An Extension of the Core Method for Continuous Values: Learning with Probabilities

Nuno C. Marques

CENTRIA/Departamento de Informática,
Faculdade de Ciências e Tecnologia, Universidade Nova de Lisboa
Quinta da Torre, 2829-516 Caparica, Portugal
nmm@di.fct.unl.pt

Abstract. This paper proposes an extension to the neuro-symbolic core method useful when observations are expressed by continuous values. Some theoretical results are presented regarding the learning process over these observations. An illustrative example is reported, demonstrating the problems of the original approach and justifying how this extension can overcome them. Results of the extended approach on irregular continuous values (simulating probabilistic data) show similar results to the original core method on clean symbolic data and point to the validity of the approach.

1 Introduction

The core method [1], allows the encoding of any logic program into a feed-forward one hidden layer neural network. This encoding expresses the immediate consequence operator T_p of a logic program. So, a direct recursive connection of the output layer to the input layer allows the computation of the result of any logic program by a neural-network with a feed-forward core. Neuro-symbolic (NeSy) integration for first order logic was recently achieved [2] with this method.

Knowledge encoding into neural networks was addressed in many works, including initial work by McCulloch and Pitts [3], knowledge based neural networks [4] and work with the core method (e.g. [1], [2], [5]). Today, we believe that more knowledge based approaches are one of the best ways to improve the applicability of standard back-propagation learning methods [6] to data mining and machine learning in general. One of the major advantages of these methods is their deep knowledge and possible formal description of what can be computed and learned from observations. From a machine learning perspective, the core method has already very good results reported (e.g. [7], [8]), namely by achieving faster convergence and better precision results than standard non-symbolic neural networks.

Unfortunately, results while applying the core method neural networks to natural language processing (e.g. [9]) were not so clear: improvements were detected and reported, but more general use of the method revealed difficult. Namely, it was too difficult to express the proper knowledge for this problem: backpropagation learns very fast how to encode easy rules in data. However too specific rules where easily forgotten during the training process.

Regarding neural-symbolic learning capabilities, the original core method proposal just focused the logic representation, and a stepwise activation function [1]. So several studies have been made regarding neural-symbolic learning capabilities, namely by replacing the stepwise function either by the standard *sigmoid* or *atan* activation functions and using gradient methods (such as error backpropagation, e.g. [8], [5]).

Recent results by [5], successfully propose a new way to encode symbolic incomplete knowledge in NeSy networks and improve them with experimental data. Also, until now, most results have always assumed binary input encodings: as it was shown in [7], the core method requires specific input encoding intervals, i.e. encoded rule output may be incorrect when inputs with intermediate values between $TRUE$ and $FALSE$ are used. In this paper we will elaborate on the results reported in [5], and conjoin them with the discussion of the problems that occur when specific probabilistic values outside core method validity intervals are used. Namely results in the learning speed and capabilities of backpropagation learning will be discussed. A specific focus will take into account the use of a probabilistic encoding, and an extension to the original core method will be proposed.

An illustrative problem will be addressed: learning AND with simulated probabilistic encoding.

2 The Core Method

Let us assume a sigmoid unit, where[1]:

$$net = w_0 + w_1 x_1 + \cdots + w_n x_n,$$

and

$$o = \sigma(net).$$

In this unit, o is the output, w_0 the bias and w_i the connection weights for each input x_i belonging to train data observation $\boldsymbol{X}$. Finally $\sigma(x)$ is the sigmoid function:

$$\sigma(x) = \frac{1}{1+e^{-x}}.$$

Generically core method rules are embedded into the network according to the specification in [1], [8] or [5]. A one hidden layer feed-forward neural network with a predefined constant ω is used. Each hidden layer neuron can encode an AND rule. The unit connections are set to ω for positive literals and to $-\omega$

[1] This presentation of MLP neural-network learning for the core method will follow Tom Mitchell's notation and derivation of the backpropagation learning process in neural networks [10], that can be consulted for further details.

