed frameworks includes probability theory, fuzzy set theory, many-valued logic, or possibilistic logic. Our approach to modelling uncertainty in human cognition and real world applications is based on the multi-adjoint logic programming paradigm.

In this paper we introduce and study a model of abduction problem. Abductive reasoning is widely recognized as an important form of reasoning with uncertain information that is appropriate for many problems in Artificial Intelligence. Broadly speaking, abduction aims at finding explanations for, or causes of, observed phenomena or facts; it is inference to the best explanation, a pattern of reasoning that occurs in such diverse places as medical diagnosis, scientific theory formation, accident investigation, language understanding, and jury deliberation. More formally, abduction is an inference mechanism where given a knowledge base and some observations, the reasoner tries to find hypotheses which together with the knowledge base explain the observations. Reasoning based on such an inference mechanism is referred to as abductive reasoning.

Abduction methods can be characterised within different frameworks, such as logical, neural, data analysis, etc. We will use the logical and the neural approach in this paper, in order to present our model of abduction reasoning. In general, as pointed out in [1], a neural network is assumed to support complex patterns of interaction between effects and causes; however, it is very difficult to model relationships such as competition between two causes when a common

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

effect shows up and cooperation between them otherwise. This is why we have chosen a hybrid approach, in which all these relationships are expressed in the rich language of multi-adjoint logic. The purpose of this work is to link, following ideas from [3, 9], the theoretical framework for abductive multi-adjoint reasoning presented in [7], and its neural net implementation [6].

Transformation rules carry multi-adjoint logic programs into corresponding neural networks, where the confidence values of rules relate to output of neurons in the net, confidence values of facts represent input values for the net, and network functions are determined by a set of conjunctors, implications and aggregation operators; the output of the net being the values of the propositional variables in the program under its minimal model. Also, some examples from a first prototype are reported.

2 Preliminary Definitions

In order to make this paper as self-contained as possible, we give here the essentials of multi-adjoint logic programming, and its neural implementation framework. Due to space limitations, neither comments nor motivations are presented, the interested reader is referred to [8] where multi-adjoint logic programs are formally introduced and its procedural semantics is given, to [7] where the framework for abductive reasoning is set, and to [6] in which a neural approach to multi-adjoint logic programming is given.

Originally, the multi-adjoint paradigm was developed for multi-adjoint lattices (a much more general structure for the set of truth-values than the unit real interval $[0, 1]$), however, for the sake of simplicity, in this specific application we will restrict our attention to $[0, 1]$. However, the other special feature of multi-adjoint logic programs, that a number of different implications are allowed in the bodies of the rules, will remain in force. Formally,

Definition 1. *A* multi-adjoint program *is a set of rules* $\langle A \leftarrow_i \mathcal{B}, \vartheta \rangle$ *satisfying:*

1. *The* head *of the rule* A *is a propositional symbol.*
2. *The* body *formula* $\mathcal{B}$ *is a formula of* $\mathfrak{F}$ *built from propositional symbols* $B_1, \ldots, B_n$ *(*$n \geq 0$*) by the use of conjunction (*$\&_j$*) operators.*
3. *The* confidence factor ϑ *is an element (a truth-value) of* $[0, 1]$.

Facts *are rules with body* $\top$ *(which usually will not be written), 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 2. *A rule* $\langle A \leftarrow_i \mathcal{B}, \vartheta \rangle$ *is said to be* homogeneous *if its body is either a propositional symbol or a* $\&_i$*-conjunction of variables.*

As usual, an *interpretation* is a mapping $I \colon \Pi \to L$. Note that each of these interpretations can be uniquely extended to the whole set of formulas. The ordering $\preceq$ of the truth-values L can be easily extended to the set of interpretations, which also inherits the structure of complete lattice.

Definition 3.

1. *An interpretation* I 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 *is a* model *of a multi-adjoint logic program* $\mathbb{P}$ *iff all weighted rules in* $\mathbb{P}$ *are satisfied by* I.
3. *An element* $\lambda \in L$ *is a* correct answer *for a program* $\mathbb{P}$ *and a query* $?A$ *if for any interpretation* I *which is a model of* $\mathbb{P}$ *we have* $\lambda \preceq I(A)$.

The immediate consequences operator, given by van Emden and Kowalski, can be easily generalised to the framework of multi-adjoint logic programs.

