PRODUCTION RULES GENERATION AND REFINEMENT IN BP NETWORKS [1]

T. DENŒUX†and R. LENGELLE‡

†*Compiègne Artificial Intelligence Laboratory*
Technopolis - ZAC de Mercières - F-60200 Compiègne - France

‡*University of Compiègne – U.R.A. 817 du CNRS*
BP 649 - F-60206 Compiègne cedex - France

ABSTRACT

The problem of extracting symbolic knowledge out of back-propagation (BP) networks is examined. A solution is proposed, based on a mapping from a class of decision rules onto a class of multilayer feedforward networks, and on learning algorithms which allow this mapping to be reversed. This approach can be applied to refine an existing rule base, or to generate classification rules from a set of positive and negative examples.

INTRODUCTION

Multilayer perceptrons and decision trees have recently become very popular in the Pattern Recognition community. Both techniques are robust in the presence of noise, require no assumption concerning the structure of the data, and are able to approximate arbitrarily complex class boundaries. Although some comparative studies have concluded to a slight superiority of the connectionist approach in terms of generalization error ([1, 9]), this method still suffers from a number of deficiencies, which are subject to intensive research: (1) the learning phase, being based on gradient descent, is slow; (2) the network architecture has to be designed empirically; and (3) the knowledge embedded in the connection weights is not easily interpretable, the latter being perhaps what best distinguishes the connectionist approach in general from the Machine Learning techniques of classical Artificial Intelligence. Attempts to provide a global answer to these problems have consisted in defining a mapping f from a set R of decision rules, expressed in some formalism, onto a set N of artificial neural networks. Such mappings have been proposed e.g. for decision trees ([8]) and production rules in Lukaciewicz or Zadeh logic ([5]). This approach allows for prior knowledge, obtained from domain experts or data preprocessing, to be used in order to initialize the network close to an "interesting" minimum of the error function, which usually results in faster convergence and reduced probability of getting stuck in a local minimum during the learning process. Furthermore, the network architecture is in that case completely defined by the correspondence with the initial representation (depth of the decision tree, number of rules or prototypes, etc.). This enables the initial problem of neural net design to be reduced to more classical problems such as decision tree pruning or prototype selection, for which well-established techniques already exist. The interpretation problem, however, is not solved unless the mapping f can be reversed: it is

[1]This work has been supported by EEC funded Esprit II project nr. 5433 (NEUFODI); partners: BIKIT, ARIAI, Elorduy Sancho y Cia, LABEIN, Lyonnaise des Eaux-Dumez.

then possible to extract a refined decision rule, that makes sense to the user, from the trained neural network. In the examples mentioned above, this condition is not met: f actually projects R onto a subset N' of N. For example, when converting a decision tree into a network with two hidden layers according to the method described in [8], the AND units are assigned weights in the set {-1,0,1}. After training by gradient descent, the performance of the network is generally improved, but it is no longer possible to interpret the AND units as computing logical functions. In that case, the learning algorithm has generated a network which no longer belongs to the set N' of networks which have an interpretation in R. To tackle this problem, one strategy consists in using a "constrained" learning procedure with operates inside the set N' of "interpretable" networks. This strategy will be applied in this paper in the case of production rules. First, we shall examine on which conditions the computation performed by a formal neuron can be interpreted as a conjunction or a disjunction, in crisp or fuzzy logic. Then, two learning procedures that preserve this interpretation will be described, after which simulation results will be presented.

IMPLEMENTATION OF LOGICAL CONNECTIVES

Consider a unit U with weights $w_1, \ldots, w_n$, bias w_0 and activation function θ such that $\theta(x) = 1$ for $x > 0$ and $\theta(x) = -1$ for $x \leq 0$. Let $x_1, \ldots, x_n$ be n binary inputs with values in $\{-1, 1\}$. The output of U is given by $y = \theta\left(\sum_{i=1}^{n} w_i x_i + w_0\right)$. Sufficient conditions for U to compute a conjunction (resp. a disjunction) of a subset of its inputs, or their negations, are

$$\forall i \in \{1, \ldots, n\} \;\; w_i \in \{-1, 0, 1\} \tag{1}$$

$$w_0 = \left(-\sum_{i=1}^{n} w_i^2 + 1\right)\epsilon \tag{2}$$

with $\epsilon = 1$ (resp. $\epsilon = -1$). Let us now assume the inputs to be continuous in the range $[-1, 1]$, and let us replace the step function $\theta(x)$ by a sigmoid function $\sigma_\alpha(x) = \tanh \alpha x$, where α is a gain parameter. Then, under the conditions expressed by Equations 1 and 2, U computes some kind of "fuzzy" conjunction (resp. disjunction). However, the property of associativity is missing: in general

$$\sigma_\alpha\left(x_1 + \sigma_\alpha\left(x_2 + x_3 + \epsilon\right) + \epsilon\right) \neq \sigma_\alpha\left(\sigma_\alpha\left(x_1 + x_2 + \epsilon\right) + x_3 + \epsilon\right) \tag{3}$$

