Modelling Derivation in Defeasible Logic Programming with Perceptron-based Neural Networks[†]

Sergio Alejandro Gómez[‡]

Laboratorio de Investigación y Desarrollo en Inteligencia Artificial (LIDIA)[*]
Depto. de Ciencias e Ingeniería de la Computación — Universidad Nacional del Sur
Av. Alem 1253 – B8000CPB Bahía Blanca – Argentina
Tel/Fax: (+54) (291) 459 5135/5136 – Email: sag@cs.uns.edu.ar

Abstract

A solution of problems in multiagent systems involves representing beliefs of agents immersed in dynamic environments. *Observation-based Defeasible Logic Programming* (ODeLP) is an argument-based logic programming language that is used to represent an agent's knowledge in the context of a multiagent system. The beliefs of the agent depends on a warrant procedure performed on its knowledge base contents. New perceptions result in changes in the agent's beliefs. In the context of real time constraints, this belief change procedure should be done efficiently.

This paper introduces an algorithm for translating an agent's knowledge base, expressed as an ODeLP rule base, into a Perceptron-based neural network. Observations in an ODeLP program can then be codified as an input pattern. The input pattern is then fed to the neural network whose propagation results in an output pattern. This output pattern contains information regarding which beliefs can be hold by the agent as well as if there exists contradiction among them. The proposal is attractive as the massivelly parallel processing intrinsic to neural networks make them appropiate for implementing parts of the aforementioned warrant procedure.

Keywords: Artificial Intelligence, Defeasible Argumentation, Observation-based Defeasible Logic Programming, Perceptron, Neural Networks.

1 Introduction

Defeasible Logic Programming (DeLP) [GS04] provides a language for knowledge and reasoning that uses *defeasible argumentation* [SL92, CRL00, CML00, PV99] to decide between contradictory *defeasible conclusions*. *Observation-based Defeasible Logic Programming* (ODeLP) [CCS04, Cap03] is a framework that aims at representing the knowledge of a single agent in a dynamic environment, where agents must perceive the changes in the world and integrate them into its existing beliefs. ODeLP solves this problem by modelling perception as new facts to be added to the agent's knowledge base.

The *Neural Networks* (NN) [FS93, RR95, Ska96, Was89] approach to knowledge representation and problem solving weakly resembles the inner workings of the animal brain. Problem solving is made in NN by feeding a network with an *input pattern* that represents a problem instance, the pattern is then propagated through the network to produce an *output pattern* representing the solution to the problem instance. The *Threshold Logical Unit* (TLU) [MP43]

[†]Sent to *Workshop de agentes y sistemas inteligentes*

[‡]Partially supported by the Agencia Nacional de Promoción Científica y Tecnológica (PICT 2002 Nro. 13096) and Secretaría de Ciencia y Tecnología of Universidad Nacional del Sur (24/N016).

[*]LIDIA is a member of IICyTI Instituto de Investigación en Ciencia y Tecnología Informática.

is an implementation of an *artificial neuron* that can be arranged into *Perceptron-based* neural network [FS93, RR95, Ska96, Was89].

As pointed out in [CCS04], real time issues play an important role when modelling agent interaction. In an argument-based multi-agent setting, a timely interaction is particularly hard to achieve, as the inference process involved is complex and computationally expensive. In particular, *dialectical databases* are proposed in [CCS04] to solve this problem by storing precompiled knowledge. In this paper, we propose an algorithm for translating a ODeLP program into a neural network. Observations can then be regarded as an input pattern fed to the obtained neural network, which will act as a black-box capable of producing an output pattern indicating if exists both an argument supporting a certain conclusion and conflict among the conclusions obtained.

The rest of the paper is organized as follows. In Section 2, we briefly describe ODeLP. Then, we describe the Perceptron-based NN approach to problem solving in Section 3. In Section 4, we describe a method for obtaining a Perceptron-based NN from a propositional ODeLP program. Section 5 presents some worked examples. Next in Section 6, we compare this work with those found in the literature. We conclude the paper in Section 7.

2 Observation-based Defeasible Logic Programming

Defeasible logic programming (DeLP) [GS04] is a particular formalization of defeasible argumentation [CML00, PV99] based on logic programming. Although DeLP can be used to model the behaviour of a single agent in a *static* environment, it lacks the appropiate mechanisms to represent knowledge in *dynamic* environments, where agents must be able to perceive the changes in the world and integrate them into its existing beliefs. The *Observation-based Defeasible Logic Programming* (ODeLP) framework [CCS04, Cap03] aims at solving this problem by modelling perception as new facts to be added to the agent's knowledge base.

The language of ODeLP is based on the language of logic programming. Literals are atoms that may be preceded by the symbol "$\sim$" denoting *strong negation*. ODeLP programs are formed by *observations* and *defeasible rules*. Observations correspond to facts in the context of logic programming, and represent the knowledge an agent has about the world; more formally, an *observation* is a grounded literal L representing some fact about the world, obtained through a perception mechanism that the agent believes to be correct. Defeasible rules provide a way of performing tentative reasoning [SL92]; formally, a *defeasible rule* R has the form $L_0 -\prec L_1, L_2, \ldots, L_n$, where L_0 is a literal and $L_1, L_2, \ldots, L_n$ is a non-empty finite set of literals. We are going to define $Head(R) = L_0$ and $Body(R) = \{L_1, L_2, \ldots, L_n\}$. An *ODeLP program* is a pair $\langle \Psi, \Delta \rangle$, where Ψ is a finite set of observations and Δ is a non-empty finite set of literals. In a program $\mathcal{P}$, the set Ψ must be *non-contradictory* (*i.e.*, it is not the case that $Q \in \Psi$ and $\sim Q \in \Psi$, for any literal Q.) A *propositional* ODeLP program is an ODeLP program where all predicates have arity 0.