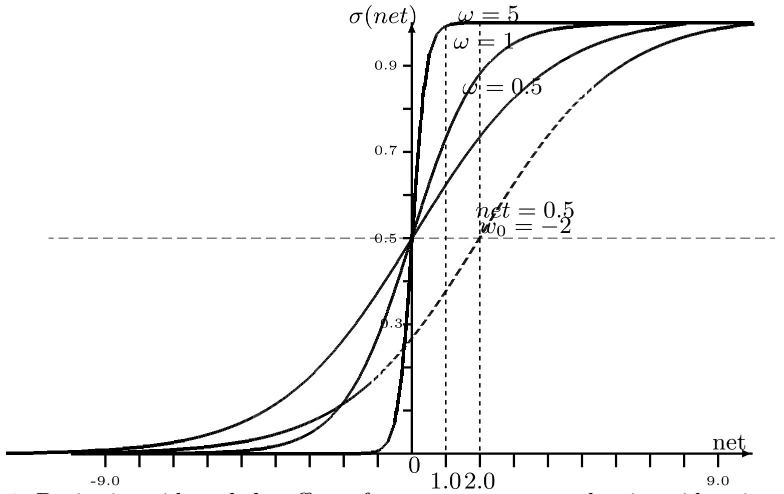


Fig. 1. Basic sigmoid, and the effect of some constants on the sigmoid activation function.

for negative literals. The bias (w_0) is set up such that this unit becomes active if and only if the input of the network coincides with the rules precondition (typically, $\omega \times (l - 0.5)$, where l is the number of literals of the rule). If no other constraints exist, the output layer can simply make an OR of hidden layer units by setting connecting weights (w_i) to 1.0 and bias to 0.0. Parameter ω is used as a multiplying factor over the predefined weights, to give additionally stability during neural network training. Bigger values of ω will make the sigmoid more similar to a step-wise activation function (figure 2), this way, high values for ω assure the encoded information is kept during learning.

Core method basic encoding can be exemplified with the following program:

$$A \wedge C \mapsto_{r_1} F$$
$$A \wedge \neg B \wedge D \mapsto_{r_2} F$$

The network for this program should have as input neurons i_A, i_B, i_C and i_D. If we take, e.g. $\omega = 5$, rule r_1 could be encoded by a hidden layer unit with $h_{r_1} = \sigma(-7.5{+}5{\times}i_A{+}5{\times}i_C)$ and rule r_2 by $h_{r_2} = \sigma(-12.5{+}5{\times}i_A{+}5{\times}i_B{+}5{\times}i_D)$. Final output F will simply be encoded by $F = \sigma(h_{r1}{+}h_{r2}{-}0)$. Usually, after encoding, but before backpropagation learning, all connections are slightly disturbed by adding some small random noise (e.g. $[-0.1, 0.1]$.).

2.1 Backpropagation Learning in the Core Method Networks

Backpropagation learns the weights w_i, by minimizing the squared error:

$$E[\boldsymbol{w}] \equiv \frac{1}{2} \sum_{d \in D} (t_d - o_d)^2,$$

where D is the set of training examples. So, the perceptron training rule can be expressed as:

$$\Delta w_i = \eta(t - o)x_i,$$

$t = c(\boldsymbol{x})$ is the target value for observation x and η is the learning rate (usually 0.1). The above training rule is convergent if training data is linearly separable and η sufficiently small [10]. So, taking into account $\nabla E[\boldsymbol{w}]$ gradient, the training rule:

$$\Delta \boldsymbol{w} = -\eta \nabla E[\boldsymbol{w}] = -\eta \frac{\partial E}{\partial w_i},$$

can be used to derive the gradient rules to train one sigmoid unit based on:

$$\frac{\partial E}{\partial w_i} = -\sum_d (t_d - o_d) \frac{\partial o_d}{\partial net_d} x_{i,d}. \tag{1}$$

