A HYBRID CONNECTIONIST-SYMBOLIC APPROACH FOR REAL-VALUED PATTERN CLASSIFICATION

Lina, Lim Tong Ming, Leow Soo Kar
Monash University Malaysia, No.2 Jalan Kolej, Bandar Sunway, 46150 Petaling Jaya, Selangor, Malaysia. Email: lynn81@gmail.com leow.soo.kar@infotech.monash.edu.my lim.tong.ming@infotech.monash.edu.my

Abstract: Knowledge-Based Artificial Neural Networks (KBANNs) offer a means for combining symbolic and connectionist approaches into a hybrid methodology capable of dealing with small datasets and requires shorter training time when compared to conventional artificial neural networks (ANNs). These approaches were developed mainly for binary-valued domain problems and when applied to real-valued data, many tedious transformation processes need to be undertaken before it could be used. In this paper we propose to extend KBANN to a Real Knowledge-Based Neural Network (RealKBNN) that is able to handle real-valued data directly, making it a more powerful and realistic methodology to be applied to real-world problem domains. Four real-valued datasets (one artificial, 2 medical and 1 classification datasets) were used to evaluate RealKBNN. The results showed that RealKBNN performs very well in terms of its ability to correctly classify unseen examples as well as its suitability for classification of scarce and complex real data.

Key words: Knowledge based Artificial Neural Networks, Hybrid systems, pattern classification

1. INTRODUCTION

Symbolic and connectionist approaches are two major techniques to machine learning. They began independently but in the mid 1990's researchers in these two areas started investigating ways of integrating both

approaches [2] to explore if better generalization can be attained when correct background knowledge is available. Imperfect domain knowledge is fed to the integrated system and then revised by training with available examples. Knowledge learned by the system is extracted as revised knowledge. This revised knowledge can then be used to provide better insight and understanding of the problem at hand.

Current knowledge based neural network systems work mainly on binary inputs. However, real world applications rarely contain only binary data. The way to deal with real-valued domains is to convert real data into the binary form but this usually increases the number of inputs and prolongs the training time as well as degrading the generalization capability of the network.

In this paper, we present a proposed technique based on KBANN, to translate inference rules that accepts real-valued inputs into neural networks. The remainder of the paper is organized as follows: Section 2 provides a brief discussion of related work in Knowledge-Based Neural Networks (KBNNs). Following this, section 3 describes the proposed technique called RealKBNN. In section 4, we present the experimental details and results to evaluate the effectiveness and performance of RealKBNN. Finally, section 5 summarizes this paper.

2. RELATED WORK

Rule Based Neural Networks (RBNN) use prior knowledge if-then inference rules. Input data variables are mapped as input units, target concepts or final consequences are mapped as output units and intermediate concepts are mapped as hidden units. The initial domain rules determine how the concepts or units are linked and how the weights are allocated to the links. KBCNN [4], (Knowledge-based conceptual neural network), KBANN [9], Sordo's KBANN [6] (knowledge based artificial neural networks) are some examples of RBNN. KBANNs accept propositional and acyclic rules and their input features are either in nominal or binary format. This restricts KBANNs in many ways. One obvious restriction is that it is only suitable for binary input data and thus is limited in its applicability.

In an attempt to use KBANN for real-valued problems, Sordo et. Al. [7] modified KBANN to accept real-valued inputs by preserving all rule-to-network translation algorithms and applied it to scarce and complex problem domain like the P Magnetic Resonance Spectroscopy (MRS) data. Sordo's KBANN treated real-valued numbers as positive antecedents and set ω as the weight on all links from the real-valued input nodes. Using this technique, she reported some encouraging results.

3. REALKBNN

Our proposed approach, RealKBNN is based on KBANN. First, domain theory in the form of if-then-else rules for real-valued inputs is translated into the forms of neurons, weights and thresholds. The novelty of this technique is in its manipulation of weight and threshold parameters by eliminating the hidden layer encoding the conjunctive relationship. This step speeds up the training time and convergence time since eliminating a layer in the neural network reduces its complexity and lessens the amount of the computational work.