Given a ODeLP program $\mathcal{P} = \langle \Psi, \Delta \rangle$ and a literal L, a defeasible derivation of L from $\mathcal{P}$, denoted $\mathcal{P} \vdash L$, consists of a finite sequence $L_1, L_2, \ldots, L_n = L$ of literals, and each literal is in the sequence because: (a) L_i is a fact in Ψ, or (b) there exists a rule R_i in $\mathcal{P}$ with head L_i and body $B_1, B_2, \ldots, B_k$ and every literal of the body is an element L_j of the sequence appearing before L_i ($j < i$). A set of rules is *contradictory* if and only if, there exists a defeasible derivation for a pair of complementary literals from this set. It must be observed that: (i) defeasible derivation is *monotonic*, i.e., if H has a defeasible derivation from $\mathcal{P}$ then H will also have a defeasible derivation from $\mathcal{P} \cup R$, where R is an arbitrary set of program rules, and (ii)

if a program $\mathcal{P}$ has no facts, then no defeasible derivation can be obtained.

Deriving literals in ODeLP results in the construction of *arguments*. An argument $\mathcal{A}$ is a (possibly empty) set of ground defeasible rules that together with the set Ψ provide a logical proof for a given literal H, satisfying the additional requirements of *non-contradiction* and *minimality*. Formally, Given an ODeLP program $\mathcal{P}$, an *argument* $\mathcal{A}$ for a ground Q, denoted $\langle \mathcal{A}, Q \rangle$, is a subset of ground defeasible rules in $\mathcal{P}$, such that (i) there exists a *defeasible derivation* for Q from $\Psi \cup \mathcal{A}$, (ii) $\Psi \cup \mathcal{A}$ is non-contradictory, and, (iii) $\mathcal{A}$ is minimal with respect to set inclusion. An argument $\langle \mathcal{A}_1, Q_1 \rangle$ is a *sub-argument* of another argument $\langle \mathcal{A}_2, Q_2 \rangle$ if $\mathcal{A}_1 \subseteq \mathcal{A}_2$. Given a DeLP program $\mathcal{P}$, we will write $Args(\mathcal{P})$ denotes the set of all possible arguments that can be derived from $\mathcal{P}$.

The notion of defeasible derivation corresponds to the usual query-driven SLD derivation used in logic programming, performed by backward chaining on both strict and defeasible rules; in this context a negated literal $\sim P$ is treated just as a new predicate name no_P. Minimality imposes a kind of 'Occam's razor principle' [SL92] on argument construction: any superset $\mathcal{A}'$ of $\mathcal{A}$ can be proven to be 'weaker' than $\mathcal{A}$ itself, as the former relies on more defeasible information.

Given two arguments $\mathcal{A}$ and $\mathcal{B}$, *conflict* (or *attack*) among arguments arises whenever $\mathcal{A}$ and $\mathcal{B}$ cannot be simultaneously accepted. Many argument systems provide a preference criterion which defines a partial order among arguments, allowing to determine when $\mathcal{A}$ defeats $\mathcal{B}$. *Specificity* [SL92] is typically used as a syntax-based criterion among conflicting arguments, preferring those arguments which are *more informed* or *more direct* [SL92, SGCS03]. However, other alternative preference criteria could also be used.

In order to determine whether a given argument $\mathcal{A}$ is ultimately undefeated (or *warranted*), a dialectical process is recursively carried out, where defeaters for $\mathcal{A}$, defeaters for these defeaters, and so on, are taken into account. We refer the interested reader to [GS04, CCS04].

3 Perceptron-Based Neural Networks

A *Neural Network* is an interconnected assembly of simple processing elements, units or nodes, whose functionality is loosely based on the animal neuron. The processing ability of the network is stored in the inter-unit connection strengths, or weights, that can be obtained by a process of adaptation to, or learning from, a set of training patterns [Gur99].

The functionality of the animal neuron is captured by the artifficial neuron known as the *Threshold Logic Unit* (TLU) originally proposed by McCulloch and Pitts [MP43]. The information processing performed in this way may be crudely summarised as follows. Signals appear at the unit's *inputs* or *synapses*. The effect of each signal has may be approximated by multiplying the signal by some number or *weight* to indicate the strength of the synapse. The weighted signals are now summed to produce an overall unit *activation*. If this activation exceeds a certain *threshold*, the unit produces a an *output* response.

Formally, we suppose there are n inputs with signals $x_1, x_2, \ldots, x_n$ and weights $w_1, w_2, \ldots, w_n$. The signals take on the values '1' or '0' only; that is, the signals are binary or Boolean valued. The activation a, is given by:

$$a = \sum_{i=1}^{n} w_i x_i. \tag{1}$$

The output y is then given by thresholding the activation by a real value θ:

$$y = \begin{cases} 1 & \text{if } a \geq \theta \\ 0 & \text{if } a < \theta \end{cases} \tag{2}$$

The threshold function is sometimes called a *step-function* or *hard-limiter* to push the analogy with real neurons where the presence of an action-potential is denoted by binary '1' and its absence by binary '0'.