Regarding learning in the core method, we should notice that by equation 1, learning depends on $\frac{\partial o_d}{\partial net_d}$. But this introduces a problem when learning with bigger values of ω. If we look at figure 2 we see that ω is indeed influencing learning: bigger values of ω make the sigmoid approximate a stepwise function and make $\frac{\partial o_d}{\partial net_d}$ go near zero, when values of net are bigger than 1.0 (this effect was observed experimentally in [5]). This is also relevant for the core method since the usual initialization of parameters is biasing the network and making harder for equation 1 to update weights (i.e. update will be very small for large values of net). This may be a good behavior, if we are certain on the rules, but not advisable if rules need revision/are wrong (e.g. [5]). Indeed, for the following discussion we should note that there is a learning zone that depends on ω and on the values of net: in practice, only small enough values of net and ω make learning possible (or, alternatively, fast enough).

3 Decisions based on distinct continuous values

Traditional error back-propagation in feed-forward neural networks deals with continuous values. However, traditional logic is based on propositions and on the truth value of those propositions. So, the usual input for the Core Method uses a 0.0/1.0 encoding for Boolean values[2]. We think that, until now, this has hidden a potential problem when generalizing the Core Method to continuous inputs on sigmoid neural networks. Let us start by showing that:

Proposition 1 *Any logic program with Boolean inputs can be encoded in a sigmoid neural network.*

[2] An alternative $-1.0/1.0$ encoding is also frequently used with the $\sigma(x) = atan(x)$ function, namely for easier encoding of negated literals. Without loss of generality, and for a more usual presentation, we have used the the equivalent sigmoid $\sigma(x)$ function.

Previous results (e.g. [7], [8]) have already shown a similar result. However, this new formulation will be useful for our discussion because it explicitly takes into account the value of ω.

Proof 1 *Previous results showed how to encode any propositional logic program inside a neural network with step-wise activation (e.g. proposition 4 in [1]). We consider a sigmoid neuron active when the sigmoid function is bigger than some appropriate value α and inactive when the sigmoid is smaller than some value $1-\alpha$ (where $\alpha \in [0.5; 1[$). Then, for any fixed α, we can make ω as big as needed, so the sigmoid function will approach the step-wise activation function up to an value ε. This way, we can make ε arbitrarily small, so that for any given problem the neurons activate in a way that follows proposition 4 in [1].* •

By conjoining this proof with preceding discussion, we should notice there is an engineering decision that must be taken when deciding the values to use for ω. As it was already pointed, backpropagation algorithm can not learn well enough for big values of ω. On the other side, if ω is too low and the encoded information is not evident from experimental data, the core method encoded information will be lost in noise and during backpropagation updates[3]. So we must choose suitable ω values to encode our knowledge inside the neural network, and, simultaneously, allow learning to be done.

Proposition 1 can now be extended for the cases when the input is not restricted to Boolean values.

Proposition 2 *Any logic program with conditions over a vector of continuous values $\boldsymbol{X}$ can be encoded in a sigmoid neural network.*

Proof 2 *Any condition over a continuous value could be given a logical value by constraints: $X_i > \mathcal{T}_{X_i}$.*

So a sigmoid neuron having an input X, and a predefined ω can implement this constraint, namely, if we consider a neuron with bias $\omega \times \mathcal{T}_{X_i}$:

$$1 - alpha < \frac{1}{1 + e^{-\omega \times (X_i - \mathcal{T}_{X_i})}} < alpha.$$

So (e.g. figure 2),

$$X_i - \mathcal{T}_{X_i} > 0,$$

will be true iff $X_i > \mathcal{T}_{X_i}$.

Let us consider a first hidden layer of such threshold units for all $X_i \in \boldsymbol{X}$, converting $\boldsymbol{X}$ to each logical value needed by program a P. Then, by proposition

[3] During experiments we have observed that -due to the distributed nature of backpropagation learning-, in some configurations, the already given information can even be harmful. This happens when backpropagation tries to re-encode (or re-learn) that information in distinct ways.

1, we can add a second hidden layer encoding this logic program P using the logic output of this first hidden layer. ●

With this result, we are no longer reduced to the single threshold bias of the rule neuron and we can have a set of threshold values, distinct for each input. Additional computational power of the neurons in this first hidden unit can also be handy for more modular encodings of logical symbols (e.g. for encoding lists). Moreover, we can increase ω value for the connection to this neurons, in order to enforce some of the intended mappings (and to make it more stable during learning) or to reduce them and so concentrate learning on those areas of the network.