After the rules are translated, the neural network is formed and is then initialized with some predetermined parametric values. The initialized network is then refined to a final network to classify unseen examples. Figure 1 shows the RealKBNN framework.

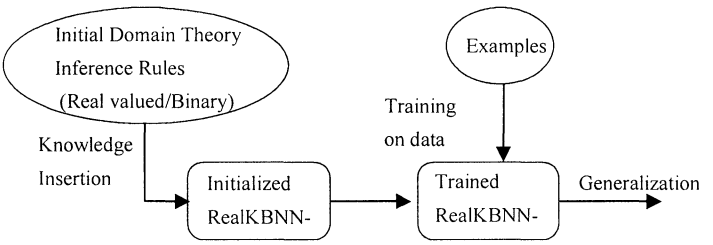


Figure. 1. RealKBNN Framework

In current knowledge refinement systems, the networks created may contain many layers depending on the domain knowledge rules used for creating the network. There are intermediate concepts in the domain knowledge and each concept is mapped as a node in the neural network and the intermediate concepts may be mapped as nodes in different layers according to the dependency of these concepts. This is a major problem because feed forward neural network learning algorithms are not good at training deep networks.

To overcome this drawback, the RealKBNN-net has only three layers, an input layer, a hidden layer and an output layer. The connection between the input layer and the hidden layer represents the conjunctive relationships while the connection between the hidden layer and the output layer represents the disjunctive relationships. For domain knowledge rules that consist of many intermediate concepts, some additional steps are required to rearrange or rewrite the propositional rules to eliminate these intermediate concepts and have a mapping only from input attributes to consequences that lead to the final results. The antecedents of a rule can be a set of arbitrary

linear conditions on the input features. For instance, the precondition of a rule could be expressed as linear constraints on the values of the input feature. Likewise, the output of the rule can be a set of arbitrary linear conditions on the output activations. For simplicity, we consider only rules where the precondition is given by a set of intervals. An example of such rules is *"IF* $A \leq a$ *and* $B \leq b$ *then* C*"*

3.1 Theoretical Results

Before we describe the knowledge integration module of RealKBNN, we present some theoretical results to support the validity of RealKBNN. The following notation is used:

$F(x) = \dfrac{1}{(1+e^{-x})}$, the standard logistic activation function.

w_j^g the weight on links corresponding to "greater than-equal to" inequality dependencies.

w_j^l the weight on links corresponding to "less than-equal to" inequality dependencies.

Y_i the antecedent in Domain D_i corresponding to "greater than-equal to" inequality dependencies.

y_i the real input value of the antecedent Y_i.

b_i the right-hand side value of Y_i corresponding to "greater than-equal to" inequality dependencies.

X_i the antecedent in Domain D_i corresponding to "less than-equal to" inequality dependencies.

x_i the real input value of the antecedent X_i.

a_i the right-hand side value of X_i corresponding to "less than-equal to" inequality dependencies.

Theorem 1. Mapping conjunctive rules of the form

"If $\underset{i}{\wedge}(X_i \leq a_i) \underset{j}{\wedge}(Y_j \geq b_j)$ then C" into a neural network using:

(i) $w_j^g = -w_i^l = \omega > 0$ (1)

(ii) $\theta = \left(\sum_{i=1}^{n} a_i - \sum_{i=1}^{m} b_i \right) \omega$ (2)

creates a network that accurately encodes the rules.

Proof. This theorem is proved by showing that these mappings are correct in the following three situations:

(A) The unit encoding the consequent of the rule is active when all of the antecedents are satisfied.
(B) The unit encoding the consequent of the rule is inactive when none of the antecedents is satisfied.
(C) When the unit encoding the consequent of the rule is inactive, then at least one of the antecedents is not satisfied.