It is possible to interpret the functionality of a TLU geometrically. In summary, it separates its input space into two parts divided by a hyperplane according to whether the input is classiffied as a '1' or a '0'. The TLU may be thought of as classifying its input patterns into two classes: those that give output '1' and those that give output '0' [RR95, Was89].

4 Modelling Arguments with Neural Networks

In this section we will describe the main aspects concerning how to model propositional ODeLP programs in terms of perceptron-based neural networks. First, in section 4.1 we will consider how to represent propositional defeasible inference rules with TLU neurons and how to arrange those neurons to represent both propositional defeasible knowledge bases and logical contradiction among them. In section 4.2, we will explain how to model derivation, arguments, and conflict among arguments from a set of observations with respect to a defeasible knowledge base using a neural network.

4.1 Modelling Rules and Observations with Neural Networks

In this section we show how defeasible rules, observations, and a propositional ODeLP program can be modelled with a neural network. We address the issue of representing rules first.

Defeasible rules in an ODeLP program can be modelled as Thresholding Logical Units in a perceptron-based neural network. As we already said earlier, in ODeLP a *defeasible rule* is an ordered pair, denoted $Head \prec Body$, whose first member, $Head$ is a literal, and whose second member, $Body$ is a finite non-empty set literals. A defeasible rule with head L_0 and body $\{L_1, L_2, \ldots, L_n\}$ can also be written as: $L_0 \prec L_1, L_2, \ldots, L_n \ \ (n > 0)$.

In our setting, each individual rule is represented as a perceptron unit for the head, i.e. L_0, with input connections for each L_i with weight equal to 1. Besides, the threshold function $\psi(x)$ for the unit is the step function with threshold θ equal to n.

$$\psi(x) = \begin{cases} 1 & \text{if } x \geq n \\ 0 & \text{if } x < n \end{cases} \tag{3}$$

The goal of the threshold function is to allow the unit to implement the logical-AND function. The rationale is that the unit will fire a one if every predecessor is 1, that is every atom in the body holds; otherwise it will fire a zero, meaning that at least one atom in the body does not hold. The situation is depicted in Figure 1.

Now we address the issue of representing facts in a propositional ODeLP program. Facts in an ODeLP program can be modelled as inputs to a neural network. As we mentioned in Section 2, a *fact* is a literal, *i.e.* an atom, or a negated atom. In our setting, a literal will be represented as a fan-in unit in a neural network. Let P be a literal, if P is known to hold then the input associated to it will be 1; otherwise, it will be 0.

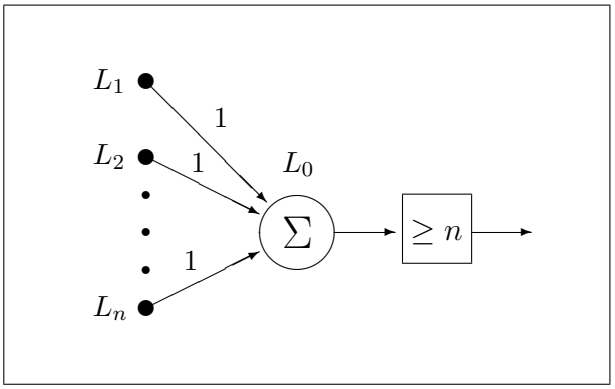

Figure 1: Generic defeasible logic rule as a TLU

We now address the issue of how a propositional ODeLP program $\mathcal{P} = \langle \Psi, \Delta \rangle$ can be modelled by a perceptron-based neural network. In this paper, we are going to restrict the defeasible rule base Δ to sets where each atom is defined by at most one rule. As a consequence of this supposition, given an (positive or negative) atom P in the language of $\mathcal{P}$, if there exists a rule R belonging to Δ such that $P = Head(R)$ then P cannot belong to Ψ. The rationale is that if both P belonged to Ψ and was the head of a rule R, then P would be defined by at least two rules, contradicting our previous suppossition.

Next, we present an algorithm for translating a defeasible logic program $\mathcal{P}$ into a multi-layer perceptron-based neural network $\mathcal{N}$. The neural network $\mathcal{N}$ is depicted as a weighted directed graph $\mathcal{N} = (V, E)$, where V stands for the set of vertices and E for the set of edges. Moreover, weighted graph edges will be denoted as $i \xrightarrow{w} j$, where i is the origin vertex, j the destination vertex, and w the weight associated to the edge, respectively.

The proposed algorithm is called *GenNeuralNet* and is presented in Figure 2. The input to *GenNeuralNet* is a propositional defeasible rule base while the output is a neural network represented as a directed graph. The algorithm first initializes the neural network as empty and then performs three loops. The first loop adds a TLU for every rule in Δ. The second loop adds fan-out connections for every possible defeasible conclusion of the program; besides, it both adds TLUs for every pair of complementary literals P and $\sim P$ (called $\perp_P$) and its corresponding fan-out connection. The third loop determines which are the possible inputs for the neural network — all literals not derived by any rule are in this category; furthermore, as a contradiction between a defeasible conclusion and a certain input can arise, the corresponding TLU with its respective fan-out connection is added.

