REVERSE ENGINEERING AND SYMBOLIC KNOWLEDGE EXTRACTION ON ŁUKASIEWICZ LOGICS USING NEURAL NETWORKS

Carlos Leandro
Área Cientifica da Matemática, Instituto Superior de Engenharia de Lisboa
Instituto Politécnico de Lisboa, Portugal

Keywords: Fuzzy logics, Łukasiewicz logic, Reverse Engineering, Symbolic Knowledge Extraction, Neural Networks, Link grammars, Levenderg-Marquardt algorithm, Optimal Brain Surgeon.

Abstract: This work describes a methodology that combines logic-based systems and connectionist systems. Our approach uses finite truth-valued Łukasiewicz logic, where we take advantage of fact, presented in (Castro and Trillas, 1998), wherein every connective can be defined by a neuron in an artificial network having, by activation function, the identity truncated to zero and one. This allowed the injection of formulas into a network architecture, and also simplified symbolic rule extraction. Neural networks are trained using the Levenderg-Marquardt algorithm, where we restricted the knowledge dissemination in the network structure, and the generated network is simplified applying the "Optimal Brain Surgeon" algorithm proposed by B. Hassibi, D. G. Stork and G.J. Wolf. This procedure reduces neural network plasticity without drastically damaging the learning performance, thus making the descriptive power of produced neural networks similar to the descriptive power of Łukasiewicz logic language and simplifying the translation between symbolic and connectionist structures. We used this method in the reverse engineering problem of finding the formula used on the generation of a given truth table. For real data sets the method is particularly useful for attribute selection, on binary classification problems defined using nominal attributes, where each instance has a level of uncertainty associated with it.

1 INTRODUCTION

There are essentially two representation paradigms, usually taken very differently. On one hand, symbolic-based descriptions are specified through a grammar that has fairly clear semantics, can codify structured objects, in some cases can support various forms of automated reasoning, and can be transparent to users. On the other hand, the usual way to see information presented using a connectionist description is its codification on a neural network. Artificial neural networks (NNs), in principle, combine - among other things - the ability to learn and robustness or insensitivity to perturbations of input data. NNs are usually taken as black boxes, thereby providing little insight into how the information is codified. The knowledge captured by NNs is not transparent to users and cannot be verified by domain experts.

It is natural to seek a synergy integrating the *white-box* character of symbolic base representation and the learning power of artificial neuronal networks. Such neuro-symbolic models are currently a very active area of research see (Bornscheuer et al., 1998) (Hitzler et al., 2004) (Hölldobler, 2000) (Hölldobler and Kalinke, 1994) (Hölldobler et al., 1999), for the extraction of logic programs from trained networks. The extraction of modal and temporal logic programs see (d'Avila Garcez, 2007) and (d'Avila Garcez et al., 2008), for connectionist representation of multi-valued logic programs see (Komendantskaya et al., 2007) and (Eklund and Klawonn, 1992).

Our approach to neuro-symbolic models and knowledge extraction is based on a comprehensive language for humans, representable directly in a NN topology and able to be used, like knowledge-based networks (Fu, 1993) (Towell and Shavlik, 1994), to generate the initial network architecture from crude symbolic domain knowledge. In the other direction, neural language can be translated into its symbolic language like presented in (Gallant, 1988) (Gallant, 1994) (Towell and Shavlik, 1993). However this processes has been used to identify the most significant determinants of decision or classification. This is a hard problem since, often, an artificial NN with good generalization does not necessarily imply involvement of hidden units with distinct meaning. Hence, any individual unit cannot essentially be associated with a single concept or feature of the problem domain. This the archetype of connectionist approaches, where all information is stored in a distributed manner among the processing units and their associated connectivity. However, in this work we used a propositional language wherein formulas are interpreted as NNs. In this framework formulas are simple to inject into a multilayer feed-forward network, and we are free from the need of giving interpretation to hidden units in the problem domain.

For this task we selected the propositional language of Łukasiewicz logic. This type of multi-valued logic has a very useful property motivated by the "linearity" of logic connectives. Every logic connective can be defined by a neuron in an artificial network having, by activation function, the identity truncated to zero and one (Castro and Trillas, 1998). This allows the direct codification of formulas in the network architecture, and simplifies the extraction of rules. Multilayer feed-forward NN, having this type of activation function, can be trained efficiently using the Levenderg-Marquardt algorithm (Hagan and Menhaj, 1999), and the generated network can be simplified using the "Optimal Brain Surgeon" algorithm proposed by B. Hassibi, D. G. Stork and G.J. Stork (Hassibi et al., 1993).

This strategy has good performance when applied to the reconstruction of formulas from truth tables. If the truth table is generated using a formula from the Łukasiewicz propositional logic language, the optimum solution is defined using only units directly translated into formulas. In this type of reverse engineering problem, we presuppose no noise. However, the process is stable for the introduction of Gaussian noise into the input data. This motivates its application to extract comprehensible symbolic rules from real data. However, often a model with good generalization can be described using configuration of neural units without exact symbolic presentation. We describe, in the following, a simple rule to generate symbolic approximation for un-representable configurations.

Our method has good performance for attribute selection from real data. We used it for data set simplification, removing potentially irrelevant attributes. This reduces the problem dimension, reducing the size of neuronal network to be trained.

2 PRELIMINARIES

We begin by presenting the basic notions we need from the subjects of many-valued logics, and by showing how formulas in a propositional language can be injected into and extracted from a feed-forward NN.

2.1 Łukasiewicz Logics

Classical propositional logic is one of the earliest formal systems of logic. The algebraic semantics of this logic are given by Boolean algebra. Both, the logic and the algebraic semantics have been generalized in many directions. The generalization of Boolean algebra can be based in the relationship between conjunction and implication given by

$$(x \wedge y) \leq z \Leftrightarrow x \leq (y \rightarrow z) \Leftrightarrow y \leq (x \rightarrow z). \tag{1}$$

These equivalences, called *residuation equivalences*, imply the properties of logic operators in Boolean algebras. They can be used to present implication as a generalized inverse for conjunction.

In applications of fuzzy logic, the properties of Boolean conjunction are too rigid, hence it is extended a new binary connective, $\otimes$, which is usually called fusion. Extending the commutativity to the fusion operation, the residuation equivalences define an implication denoted in this work by $\Rightarrow$:

$$(x \otimes y) \leq z \Leftrightarrow x \leq (y \Rightarrow z) \Leftrightarrow y \leq (x \Rightarrow z). \tag{2}$$

These two operators are defined in a partially ordered set of truth values, $(P, \leq)$, thereby extending the two-valued set of an Boolean algebra. This defines a *residuated poset* $(P, \otimes, \Rightarrow, \leq)$, where we interprete P as a set of truth values. This structure has been used in the definition of many types of logics. If P has more than two values, the associated logics are called a *many-valued logics*.

We focused our attention on many-valued logics having $[0,1]$ as set of truth values. In this type of logics the fusion operator $\otimes$ is known as a t-norm. In (Gerla, 2000), it is described as a binary operator defined in $[0,1]$ commutative and associative, non-decreasing in both arguments, $1 \otimes x = x$ and $0 \otimes x = 0$.

The following are example of continuous t-norms:

1. *Łukasiewicz* t-norm: $x \otimes y = \max(0, x + y - 1)$.
2. Product t-norm: $x \otimes y = xy$ usual product between real numbers.
3. Gödel t-norm: $x \otimes y = \min(x, y)$.

In (Frank, 1979), all continuous t-norms are characterized using only Łukasiewicz, Gödel and product t-norms.

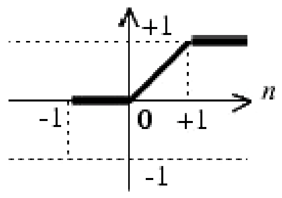


Figure 1: Saturating linear transfer function.

Many-valued logics can be conceived of as a set of formal representation languages that have proven to be useful for both real-world and computer science applications. When they are defined by continuous t-norms they are known as *fuzzy logics*. The fuzzy logic defined using *Łukasiewicz t*-norm is called Łukasiewicz logic and the corresponding propositional calculus has a nice complete axiomatization (Hájek, 1995).

2.2 Processing Units

As mentioned in (Amato et al., 2002) there is a lack of a deep investigation of the relationships between logics and NNs. In this work we present a methodology using NNs to learn formulas from data.