Definition 4. *Let* $\mathbb{P}$ *be a multi-adjoint program. The* immediate consequences operator $T_{\mathbb{P}}$ *maps interpretations to interpretations, and is defined by*

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

The semantics of a multi-adjoint logic program can be characterised, as usual, 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 [8], and it is remarkable that these results are true even for non-commutative and non-associative conjunctors. In particular, by continuity, the least model can be reached in at most countably many iterations of $T_{\mathbb{P}}$ on the least interpretation.

3 Model of Neural Network

In this section we briefly describe the model of neural net chosen to implement the immediate consequences operator $T_{\mathbb{P}}$ for multi-adjoint logic programming.

Before describing the model, some considerations are needed: The set of operators to be implemented will consist of the three most important adjoint pairs: product $(\&_P, \leftarrow_P)$, Gödel $(\&_G, \leftarrow_G)$ and Łukasiewicz $(\&_L, \leftarrow_L)$. Regarding the selection of operators implemented, just recall that every t-norm, the type of conjunctor more commonly used in the context of fuzzy reasoning, is expressible as a direct sum of these three basic conjunctors [5]. Regarding the aggregation operators, we will implement a family of weighted sums, which are denoted $@_{(n_1,\ldots,n_m)}$ and defined as follows:

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

A neural net is considered in which each process unit is associated to either a propositional symbol or an homogeneous rule. The state of the i-th neuron in the instant t is expressed by its output $I_i(t)$. Thus, the state of the network can be expressed by means of a state vector $\boldsymbol{I}(p)$, whose components are the output of the neurons in the net, and its initial state is the null vector.

Regarding the user interface, there are two layers, a visible one, whose output is part of the overall output of the net, and a hidden layer, whose outputs are only used as input values for other neurons.

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

We will work with two vectors in the internal register: the first one stores the confidence values $\boldsymbol{v}$ of atoms and homogeneous rules, the second vector, $\boldsymbol{m}$, stores the functioning mode of each neuron in the net as a signal m_i. The different functioning modes are described below:

If $m_i = 1$ the neuron is assumed to be associated to a propositional symbol, and its next state is the maximum value among all its input, its previous state, and the initial confidence values v_i. More precisely:

$$\textbf{Case } p,\ m_i = 1\text{: } I_i(t+1) = \max\left\{\max_{k|w_{ik}>0}\{I_k(t)\}, I_i(t), v_i\right\}$$

When a neuron is associated to the product, Gödel, or Łukasiewicz implication, then the signal m_i is set to 2, 3, and 4, respectively. Its input is formed by the external value v_i of the rule, and the outputs of the neurons associated to the body of the implication. The output of the neuron somehow mimics the behaviour of the procedural semantics when a rule of type m_i has been used; specifically, the output in the next instant will be:

$$\textbf{Case } \leftarrow_P,\ m_i = 2\text{: } I_i(t+1) = \max\left\{I_i(t), v_i \cdot \prod_{k|w_{ik}>0} I_k(t)\right\}$$

$$\textbf{Case } \leftarrow_G,\ m_i = 3\text{: } I_i(t+1) = \max\left\{I_i(t), \min\left\{v_i, \min_{k|w_{ik}>0}\{I_k(t)\}\right\}\right\}$$

$$\textbf{Case } \leftarrow_L,\ m_i = 4\text{: } I_i(t+1) = \max\left\{I_i(t), v_i + \sum_{k|w_{ik}>0} I_k(t) - N_i\right\},$$

where N_i indicates the number of arguments of the body of the rule.

Case @, $m_i = 5$: the aggregators considered as weighted sums, therefore

$$I_i(t+1) = \sum_{k|w_{ik}>0} w'_{ik} \cdot I_k(t) \qquad \text{where} \qquad w'_{ik} = \frac{w_{ik}}{\sum_{r|w_{ir}>0} w_{ir}}$$

Finally, neurons associated to the adjoint conjunctors have signals $m_i = 6, 7, 8$, for product, Gödel, or Łukasiewicz conjunctions, respectively. Its output is:

$$\textbf{Case } \&_P,\ m_i = 6\text{: } I_i(t+1) = \prod_{k|w_{ik}>0} I_k(t)$$

$$\textbf{Case } \&_G,\ m_i = 7\text{: } I_i(t+1) = \min_{k|w_{ik}>0} I_k(t)$$

$$\textbf{Case } \&_L,\ m_i = 8\text{: } I_i(t+1) = \max\left\{0, \sum_{k|w_{ik}>0} I_k(t) - N_i + 1\right\}$$

Note that the output of the neurons is never decreasing.

By means of an external reset signal r, common to all the neurons, one can modify both the values of the internal registers of the neurons and their state vector $\boldsymbol{I}(t)$.