The input and output pattern vectors are binary valued. An input or output equal to one means that the literal is known to hold. In addition, we assume that the features in both the input and output pattern are ordered lexicographically.

The input layer requires some explanation. As we already mentioned, we only consider programs where literals are defined by at most one rule. As a consequence, if a literal is defined by a rule, then it cannot be an input in Ψ. Because all inputs belong to Ψ, they will hold strictly and will be considered as observations in the resulting ODeLP program $\mathcal{P} = \langle \Psi, \Delta \rangle$.

Furthermore, the algorithm does not produce units for testing consistency in the set Ψ of observations, as it is assumed consistent [CCS04][1]. On the other hand, as we already said

[1] As the set of observations can be regarded as a set of facts, this stance is also adopted in other argumentative systems as well [CRL00, CML00, PV99].

contradiction between an input from Ψ and the conclusion of a rule of Δ is tested as it can be cause of disagreement.

The output layer consists of binary features for every posible literal (positive or negative) that could be derived by the program as well as indicators of contradictions. As consequences of the program are derived by defeasible rules, they will hold defeasibly. One exception is given by input facts which have output connections but hold strictly.

4.2 Modelling Derivation and Arguments with Neural Networks

Next we show how derivation of literals from a propositional logic program can be represented using our proposal as the propagation of patterns through a neural network. Given a propositional ODeLP program $\mathcal{P} = (\Psi, \Delta)$, it can be modelled executing the algorithm *GenNeuralNet* on Δ and obtaining a neural network $\mathcal{N}$. The neural network thus obtained $\mathcal{N}$ can be regarded as a partial function $\mathcal{N} : 2^{language(\mathcal{P})} \mapsto 2^{language(\mathcal{P})} \cup \{\bot_p : p \in language(\mathcal{P})\}$, i.e., it takes a set of literals from $\mathcal{P}$ and returns another one or an indication of contradiction associated to a certain literal. Then a set S of literals can be calculated from $\mathcal{N}$, where $S = \mathcal{N}(\Psi)$ is the set of literals that can be derived from $\mathcal{P}$ when Ψ is known to be true.

Then, the literals belonging to the language of $\mathcal{P}$ except those that are the head of some rule in Δ can be arranged into a pattern vector. An entry of this pattern vector will be one if the corresponding literal is known to hold; otherwise, the entry will be zero indicating that it is not known if the literal holds. This pattern vector will represent a set Ψ of observations that can then be propagated through the neural network to obtain another output pattern vector indicating which literals can be derived from the program and whether exist or not contradiction among them.

Moreover, both argument and conflict can be represented in this setting. Arguments can be regarded as subsets of a neural network. Conflict appears when opposite claims have to be accepted; in this setting, conflict is represented as the positive output associated to the $\bot_P$ for a literal P.

It must be remarked that our proposal satisfies two observations made by García and Simari [GS04, Obs. 2.1 and 2.2]. Observation [GS04, Obs. 2.1] says that defeasible derivation is *monotonic*, i.e., if H has a defeasible derivation from $\mathcal{P}$ then H will also have a defeasible derivation from $\mathcal{P} \cup R$, where R is an arbitrary set of program rules. In our setting, by adding rules to a defeasible program, the application of algorithm *GenNeuralNet* will result in the extension of an existing neural network, thus not invalidating previous propagation of patterns. Observation [GS04, Obs. 2.2] says that if a program $\mathcal{P}$ has no facts, then no defeasible derivation can be obtained. In our setting, that is equivalent to propagating the input vector $(0, \ldots, 0)$ through the network resulting in the output vector $(0, \ldots, 0)$ meaning that no literal could be derived from a program with no observations.

Moreover, it must be noted that, as it is customary in both DeLP [GS04] and ODeLP [CCS04], we have supposed that the set of facts (observations, resp.) is consistent. However, the *GenNeuralNet* algorithm can be straightforwadly tuned to detect inconsistencies in it by adding the corresponding $\bot_Q$ unit for literals Q and $\sim Q$.

Algorithm GenNeuralNet(Δ : DefeasibleRuleBase; **var** $\mathcal{N} = (V, E)$: NeuralNet)
var rule_for_p, rule_for_neg_p : boolean
begin
$V := \emptyset$; $E := \emptyset$
for each rule $R_i \in \Delta$ **do**
(* Suppose that rule $R_i \equiv P_i \prec Q_{i1}, \ldots, Q_{in_i}$. *)
$P_i := Head(R_i)$
$V := V \cup \{P_i\}$
for each atom $Q_{ij} \in Body(R_i)$ **do**
$V := V \cup \{Q_{ij}\}$
$E := E \cup \{Q_{ij} \xrightarrow{1} P_i\}$
end for
Set $\psi(x) =$ **if** $x \geq n_i$ **then** 1 **else** 0 **fi** as the threshold function for unit P_i in $\mathcal{N}$
end for

for each literal $P_i \in language(\mathcal{P}) \cap V$ **do**
if $\{P_i, \sim P_i\} \subseteq V$ **then** AddDisagreementUnit(P_i, $\mathcal{N}$)
elsif $P_i \in V$ **then**
Add a fan-out unit and connection for P_i to $\mathcal{N}$
elsif $\sim P_i \in V$ **then**
Add a fan-out unit and connection for $\sim P_i$ to $\mathcal{N}$
end if
end for