In (Castro and Trillas, 1998) it is shown how, by taking as activation function, ψ, the identity truncated to zero and one,

$$\psi(x)=\min(1,\max(x,0)), \tag{3}$$

it is possible to represent the corresponding NN as a combination of propositions of Łukasiewicz calculus and *viceversa* (Amato et al., 2002).

In Łukasiewicz logic sentences are usually built from a (countable) set of propositional variables, a conjunction $\otimes$ (the fusion operator), an implication $\Rightarrow$, and the truth constant 0. Further connectives are defined as follows:

1. $\neg\varphi_1$ is $\varphi_1 \Rightarrow 0$
2. $\varphi_1 \oplus \varphi_2$ is $\neg\varphi_1 \Rightarrow \varphi_2$,
3. $\varphi_1 \wedge \varphi_2$ is $\varphi_1 \otimes (\varphi_1 \Rightarrow \varphi_2)$,
4. $\varphi_1 \vee \varphi_2$ is $((\varphi_1 \Rightarrow \varphi_2) \Rightarrow \varphi_2) \wedge ((\varphi_2 \Rightarrow \varphi_1) \Rightarrow \varphi_1)$
5. $\varphi_1 \Leftrightarrow \varphi_2$ is $(\varphi_1 \Rightarrow \varphi_2) \otimes (\varphi_2 \Rightarrow \varphi_1)$
6. 1 is $0 \Rightarrow 0$

The interpretation for a well-formed formula φ is defined by assigning a truth value to each propositional variable. However, if we want to apply a NN in order to learn Łukasiewicz sentences, it seems more promising if we take a non-recursive approach to proposition evaluation. We can do this by defining the language as a set of molecular components generated from the plugging of atomic components. For this, we used the library of components presented in figure 2, interpreted as neural units and linked them together, to form NNs having only one output, without loops. These NNs are interpretation for formulas, having its structure where each neuron defines the connective identified by its label. This task of construct complex structures based on simplest ones can be formalized using generalized programming (Fiadeiro and Lopes, 1997).

In other words the language for Łukasiewicz logic is defined by the set of all NNs, wherein neurons assume one of the configurations presented in figure 2.

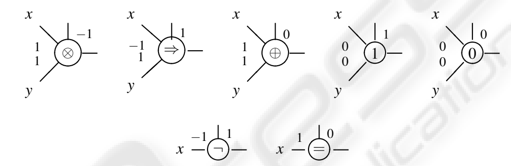


Figure 2: Neural networks codifying formulas $x \otimes y$, $x \Rightarrow y$, $x \oplus y$, 1, 0, $\neg x$ and x.

The neurons of these types of networks, which have two inputs and one output, can be interpreted as a function (see figure 3) and are generically denoted, in the following, by $\psi_b(w_1x_1, w_2x_2)$, where b represent the bias, w_1 and w_3 are the weights and, x_1 and x_2 input values. In this context a network is the *functional interpretation* of a sentence in the string-based notation when the relation, defined by network execution, corresponds to the sentence truth table.

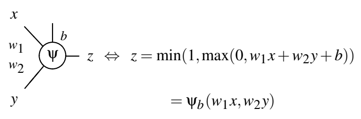


Figure 3: Functional interpretation for a neural network.

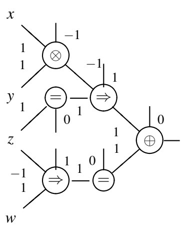
The use of NNs as interpretation of formulas simplifies the transformation between string-based representations and the network representation, allowing one to write:

Proposition 1. *Every well-formed formula in the Łukasiewicz logic language can be codified using a NN, and the network defines the formula interpretation, when the activation function is the identity truncated to zero and one.*

For instance, the semantic for sentence

$$\varphi=(x\otimes y\Rightarrow z)\oplus(z\Rightarrow w),$$

can be described using the bellow network or can be codified by the presented set of matrices. From this matrices we must note that the partial interpretation of each unit can be seen as a simple exercise of pattern checking, where we must take by reference relation between formulas and configuration described in table 1.



$$
\begin{array}{lcccc}
 & \begin{array}{cccc} x & y & z & w \end{array} & b\text{'s} & \text{partial interpretation} \\
\begin{array}{l} i_1 \\ i_2 \\ i_3 \end{array} & \left[\begin{array}{cccc} 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & -1 & 1 \end{array}\right] & \left[\begin{array}{c} -1 \\ 0 \\ 1 \end{array}\right] & \begin{array}{l} x \otimes y \\ z \\ z \Rightarrow w \end{array} \\
 & \begin{array}{ccc} i_1 & i_2 & i_3 \end{array} & & \\
\begin{array}{l} j_1 \\ j_2 \end{array} & \left[\begin{array}{ccc} -1 & 1 & 0 \\ 0 & 0 & 1 \end{array}\right] & \left[\begin{array}{c} 1 \\ 0 \end{array}\right] & \begin{array}{l} i_1 \Rightarrow i_2 \\ i_3 \end{array} \\
 & \begin{array}{cc} j_1 & j_2 \end{array} & & \\
 & \left[\begin{array}{cc} 1 & 1 \end{array}\right] & \left[\begin{array}{c} 0 \end{array}\right] & j_1 \oplus j_2
\end{array}
$$

INTERPRETATION:

$$j_1 \oplus j_2 = (i_1 \Rightarrow i_2) \oplus (i_3) = ((x \otimes y) \Rightarrow z) \oplus (z \Rightarrow w)$$

In this sense this NN can be seen as an interpretation for sentence φ; it codifies f_φ, the proposition truth table. This relationship is presented in string-base notation by writing:

$$f_\varphi(x,y,z,w)=\psi_0(\psi_0(\psi_1(-z,w)),\psi_1(\psi_0(z),-\psi_{-1}(x,y)))$$

However truth table f_φ is a continuous structure, for our goal, it must be discretized using a finite structure, ensuring sufficient information to describe the original formula. A truth table f_φ for a formula φ, in a fuzzy logic, is a map $f_\varphi : [0,1]^m \to [0,1]$, where m is the number of propositional variables used in φ. For each integer $n > 0$, let S_n be the set $\{0, \frac{1}{n}, \ldots, \frac{n-1}{n}, 1\}$. Each $n > 0$, defines a sub-table for f_φ defined by $f_\varphi^{(n)} : (S_n)^m \to [0,1]$, given by $f_\varphi^{(n)}(\bar{v}) = f_\varphi(\bar{v})$, and called the φ *(n+1)-valued truth sub-table*.

2.3 Similarity between a Configuration and a Formula

We call a *Castro neural network* (CNN) a type of NN having as activation function $\psi(x) = \min(1, max(0,x))$, where its weights are -1, 0 or 1 and having by bias an integer. A CNN is called *representable* if it is codified as a binary NN: i.e. a CNN where each neuron has one or two inputs. A network is called *un-representable* if is impossible to codify using a binary CNN. In figure 4, we present the example of an un-representable network configuration, as we will see in the following.

Note that, a binary CNN can be translated directly into Łukasiewicz logic language, using the correspondences described in table 1; in this sense, we called them *Łukasiewicz neural network* (ŁNN).

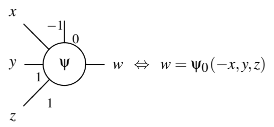
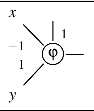
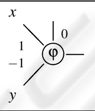
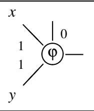
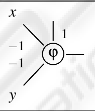
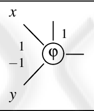
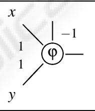
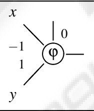
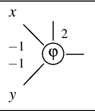


Figure 4: An un-representable neural network.

Table 1: Possible configurations for a neuron in a Łukasiewicz neural network a its interpretation.

Formula:	Configuration:	Formula:	Configuration:
$\neg x \oplus y$		$x \otimes \neg y$	
$x \oplus y$		$\neg x \otimes \neg y)$	
$x \oplus \neg y$		$x \otimes y$	
$\neg x \otimes y$		$\neg x \oplus \neg y$	

Below we present functional interpretation for formulas defined using a neuron with two inputs. These interpretation are classified as disjunctive interpretations ou conjunctive interpretations.