– $r = 1$. The initial truth-value v_i, the type of formula m_i, and the i-th row of the matrix of weights $\boldsymbol{w_{i\bullet}}$ are set in the internal registers. This allows to reinitialise the network for working with a new problem.
– $r = 0$. The neurons evolve with the usual dynamics, and it is only affected by the state vector of the net $\boldsymbol{I}(t)$. The value m_i, set in their internal register, selects the function which is activated in the neuron. By using a delay, the output of the activated function is compared with the previous value of the neuron.

Once the corresponding values for both the registers and the initial state of the net have been loaded, the signal r is set to 0, and each neuron will only be affected by the neurons given by $\boldsymbol{I}(t)$, its state vector at step t.

A number of programs have been carried out with the implementation. Here we present two toy examples:

Example 1. Consider the program with rules

$$\begin{array}{lll} \langle h \leftarrow_G (r \&_P o), 0.9\rangle & \langle v \leftarrow_G @_{(1,2)}(o, w), 0.8\rangle & \\ \langle n \leftarrow_P r, 0.8\rangle & \langle n \leftarrow_P w, 0.9\rangle & \langle w \leftarrow_P v, 0.75\rangle \end{array}$$

and facts $\langle o, 0.2\rangle$, $\langle w, 0.2\rangle$, $\langle r, 0.5\rangle$.

As there are non-homogeneous rules, the rules of the program are transformed as follows

$$\begin{array}{lll} \langle i \leftarrow_P r \&_P o, 1\rangle & \langle j \leftarrow_P @_{(1,2)}(o, w), 1\rangle & \\ \langle h \leftarrow_G i, 0.9\rangle & \langle v \leftarrow_G j, 0.8\rangle & \langle n \leftarrow_P r, 0.8\rangle \\ \langle n \leftarrow_P w, 0.9\rangle & \langle w \leftarrow_P v, 0.75\rangle & \end{array}$$

Therefore, we will need 13 neurons (7 in the hidden layer) associated to variables h, n, o, r, v, w, i, j and to the last five rules.

The initial values of the registers are:

– The vector $\boldsymbol{v} = (0, 0, 0.2, 0.5, 0, 0.2, 1, 1, 0.9, 0.8, 0.8, 0.9, 0.75)$
– The vector $\boldsymbol{m} = (1, 1, 1, 1, 1, 1, 6, 5, 3, 3, 2, 2, 2)$
– The matrix W_1 is given in Figure 1 (left).

After five iterations, the net gets a stable state:

$$\boldsymbol{I} = (0.1, 0.4, 0.2, 0.5, 0.2, 0.2, 0.1, 0.2, 0.1, 0.2, 0.4, 0.18, 0.15)$$

with the following values for the variables: $v_h = 0.1$; $v_n = 0.4$; $v_o = 0.2$; $v_r = 0.5$; $v_v = 0.2$; $v_w = 0.2$.

Example 2. Consider the program with rules

$$\begin{array}{ll} \langle p \leftarrow_G @_{(1,2,3)}(q, r, s), 0.8\rangle & \langle q \leftarrow_P (t \&_L u), 0.6\rangle \\ \langle t \leftarrow_P (v \&_G u), 0.5\rangle & \langle v \leftarrow_P u, 0.8\rangle \end{array}$$

and facts $\langle u, 0.75\rangle$, $\langle r, 0.7\rangle$, $\langle s, 0.6\rangle$.

$$W_1 = \begin{pmatrix} \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 & 1 & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 \\ \cdot & \cdot & 1 & 1 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & 1 & \cdot & \cdot & 2 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & 1 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & 1 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & 1 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \end{pmatrix} \qquad W_2 = \begin{pmatrix} \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 \\ \cdot & 1 & 2 & 3 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & 1 & 1 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & 1 & 1 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & 1 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \end{pmatrix}$$

Fig. 1. Matrices for Examples 1 and 2.

Firstly, the rules of the program are homogenised as follows

$$\langle h \leftarrow_P @_{(1,2,3)}(q, r, s), 1\rangle \qquad \langle i \leftarrow_P t \&_L u, 1\rangle \qquad \langle j \leftarrow_P v \&_G u, 1\rangle$$
$$\langle p \leftarrow_G h, 0.8\rangle \qquad \langle q \leftarrow_P i, 0.6\rangle$$
$$\langle t \leftarrow_P j, 0.5\rangle \qquad \langle v \leftarrow_P u, 0.8\rangle$$

The net will consist of 14 neurons, to represent variables $p, q, r, s, t, u, v, h, i, j$ together with the last four (homogeneous) rules in the program.