for each literal $P_i \in language(\mathcal{P})$ **do**
rule_for_p := $\exists R \in \Delta : P_i = Head(R)$
rule_for_neg_p := $\exists R \in \Delta : \sim P_i = Head(R)$
if not rule_for_p **then**
$V := V \cup \{P_i\}$
Add a fan-in connection for P_i to $\mathcal{N}$
if rule_for_neg_p **then** AddDisagreementUnit(P_i, $\mathcal{N}$) **end if**
end if
if not rule_for_neg_p **then**
$V := V \cup \{\sim P_i\}$
Add a fan-in connection for $\sim P_i$ to $\mathcal{N}$
if rule_for_p **then** AddDisagreementUnit(P_i, $\mathcal{N}$) **end if**
end if
end for
end

procedure AddDisagreementUnit(P_i : Literal; **var** $\mathcal{N} = (V, E)$: NeuralNet)
begin
$V := V \cup \{\bot_{P_i}\}$
$E := E \cup \{(P_i \xrightarrow{1} \bot_{P_i}), (\sim P_i \xrightarrow{1} \bot_{P_i})\}$
Add fan-out units and connections for P_i, $\sim P_i$, and $\bot_{P_i}$ to $\mathcal{N}$ if not already there
Set $\psi(x) =$ **if** $x \geq 2$ **then** 1 **else** 0 **fi** as the threshold function for unit $\bot_{P_i}$
end

Figure 2: An algorithm for translating a propositional ODeLP program $\mathcal{P}$ into a Perceptron-based neural network $\mathcal{N}$

5 Some Worked Examples

Now we present some examples of how the proposed approach works.

Example 1 *Consider the following defeasible knowledge base Δ_1 based on [GS04, Example 2.1]:*

$$\Delta_1 = \left\{ \begin{array}{l} \sim flies \prec chicken. \\ flies \prec chichen, scared. \\ nest\_in\_trees \prec flies. \end{array} \right\}$$

The defeasible rule set Δ_1 expresses that chickens usually do not fly unless they are scared. Besides, anything that flies usually nest in trees.

If we consider the set of observations $\Psi_1 = \{chicken\}$, the ODeLP program $\mathcal{P}_1 = \langle \Psi_1, \Delta_1 \rangle$ is obtained. From $\mathcal{P}_1$ the argument $\langle \mathcal{A}_1, \sim flies \rangle$ can be derived, where $\mathcal{A}_1 = \{\sim flies \prec chicken\}$.

Analogously, if we consider the set of observations $\Psi_1' = \{chicken, scared\}$, the ODeLP $\mathcal{P}_1' = \langle \Psi_1', \Delta_1 \rangle$ is now obtained. From $\mathcal{P}_1'$, besides $\langle \mathcal{A}_1, \sim flies \rangle$, arguments $\langle \mathcal{B}_1, flies \rangle$ and $\langle \mathcal{C}_1, nest\_in\_trees \rangle$ can be also derived where

- $\mathcal{B}_1 = \{\sim flies \prec chicken\}$
- $\mathcal{C}_1 = \{(flies \prec chicken, scared); (nest\_in\_trees \prec flies)\}$

The defeasible rule set Δ_1 will be represented by the neural network in Figure 3 obtained by the application of the GenNeuralNet *algorithm. Input patterns of the network obtained will be made up of the following features:*

$$(chicken, scared, \sim chicken, \sim nest\_in\_trees, \sim scared).$$

Output patterns wil be composed of the following features:

$$\begin{array}{c} (chicken, flies, nest\_in\_trees, scared, \sim chicken, \\ \sim flies, \sim nest\_in\_trees, \sim scared, \bot_{flies}, \bot_{nest\_in\_trees}) \end{array}$$

Therefore the set Ψ_1 of observations will be codified as the input pattern $(1, 0, 0, 0, 0)$. When the input pattern is fed to the neural network $\mathcal{N}_1$, it will output a pattern indicating which literals can be derived. The existence of the argument $\langle \mathcal{A}_1, \sim flies \rangle$ and the observations Ψ_1 will be represented as the output pattern $(1, 0, 0, 0, 0, 1, 0, 0, 0, 0)$. On the other hand, the set Ψ_1' of observations will be codified as the input vector $(1, 1, 0, 0, 0)$, and, when this pattern is fed to $\mathcal{N}_1$, the existence of the arguments $\langle \mathcal{A}_1, \sim flies \rangle$, $\langle \mathcal{B}_1, flies \rangle$, and $\langle \mathcal{C}_1, nest\_in\_trees \rangle$ as well as the existence of contradiction on literal $flies$ will be represented as the output pattern $(1, 1, 1, 1, 0, 1, 0, 0, 1, 0)$.

We now introduce another example that models some decision criteria in the stock market domain based on [GS04, Example 2.4].