Case A: Assume that all of the antecedents are satisfied and show that the unit encoding the consequent is active. When all of the antecedents are true, it means that all the inputs satisfy their respective inequality conditions.
This implies: $x_i \le a_i$, i = 1,...n and $y_j \ge b_j$, j = 1,....m
Summing both sides of all inequalities we get,

$$\sum_{i=1}^{n} x_i \le \sum_{i=1}^{n} a_i \quad \text{i.e.} \quad \sum_{i=1}^{n} a_i - \sum_{i=1}^{n} x_i \ge 0 \tag{3}$$

$$\sum_{j=1}^{m} y_j \ge \sum_{j=1}^{m} b_j \quad \text{i.e.} \quad \sum_{j=1}^{m} y_j - \sum_{j=1}^{m} b_j \ge 0 \tag{4}$$

Thus, the activation function is given as

$$F\left(\sum_{i=1}^{n} x_i w_i^{\ l} + \sum_{j=1}^{m} y_j w_j^{\ g} + \theta\right) = F\left(\sum x_i(-\omega) + \sum y_j(\omega) + \left(\sum a_i - \sum b_j\right)\omega\right)$$

$$= F\left(\omega\left(\left(\sum a_i - \sum x_i\right) + \left(\sum y_j - \sum b_j\right)\right)\right)$$

$$= F(+ve\ value) > 0.5$$

Hence, the output unit is activated.

Case B: Assume that all antecedents of the rule are not satisfied and show that the unit encoding the consequent is inactive.
When all antecedents are false, it means that all the inputs do not satisfy their respective inequality conditions.
This implies: $x_i > a_i$, i = 1...n $y_j < b_j$, j = 1...m

$$\forall_i (x_i > a_i) \text{ implies } \sum_{i=1}^{n} x_i > \sum_{i=1}^{n} a_i \quad \text{i.e.} \sum_{i=1}^{n} a_i - \sum_{i=1}^{n} x_i < 0 \tag{5}$$

$$\forall_j (y_j < b_j) \text{ implies } \sum_{j=1}^{m} y_j < \sum_{j=1}^{m} b_j \text{ i.e.} \sum_{j=1}^{m} y_j - \sum_{j=1}^{m} b_j < 0 \tag{6}$$

Hence, the activation function is as follows:

$$F\left(\sum_{i=1}^{n} x_i w_i^{\ l} + \sum_{j=1}^{m} y_j w_j^{\ g} + \theta\right)$$
$$= F\left(\sum x_i(-\omega) + \sum y_j(\omega) + \left(\sum a_i - \sum b_j\right)\omega\right)$$
$$= F\left(\omega\left(\left(\sum a_i - \sum x_i\right) + \left(\sum y_j - \sum b_j\right)\right)\right)$$
$$= F(\textit{-ve value}) < 0.5$$

Hence, the output unit is deactivated.

Case C: Assume that the consequent of the rule is inactive, then the activation function is

$$F\left(\omega\left(\left(\sum a_i - \sum x_i\right) + \left(\sum y_j - \sum b_j\right)\right)\right) < 0.5$$

This implies that $\left(\omega\left(\left(\sum a_i - \sum x_i\right) + \left(\sum y_j - \sum b_j\right)\right)\right) < 0$

i.e. $\left(\sum_{i=1}^{n} a_i - \sum_{i=1}^{n} x_i\right) + \left(\sum_{j=1}^{m} y_j - \sum_{j=1}^{m} b_j\right) < 0$ since $\omega > 0$

Hence, at least one of the antecedents is not satisfied.
Case C of theorem 1 shows that by defining the threshold function as in (2), it eliminates the occurrence of false negative errors (type 1 errors). However, the function is unable to eliminate false positive errors (type II errors). Despite this fact, the threshold function performed creditably in our experiments and proved empirically to be somewhat powerful in translating the inequality rules into the neural network for training. Even though the rules translated are very rough rules, the network manages to learn and classify the output very accurately.

Theorem 2. Mapping disjunctive rules of the form "if $\vee_i X_i \geq 0.5$ then C" into a neural network using:

(i) $w_i = \omega > 0$ for $i = 1, 2, \ldots n$ (7)

(ii) $\theta = -\dfrac{\omega}{2}$ (8)

creates a network that accurately encodes the disjunctive rules.

Proof. This theorem is proved by showing that these mappings are correct in the following two situations:

(A) The unit encoding the consequent of the rule is active when at least one of the antecedents is satisfied.

(B) When the unit encoding the consequent of the rule is inactive, then none of the antecedents is satisfied.

Case A: Assume that at least one of the antecedents is satisfied and show that the unit encoding the consequent of the rule is active.