The initial values of the net are:

- The vector $\boldsymbol{v} = (0, 0, 0.7, 0.6, 0, 0.75, 0, 1, 1, 1, 0.8, 0.6, 0.5, 0.8)$.
- The vector $\boldsymbol{m} = (1, 1, 1, 1, 1, 1, 1, 5, 8, 7, 3, 2, 2, 2)$.
- The matrix W_2 is given in Figure 1 (right).

After running the net, its state vector get stabilised at

$$\boldsymbol{I} = (0.5383, 0.03, 0.7, 0.6, 0.3, 0.75, 0.6, 0.5383, 0.05, 0.6.0.5383, 0.03, 0.3, 0.6)$$

where the first nine components correspond to the visible layer, which are interpreted as the obtained truth-value for p, q, etc.

4 Abduction in multi-adjoint logic programs

In this section we introduce the basics of abductive reasoning in a multi-adjoint context; specifically, we will define the concept of (multi-adjoint) abduction problem and correct explanation for an abduction problem. Later, we will make the necessary adaptations to the neural model to cope with abductive reasoning.

Definition 5. *An* abduction problem *is a tuple* $\mathcal{A} = \langle \mathbb{P}, OBS, H\rangle$*, where*

1. $\mathbb{P}$ *is a multi-adjoint logic program.*
2. H *is a (finite) subset of the set of propositional symbols, the set of hypotheses.*

3. $OBS\colon OV \to [0,1]$ *is the theory of observations (where* OV *is a set of observation variables such that* $OV \cap H = \varnothing$*).*

The intended meaning of $OV \cap H = \varnothing$ is that observation variables should not be explained by themselves.

Definition 6. *A theory* $E\colon H \to [0,1]$ *is a* correct explanation *to* $\langle \mathbb{P}, OBS, H\rangle$ *if*

1. $\mathbb{P} \cup E$ *is satisfiable.*
2. *Every model of* $\mathbb{P} \cup E$ *is also a model of* OBS*.*

In [7] it was given a the procedural semantics based on the $T_{\mathbb{P}}$ operator which is sound and complete and, furthermore, it was proved that the surface corresponding to all the solutions for particular observations has the shape of a convex body and the set of all solutions is the union of such surfaces.

4.1 Neural model for multi-adjoint abduction

Our main goal here is to adapt the neural model above to the abductive framework for multi-adjoint logic programming. The general approach to abduction is, given a program $\mathbb{P}$ and a set of observations, to obtain a set of explanations for these observations, as a number of *abduced* facts. In addition, we are also interested in allowing the possibility of changing the confidence values of the rules in the given program for a number of reasons; for instance, it could happen that no explanation exists simply because the confidence values of the rules have not been suitably assigned although, obviously, it might also happen that no explanation can be obtained for a given problem, for instance, in the case of badly posed problems.

Our neural model for abductive reasoning will allow to divide the set of rules as rules with 'hard' confidence value and rules with 'soft' confidence value; the former assumed to have a fixed confidence value throughout all the computation and the latter whose confidence value could be modified if necessary.

Once the parameters $\boldsymbol{v}$, $\boldsymbol{m}$ and $\mathbf{W}$ have been set in the initial registers of the net, the program can be run in order to obtain the minimal model, which may or may not explain all the observed values (loaded in a vector of observed values $\boldsymbol{ov}$). Obviously, the interesting case from an abductive point of view is when the minimal model does not explain all the observed values.

The neural model for abduction will be a modification on that given in the previous section which includes, apart from the vector of observed values $\boldsymbol{ov}$, another vector for setting the rules whose confidence values will remain unmodified $\boldsymbol{u}$. Now, our goal will be to find either an explanation based solely on the set of hypotheses or set new confidence values to rules (determined by vector $\boldsymbol{u}$) so that the observed values are attained. The search for these new confidence values $\boldsymbol{v}$ is obtained by training the net.

If there are n neurons, and we have b observed values and h rules have hard confidence values, then the net implements a function $f\colon [0,1]^{n-b-h} \to [0,1]^b$, since the b components of the observations and the h components of the hard rules will remain fixed. Therefore we can consider the space $[0,1]^{n-b-h}$ as the search space and, given $\boldsymbol{v} \in [0,1]^n$ we will denote its projection on the search space as $\boldsymbol{\pi_v} \in [0,1]^n$.