Example 2 *Consider the defeasible rule base Δ_2:*

$$\Delta_2 = \left\{ \begin{array}{l} buy\_stock \prec good\_price. \\ \sim buy\_stock \prec good\_price, risky\_co. \\ risky\_co \prec infusion. \\ \sim risky\_co \prec infusion, strong\_partner. \end{array} \right\}$$

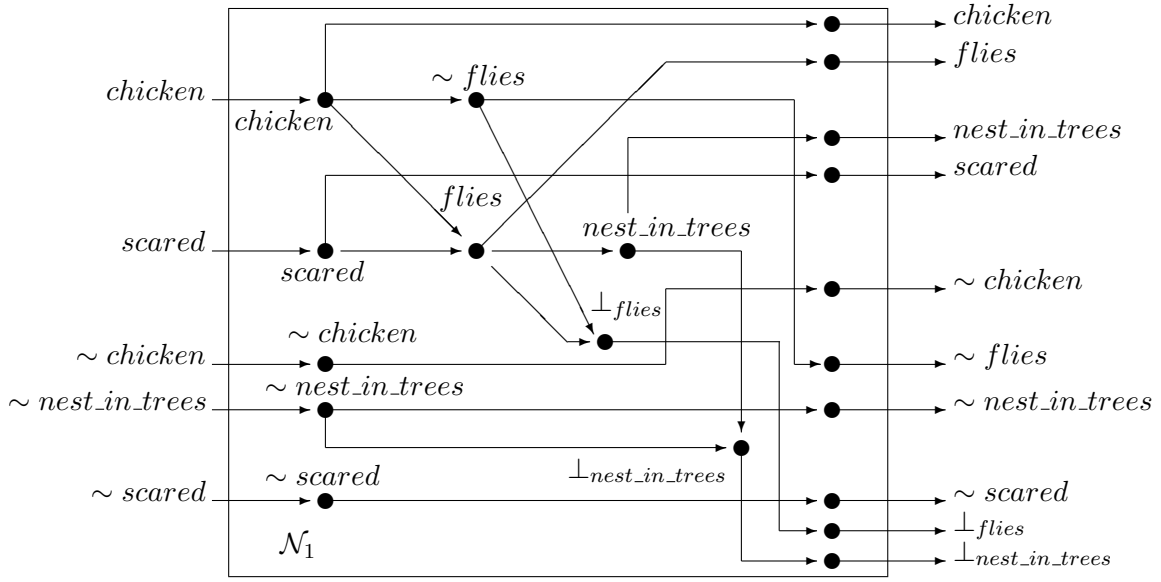

Figure 3: Perceptron based neural network $\mathcal{N}_1$ obtained by the application of the *GenNeuralNet* algorithm to the defeasible logic program in Example 1. The connection weights between neurons are all equal to one and are not shown.

The defeasible rule set Δ_2 expresses that is usually good to buy stock shares of a company when they have a good price. However, it is not usually good to buy them when the company is risky. Besides, a company is usually considered risky if it is undergoing a merger unless it has an strong partner. This base will be represented by the neural network in Figure 4 obtained by the application of GenNeuralNet *algorithm.*

If we consider the set of observations $\Psi_2 = \{good\_price, infusion\}$, the ODeLP $\mathcal{P}_2 = \langle \Psi_2, \Delta_2 \rangle$ is obtained. Then the arguments $\langle \mathcal{A}_2, buy\_stock \rangle$, $\langle \mathcal{B}_2, risky\_co \rangle$, $\langle \mathcal{C}_2, \sim buy\_stock \rangle$ can be derived from $\mathcal{P}_2$, where

- $\mathcal{A}_2 = \{buy\_stock \prec good\_price\}$
- $\mathcal{B}_2 = \{risky\_co \prec infusion\}$
- $\mathcal{C}_2 = \{(\sim buy\_stock \prec good\_price, risky\_co); (risky\_co \prec infusion)\}$

If we now consider the set of observations $\Psi'_2 = \{good\_price, infusion, strong\_partner\}$, the ODeLP program $\mathcal{P}'_2 = \langle \Psi'_2, \Delta_2 \rangle$ is obtained. From $\mathcal{P}'_2$, the arguments obtained from Ψ_2 can be also obtained besides the argument $\langle \mathcal{D}_2, \sim risky\_co \rangle$ where

$$\mathcal{D}_2 = \{\sim risky\_co \prec infusion, strong\_partner\}.$$

The defeasible rule set Δ_2 will be represented by the neural network in Figure 4 obtained by the application of the GenNeuralNet *algorithm. Input patterns of the network obtained will be made up of the following features*

$$(good\_price, infusion, strong\_partner, \sim good\_price, \sim infusion, \sim strong\_partner)$$

Output patterns wil be composed of the following features

$$(buy\_stock, good\_price, infusion, risky\_co, strong\_partner,$$
$$\sim buy\_stock, \sim good\_price, \sim infusion, \sim risky\_co,$$
$$\sim strong\_partner, \bot_{buy\_stock}, \bot_{risky\_co})$$

Therefore the set Ψ_2 *of observations will be codified as* $(1,1,0,0,0,0)$ *and the existence of the arguments* $\langle \mathcal{A}_2, buy\_stock\rangle$, $\langle \mathcal{B}_2, risky\_co\rangle$, $\langle \mathcal{C}_2, \sim buy\_stock\rangle$, *the observations* Ψ_2, *and the contradiction in literal buy_stock will be represented as the output pattern* $(1,1,1,1,0,1,0,0,1,0)$ *obtained from feeding the input vector to the network* $\mathcal{N}_2$. *On the other hand, the set* Ψ'_2 *of observations will be codified as* $(1,1,1,0,0,0)$ *and the existence of the arguments* $\langle \mathcal{A}_2, buy\_stock\rangle$, $\langle \mathcal{B}_2, risky\_co\rangle$, $\langle \mathcal{C}_2, \sim buy\_stock\rangle$, *and* $\langle \mathcal{D}_2, \sim risky\_co\rangle$, *along with the observations* Ψ'_2 *and the existence of contradiction on literals buy_stock and risky_co will be represented as the output pattern* $(1,1,1,1,1,1,0,0,1,1)$.