Therefore, the interpretation of U in terms of fuzzy logic is not strictly appropriate. However, one special case deserves attention: for some values of α, $\sigma_\alpha(x)$ is a good approximation of $\min(x, 1)$ on $[-1, \infty[$ and of $\max(-1, x)$ on $]-\infty, 1]$. In that case, U approximates the conjunction (with $\epsilon = 1$) or the disjunction (with $\epsilon = -1$) in Lukaciewicz logic ([5]), defined by:

$$\begin{aligned} x \wedge y &= \max(-1, x + y - 1) \\ x \vee y &= \min(1, x + y + 1) \end{aligned}$$

The gain parameter α can be chosen so as to minimize the quantity:

$$\int_0^\infty \left(\tanh \alpha x - \min(1, x)\right)^2 \, dx \tag{4}$$

One obtains numerically $\alpha \approx 1.325$.

Since logical connectives can be computed in BP units, any logical expression in conjunctive normal form can be represented in a network with one hidden layer of AND units, and one output layer of OR units. In order to represent arbitrary decision rules in $\mathbb{R}^p$, one additional "partitioning" layer can be added before the AND layer ([8]), each partitioning unit with weights $u_1, \ldots, u_p$ and bias u_0 evaluating a test of the form: $u_0 + \sum_{i=1}^{p} u_i x_i > 0$?

LEARNING PROCEDURES

Relationship between the bias and the weights

In order to preserve the interpretation of the neural network in terms of production rules, one needs a learning procedure which attempts to respect the conditions expressed by Equations 1 and 2. The first one can easily be taken into account by replacing the bias term w_0 by its expression as a function of the weights $w_1, \ldots, w_n$ in the activation equation of logical units. The derivative of y with respect to w_i can then be easily computed and used in the expression of the gradient of the error F with respect to the weights. In order to account for the second above-mentioned condition, two learning rules have been tested. They are described in the two following sections.

Inclusion of a penalty term in the cost function

Both rules operate in batch mode, i.e. the error function F is defined as the expected value of $\|\mathbf{t} - \mathbf{o}\|^2$ where $\mathbf{t}$ and $\mathbf{o}$ are the desired and obtained output vectors, respectively. The first procedure consists in adding to the error function F a penalty term P, which "incites" the weights of the logical units to tend towards values in $\{-1, 0, 1\}$: $P = \lambda \sum_{i=1}^{N} h(W_i)$ where W_i is the i-th component of the vector $\mathbf{W}$ containing the N weights associated to the AND and OR units, h is a function with three stable points at -1, 0 and 1 ([3]), and λ is a coefficient. The relative importance of the penalty term in the global cost function, represented by λ, is difficult to determine *a priori*. One solution consists in adjusting it dynamically during the learning process, e.g. according to the heuristics proposed in [10] in the case of weight elimination. Note that this adaptation mechanism does not aim at satisfying $P = 0$ exactly, but just attempts to find the best compromise between the minimization of F and P.

In order to compensate for the introduction of a rather strict condition on the weights, the network can profitably be given additional degrees of freedom by allowing the gains of the sigmoids to vary, as suggested in [6]. This adaptation of gains by gradient descent on F amounts to tuning the "degree of fuzziness" of the units. If one is seeking to obtain decision rules in Lukaciewicz logic, the gains of the logical units can be fixed to their optimal value of 1.325 after convergence of the network, and the weights in the partitioning layer can be updated separately to account for this modification.