Given the observations $\boldsymbol{ov} \in [0,1]^b$, we define the **feasible region** as the set $\mathcal{F} \subset [0,1]^{n-b-h}$ such that if $\boldsymbol{\pi_v} \in \mathcal{F}$ then $\pi_{v_j} \geq ov_j$ for all $j = 1, \ldots, b$.

The function implemented by the net has the following properties:

1. It is non-decreasing in all its components.
2. If there is some correct explanation, then $f(\mathbf{1}) \in \mathcal{F}$.
3. If every interpretation is a correct explanation, then $f(\mathbf{0}) \in \mathcal{F}$.

4.2 Training the Net

Given an abduction problem, firstly, we have to check whether there is at least a model for the program and the observations. This is done by checking that the vector $\boldsymbol{\pi_v} = \mathbf{1}$, changed by including the observed values, is a point of the feasible region. If we get affirmative answer, then the effective training of the net begins, having in mind that the components of $\boldsymbol{v}$ corresponding to the observations will be fixed for all the training process, as well as $\boldsymbol{m}$ and $\mathbf{W}$.

We have chosen to randomly search for explanations, so that we have chance to obtaining a wide range of possible explanations to a given abduction problem. The training process aims at obtaining a vector of confidence values for hypotheses and soft rules such that the resulting least model (that is, the output of the function implemented by the net) is as close as possible to the frontier of the feasible region.

The training is based on an iterative procedure which begins with the initial vector $\boldsymbol{v}_0 = \boldsymbol{\pi_v}$, where $\boldsymbol{v}$ corresponds to the confidence values of rules and facts in the program $\mathbb{P}$, and zeroes assigned to variables which are not facts. Now, assume that the net gets stable at point $f(\boldsymbol{v}_0)$, and randomly take another vector $\boldsymbol{v}_1 \in [0,1]^{n-b-h}$, and assume the net stabilises at $f(\boldsymbol{v}_1)$. Then, calculate the values $0 \leq k \leq 1$, such that the point $kf(\boldsymbol{v}_0) + (1-k)f(\boldsymbol{v}_1)$ is the closest (using euclidean distance) to vector $\boldsymbol{ov}$. The new initial vector will be $\boldsymbol{v_2} = k\boldsymbol{v_0} + (1-k)\boldsymbol{v_1}$, which by convexity is in the search space.

The procedure is repeated by choosing new random vectors, until the resulting confidence values $\boldsymbol{v}_n$ are such that $f(\boldsymbol{v}_n)$ can be no longer improved, in the sense of getting closer to $\boldsymbol{ov}$. This occurs if in several trials (in a number greater than the dimension $n-b-h$ of the search space) the obtained point gets fixed. This point is checked to be in the feasible region, if affirmative the training is finished, otherwise, we will find the point in the frontier of the feasible region contained in the segment $[\boldsymbol{v}_n, \mathbf{1}]$.

As a result, after the training process, the net is able to explain the observed facts, in the sense that new confidence values are assigned to rules and facts, and possibly new facts are added to the program, obtaining a modified program $\mathbb{P}'$, so that the observations are logically implied by $\mathbb{P}'$.

4.3 Simulations for multi-adjoint abduction

A number of problems have been carried out with the resulting implementation. Here we present some toy examples:

Example 3. Consider the program with rules

$$\langle p \leftarrow_P (q \&_P r), 0.8\rangle \qquad \text{and} \qquad \langle r \leftarrow_G s, 0.7\rangle$$

and the observation $\langle p, 0.7\rangle$.

By assigning neurons with the variables p, q, r, s and with the two rules, the initial registers will be $\boldsymbol{v} = (0, 0, 0, 0, 0.8, 0.7)$, $\boldsymbol{m} = (1, 1, 1, 1, 2, 3)$, the matrix W whose entries are all zeroes but $w_{15}, w_{36}, w_{52}, w_{53}$ and w_{64} which are 1, and the observed value $p = 0.7$.

After training the net, without considering any hard rule, we get the new vector of confidence values $\boldsymbol{v} = (0, 0.8599, 0.9024, 0.9268, 0.8783, 0.9641)$, which gives the new program with rules

$$\langle p \leftarrow_P (q \&_P r), 0.8783\rangle \quad \text{and} \quad \langle r \leftarrow_G s, 0.9641\rangle$$

and facts $\langle q, 0.8599\rangle$, $\langle r, 0.9268\rangle$,$\langle s, 0.9268\rangle$.