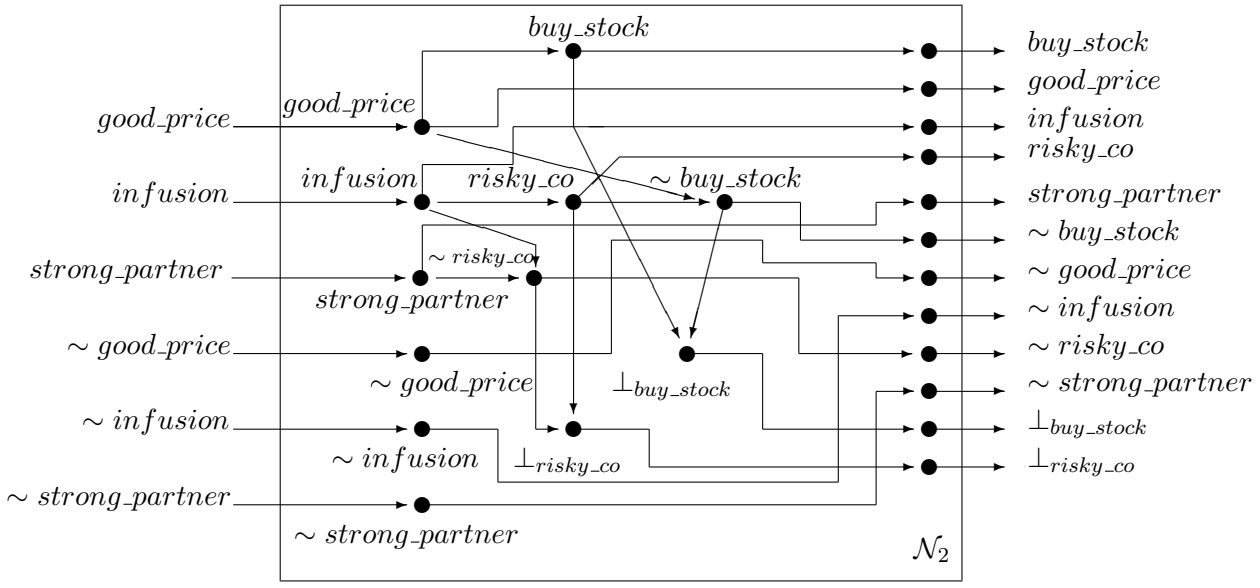

Figure 4: Perceptron based neural network $\mathcal{N}_2$ obtained by the application of the *GenNeuralNet* algorithm to the defeasible logic program in Example 2. The connection weights between neurons are all equal to one and are not shown.

6 Related Work

Defeasible argumentation is a relatively young area but already mature enough to provide solutions for other areas [CRL00, CML00, PV99]. Neural networks are also a mature area with numerous practical applications [FS93, Koh97, KKL+00, RR95, Ska96, Was89, ZSS97].

In [CCS04] Capobianco et al. present the ODeLP framework where the process of computing beliefs in changing environment is aided by integrating a "dialectical database" with an agent's program; our proposal is not aimed at replacing that database but at improving the efficiency of mentioned process. Given an ODeLP program $\mathcal{P} = \langle \Psi, \Delta\rangle$, the neural network $\mathcal{N}$ obtained from Δ and fed with Ψ can be used to determine if there exists contradiction in the agent's knowledge base. If there is none, then there is no need of performing a dialectical analysis; hence improving the efficiency of the argumentation process. Moreover, the intrinsically parallel nature of neural

networks makes them appealling for efficient implementation of parts of an agent internal state; thus speeding up the argumentation process again.

In an early work for combining neural networks and rules sets [ST89], rules are used to initialize a backpropagation neural networks weights, whereas we build a neural network from a set of defeasible rules. Balduccini [Bal02] proposes an algorithm for inducing a neural network from a propositional A-Prolog logic program; while his proposal is aimed at calculating the answer set of a logic program, our proposal is aimed at determining which literals can be derived from an ODeLP program.

In [GC03, GC04a, GC04c], Gómez and Chesñevar propose combining a set of criteria specified as a DeLP program with a Fuzzy ART neural network model for solving ambiguities in clustering problems. Also Gómez and Chesñevar propose combining a Counterpropagation neural network with a DeLP program for HTML document filtering in [GC04b]. In those works, a neural network is modelled as a DeLP program while in this work the opposite is done.

7 Conclusions and Future Work

Defeasible logic programming and neural networks are powerful knowledge representation and problem solving tools whose combination can be very fuitful. We have presented an approach for building a neural network from a set of propositional defeasible logic rules. The neural network built can be used to determine the set of possible inferences and contradictions that can be obtained from that set of rules and a set of observations provided as an input pattern.

