Extraction of Knowledge from Data Using Constrained Neural Networks

Raqui Kane¹, Irina Tchoumatchenko², Maurice Milgram¹ *

¹ LRP-CNRS, Université Paris-VI, 4 Place Jussieu, 75252 Paris, CEDEX 05 France

² ACASA, LAFORIA-CNRS, Université Paris-VI, 4 Place Jussieu, 75252 Paris,
CEDEX 05 France

Abstract. This paper deals with two complementary problems: the problem of extracting knowledge from neural networks and the problem of inserting knowledge into neural networks. Our approach to the extraction of knowledge is essentially constraints-based. Local constraints are imposed on the neural network's weights and activities to make neural network units work as logical operators. We have modified two well-known learning algorithms, namely the simulated annealing and the backpropagation, with respect to imposed constraints. In the case of the non-empty domain theory, the knowledge insertion technique is used to impose global constraints to determine the neural network's topology and initialization according to a priori knowledge about the problem under study. The knowledge to be inserted can be expressed as a set of propositional rules. We report simulation results obtained by running our algorithms to extract boolean formulae.

1 Introduction

Last decade artificial neural networks have demonstrated capabilities as efficient mapping devices. The criticism usually made about neural networks is that the way work has little explanation in humanly comprehensible terms. Despite a growing number of research efforts to interpret trained neural networks in logical terms, a general method for the extraction of logical rules from *arbitrary* neural networks has not yet been found. Both propositional [4] [1] and fuzzy [2] logic approaches were used for extracting knowledge from a trained neural network. The main difference between the extraction method presented in this paper and the previously developed ones is the intensive use of constraints. To facilitate knowledge extraction as well as to bias neural learning to explore the most interesting part of a weight space, we adapt the knowledge insertion technique. Some logical formalisms have been mapped into neural networks [4] [3]. Unfortunately, the expressive power of neurally-mapped logic is restricted, as a satisfactory decision for variables handling has not been found.

---

* This work is supported by a grant from the DRET

2 Our Approach to Knowledge Extraction

Extracting knowledge from an arbitrary neural network is recognized as a complex task. One possible approach to the extraction problem is to constrain appropriately a neural network making it logically-interpretable. Naturally, neural network learning algorithm should be modified with respect to the imposed constraints. We use symmetric sigmoid function $f(x) = \tanh(kx)$, where the steepness parameter k is chosen appropriately to allow both a step function approximation and a reasonable time convergency. This choice of a steepness parameter realizes the first constraint on the units' activities, namely the unit-activities are arbitrarily close to -1 or 1 . We call such a unit a *logical unit*. Let i be a logical unit with n inputs $x_1^i \cdots x_n^i$. The output of the unit is defined by the following equation:

$$o^i = f(\sum_{j=1}^n w_{ij} * x_j^i - \theta_i)$$

Let's impose the constraint on the neural network's weights to be in the finite set of values $\{-1, 0, 1\}$. As it directly follows from the conjunction (disjunction) truth table analysis, a so-constrained unit i realizes conjunction (disjunction) from its inputs if and only if $\theta_i = \epsilon \sum_{j=1}^n w_{ij}^2 - 1$ and $\epsilon = 1$ ($\epsilon = -1$), where $\sum_{j=1}^n w_{ij}^2$ represents a number of non-zero unit inputs.

To summarize, we impose the following constraints:

1. $\forall x_j^i, x_j^i \in \{-1, 1\}$
2. $\forall w_{ij}, w_{ij} \in \{-1, 0, 1\}$
3. $\theta_i = \epsilon \sum_{j=1}^n w_{ij}^2 - 1$, where $\epsilon = 1$ ($\epsilon = -1$) if unit i realizes a conjunctive (disjunctive) operator.

So, we can formulate the following recursive method for extracting logical rules:

- For each hidden (output) unit i :
  - Analyse the bias of the unit:
    - * If its bias is positive, Then unit realizes a conjunction of its inputs
    - * If its bias is negative, Then unit realizes a disjunction of its inputs
    - * If its bias is null, Then unit is inactive
  - For each weight coming to the unit:
    - * If weight w_{ij} from unit j to unit i is negative, Then the logical operator realized by unit i is applied to $\neg x_j^i$
    - * If weight w_{ij} from unit j to unit i is positive, Then the logical operator realized by unit i is applied to x_j^i
    - * If weight w_{ij} from unit j to unit i is null, Then x_j^i is not considered

Example 1. Given the simple one-hidden-layer neural network presented in Fig.1. Running recursively our algorithm, we will obtain:

- for the first layer:
  - the bias of unit 1 is positive, so unit 1 realizes the conjunction of its inputs $\neg x_1 \wedge \neg x_2 \wedge x_3$, where $\neg x_1$ and $\neg x_2$ result from the negative weights