Example 4. Consider the following program

$$hi\_fuel\_comp \overset{0.8}{\leftarrow}_G @_{(2,1)}(ri\_mix, lo\_oil) \tag{1}$$

$$overheating \overset{0.5}{\leftarrow}_P lo\_oil \tag{2}$$

$$overheating \overset{0.9}{\leftarrow}_L lo\_water \tag{3}$$

This program is intended to represent some kind of knowledge about the behaviour of a car.[1] Let us assume that we have two observed facts, namely

$$\langle hi\_fuel\_comp, 0.75\rangle \qquad \langle overheating, 0.5\rangle$$

The vector of observed values is $\boldsymbol{ov} = (0.75, 0.5)$

We have trained the net twice: the first one considering no hard rule, and the second one considering no soft rule.

The non-homogeneous rule has been separated by introducing a hidden neuron implementing its body The obtained results in either case are the following:

1. No hard rules: The obtained explanation, regarding the hypotheses, is $ri\_mix = 0.853$, $lo\_oil = 0.5656$, $lo\_water = 0.6214$, and the updated confidence values for the rules are (1)= 0.75, (2)= 0.9519 and (3)=0.8837.
   The values above give the following results to the observed variables is $hi\_fuel\_comp = 0.75$ and $overheating = 0.5384$.
2. No soft rules: The obtained explanation is $ri\_mix = 0.8335$, $lo\_oil = 0.5864$, and $lo\_water = 0.6$.
   The values above give the following results to the observed variables is $hi\_fuel\_comp = 0.7511$ and $overheating = 0.5$.

[1] We do not intend that these relationships correspond to an actual case.

5 Conclusions and Related Work

A neural model has been introduced, which implements the procedural semantics recently given to multi-adjoint logic programming, in addition, it has been adapted to model abductive reasoning. As a result, it is possible to adjust the confidence values of the rules and facts of a given program which is supposed to explain a set of given observations. An advantage of the use of multi-adjoint programs is that one has a uniform computational model for a number of fuzzy rules and, thus, the implementation can be easily modified to add new connectives.

Some authors have addressed similar problems as those stated here; for instance [4] introduces a neural implementation of the fixed point of the $T_{\mathbb{P}}$ operator but only for classical logic. On the other hand [1, 2] introduces some neural approaches to the simulation of abductive reasoning, in which the complexity of the relationships between causes and effects is reproduced at the neural level; in our case, this complexity is dealt with by using the great expressive power of multi-adjoint logic, and the neural approach is used to provide a massively parallel computational model.

As future work, we will have to study different training strategies for the net in order to minimise its complexity and improving the approximation to the observed values, in order to apply some criteria for selecting the "best" explanation are used, such as the *parsimony covering* (the best explanation should include a minimal set of causes) or *maximal plausibility* (the best explanation must be the most likely wrt a given belief function).

Acknowledgements. We thank P. Eklund for providing interesting comments on previous versions of this work.

References

1. B E. Ayeb and S. Wang. Abduction based diagnosis: A competition based neural model simulating abductive reasoning. *J. of Parall. and Distributed Computing*, 24(2):202–212, 1995.
2. B.E. Ayeb, S. Wang, and J. Ge. A unified model for neural based abduction. *IEEE Transactions on Systems, Man and Cybernetics*, 28(4):408–425, 1998.
3. P. Eklund and F. Klawonn. Neural fuzzy logic programming. *IEEE Trans. on Neural Networks*, 3(5):815–818, 1992.
4. 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.
5. E.P. Klement, R. Mesiar, and E. Pap. *Triangular norms.* Kluwer academic, 2000.
6. J. Medina, E. Mérida-Casermeiro, and M. Ojeda-Aciego. Multi-adjoint logic programming: a neural net approach. In *Logic Programming. ICLP'02*. Lect. Notes in Computer Science, 2002. To appear.
7. J. Medina, M. Ojeda-Aciego, and P. Vojtáš. A multi-adjoint logic approach to abductive reasoning. In *Logic Programming, ICLP'01*, pages 269–283. Lect. Notes in Computer Science 2237, 2001.
8. J. Medina, M. Ojeda-Aciego, and P. Vojtáš. A procedural semantics for multi-adjoint logic programming. In *Progress in Artificial Intelligence, EPIA'01*, pages 290–297. Lect. Notes in Artificial Intelligence 2258, 2001.
9. E. Mérida-Casermeiro, G. Galán-Marín, and J. Muñoz Pérez. An efficient multivalued Hopfield network for the traveling salesman problem. *Neural Processing Letters*, 14:203–216, 2001.