Probabilistic Encoding Proposition 2 is important for problems were we don't have a direct way to describe a given object in terms of its explicit features. Usually, in these cases knowledge is better expressed using probability theory. E.g. in [9], the probability that some word w being tagged as T was calculated based on:

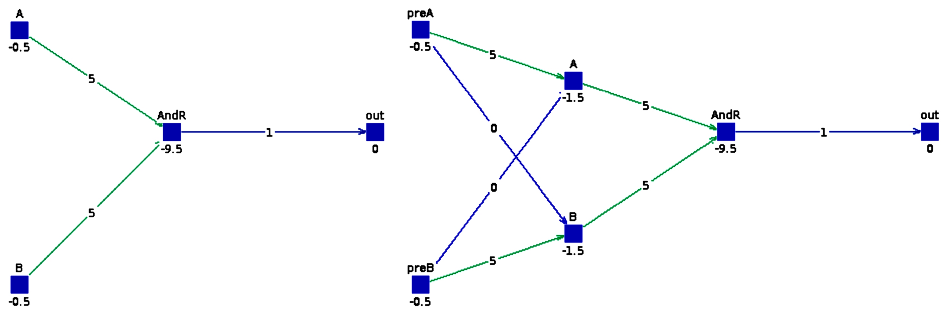
$$p_w(T|event) = \frac{freq(event, T)}{freq(event)}.$$

In this equation, $freq(event, T_i)$ denoted the number of times $event$ had mark T and $freq(event)$ denoted the total number of observations of $event$. In this encoding a probability value of, e.g. 0.1, would represent ten observations per 100 occurrences of that event (e.g. a given word). This is a meaningful value and could be modeled by some rule related to the presence of T. However there is a language problem when we talk about probabilistic events in a logical format: Assume P_b is the probability for a feature b and that P_c is the probability for an independent feature c. If we know that a is the logic consequence of both b and c, we can write:

$$a \leftarrow b(P_b), c(P_c). \tag{2}$$

According to this equation, if both b and c are probable enough to hold, i.e. $P(x) \geq \mathcal{T}_x$ then we should expect a to also hold. However, due to statistical independence, $P_a = P_b \times P_c$, i.e. the minimal value for P_a to hold is ${P_a}^{min} = \mathcal{T}_b \times \mathcal{T}_c$, a value that should be in the non-accepting region of the usual $T \geq 0.5$ core method perceptron (e.g. if $\mathcal{T}_a = \mathcal{T}_b = 0.2$, but $\mathcal{T}_c = 0.6$, then the independent event has $P_a = 0.12 < \mathcal{T}_a$). In this case, it might not be advisable to change the threshold of all the inputs by using the bias of the hidden neuron (i.e. either there is an error in considering a and b $FALSE$ or an error by considering c $TRUE$). So, there could be a problem when representing this probability knowledge for a neural network using the original core method. I.e., we may be interested in the logical consequence of two events considered true, and not on the probability of its joint occurrence. In this cases we should use the encoding proposed as a result of proposition 2, so the decision surface of the core method perceptron can be initialized according to the intended logical consequence function of individual threshold values $\mathcal{T}_b$ and $\mathcal{T}_c$.

4 An Illustrative Example while Learning an AND



1HL– Original Core Method *2HL– Extended Core Method*

Fig. 2. Initial core methods neural networks, before training and initial weight disturbance.

For better illustrating the described problem, a simple $C \leftarrow A \wedge B$ rule was encoded in a feed-forward sigmoid based neural network with two inputs (A and B) and a single output (C). According with the original core method, a neural network with a single hidden unit was defined. We set $\omega = 5.0$, so this unit had the two initial weights (one for each possible value of input) also set to 5.0. However the initial bias on all these networks was set to -9.5 (i.e. a slightly higher bias that assumes $\mathcal{T} \geq 0.9$). This hidden unit was then connected with the single output layer unit (C) with initial weight set to 1.0. The neural network is represented in figure 2 – 1HL.