Disjunctive interpretations	Conjunctive interpretations
$\psi_0(x_1,x_2) = f_{x_1 \oplus x_2}$	$\psi_{-1}(x_1,x_2) = f_{x_1 \otimes x_2}$
$\psi_1(x_1,-x_2) = f_{x_1 \oplus \neg x_2}$	$\psi_0(x_1,-x_2) = f_{x_1 \otimes \neg x_2}$
$\psi_1(-x_1,x_2) = f_{\neg x_1 \oplus x_2}$	$\psi_0(-x_1,x_2) = f_{\neg x_1 \otimes x_2}$
$\psi_2(-x_1,-x_2) = f_{\neg x_1 \oplus \neg x_2}$	$\psi_1(-x_1,-x_2) = f_{\neg x_1 \otimes \neg x_2}$

These correspond to all possible configurations of neurons with two inputs. The other possible configurations are constant and can also be seen as representable configurations. For instance, $\psi_b(x_1,x_2) = 0$, if $b < -1$, and $\psi_b(-x_1,-x_2) = 1$, if $b > 1$.

In this sense, every representable network can be codified by a NN where the neural units satisfy one of the above patterns. Below we can see also examples of representable configurations for a neuron with three inputs. In the table we presente how they can be codified using representable NNs having units with two inputs, and the corresponding interpreting formula in the sting-based notation.

Conjunctive configurations
$\psi_{-2}(x_1,x_2,x_3) = \psi_{-1}(x_1,\psi_{-1}(x_2,x_3)) = f_{x_1 \otimes x_2 \otimes x_3}$
$\psi_{-1}(x_1,x_2,-x_3) = \psi_{-1}(x_1,\psi_0(x_2,-x_3)) = f_{x_1 \otimes x_2 \otimes \neg x_3}$
$\psi_0(x_1,-x_2,-x_3) = \psi_{-1}(x_1,\psi_1(-x_2,-x_3)) = f_{x_1 \otimes \neg x_2 \otimes \neg x_3}$
$\psi_1(-x_1,-x_2,-x_3) = \psi_0(-x_1,\psi_1(-x_2,-x_3)) = f_{\neg x_1 \otimes \neg x_2 \otimes \neg x_3}$

Disjunctive interpretations
$\psi_0(x_1,x_2,x_3)=\psi_0(x_1,\psi_0(x_2,x_3))=f_{x_1\oplus x_2\oplus x_3}$
$\psi_1(x_1,x_2,-x_3)=\psi_0(x_1,\psi_1(x_2,-x_3))=f_{x_1\oplus x_2\oplus\neg x_3}$
$\psi_2(x_1,-x_2,-x_3)=\psi_0(x_1,\psi_2(-x_2,-x_3))=f_{x_1\oplus\neg x_2\oplus\neg x_3}$
$\psi_3(-x_1,-x_2,-x_3)=\psi_1(-x_1,\psi_2(-x_2,-x_3))=f_{\neg x_1\oplus\neg x_2\oplus\neg x_3}$

Constant configurations like $\psi_b(x_1,x_2,x_3)=0$, if $b<-2$, and $\psi_b(-x_1,-x_2,-x_3)=1$, if $b>3$, are also representable. However there are examples of un-representable networks with three inputs like the configuration presented in figure 4.

Naturally, a neuron configuration - when representable - can by codified by different structures using a ŁNN. Particularly, we have:

Proposition 2. *If the neuron configuration $\alpha=\psi_b(x_1,x_2,\dots,x_{n-1},x_n)$ is representable, but not constant, it can be codified in a ŁNN with the following structure:*

$$\alpha=\psi_{b_1}(x_1,\psi_{b_2}(x_2,\dots,\psi_{b_{n-1}}(x_{n-1},x_n)\dots)), \tag{4}$$

where $b_1,b_2,\dots,b_{n-1}$ are integers, and $b=b_1+b_2+\dots+b_{n-1}$.

And, since the n-nary operator ψ_b is commutative, variables $x_1,x_2,\dots,x_{n-1},x_n)$ could interchange its position in function $\alpha=\psi_b(x_1,x_2,\dots,x_{n-1},x_n)$ without changing the operator output. By this we mean that, for a three input configuration, when we permutate variables, we generate equivalent configurations:

$$\psi_b(x_1,x_2,x_3)=\psi_b(x_2,x_3,x_1)=\psi_b(x_3,x_2,x_1)=\dots \tag{5}$$

When these are representa, they can be codified in string-based notation using logic connectives. But these diferente configuration only generate equivalente formulas if these formulas are disjunctive ou conjunctive formulas. A disjunctive formulas is formula written using the disjunction of propositional variables or negation of propositional variable. Similarly, a conjunctive formulas are formulas written using only the conjunction of propositional variables or its negation.

Proposition 3. *If $\alpha=\psi_b(x_1,x_2,\dots,x_{n-1},x_n)$ is representable, it is the interpretation of a disjunctive formula or a conjunctive formula.*

This leave us with the task of classifying a neuron configuration according to its representation. For that, we established a relationship using the configuration bias and the number of negative and positive weights.

Proposition 4. *(Dubois and Prade, 2000) Given the neuron configuration*

$$\alpha=\psi_b(-x_1,-x_2,\dots,-x_n,x_{n+1},\dots,x_m) \tag{6}$$

with $m=n+p$ inputs and where n and p are, respectively, the number of negative and the number of positive weights, on the neuron configuration:

1. *If $b=-p+1$ the neuron is called a* conjunction *and it is a interpretation for*

$$\neg x_1\otimes\dots\otimes\neg x_n\otimes x_{n+1}\otimes\dots\otimes x_m. \tag{7}$$

2. *When $b=n$ the neuron is called a* disjunction *and it is a interpretation of*

$$\neg x_1\oplus\dots\oplus\neg x_n\oplus x_{n+1}\oplus\dots\oplus x_m. \tag{8}$$

From the structure associated with this type of formula, we proposed the following structural characterization for representable neurons:

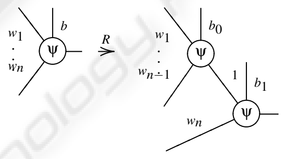
Proposition 5. *Every conjunctive or disjunctive configuration $\alpha=\psi_b(x_1,x_2,\dots,x_{n-1},x_n)$, can be codified by a ŁNN*

$$\beta=\psi_{b_1}(x_1,\psi_{b_2}(x_2,\dots,\psi_{b_{n-1}}(x_{n-1},x_n)\dots)), \tag{9}$$

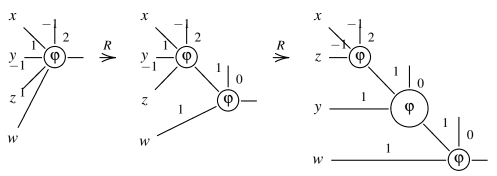
where

$$b=b_1+b_2+\dots+b_{n-1} \text{ and } b_1\leq b_2\leq\dots\leq b_{n-1}. \tag{10}$$

This property can be translated in the following neuron rewriting rule,



linking equivalent networks, when values b_0 and b_1 satisfy $b=b_0+b_1$ and $b_1\leq b_0$, and are such that neither of the involved neurons have constant output. This rewriting rule can be used to join equivalent configurations like:



Note that, a representable CNN can be transformed by the application of rule R in a set of equivalent ŁNN with simplest neuron configuration. Then we have:

Proposition 6. *Un-representable neuron configurations are those transformed by rule R in, at least, two non-equivalent NNs.*

For instance, the un-representable configuration $\psi_0(-x_1,x_2,x_3)$, presented in figure 4, is transformed by rule R in three non-equivalent configurations:

1. $\psi_0(x_3,\psi_0(-x_1,x_2))=f_{x_3\oplus(\neg x_1\otimes x_2)}$,
2. $\psi_{-1}(x_3,\psi_1(-x,x_2))=f_{x_3\otimes(\neg x_1\otimes x_2)}$, or
3. $\psi_0(-x_1,\psi_0(x_2,x_3))=f_{\neg x_1\otimes(x_2\oplus x_3)}$.

The representable configuration $\psi_2(-x_1,-x_2,x_3)$ is transformed by rule R on only two distinct but equivalent configurations:

1. $\psi_0(x_3, \psi_2(-x_1, -x_2)) = f_{x_3 \oplus \neg(x_1 \otimes x_2)}$, or
2. $\psi_1(-x_2, \psi_1(-x_1, x_3)) = f_{\neg x_2 \oplus (\neg x_1 \oplus x_3)}$

From this case we can concluded that CNNs have more expressive power than Łukasiewicz logic language. Since there are structures defined using CNNs but not codified in the Łukasiewicz logic language.

For the extraction of knowledge from trained NNs, we translate neuron configuration in propositional connectives to form formulas. However, not all neuron configurations can be translated in formulas, but they can be approximate by formulas. To quantify the approximation quality we defined the notion of interpretation λ-similar to a formula.