- the bias of unit 2 is negative, so unit 2 realizes the disjunction of its inputs $x_1 \vee \neg x_3$, where $\neg x_3$ results from the negative weight and x_2 is not included due to its zero weight
- for the second layer:
  - the bias of unit 3 is negative, so unit 2 realizes the disjunction of its inputs $(\neg x_1 \wedge \neg x_2 \wedge x_3) \vee \neg(x_1 \vee \neg x_3)$ and, after transformations, we obtain $\neg x_1 \wedge x_3$

Fig. 1. Extracting rules from a sample constrained neural network

3 Modifications of the Learning Algorithm

The main problem for extracting logical rules using the proposed constrained-based method is to modify a standard neural network learning algorithm to keep the neural network's weights into a finite set of values $\{-1, 0, 1\}$. In this section, we propose two minor, but efficient modifications of the simulated annealing learning algorithm and the penalty function modification for the back-propagation. To validate each method, we have trained networks with different architectures to learn boolean formulae.

3.1 Modifications of the Simulated Annealing Algorithm

– *Modification1.* To force a neural network to respect the imposed constraints, we change the initialization and the update steps of the standard simulated annealing learning algorithm as follows:

- *Initialization:* Select randomly initial weights in $\{-1, 0, 1\}$. Compute the resulting biases $\theta_i = \epsilon \sum_{j=1}^n w_{ij}^2 - 1$, where ϵ is randomly chosen in $\{-1, 1\}$.
- *Update:* A fixed number of values is randomly chosen in the set of weights and biases. If a bias is chosen, we change its sign. If a weight is chosen, we randomly change its value in $\{-1, 0, 1\}$ and then compute the resulting bias.

– *Modification2 (simulated annealing controled by gradient)*. Changes of weights can be controled by the sign of the gradient. For the weights from $\{-1, 0, 1\}$, the variation can be performed in two cases:

- the gradient is positive and the initial weight is -1 or 0
- the gradient is negative and the initial weight is 1 or 0

Thermal equilibrium is reached if there are no more changes possible when computing the gradients of the weights.

Example 2. The one-hidden-layer neural network $3 - 3 - 1$ with 3 input units, 3 hidden units and 1 output unit, was trained to learn the boolean formula $(x_1 \wedge x_2) \vee \neg x_3$ using the above-described modifications of the simulated annealing. Both algorithms have achieved the desired results, namely the weights were kept in $\{-1, 0, 1\}$. The first algorithm converged in about 2000 iterations, while the second one converged in about 1000 iterations. The neural network's state after training and its interpretation in logical terms using our extraction method are presented in Table 1.

Table 1. Extraction of a boolean formula from the $3 - 3 - 1$ neural network trained with the modified simulated annealing algorithms

Links from First To Second Layer	Links from Second to Third Layer	Extracted Formula
$w_{11}^{(1)} = -1.000000$ $w_{12}^{(1)} = -1.000000 \neg x_1 \vee \neg x_2$ $w_{13}^{(1)} = 0.000000$ $\theta_1^{(1)} = -3.000000$	$w_{11}^{(2)} = -1.000000$ $w_{12}^{(2)} = 0.000000$ $\neg x_1 \vee x_3$	$(x_1 \wedge x_2) \vee \neg x_3$
$w_{21}^{(1)} = 0.000000$ $w_{22}^{(1)} = 1.000000 x_2$ $w_{23}^{(1)} = 0.000000$ $\theta_2^{(1)} = 1.000000$	$w_{13}^{(2)} = 1.000000$	
$w_{31}^{(1)} = 0.000000 \neg x_3$ $w_{32}^{(1)} = 0.000000$ $w_{33}^{(1)} = -1.000000$ $\theta_3^{(1)} = 1.000000$	$\theta_1^{(2)} = -3.000000$	

3.2 Modification of the Backpropagation Algorithm

We have found that the following modification of the backpropagation algorithm provides both better performance and flexibility. We penalize any unwanted weight changes during training by adding an extra term $P(W)$ to the standard error function $E(W)$. The penalty term $P(W)$ will force weights to approach -1 or 0 or 1 during training. Thus, the function to minimize using the standard gradient descent method becomes $E(W) + \lambda P(W)$, where λ is an adjustable

parameter to get the best convergence properties ($\lambda = 0.01$). We have chose $P(W)$ to be a differentiable juxtaposition of 5 paraboles:

$$P(x) = 8(x+1)^2 \mathbb{I}(x)_{]-\infty; -\frac{3}{4}]} + (1 - 8(x + \frac{1}{2})^2) \mathbb{I}(x)_{[-\frac{3}{4}; -\frac{1}{4}]} + 8x^2 \mathbb{I}(x)_{[-\frac{1}{4}; \frac{1}{4}]} + (1 - 8(x - \frac{1}{2})^2) \mathbb{I}(x)_{[\frac{1}{4}; \frac{3}{4}]} + 8(x-1)^2 \mathbb{I}(x)_{[\frac{3}{4}; \infty[}$$

Example 3. The 3 - 2 - 1 neural network was trained by the modified back-propagation algorithm to learn boolean formula $(x_1 \wedge x_2) \vee \neg x_3$. Convergence was reached as fast as with standard backpropagation. The neural network's state after training and its interpretation in logical terms according to our extraction method are presented in Table 2.