Let x_i be the input value of an antecedent, $i = 1....n$.

Let w_i be the weight associated with x_i.

The minimum activation value, to be considered as active is 0.5. Assume at least one of the antecedents x_p is active.

Given $x_i \geq 0$, $i = 1,2,...(p-1),(p+1),.....,n$ and , assume $x_p \geq 0.5$

Then $\sum_{i=1}^{n} x_i \geq 0.5$.

This implies $\sum_{i=1}^{n} x_i w_i - \frac{\omega}{2} = \sum_{i=1}^{n} x_i \omega - \frac{\omega}{2} > 0$

Therefore, $F\left(\sum_{i=1}^{n} x_i \omega - \frac{\omega}{2}\right) = F(+ve\ value) > 0.5$

Hence, the unit will be activated.

Case B: Assume that the unit encoding the consequent of the rule is inactive and show that none of the antecedents is active.

$$F\left(\sum_{i=1}^{n} x_i w_i + \theta\right) < 0.5$$

$$\sum_{i=1}^{n} x_i \omega - \frac{\omega}{2} < 0$$

$$\omega\left(\sum_{i=1}^{n} x_i - \frac{1}{2}\right) < 0$$

$$\sum_{i=1}^{n} x_i < 0.5$$

Thus, $\forall i$, $x_i < 0.5 \Rightarrow$ none of the antecedents is active.

3.2 Translating conjunctive rules

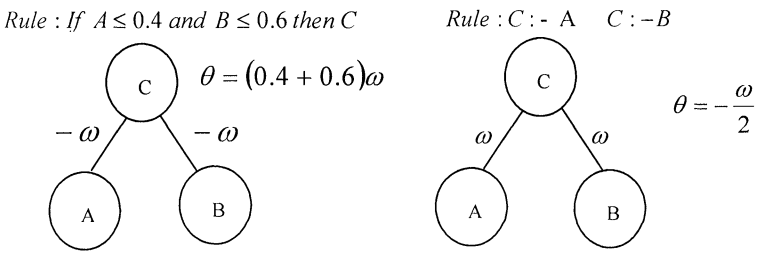
The weight and threshold values depend on the inequality ("≤" or "≥") of the antecedent to be inserted. For rules with "≤" inequality, the weight ω is assigned a negative value (e.g. -4) and the threshold θ is equal to $\sum$ +(RHS constant of inequality)ω. As for rules with "≥" inequality, it works the opposite way, the weight ω is given positive value (e.g. 4) while

threshold value θ is set as $\sum$ –(RHS constant of inequality) ω. Figure 3 illustrates how conjunctive and disjunctive rules are translated into ANNs.

3.3 Translating disjunctive rules

Disjunctive rules are translated into a neural network by setting weights on the link corresponding to a disjunctive antecedent to ω. The threshold of the unit encoding the consequent is set to be $-\frac{\omega}{2}$. Figure 2 illustrates an example of the translation of the rules.

Figure 2. Conjunctive Rules Disjunctive Rules



This is a reasonable strategy as the output is activated whenever one of the antecedents is true. After the rules are translated into the neural networks, the next step would be to refine the RealKBNN-net so that they are capable of predicting the unseen examples. Generally, learning in artificial neural networks means the process of modifying connection weights in order to achieve correct inference behavior [4]. In RealKBNN we use a generalized back-propagation algorithm as the learning algorithm.

4. PERFORMANCE OF REALKBNN

The performance of RealKBNN was evaluated using three groups of experiments operating on different datasets. The first experiment was conducted on the Wisconsin Breast Cancer Data (WBCD) set obtained from the UCI Repository [5]. The second test was run on an artificial dataset while the third experiment was run on the Iris data [3] and a Diabetes data set [8] obtained from the UCI Repository.