For this example two datasets were built simulating different threshold values over the AND truth table: one had input $TRUE$ values encoded as 0.4 and in the other $TRUE$ was encoded as 0.9. Output $TRUE$ values were always encoded as 1.0 and all $FALSE$ values were encoded as 0.0.

A second neural network (represented in figure *2HL*) was also built according with proposition 2: a first hidden layer was added between input layer and the original hidden layer. As described, this layer implements the threshold detection for the 0.4 encoding. This layer has two hidden neurons (one representing each input neuron), connected with initial weight of 5.0 to the neuron that is being mapped and with 0.0 to the other neuron. Initial bias was set to -1.5 (i.e. $\mathcal{T}_X \geq .3$, with $\omega = 5.0$). Finally weights in *1HL* network were randomized (between -0.1 and 0.1), to provide a control case (*RND* network).

The three neural networks were trained with standard backpropagation method ($\eta = 0.2$). Learning stopped when error variation was smaller than 0.01. An initial disturbance of 0.1 was added to all networks. During training a weight disturbance of 0.05 was done every 100 iterations. The SSE values for all networks during the train process are represented in figure 3. Several other experiments were also preformed with similar results. E.g., we should point that for the more standard bias of 7.5 (i.e. $\mathcal{T} \geq 0.5$) a similar behavior was observed when $TRUE$ was encoded as 0.25. Namely the initial core method network revealed itself much more sensible to noise than the network derived from proposition 2.

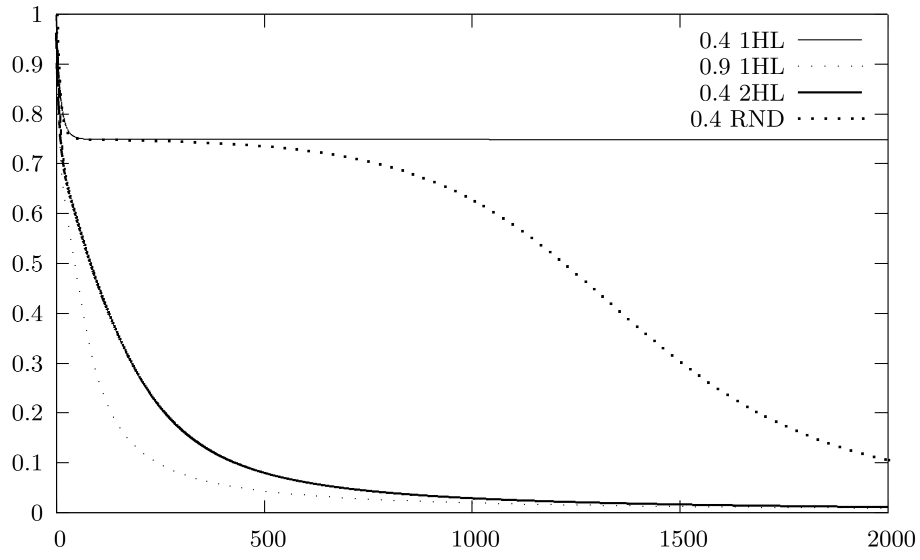


Fig. 3. Train error for learning an AND with four distinct encodings.

5 Discussion and Conclusions

The standard core method neural network on the 0.4 dataset (*0.4 1HL*) didn't converge in more than 100000 iterations[4]. We should also mention the importance of setting ω to appropriate values in *0.4 2HL* network. Indeed, the first hidden layer value for ω was first set to 1.0. However then, although the network was still faster to learn than the control (*RND*) network, a much bigger sensibility to the initial *bias* in the first hidden layer was detected, with convergence only starting around iteration 500 and 0.1 SSE achieved around iteration 1000. When $\omega = 5.0$ the behavior was more stable, and even if wrong initial bias

[4] This was due to the noise added every 100 iterations. But even if jog weighting is eliminated, *0.4 1HL* network only starts convergence around iteration 5000.

were given (e.g. 0), the backpropagation quickly found the correct bias values. Moreover, if we set ω to 10.0 the behavior of *0.4 2HL* is indistinguishable of *0.9 1HL*.