Two neuron configurations $\alpha = \psi_b(x_1, x_2, \ldots, x_n)$ and $\beta = \psi_{b'}(y_1, y_2, \ldots, y_n)$, are called λ-similar, in a $(m+1)$-valued Łukasiewicz logic, if λ is the exponential of mean absolute error symmetric, evaluated taking the same cases in the truth sub-table of α and β. When we have

$$\lambda = e^{-\sum_{\bar{x} \in T} \frac{|\alpha(\bar{x}) - \beta(\bar{x})|}{\sharp T}}, \tag{11}$$

write $\alpha \sim_\lambda \beta$. If α is un-representable and β is representable, the second configuration is called *a representable approximation* to the first.

On the 2-valued Łukasiewicz logic (the Boolean logic case), we have for the un-representable configuration $\alpha = \psi_0(-x_1, x_2, x_3)$:

1. $\psi_0(-x_1, x_2, x_3) \sim_{0.883} \psi_0(x_3, \psi_0(-x_1, x_2))$,
2. $\psi_0(-x_1, x_2, x_3) \sim_{0.883} \psi_{-1}(x_3, \psi_1(-x_1, x_2))$, and
3. $\psi_0(-x_1, x_2, x_3) \sim_{0.883} \psi_0(-x_1, \psi_0(x_2, x_3))$.

In this case, the truth sub-tables of, formulas $\alpha_1 = x_3 \oplus (\neg x_1 \otimes x_2)$, $\alpha_1 = x_3 \otimes (\neg x_1 \otimes x_2)$ and $\alpha_1 = \neg x_1 \otimes (x_2 \oplus x_3)$ are both λ-similar to $\psi_0(-x_1, x_2, x_3)$, where $\lambda = 0.883$, since they differ in one position on 8 possible positions. This means that both formulas are 87.5% accurate. The quality of this approximations was checked analyzing the similarity level λ on others finite Łukasiewicz logics. In every selected logic formula α_1, α_2 and α_3 had the some similarity level when compared to α:

- 3-valued logic, $\lambda = 0.8779$, 4-valued logic, $\lambda = 0.8781$,
- 5-valued logic, $\lambda = 0.8784$, 10-valued logic, $\lambda = 0.8798$,
- 20-valued logic, $\lambda = 0.8809$, 30-valued logic, $\lambda = 0.8814$,
- 50-valued logic, $\lambda = 0.8818$.

For a more complex configuration like $\alpha = \psi_0(-x_1, x_2, -x_3, x_4, -x_5)$, we can derive, using rule R, configurations:

1. $\beta_1 = \psi_0(-x_5, \psi_0(x_4, \psi_0(-x_3, \psi_0(x_2, -x_1))))$
2. $\beta_2 = \psi_{-1}(x_4, \psi_{-1}(x_2, \psi_0(-x_5, \psi_0(-x_3, -x_1))))$
3. $\beta_3 = \psi_{-1}(x_4, \psi_0(-x_5, \psi_0(x_2, \psi_1(-x_3, -x_1))))$
4. $\beta_4 = \psi_{-1}(x_4, \psi_0(x_2, \psi_0(-x_5, \psi_1(-x_3, -x_1))))$

Since these configurations are not equivalents, we concluded that α is un-representable. In this case we can see a change in the similarity level between α and each β_i when the number of truth valued is changed:

- In the 2-valued logic $\alpha \sim_{0.8556} \beta_1$, $\alpha \sim_{0.9103} \beta_2$, $\alpha \sim_{0.5189} \beta_3$ and $\alpha \sim_{0.5880} \beta_4$;
- In the 3-valued logic $\alpha \sim_{0.8746} \beta_1$, $\alpha \sim_{0.9213} \beta_2$, $\alpha \sim_{0.4829} \beta_3$ and $\alpha \sim_{0.5483} \beta_4$;
- In the 4-valued logic $\alpha \sim_{0.8860} \beta_1$, $\alpha \sim_{0.9268} \beta_2$, $\alpha \sim_{0.4667} \beta_3$ and $\alpha \sim_{0.5299} \beta_4$;
- In the 5-valued logic $\alpha \sim_{0.8940} \beta_1$, $\alpha \sim_{0.9315} \beta_2$, $\alpha \sim_{0.4579} \beta_3$ and $\alpha \sim_{0.6326} \beta_4$;
- In the 10-valued logic $\alpha \sim_{0.9085} \beta_1$, $\alpha \sim_{0.9399} \beta_2$, $\alpha \sim_{0.4418} \beta_3$ and $\alpha \sim_{0.4991} \beta_4$.

From observed similarity we selected β_2 as the best approximation to α. Its quality, as an approximation, improves when we increase the logics number of truth values. Similarity increases with the increase in the number of evaluations.

In this sense, rule R can be used for configuration classification and configuration approximation. From an un-representable configuration, α, we can generate the finite set $S(\alpha)$, with representable networks similar to α, using rule R. Given a $(n+1)$-valued logic, from that set of formulas we can select as an approximation to α; the formula having the interpretation more similar to α. This identification of un-representable configuration, using representable approximations, is used to transform networks with un-representable neurons into representable structures. The stress associated with this transformation characterizes the translation accuracy.

2.4 Neural Network Crystallization

Weights in CNNs assume the values -1 or 1. However, the usual learning algorithms process NNs weights, presupposing the continuity of the weights domain. Naturally, every NN with weighs in $[-1,1]$ can be seen as an approximation to a CNNs. The process of identifying a NN with weighs in $[-1,1]$ as a ŁNNs is called *crystallization*, and essentially consists in rounding each neural weight w_i to the nearest integer less than or equal to w_i, denoted by $\lfloor w_i \rfloor$.

In this sense the crystallization process can be seen as a pruning on the network structure, where links between neurons with weights near 0 are removed and weights near -1 or 1 are consolidated. However this process is very crispy. We need a smooth procedure to crystallize a network, in each learning iteration, to avoid the drastic reduction in learning performance. In each iteration we restricted the NN representation bias, making the network representation bias converge to a structure similar to a CNN. For that, we defined by *representation error* for a network N with weights $w_1, \ldots, w_n$, as

$$\Delta(N) = \sum_{i=1}^{n} (w_i - \lfloor w_i \rfloor). \tag{12}$$

When N is a CNNs we have $\Delta(N) = 0$. Our *smooth crystallization* process results from the iterating of function:

$$\Upsilon_n(w) = sign(w).((\cos(1 - abs(w) - \lfloor abs(w) \rfloor).\tfrac{\pi}{2})^n + \lfloor abs(w) \rfloor),$$

where $sign(w)$ is the sign of w and $abs(w)$ its absolute value. Denoting by $\Upsilon_n(N)$ the function having by input and output a NN, where the weights on the output network results of applying Υ to all the input network weights and neurons biases. Each interactive application of Υ produce a networks progressively more similar to a CNNs. Since, for every network N and $n > 0$, $\Delta(N) \geq \Delta(\Upsilon_n(N))$, we have:

Proposition 7. *Given a NNs N with weights in the interval $[0,1]$. For every $n > 0$ the function $\Upsilon_n(N)$ has, by fixed points, a CNNs.*

The convergence speed depends on parameter n. Increasing n speeds up crystallization but reduces the network's plasticity to the training data. For our applications, we selected $n = 2$ based on the learning efficiency of a set of test formulas. Greater values for n imposes stronger restrictions to learning. This procedure induces a quicker convergence to an admissible configuration of CNNs.

3 LEARNING PROPOSITIONS

We began the study of knowledge extraction using a CNN by reverse engineering a truth table. By this we mean that, for a given truth table on a $(n+1)$-valued Łukasiewicz logic, generated using a formula in the Łukasiewicz logic language, we will try to find its interpretation in the form of a ŁNN, and from it, rediscover the original formula.

For that we trained a feed-forward NN using a truth table. Our methodology trains progressively more complex networks until a crystallized network with good performance has been found. Note that convergence depends on the selected training algorithm.

The methodology is described in algorithm 1 that is used on the truth table reverse engineering task.