All the experiments were conducted using the hold-out method [1]. In short, the data is divided into two sets. One set whose elements were

randomly selected, is used for training the classifier while the other set is reserved for testing the generalization capability of the trained classifier. Only 10% of the data is used as the training data while the other 90% were used for testing. This proportion was chosen to investigate how well RealKBNN can perform with scarce data

4.1 Wisconsin Breast Cancer Data

Table 1. Performance of RealKBNN on WBCD

Method	Accuracy	Training Cycle	MSE
RealKBNN	96.2%	7	0.083
KBANN	27.1%	7	0.670
SordoKBANN	35.3%	6	0.668
ANN	90.5%	158	0.093

The first experiment was conducted to evaluate RealKBNN's performance against 3 other methods, namely, KBANN, Sordo's KBANN (SordoKBANN) and the conventional ANN. The results showed that RealKBNN performed much better compared to the other three methods. KBANN was very inefficient in dealing with real-valued data showing only 27.1% compared to 96.2% average accuracy of RealKBNN. A small improvement was seen in Sordo's KBANN technique. It only improved the KBANN's performance by a mere 8%. The conventional ANN performed relatively well (90.5% average accuracy), but it was much more computationally expensive as it required 158 training cycles to converge while RealKBNN only needed 7 training cycles.

4.2 Artificial Data

Table 2. Performance on Artificial Data

Noise Level (%)	Accuracy	Training Cycle	MSE
0%	100%	6	0.01
10%	100%	6	0.098
20%	100%	6	0.201
30%	99.6%	6	0.289
40%	95.4%	6	0.417
50%	90.9%	6	0.5

The second experiment was conducted on an artificial dataser, which was generated to test the generalizability of RealKBNN. Table 2 presents the results of the experiment at different noise levels. The results showed that RealKBNN is a consistent and reliable system as it could learn from mistakes that were present in the training data. At the 30% noise level, RealKBNN can still learn perfectly and shows 100%

accuracy. Two other experiments were conducted on real-world classification problems, the Iris database and a Diabetes data set; both obtained from UCI repository. The objective of the experiments was to conduct a comparison of RealKBNN with other techniques available in the literature. For the Iris Data, RealKBNN performed credibly with an average accuracy of 96.3% when compared to the other methods. On the Diabetes Data, RealKBNN out performed all the other techniques achieving an average accuracy of 76.6%.

5. CONCLUSION

Overall, RealKBNN has dealt with real-valued data very successfully and is also a robust technique in dealing with noise levels present in data. Moreover, RealKBNN performed consistently in all experiments as many runs were conducted and the results produced were consistent with a low accuracy variance, error rates and training cycles. This observation is important especially when applied to real-world critical problems where system crashes or unpredictable response times or results cannot be tolerated.

REFERENCES

1. Anonymous (2004), "Lecture 13: Validation", Retrieved: September 24, 2004, from http://research.cs.tamu.edu/prism/lectures/iss/iss_l13.pdf.
2. Boz, O. (1995), "Knowledge Integration and Rule Extraction in Neural Networks", Ph.D Proposal, Lehigh University.
3. Fisher, R. A. (1936), "The use of multiple measurements in taxonomic problems", *Annual Eugenics*, vol. 7, Part II, pp. 179-188.
4. Fu, L-M. and Fu, L-C. (1990), "Mapping Rule-based Systems into Neural Architecture", *Knowledge-Based Systems.*, vol. 3, no. 1, pp.48-56.
5. Mertz, C. J. and Murphy, P. M. (2004), *UCI repository of machine learning databases*, Retrieved: July 25, 2004, from http://www.ics.uci.edu/pub/machine-learning-data-bases.
6. Sordo, M. (1999). "A Neurosymbolic Approach to the Classification of Scarce and Complex Data", Ph.D thesis, University of Sussex
7. Sordo, M., Buxton, H. and Watson, D. (2001) "A Hybrid Approach to Breast Cancer Diagnosis", In Wilde, D. and Jain (Eds), *Practical Applications of Computational Intelligence Techniques*, Kluwer.
8. Smith, J. W., Everhart, J. E., Dickson, W. C., Knowler, W. C. and Johannes, R. S. (1998), "Using the ADAP learning algorithm to forecast

the onset of diabetes mellitus", In *Proceedings of the Symposium on Computer Applications and Medical Care*, IEEE Computer Society Press, pp.261-265.

9. Towell, G. and Shavlik, J. W. (1994), "Knowledge-Based Artificial Neural Networks", *Artificial Intelligence*, vol. 70, no. 1-2, pp.119-165.