Table 2. Extraction of a boolean formula from the 3 - 2 - 1 neural network trained with the modified backpropagation

Links from First To Second Layer		Links from Second to Third Layer	Extracted Formula
$w_{11}^{(1)} = 1.083391$ $w_{12}^{(1)} = -0.063624$ $w_{13}^{(1)} = -1.143470$ $\theta_1^{(1)} = -1.0$	$x_1 \vee \neg x_3$	$w_{11}^{(2)} = 0.813960$ $w_{12}^{(2)} = 0.920622$	$(x_1 \wedge x_2) \vee \neg x_3$
$w_{21}^{(1)} = 0.0430180$ $w_{22}^{(1)} = 0.920361$ $w_{23}^{(1)} = -0.822301$ $\theta_2^{(1)} = -1.0$	$x_2 \vee \neg x_3$	$\theta_1^{(2)} = 1.0$	

Remark. We encountered the problem of tautologically true rule extraction after training the network both by the modified simulated annealing algorithms and by the modified backpropagation. This problem is related to the initialization of the neural network and the local mimina problem. The lowest rate of tautologically true rule extraction was obtained for computational experiments with the modified backpropagation.

4 Inserting Rules into a Constrained Neural Network

Inserting a logical rule into a neural network means building a neural network which realizes the same input-output mapping as the rule to be inserted. Below, we propose a method for inserting simple propositional rules into constrained . This method is based on the truth-table analysis of the rule to be inserted.

- *Inserting conjunctive rules.* Using logical units with weights into $\{-1, 1\}$, it is straightforward to put a logical rule

$$A : -B_1 \wedge \dots \wedge B_p \wedge \neg C_1 \wedge \dots \wedge \neg C_n$$

with p positive and n negative literals into correspondance with a logical unit realizing the same mapping. This rule is translated into a neural network unit by setting weights on all links corresponding to positive literals to 1 and weights on all links corresponding to negative literals to -1 . Analysing the truth table of a conjunctive logical rule, it is easy to see that for the ideal step function the bias of the unit corresponding to the rule's consequent can be chosen from $[p - n - 2; p - n]$. Practically, to insure good convergence properties, we use a quasi-step function; hence it is reasonable to set the bias equal to $p - n - 1$.

– *Inserting disjunctive rules.* Mapping of disjunctive logical rule with p positive and n negative literals

$$A : -B_1 \vee \dots \vee B_p \vee -C_1 \vee \dots \vee -C_n$$

is done as follows. Weights on all links corresponding to positive literals are equal to 1, weights on all links corresponding to negative literals are equal to -1 and the bias can be set to any values from $[-p - n; -p - n + 2]$. We set it equal to $-p - n + 1$ to provide better convergence properties.

Using this approach based on the thruth table analysis, we are able to insert any propositional rule into our constrained neural network.

5 Conclusion

The method for knowledge extraction from strongly constrained neural networks is presented and validated on the simple problem of boolean formulae extraction. Two widely used learning algorithms were modified with respect to the imposed constraints. It was found that though slightly modified simulated annealing learning algorithm works reasonably well, the proposed penalty function modification for standard backpropagation provides both better performance and flexibility. To facilitate neural learning as well as further rule extraction for a given problem the propositional rule insertion technique was proposed. Currently we are working on validation of our extraction/insertion method on the real-world problem of the protein secondary structure prediction. A constraints-based method for numerical dependencies extraction is under study.

References

1. Bochereau, L., Bourguine, P.: Extraction of semantic features from a multilayer neural network, Proceedings of the IJCNN90, Washington DC.
2. Masuoka, R., Watanabe, N., Kawamura, A.: Neuro-fuzzy system - fuzzy inference using a structured neural network. In Proc. of the International Conference on Fuzzy Logic and Neural Networks, 173–177, 1990.
3. Sethi, I.K.: Entropy nets: From decision trees to neural networks. Proc. of the IEEE, vol.78, No.10, 1990.
4. Towell, G.G., Shavlik, J.W.: The extraction of refined rules from knowledge-based neural networks. Submitted to Machine Learning (8/91).

This article was processed using the $\text{\LaTeX}$ macro package with LLNCS style