Introduction of a probabilistic update mechanism

The method described above used real-valued weights, which were forced to gradually approach integer values in {-1, 0, 1}. Although this method yields satisfactory results for problems of limited complexity (see examples in the following section), the introduction of a penalty term has been found to make the learning process much harder

by creating deep local minima. Although this problem can probably be solved by improving the optimization scheme, this has incited us to investigate means of operating directly on integer values. In this second approach, the initial weights are chosen in {-1, 0, 1}, and integer weight updates ΔW_i are generated randomly between -1 and 1, according to a probability law which verifies the following condition:

$$\mathbf{E}\left[\Delta W_i\right] = -\eta \left(\frac{\partial F}{\partial W_i}\right) \tag{5}$$

where η is a positive coefficient. If p_1, p_2 and p_3 are defined as:

$$p_1 = P(\Delta W_i = -1) \quad p_2 = P(\Delta W_i = 0) \quad p_3 = P(\Delta W_i = 1) \tag{6}$$

then the condition expressed by Equation 5 is equivalent to:

$$\begin{cases} p_3 - p_1 = -\eta \left(\frac{\partial F}{\partial W_i}\right) \\ p_1 + p_2 + p_3 = 1 \end{cases} \tag{7}$$

This system of equations is obviously underdetermined. A third condition can be added by choosing the value of p_2, which is the probability that a given weight remains unchanged by the procedure at one iteration.

This update mechanism is a probabilistic version of gradient descent. It ensures that, *on average*, the weights are updated in the direction of steepest descent, by an amount proportional to the gradient. However, owing to the probabilistic nature of this rule, the probability of an increase of the error function is not negligible. Therefore, the modification of the weights is accepted only if the error decreases, or after a pre-defined number of unsuccessful trials. A simulated annealing version of this algorithm could also be considered, in which a weight change resulting in an increase of the error would be accepted with some time-decreasing probability (depending also on the magnitude of the error). This possibility has not yet been investigated.

Note that this procedure makes it difficult to train a partitioning layer and the conjunction layer at the same time. One solution is to train the partitioning layer independently using e.g. the procedure described in [7].

Automatic construction of the AND layer

The learning procedures described above leave open the problem of determining the right number of conjunction units. To overcome this problem, we have tested a very simple algorithm which automatically constructs the AND layer. This algorithm treats AND units as target units. For each input class c, AND units are added iteratively and labeled according to class c until the misclassification error for that class no longer decreases. The error is computed on the learning set or — if good generalization performance is needed — on a cross-validation set. Details concerning this algorithm can be found in [4].

EXPERIMENTS

The learning procedures described above have been tested successfully on several learning tasks. Some results concerning the penalty method are reported in [3]. The problem that will be used here to illustrate the second method has been proposed in [2]. It consists in recognizing digits composed of seven horizontal and vertical lines in

Table 1: Weight values and their interpretation after learning to classify the digits data

	class	*Weights*							*Symbolic interpretation*
AND units	1	-1	-1	0	-1	-1	0	0	$\neg u \wedge \neg ul \wedge \neg m \wedge \neg dl$
	1	-1	0	0	-1	0	0	-1	$\neg u \wedge \neg m \wedge \neg d$
	2	0	-1	1	0	1	-1	0	$\neg ul \wedge ur \wedge dl \wedge \neg dr$
	3	0	-1	1	1	-1	1	1	$\neg ur \wedge ul \wedge m \wedge \neg dl \wedge dr \wedge d$
	4	-1	1	0	1	-1	0	-1	$\neg u \wedge ul \wedge m \wedge \neg dl \wedge \neg d$
	5	0	0	-1	1	-1	0	1	$\neg ur \wedge m \wedge \neg dl \wedge d$
	6	0	1	-1	0	1	1	0	$ul \wedge \neg ur \wedge dl \wedge dr$
	7	1	-1	0	-1	0	0	-1	$u \wedge \neg ul \wedge \neg m \wedge \neg d$
	8	1	1	1	1	1	1	1	$u \wedge ur \wedge ul \wedge m \wedge dr \wedge dl \wedge d$
	9	1	1	1	0	-1	0	1	$u \wedge ul \wedge ur \wedge \neg dl \wedge d$
	0	0	1	1	-1	1	0	0	$ul \wedge ur \wedge \neg m \wedge dl$

Table 2: Weight values and their interpretation after learning to discriminate between odd and even numbers

	Weights							*Symbolic interpretation*
unit 1	0	0	0	0	1	0	0	$dl \Rightarrow$ even
unit 2	-1	1	0	1	0	0	0	$\neg u \wedge ul \wedge m \Rightarrow$ even

on-of combinations, such as usually displayed on electronic watches and calculators. The problem is made more difficult by assuming that each light has a probability 0.1 of not doing what it is supposed to do. With equiprobable classes, the Bayes misclassification rate for this problem is 0.26. The data set consisted of 200 learning samples, and 2000 test samples.