An obvious extension to this work consists of allowing atoms being defined by more of a rule. This can be implemented by defining TLUs as described in this article and next adding another TLU implementing the logical-OR function for rules sharing the same head.

It remains as an open issue to consider the representational power of neural networks for the warrant process in argumentation systems. Further research will be needed in order to establish the feasibility of that approach.

Moreover, another extension to this work consists of extending our proposal to first-order languages. As in first-order DeLP and ODeLP arguments are only formed by ground formulas, our proposal extends naturally to cope with the latter. Nevertheless, allowing for rules with recursive definition has to be further studied.

References

[Bal02] Marcello Balduccini. A neural network-based approach for the computation of the answer set of logic programs, February 1, 2002. Slides available on the web.

[Cap03] Marcela Capobianco. *Argumentación Rebatible en Entornos Dinámicos*. PhD thesis, Universidad Nacional del Sur, Bahía Blanca, Argentina, 2003.

[CCS04] Marcela Capobianco, Carlos Iván Chesñevar, and Guillermo Simari. An argument-based framework to model an agent's beliefs in a dynamic environment. In *Proc. of First International Workshop of on Argumentation in Multi-Agent Systems (ArgMAS 2004). Third International Joint Conference on Autonomous Agentes and Multi-Agent Systems (AAMAS 2004) New York, USA*, pages 163–178, July 19-23 2004.

[CML00] Carlos Iván Chesñevar, Ana Maguitman, and Ronald Loui. Logical Models of Argument. *ACM Computing Surveys*, 32(4):337–383, December 2000.

[CRL00] Daniela Carbogim, David Robertson, and John Lee. Argument-based applications to knowledge engineering. *The Knowledge Engineering Review*, 2000.

[FS93] J. Freeman and D. Skapura. *Redes Neuronales. Algoritmos, aplicaciones y técnicas de programación.* Addison-Wesley/Díaz de Santos, 1993.

[GC03] Sergio Alejandro Gómez and Carlos Iván Chesñevar. Combining Argumentation and Clustering Techniques in Pattern Classification Problems. In *Proc. of the IX Argentinian Conference in Computer Science (CACIC 2003)*, pages 601–612, 2003.

[GC04a] Sergio Alejandro Gómez and Carlos Iván Chesñevar. A Hybrid Approach to Pattern Classification Using Neural Networks and Defeasible Argumentation. In *Procs. of the 17th International FLAIRS Conference, Palm Beach, Florida,USA*, pages 393–398, May 2004.

[GC04b] Sergio Alejandro Gómez and Carlos Iván Chesñevar. Combining Counterpropagation Neural Networks and Defeasible Logic Programming for Text Classification. In *Proc. of the VI Workshop of Researchers in Computer Science (WICC 2004)*, pages 480–484, 2004.

[GC04c] Sergio Alejandro Gómez and Carlos Iván Chesñevar. Integrating Defeasible Argumentation with Fuzzy ART Neural Networks for Pattern Classification. In *Journal of Computer Science and Technology*, volume 4(1), pages 45–51, April 2004.

[GS04] Alejandro J. García and Guillermo R. Simari. Defeasible Logic Programming: An Argumentative Approach. *Theory and Practice of Logic Programming*, 4(1):95–138, 2004.

[Gur99] Kevin Gurney. Computers and symbols versus nets and neurons, 1999.

[KKL+00] Teuvo Kohonen, Samuel Kaski, Krista Lagus, Jarkko Salojärvi, Jukka Honkela, Vesa Paatero, and Antti Saarela. Self organization of a massive document collection. *IEEE Transactions on Neural Networks, Vol. 11, No. 3, May 2000*, 2000.

[Koh97] Teuvo Kohonen. *Self-Organizing Maps. Second Edition.* Springer-Verlag Berlin Heidelberg, 1997.

[MP43] W. McCulloch and W. Pitts. A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*, (7):115–133, 1943.

[PV99] Henry Prakken and Gerard Vreeswijk. Logics for Defeasible Argumentation. In Dov Gabbay, editor, *Handbook of Philosophical Logic.* Kluwer Academic Publisher, 1999.

[RR95] V. Rao and H. Rao. *C++ Neural Networks and Fuzzy Logic, Second Edition.* MIS Press, 1995.

[SGCS03] Frieder Stolzenburg, Alejandro García, Carlos I. Chesñevar, and Guillermo R. Simari. Computing Generalized Specificity. *Journal of Non-Classical Logics*, 13(1):87–113, 2003.

[Ska96] David Skapura. *Building Neural Networks.* ACM Press, Addison-Wesley, 1996.

[SL92] Guillermo R. Simari and Ronald P. Loui. A Mathematical Treatment of Defeasible Reasoning and its Implementation. *Artificial Intelligence*, 53:125–157, 1992.

[ST89] J. Shavlik and G. Towell. An approach to combining explanation-based and neural learning algorithms. *Connection Science*, 1(3):233–255, 1989.

[Was89] Philip D. Wasserman. *Neural Computing. Theory and Practice.* Van Nostrand Reinhold, 1989.

[ZSS97] M. Zeller, R. Sharma, and K. Schulten. Motion planning of a pneumatic robot using a neural network. *IEEE Control Systems*, June 1997.