Algorithm 1 : Reverse Engineering algorithm.

```
1:  Given a $(n+1)$-valued truth sub-table for a Łukasiewicz logic proposition
2:  Define an inicial network complexity
3:  Generate an inicial NN
4:  Apply (the selected) Backpropagation algorithm using the data set
5:  **if** the generated network have bad performance **then**
6:      If need increase network complexity
7:      Try a new network. Go to 3
8:  **end if**
9:  Do neural network crystallization using the crisp process.
10: **if** crystalized network have bad performance **then**
11:     Try a new network. Go to 3
12: **end if**
13: Refine the crystalized network
```

Given part of a truth table we try to find a ŁNN that codifies the data. For this we generated NNs with a fixed number of hidden layers (our implementation uses three hidden layers). When the process detects bad learning performances, it aborts the training, generating a new network with random heights. After a fixed number of tries, the network topology is changed. The number of tries for each topology depends on the number of network inputs. After trying to configure a set of networks for a given complexity with bad learning performance, the system tries to apply the selected back-propagation algorithm to a more complex set of networks. In the following we presented a short description for the selected learning algorithm.

If the continuous optimization process converges, i.e. if the system finds a network codifying the data, the network is crystallized. When the error associated to this process increase, the system returns to the learning phase and tries to configure a new network.

When the process converges and the resulting network can be codified as a crisp ŁNN the system prunes the network. The goal of this phase is network simplification. For this, we selected the Optimal Brain Surgeon algorithm proposed by G.J. Wolf, B. Hassibi and D.G. Stork in (Hassibi et al., 1993).

Figure 5 presents an example of the reverse engineering algorithm input data set (a truth table in a 2-valued logic generated using 'xor') and the resulting NN output structure.

x_1	x_2	$x_1 \text{xor} x_2$
1	1	0
1	0	1
0	1	1
0	0	0

$$\Rightarrow \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix} \begin{bmatrix} 1 & 1 \end{bmatrix} \begin{bmatrix} 1 \end{bmatrix} \qquad \begin{bmatrix} 0 \\ 0 \end{bmatrix} \begin{bmatrix} 0 \end{bmatrix} \begin{bmatrix} 0 \end{bmatrix} \quad \begin{array}{l} x_1 \otimes \neg x_2 \\ \neg x_1 \otimes x_2 \\ i_1 \oplus i_3 \end{array}$$

Figure 5: Input and Output structures.

3.1 Training

Standard error back-propagation algorithm (EBP) is a gradient descent algorithm, in which the network weights are moved along the negative of the gradient of the performance function. EBP algorithm has been a significant improvement in NN research, but it has a weak convergence rate. Many efforts have been made to speed up the EBP algorithm (Bello, 1992) (Samad, 1990) (Solla et al., 1988) (Miniani and Williams, 1990) (Jacobs, 1988). The Levenderg-Marquardt algorithm (LM) (Hagan and Menhaj, 1999) (Andersen and Wilamowski, 1995) (Battiti, 1992) (Charalambous, 1992) ensued from the development of EBP algorithm-dependent methods. It gives a good exchange between the speed of the Newton algorithm and the stability of the steepest descent method (Battiti, 1992).

The basic EBP algorithm adjusts the weights in the steepest descent direction. This is the direction in which the performance function is decreasing most rapidly. In the EBP algorithm, the performance index $F(w)$ to be minimized is defined as the sum of squared erros between the target output and the network's simulated outputs. When training with the EBP method, an iteration of the algorithm defines the change of weights and has the form

$$w_{k+1}=w_k-\alpha G_k, \tag{13}$$

where G_k is the gradient of F on w_k, and α is the learning rate.

Note that the basic step of Newton's method can be derived from Taylor formula and is:

$$w_{k+1}=w_k-H_k^{-1}G_k, \tag{14}$$

where H_k is the Hessian matrix of the performance index at the current values of the weights.

Since Newton's method implicitly uses quadratic assumptions (arising from the neglect of higher-order terms in a Taylor series), the Hessian matrix dos not need be evaluated exactly. Rather, an approximation can be used, such as

$$H_k \approx J_k^T J_k, \tag{15}$$

where J_k is the Jacobian matrix that contains first derivatives of the network errors with respect to the weights w_k. The Jacobian matrix J_k can be computed through a standard back-propagation technique (Mehrotra et al., 1997) that is much less complex than computing the Hessian matrix.

The simple gradient descent and newtonian iteration are complementary in the advantages they provide. Levenberg proposed an algorithm based on this observation, whose update rule blends aforementioned algorithms and is given as

$$w_{k+1}=w_k-[J_k^T J_k+\mu I]^{-1}J_k^T e_k, \tag{16}$$

where J_k is the Jacobian matrix evaluated at w_k and μ is the learning rate. This update rule is used as follows. If the error goes down following an update, it implies that our quadratic assumption on the function is working and we reduce μ (usually by a factor of 10) to reduce the influence of gradient descent. In this way, the performance function is always reduced at each iteration of the algorithm (Hagan et al., 1996). On the other hand, if the error goes up, we would like to follow the gradient more and so μ is increased by the same factor.

The algorithm has the disadvantage that if the value of μ is large, the approximation to the Hessian matrix is not used at all. We can obtain some advantage out of the second derivative, even in such cases, by scaling each component of the gradient according to the curvature. This should result in larger movements along the direction where the gradient is smaller so the classic "error valley" problem does not occur any more. This crucial insight was provided by Marquardt. He replaced the identity matrix in the Levenberg update rule with the diagonal of Hessian matrix approximation resulting in the Levenberg-Marquardt update rule.

$$w_{k+1}=w_k-[J_k^T J_k+\mu.diag(J_k^T J_k)]^{-1}J_k^T e_k. \tag{17}$$

We changed the Levenberg-Marquardt algorithm by applying a soft crystallization step after the Levenberg-Marquardt update rule:

$$w_{k+1}=\Upsilon_2(w_k-[J_k^T J_k+\mu.diag(J_k^T J_k)]^{-1}J_k^T e_k) \tag{18}$$

This drastically improves the convergence to a CNN.

In our methodology regularization is made using three different strategies:

1. using soft crystallization, where knowledge dissemination is restricted on the network, information is concentrated on some weights;
2. using crisp crystallization where only the heavier weights survive defines the network topology;
3. pruning the resulting crystallized network.

The last regularization technic avoids redundancies, in the sense that the same or redundant information can be codified at different locations. We minimized this by selecting weights to eliminate. For this task, we used "*Optimal Brain Surgeon*" (OBS) method proposed by B. Hassibi, D. G. Stork and G.J. Stork in (Hassibi et al., 1993), which uses the criterion of minimal increase in training error. It uses information from all second-order derivatives of the error function to perform network pruning.

Our method is in no way optimal, it is just a heuristic, however works extremely well for learning CNNs.

4 REVERSE ENGINEERING

Given a ŁNN it can be translated in the form of a string base formula if every neuron is representable. Proposition 4 defines a tool to translate from the connectionist representation to a symbolic representation. It is remarkable that, when the truth table sample used in the learning was generated by a formula, the Reverse Engineering algorithm converges to a representable ŁNN equivalent to the original formula, when evaluated on the cases used in the truth table sample.

When we generate a truth table in the 4-valued Łukasiewicz logic using formula

$$(x_4 \otimes x_5 \Rightarrow x_6) \otimes (x_1 \otimes x_5 \Rightarrow x_2) \otimes (x_1 \otimes x_2 \Rightarrow x_3) \otimes (x_6 \Rightarrow x_4)$$

it has 4096 cases, the result of applying the algorithm is the 100% accurate NN:

$$\begin{array}{lll}
\begin{bmatrix} 0 & 0 & 0 & -1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & -1 \\ 1 & 1 & -1 & 0 & 0 & 0 \\ -1 & 1 & 0 & 0 & -1 & 0 \end{bmatrix} & \begin{bmatrix} 0 \\ -1 \\ -1 \\ 2 \end{bmatrix} & \begin{array}{l} \neg x_4 \otimes x_6 \\ x_4 \otimes x_5 \otimes \neg x_6 \\ x_1 \otimes x_2 \otimes \neg x_3 \\ \neg x_1 \oplus x_2 \oplus \neg x_5 \end{array} \\
\begin{bmatrix} -1 & -1 & -1 & 1 \end{bmatrix} & \begin{bmatrix} 0 \end{bmatrix} & \neg i_1 \otimes \neg i_2 \otimes \neg i_3 \otimes i_4 \\
\begin{bmatrix} 1 \end{bmatrix} & \begin{bmatrix} 0 \end{bmatrix} & j_1
\end{array}$$