This problem has been used for testing the probabilistic update algorithm, with incremental construction of the AND layer. The results are shown in Table 1 (The variable names u, ur, ul, m, dr, dl, d are short for "up", "up-right", "up-left", "middle", "down-right", "down-left", and "down", respectively). Eleven AND units have been generated, corresponding to eleven rules which are all correct, given our knowledge of the model that served to generate the data. Note that the minimum number of rules needed to logically characterize the data was 10 in that case, i.e. slightly less that the number obtained. The estimated misclassification rate is 0.32.

Variants of this problem can be constructed by grouping some digits together to form fewer, more complex classes. Table 2 shows the rules obtained for the discrimination of odd from even numbers. Once again, these rules are in agreement with the underlying model. The constructive procedure described above has been run 40 times, yielding 11 times 1 AND unit (corresponding to a 90 % correct rule), 24 times 2 units, and 5 times 3 units, which shows the relatively good robustness of the procedure.

CONCLUSIONS

We have shown how the definition of a correspondence between a symbolic and a connectionist formalism allows not only to introduce prior knowledge in a neural network, but also to extract new or refined knowledge out of the connection weights. In the case of production rules, which has been addressed in this paper, this knowledge extraction can be achieved by applying learning procedures which preserve the interpretation of

the network as computing crisp of fuzzy logical expressions. Beyond the improvement of these procedures, particularly with respect to robustness against local minima, a continuation of this work will be to clarify the situation of this approach compared to the large number of rule generation techniques which have been described in the Machine Learning literature. On the one hand, a comparison with the main symbolic methods (e.g. decision tree generation, etc.) has to be undertaken, from the points of view of explainability and generalization capability of the rules generated. On the other hand, this work might lead to consider some kind of cooperation between symbolic and connectionist techniques.

References

[1] L. Atlas, R. Cole, J. Connor, M. El-Sharkawi, R. J. Marks II, Y. Muthusamy, and E. Barnard. Performance comparisons between backpropagation networks and classification trees on three real-world applications. In D. S. Touretzky, editor, *Advances in Neural Information Processing Systems 2*, pages 622–629. Morgan Kaufmann, San Mateo, CA, 1990.

[2] L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone. *Classification and Regression Trees*. Wadsworth, Belmont, CA, 1984.

[3] T. Denoeux. Generation of symbolic rules in back-propagation networks. In Igor Aleksander and John Taylor, editors, *Artificial Neural Networks II*. North-Holland, Amsterdam, 1992.

[4] T. Denoeux. Production rules generation and refinement in back-propagation networks. Technical Report Neufodi/TR/202/LY03/1, Lyonnaise des Eaux - Dumez / LIAC, Compiegne, France, 1992.

[5] Pierre-Yves Glorennec. Un réseau neuro-flou evolutif. In *Proceedings Neuro-Nîmes '91*, pages 301–314, Nanterre, France, 1991. EC2.

[6] K. Kruschke and J. V. Movellan. Benefits of gain: Speeded learning and minimal hidden layers in back-propagation networks. *IEEE Trans. Systems, Man and Cybernetics*, 21(1):273–280, 1991.

[7] R. Lengellé and T. Denoeux. Optimizing multilayer networks layer per layer without back-propagation. In Igor Aleksander and John Taylor, editors, *Artificial Neural Networks II*. North-Holland, Amsterdam, 1992.

[8] I. K. Sethi. Entropy nets: From decision trees to neural networks. *Proceeding of the IEEE*, 78(10):1605–1613, 1990.

[9] A. C. Tsoi and R. A. Pearson. Comparison of three classification techniques, cart, c4.5 and multi-layer perceptrons. In R. P. Lippman, J. E. Moody, and D. S. Touretzky, editors, *Neural Information Processing 3*, pages 963–969. Morgan Kaufmann, San Mateo, CA, 1991.

[10] A. S. Weigend, D. E. Rumelhart, and B.A. Huberman. Generalization by weight-elimination with application to forecasting. In R. P. Lippman, J. E. Moody, and D. S. Touretzky, editors, *Neural Information Processing 3*, pages 875–882. Morgan Kaufmann, San Mateo, CA, 1991.