This small example shows how sensible the initial core method proposal may be, when appropriate distinct continuous input conditions exist. Indeed we should notice that $1HL$ network is much worse than the control (RND) network on this peculiar case. Even minor weight changes can condition the proper convergence of this network. From what we have observed the correct initial knowledge encoded into the network was unused and made learning almost impossible for backpropagation algorithm. From ongoing experiments we have noticed this effect to become even worst in real scenarios.

This result is clearly demonstrating the need of additional computational structures for helping the network to find the truth values when inputs are sent as continuous irregular values. It should be stressed that this was not a theoretical example. Indeed there are many cases (as, e.g., the one mentioned in [9]) where the additional information encoded in probability values is needed for the system to perform properly. The main problem was that, although this information should be provided to the neural network (so that it could learn other relations with on free neurons), it was then difficult to encode the symbolic knowledge into the network.

The solution presented by proposition 2 is quite valuable in these terms, since it allows the network and training algorithm to protect encoded knowledge from random fluctuations in data. Also the setting of ω values revealed itself as a quite powerful tool to help in optimizing and even directing learning effort into the appropriate areas of the network. Indeed, the author is now convinced that we are now approaching the point were Neuro-Symbolic integration is starting to prove its capabilities for general machine learning problems. The general perspective is that neural network programming and tuning for known knowledge is now increasingly possible while still keeping the network open to learn other information contained in training data.

6 Acknowledgments

This work would not have been possible without initial joint work done with Sebastian Bader and Steffen Hölldobler, namely during author's stay as an invited researcher in the Technical University of Dresden. I thank both of them, not only their excellent help and support, but also the very good working conditions and environment they made available in TU-Dresden.

References

1. Hölldobler, S., Kalinke, Y.: Towards a massively parallel computational model for logic programming. In: Proceedings ECAI94 Workshop on Combining Symbolic and Connectionist Processing, ECCAI (1994) 68–77

2. Bader, S., Hitzler, P., Hölldobler, S.: Connectionist model generation: A first-order approach. Neurocomputing **51** (August 2008) 2420–2432
3. McCulloch, W.S., Pitts, W.: A logical calculus of the ideas immanent in nervous activity. Bulletin of Mathematical Biophysics **5** (1943) 115–133
4. Towell, G.G., Shavlik, J.W.: Knowledge-based artificial neural networks. Artificial Intelligence **70**(1–2) (1994) 119–165
5. Bader, S., Hölldobler, S., Marques, N.C.: Guiding backprop by inserting rules. In d'Avila Garcez, A.S., Hitzler, P., eds.: Proceedings of the 4th International Workshop on Neural-Symbolic Learning and Reasoning, Patras, Greece, July 2008. CEUR Workshop Proceedings, Vol. 366, 2008. ISSN 1613-0073. Volume 366. CEUR Workshop Proceedings (2008) ISSN: 1613-0073.
6. McClelland, J., Rumelhart, D.: Parallel Distributed Processing. MIT Press (1986)
7. d'Avila Garcez, A.S., Zaverucha, G.: The connectionist inductive learning and logic programming system. Applied Intelligence, Special Issue on Neural networks and Structured Knowledge **11**(1) (1999) 59–77
8. d'Avila Garcez, A.S., Broda, K.B., Gabbay, D.M.: Neural-Symbolic Learning Systems — Foundations and Applications. Perspectives in Neural Computing. Springer, Berlin (2002)
9. Marques, N.C., Bader, S., Rocio, V., Hölldobler, S.: Neuro-symbolic word tagging. In José Neves, Manuel Filipe Santos, J.M., ed.: New Trends in Artificial Intelligence, APPIA - Associação Portuguesa para a Inteligência Artificial (12 2007) 779–790
10. Mitchell, T.M.: Machine Learning. McGraw-Hill (March 1997)