Using local interpretation we may reconstruct the formula:

$$\begin{gathered}
j_1 = \neg i_1 \otimes \neg i_2 \otimes \neg i_3 \otimes i_4 = \\
\neg(\neg x_4 \otimes x_6) \otimes \neg(x_4 \otimes x_5 \otimes \neg x_6) \otimes \neg(x_1 \otimes x_2 \otimes \neg x_3) \otimes (\neg x_1 \oplus x_2 \oplus \neg x_5) = \\
= (x_4 \oplus \neg x_6) \otimes (\neg x_4 \oplus \neg x_5 \oplus x_6) \otimes (\neg x_1 \oplus \neg x_2 \oplus x_3) \otimes (\neg x_1 \oplus x_2 \oplus \neg x_5) = \\
= (x_6 \Rightarrow x_4) \otimes (x_4 \otimes x_5 \Rightarrow x_6) \otimes (x_1 \otimes x_2 \Rightarrow x_3) \otimes (x_1 \otimes x_5 \Rightarrow x2)
\end{gathered}$$

Note, however, that the restriction imposed in our implementation of three hidden layers wherein the last hidden layer has only one neuron, restrict the complexity of reconstructed formula. For instance, in order for

$$((x_4 \otimes x_5 \Rightarrow x_6) \oplus (x_1 \otimes x_5 \Rightarrow x_2)) \otimes (x_1 \otimes x_2 \Rightarrow x_3) \otimes (x_6 \Rightarrow x_4)$$

to be codified in a three hidden layer network the last layer needs two neurons one to codify the disjunction and the other to codify the conjunctions. When the algorithm was applied to the truth table generated in the 4-valued Łukasiewicz logic by using a stopping criterion a mean square error less than 0.0007 it produced the representable network:

$$\begin{array}{lll}
\begin{bmatrix} 0 & 0 & 0 & 1 & 0 & -1 \\ 1 & -1 & 0 & 1 & 1 & -1 \\ 1 & 1 & -1 & 0 & 0 & 0 \end{bmatrix} & \begin{bmatrix} 1 \\ -2 \\ -1 \end{bmatrix} & \begin{array}{l} x_4 \oplus \neg x_6 \\ x_1 \otimes \neg x_2 \otimes x_4 \otimes x_5 \otimes \neg x_6 \\ x_1 \otimes x_2 \otimes \neg x_3 \end{array} \\
\begin{bmatrix} 1 & -1 & -1 \end{bmatrix} & \begin{bmatrix} 0 \end{bmatrix} & i_1 \otimes \neg i_2 \otimes \neg i_3 \\
\begin{bmatrix} 1 \end{bmatrix} & \begin{bmatrix} 0 \end{bmatrix} & j_1
\end{array}$$

By this we may conclude what original formula can be approximate, or is λ-similar with $\lambda = 0.998$ to:

$$\begin{gathered}
j_1 = i_1 \otimes \neg i_2 \otimes \neg i_3 = (x_4 \oplus \neg x_6) \otimes \neg(x_1 \otimes \neg x_2 \otimes x_4 \otimes x_5 \otimes \neg x_6) \otimes \neg(x_1 \otimes x_2 \otimes \neg x_3) = \\
= (x_4 \oplus \neg x_6) \otimes (\neg x_1 \oplus x_2 \oplus \neg x_4 \oplus \neg x_5 \oplus x_6) \otimes (\neg x_1 \oplus \neg x_2 \oplus x_3) = \\
= (x_6 \Rightarrow x_4) \otimes ((x_1 \otimes x_4 \otimes x_5) \Rightarrow (x_2 \oplus x_6)) \otimes (x_1 \otimes x_2 \Rightarrow x_3)
\end{gathered}$$

Note that j_1 is 0.998-similar to the original formula in the 4-valued Łukasiewicz logic but it is equivalent to the original in the 2-valued Łukasiewicz logic, i.e. in Boolean logic.

The fixed number of layers also imposes restrictions to reconstruction of formula. A truth table generated by:

$$(((i_1 \otimes i_2) \oplus (i_2 \otimes i_3)) \otimes ((i_3 \otimes i_4) \oplus (i_4 \otimes i_5))) \oplus (i_5 \otimes i_6)$$

requires at least 4 hidden layers, to be reconstructed; this is the number of levels required by the associated parsing tree.

Table 2 presents the mean CPU times need to find a configuration with a mean square error of less than 0.002. The mean time is computed using 6 trials on a 5-valued truth Łukasiewicz logic for each formula. We implemented the algorithm using the MatLab NN package and executed it in an AMD Athlon 64 X2 Dual-Core Processor TK-53 at 1.70 GHz on a Windows Vista system with 959MB of memory.

Table 2: Reverse engineering test formulas.

formula	mean	stdev
$i_1 \otimes i_3 \Rightarrow i_6$	7.68	6.27
$i_4 \Rightarrow i_6 \otimes i_6 \Rightarrow i_2$	25.53	11.14
$((i_1 \Rightarrow i_4) \oplus (i_6 \Rightarrow i_2)) \otimes (i_6 \Rightarrow i_1)$	43.27	14.25
$(i_4 \otimes i_5 \Rightarrow i_6) \otimes (i_1 \otimes i_5 \Rightarrow i_2)$	51.67	483.85
$((i_4 \otimes i_5 \Rightarrow i_6) \oplus (i_1 \otimes i_5 \Rightarrow i_2)) \otimes (i_1 \otimes i_3 \Rightarrow i_2)$	268.31	190.99
$((i_4 \otimes i_5 \Rightarrow i_6) \oplus (i_1 \otimes i_5 \Rightarrow i_2)) \otimes (i_1 \otimes i_3 \Rightarrow i_2) \otimes (i_6 \Rightarrow i_4)$	410.47	235.52

In table 2 the last two formula was approximated, since we restricted the structure for NNs to three hidden layers, for others each extraction process made equivalent reconstructions.

5 REAL DATA

Extracting symbolic rules from a real data set can be a very different task than reverse-engineering the rule used on the generation of an artificial data set, in sense that, in the reverse engineering task, we know the existence of a perfect description. In particular, we know the appropriate logic language to describe it and we have no noise. The process of symbolic extraction from the real data set is made by establishing a stopping criterion and having a language bias defined by the extraction methodology. The expressive power of this language characterizes the learning algorithm plasticity. Very expressive languages produce good fitness to data, but usually bad generalization, and the extracted sentences usually are difficult to understand by human experts.

The described extraction process, when applied to real data, expresses the information using CNNs. This naturally means that the process searches for simple and understandable models for the data, able to be codify directly or approximated using Łukasiewicz logic language. The process gives preference to the simplest models and subject them to a strong pruning criteria. With this strategy we avoid overfetting and the problems associated with the algorithm complexity.

The process, however, can be prohibitive to train complex models having a great number of links. To avoid this, the rule extraction must be preceded by a phase of attribute selection.

5.1 Mushrooms

Mushroom is a data set available in the *UCI Machine Learning Repository*. This data set includes descriptions of hypothetical samples corresponding to 23

Table 3: *Mushroom* data set attribute Information.

N.	Attribute	Values
0	classes	edible=e, poisonous=p
1	cap.shape	bell=b,conical=c,convex=x,flat=f,knobbed=k, sunken=s
2	cap.surface	fibrous=f,grooves=g,scaly=y,smooth=s
3	cap.color	brown=n,buff=b,cinnamon=c,gray=g,green=r, pink=p,purple=u,red=e,white=w,yellow=y
4	bruises?	bruises=t,no=f
5	odor	almond=a,anise=l,creosote=c,fishy=y,foul=f, musty=m,none=n,pungent=p,spicy=s
6	gill.attachment	attached=a,descending=d,free=f,notched=n
7	gill.spacing	close=c,crowded=w,distant=d
8	gill.size	broad=b,narrow=n
9	gill.color	black=k,brown=n,buff=b,chocolate=h,gray=g, green=r,orange=o,pink=p,purple=u,red=e,white=w, yellow=y
10	stalk.shape	enlarging=e,tapering=t
11	stalk.root	bulbous=b,club=c,cup=u,equal=e,rhizomorphs=z, rooted=r,missing=?
12	stalk.surface.above.ring	ibrous=f,scaly=y,silky=k,smooth=s
13	stalk.surface.below.ring	ibrous=f,scaly=y,silky=k,smooth=s
14	stalk.color.above.ring	brown=n,buff=b,cinnamon=c,gray=g,orange=o, pink=p,red=e,white=w,yellow=y
15	stalk.color.below.ring	brown=n,buff=b,cinnamon=c,gray=g,orange=o, pink=p,red=e,white=w,yellow=y
16	veil.type	partial=p,universal=u
17	veil.color	brown=n,orange=o,white=w,yellow=y
18	ring.number	none=n,one=o,two=t
19	ring.type	cobwebby=c,evanescent=e,flaring=f,large=l,none=n, pendant=p,sheathing=s,zone=z
20	spore.print.color	black=k,brown=n,buff=b,chocolate=h,green=r, orange=o,purple=u,white=w,yellow=y
21	population	abundant=a,clustered=c,numerous=n,scattered=s, several=v,solitary=y
22	habitat	grasses=g,leaves=l,meadows=m,paths=p,urban=u, waste=w,woods=d

species of gilled mushrooms in the Agaricus and Lepiota Family. Each species is identified as definitely edible, definitely poisonous, or of unknown edibility and not recommended. This latter class was combined with the poisonous one. The Guide clearly states that there is no simple rule for determining the edibility of a mushroom. However, we will try to find one using the data set as a truth table.

The data set has 8124 instances defined using 22 nominally valued attributes presented in the table below. It has missing attribute values, 2480, all for attribute #11. 4208 instances (51.8%) are classified as edible and 3916 (48.2%) are classified as poisonous.

An example of a known rule for edible mushrooms is:

odor=(almond.OR.anise.OR.none).AND.spore-print-color=NOT.green

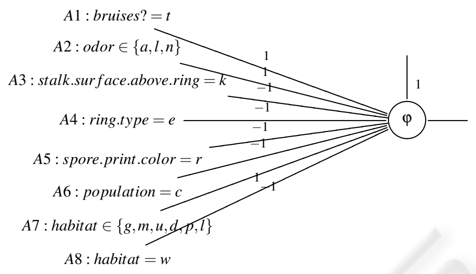
gives 48 errors, or 99.41% accuracy on the whole dataset

We used an unsupervised filter that converted all nominal attributes into binary numeric attributes. An attribute with k values was transformed into k binary attributes. This produced a data set containing 111 binary attributes.

After the binarization we used the described method to select relevant attributes for mushroom classification by fixing a weak stoping criterion. As a result, the method produced a model, with 100% accuracy, depending on 23 binary attributes defined by values of:

odor,gill.size,stalk.surface.above.ring, ring.type, spore.print.color.

We used the values assumed by these attributes to produce a new data set. After 3 tries we selected the model less complex:


$A1 : bruises? = t$
$A2 : odor \in \{a,l,n\}$
$A3 : stalk.surface.above.ring = k$
$A4 : ring.type = e$
$A5 : spore.print.color = r$
$A6 : population = c$
$A7 : habitat \in \{g,m,u,d,p,l\}$
$A8 : habitat = w$
1
1
−1
−1
−1
−1
1
−1
1
φ

This model has an accuracy of 100%. From it, and since attribute values in A2 and A3, as well as the values in A7 and A8 are auto-exclusive, we used propositions A1, A2, A3, A4, A5, A6 and A7 to define a new data set. This new data set was enriched with new negative cases by introducing, for each original case, a new one where the truth value of each attribute was multiplied by 0.5. For instance, the "eatable" mushroom case:

(A1=0, A2=1, A3=0, A4=0, A5=0, A6=1, A7=0)

was used on the definition of a new "poison" case

(A1=0, A2=0.5, A3=0, A4=0, A5=0, A6=0.5, A7=0)

This resulted in a convergence speedup and reduced the occurrence of un-representable configurations.

When we applied our "reverse engineering" algorithm to the enriched data set, having as stopping criterion the mean square error (*mse*) less than 0.003, the method produced the model:

$$\begin{bmatrix} 0 & 1 & 0 & 0 & -1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 & 0 & -1 \end{bmatrix} \quad \begin{bmatrix} -1 \\ -1 \end{bmatrix} \quad \begin{matrix} A2 \otimes \neg A5 \otimes A7 \\ A2 \otimes A4 \otimes \neg A7 \end{matrix}$$
$$\begin{bmatrix} 1 & 1 \end{bmatrix} \quad \begin{bmatrix} 0 \end{bmatrix} \quad i_1 \oplus i_2$$

This model codifies the proposition

$$(A2 \otimes \neg A5 \otimes A7) \oplus (A2 \otimes A4 \otimes \neg A7)$$

and misses the classification of 48 cases. It has 99.41% accuracy and can be interpreted as the rule for eatable mushrooms given by: "a mushroom is eatable if its *odor*=almond.OR.anise.OR.none and *spore.print.color*=black.AND.*habitat*=NOT.waste or *ring.type*=evanescent.AND.*habitat*=NOT.waste."

More precise model can be produced, by restricting the stopping criteria. However, this in general, produces more complex propositions and is more difficult to understand. For instance with a stopping criterion $mse < 0.002$ the systems generated the below model. It misses 32 cases, has an accuracy of 99.2%, and it is easy to convert in a proposition.

$$\begin{bmatrix} 0 & 0 & 0 & -1 & 0 & 0 & 1 \\ 1 & 1 & 0 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & -1 & -1 & 1 \end{bmatrix} \quad \begin{bmatrix} 1 \\ -1 \\ 0 \\ -1 \end{bmatrix} \quad \begin{matrix} \neg A4 \oplus A7 \\ A1 \otimes A2 \otimes \neg A4 \\ A7 \\ A2 \otimes \neg A5 \otimes \neg A6 \otimes A7 \end{matrix}$$
$$\begin{bmatrix} -1 & 0 & 1 & 0 \\ 1 & -1 & 0 & -1 \end{bmatrix} \quad \begin{bmatrix} 1 \\ 0 \end{bmatrix} \quad \begin{matrix} \neg i_1 \oplus i_3 \\ i_1 \otimes \neg i_2 \otimes \neg i_4 \end{matrix}$$
$$\begin{bmatrix} 1 & -1 \end{bmatrix} \quad \begin{bmatrix} 0 \end{bmatrix} \quad j_1 \otimes \neg j_2$$

This NN can be used to interprete formula:

$$\begin{gathered} j_1 \otimes \neg j_2 = (\neg i_1 \oplus i_3) \otimes \neg (i_1 \otimes \neg i_2 \otimes \neg i_4) = (\neg(\neg A4 \oplus A7) \oplus A7) \otimes \neg((\neg A4 \oplus A7) \otimes \\ \neg(A1 \otimes A2 \otimes \neg A4) \otimes \neg(A2 \otimes \neg A5 \otimes \neg A6 \otimes A7))) = \\ = ((A4 \otimes \neg A7) \oplus A7) \otimes ((A4 \otimes \neg A7) \oplus (A1 \otimes A2 \otimes \neg A4) \oplus (A2 \otimes \neg A5 \otimes \neg A6 \otimes A7))) \end{gathered}$$

Some times the algorithm converged to un-representable configurations like the one presented below, with 100% accuracy. The frequency of this type of configurations increases with the increase of required accuracy.

$$\begin{array}{ll} \begin{bmatrix} -1 & 1 & -1 & 1 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & -1 \\ 1 & 1 & 0 & 0 & 0 & 0 & -1 \end{bmatrix} & \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} \begin{array}{l} i_1 \text{ un-representable} \\ A4 \otimes A5 \otimes \neg A7 \\ i_3 \text{ un-representable} \end{array} \\ \begin{bmatrix} 1 & -1 & 1 \end{bmatrix} & \begin{bmatrix} 0 \end{bmatrix} \; j_1 \text{ un-representable} \end{array}$$

Using rule R and selecting the best approximation in data set to each un-representable formula, evaluated in the data set, we have:

1. $i_1 \sim_{0.9297} ((\neg A1 \otimes A4) \oplus A2) \otimes \neg A3 \otimes \neg A6$
2. $i_3 \sim_{1.0} (A1 \oplus \neg A7) \otimes A2$
3. $j_1 \sim_{0.9951} (i_1 \otimes \neg i_2) \oplus i_3$

The extracted formula

$$\alpha = (((((\neg A1 \otimes A4) \oplus A2) \otimes \neg A3 \otimes \neg A6) \otimes \neg(A4 \otimes A5 \otimes \neg A7)) \oplus ((A1 \oplus \neg A7) \otimes A2)$$

is λ-similar, with $\lambda = 0.9951$ to the original NN. Formula α misses the classification for 40 cases. Note that the symbolic model is stable, the bad performance of i_1 representation do not affect the model.

The CNN structure can codify the dataset with 100% accuracy. Bellow we present a prefect description for edible mushrooms.

$$\begin{array}{ll} \begin{bmatrix} 0 & 1 & 1 & -1 & -1 & 0 & 0 \\ 0 & 1 & -1 & 0 & -1 & -1 & 0 \\ 0 & 1 & -1 & 0 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 & -1 & 1 & 1 \end{bmatrix} & \begin{bmatrix} -1 \\ 0 \\ -1 \\ 3 \end{bmatrix} \\ \begin{bmatrix} 1 & 1 & 1 & 1 \end{bmatrix} & \begin{bmatrix} 0 \end{bmatrix} \end{array}$$

This structure have by interpretation the rule for edible mushrooms:

$$\begin{gathered} (A2.and.A3.and.NOT(A4).and.NOT(A5)).or. \\ (A2.and.NOT(A3).and.NOT(A5).and.NOT(A6)).or. \\ (A2.and.NOT(A3).and.NOT(A5).and.A6.and.NOT(A7)).or. \\ (A1.and.A2.and.NOT(A3).and.NOT(A4).and.NOT(A5).and.A6.and.A7) \end{gathered}$$

6 CONCLUSIONS AND FUTURE WORK

This methodology to codify and extract symbolic knowledge from a NN is very simple and efficient for the extraction of comprehensible rules from medium-sized data sets. It is, moreover, very sensible to attribute relevance.

In the theoretical point of view it is particularly interesting that restricting the values assumed by neurons weights restrict the information propagation in the network, thus allowing the emergence of patterns in the neuronal network structure. For the case of linear neuronal networks, having by activation function the identity truncate to 0 and 1, these structures are characterized by the occurrence of patterns in neuron configuration directly presentable as formulas in Łukasiewicz logic.

Generated fuzzy rules might do a good approximation of the data, but often are not interpretable. In your point of view the interpretability of such symbolic rules are strictly related to the type of fuzzy logic associated to the problem. When we applied our method on the extraction of rules from truth tables, generated on Product logic or on Gödel logic, this rules were very dificulte to interprete. For the extraction of knowledge from this types of fuzzy logic extraction processed governed by appropriated logic must be developed.

We are using this methodology for fuzzy regression tree generation. Where we use CNN for finding slitting formulas in the algorithm pruning fase (Algara, 2007).

ACKNOWLEDGEMENTS

I thanks Helder Pita for reading and commenting on the manuscript. I acknowledge the support of the Instituto Superior de Engenharia de Lisboa and the Área Cientifica da Matemática.

REFERENCES

Algara, E. (2007). *Soft Operators Decision Trees: Uncertainty and stability related issues*. Vom Fachbereich Mathematik der Technischen Universitt Kaiserslautern zur Verleihung des Akademischen Grades Doktor der Naturwissenschaften, 2007.

Amato, P., Nola, A., and Gerla, B. (2002). Neural networks and rational łukasiewicz logic. *IEEE Transaction on Neural Networks, vol. 5 no. 6, (2002)506-510.*

Andersen, T. and Wilamowski, B. (1995). A modified regression algorithm for fast one layer neural network training. *World Congress of Neural Networks, Washington DC, USA, Vol. 1 no. 4, CA, (1995)687-690.*

Battiti, R. (1992). Frist- and second-order methods for learning between steepest descent and newton's method. *Neural Computation, Vol. 4 no. 2, (1992)141-166.*

Bello, M. (1992). Enhanced training algorithms, and intehrated training/architecture selection for multilayer perceptron network. *IEEE Transaction on Neural Networks, vol. 3, (1992)864-875.*

Bornscheuer, S., Hölldobler, S., Kalinke, Y., and Strohmaier, A. (1998). Massively parallel reasoning. *in: Automated Deduction - A Basis for Applications, Vol. II, Kluwer Academic Publisher, (1998)291-321.*

Castro, J. and Trillas, E. (1998). The logic of neural networks. *Mathware and Soft Computing, vol. 5, (1998)23-27.*

Charalambous, C. (1992). Conjugate gradient algorithm for efficient training of artificial neural networks. *IEEE Proceedings, Vol. 139 no. 3, (1992)301-310.*

d'Avila Garcez, A. S. (2007). Advances in neural-symbolic learning systems: Modal and temporal reasoning. *In B. Hammer and P. Hitzler (ed.), Perspectives of Neural-Symbolic Integration, Studies in Computational Intelligence, Volume 77, Springer, 2007.*

d'Avila Garcez, A. S., Lamb, L. C., and Gabbay, D. M. (2008). *Neural-simbolic Cognitive Reasoning*. Cognitive Technologies, Springer.

Dubois, D. and Prade, H. (2000). *Fundamentals of fuzzy sets*. Kluwer, 2000.

Eklund, P. and Klawonn, F. (1992). Neural fuzzy logic programming. *IEEE translations on neural networks, Vol. 3, No. 5, 1992.*

Fiadeiro, J. and Lopes, A. (1997). Semantics of architectural connectors. *TAPSOFT'97 LNCS, v.1214, p.505-519, Springer-Verlag, 1997.*

Frank, M. (1979). On the simultaneous associativity of $f(x,y)$ and $x+y-f(x,y)$. *Aequations Math., vol. 19, (1979)194-226.*

Fu, L. (1993). Knowledge-based connectionism from revising domain theories. *IEEE Trans. Syst. Man. Cybern, Vol. 23 ,(1993)173-182.*

Gallant, S. (1988). Connectionist expert systems. *Commun. ACM, Vol. 31 ,(1988)152-169.*

Gallant, S. (1994). *Neural Network Learning and Expert Systems*. Cambridge, MA, MIT Press.

Gerla, B. (2000). Functional representation of many-valued logics based on continuous t-norms. *PhD thesis, University of Milano, 2000.*

Hagan, M., Demuth, H., and Beal, M. (1996). *Neural Network Design*. PWS Publishing Company, Boston.

Hagan, M. and Menhaj, M. (1999). Training feedforward networks with marquardt algorithm. *IEEE Transaction on Neural Networks, vol. 5 no. 6, (1999)989-993.*

Hájek, P. (1995). Fuzzy logic from the logical point of view. *In Proceedings SOFSEM'95, LNCS, Springer-Verlag, 1995.*

Hassibi, B., Stork, D., and Wolf, G. (1993). Optimal brain surgeon and general network pruning. *IEEE International Conference on Neural Network, vol. 4 no. 5, (2003)740-747.*

Hitzler, P., Hölldobler, S., and Seda, A. (2004). Logic programs and connectionist networks. *Journal of Applied Logic, 2, (2004)245-272.*

Hölldobler, S. (2000). Challenge problems for the integration of logic and connectionist systems. *in: F. Bry, U.Geske and D. Seipel, editors, Proceedings 14. Workshop Logische Programmierung, GMD Report 90, (2000)161-171.*

Hölldobler, S. and Kalinke, Y. (1994). Towards a new massively parallel computational model for logic programming. *in: Proceedings ECAI94 Workshop on Combining symbolic and Connectionist Processing, (1994)68-77.*

Hölldobler, S., Kalinke, Y., and Störr, H. (1999). Approximating the semantics of logic programs by recurrent neural networks. *Applied Intelligence 11, (1999)45-58.*

Jacobs, R. (1988). Increased rates of convergence through learning rate adaptation. *Neural Networks, Vol. 1 no. 4, CA, (1988)295-308.*

Komendantskaya, E., Lane, M., and Seda, A. K. (2007). Connectionistic representation of multi-valued logic programs. *In B. Hammer and P. Hitzler (ed.), Perspectives of Neural-Symbolic Integration, Studies in Computational Intelligence, Volume 77, Springer, 2007.*

Mehrotra, K., Mohan, C., and Ranka, S. (1997). *Elements of Artificial Neural Networks*. The MIT Press.

Miniani, A. and Williams, R. (1990). Acceleration of back-propagation through learning rate and momentum adaptation. *Proceedings of International Joint Conference on Neural Networks, San Diego, CA, (1990)676-679.*

Samad, T. (1990). Back-propagation improvements based on heuristic arguments. *Proceedings of International Joint Conference on Neural Networks, Washington (1990)565-568.*

Solla, S., Levin, E., and Fleisher, M. (1988). Accelerated learning in layered neural networks. *Complex Sustems, 2, (1988)625-639.*

Towell, G. and Shavlik, J. (1993). Extracting refined rules from knowledge-based neural networks. *Mach. Learn., Vol. 13 ,(1993)71-101.*

Towell, G. and Shavlik, J. (1994). Knowledge-based artificial neural networks. *Artif. Intell., Vol. 70 ,(1994)119-165.*