Improving Rule Extraction from Neural Networks by Modifying Hidden Layer Representation

Thuan Q. Huynh

***Abstract*— This paper describes a new method for extracting symbolic rules from multilayer feedforward neural networks. Our approach is to encourage backpropagation to learn a sparser representation at the hidden layer and to use the improved representation to extract fewer, easier to understand rules. A new error term defined over the hidden layer is added to the standard sum of squared error so that the total squared distance between hidden activation vectors is increased. We show that this method helps extract fewer rules without decreasing classification accuracy in four publicly available data sets.**

I. Introduction

Error backpropagation is the most widely used supervised learning method for neural networks and has achieved success in many classification and prediction applications. A typical network has an architecture consisting of an input layer, one or more hidden layers, and an output layer (Figure 1). There are several variants of the algorithm, all driven by minimizing the sum of squared error $E = \sum(correct_i - output_i)^2$ at output units. The network learns a mapping between the input and output units, while the hidden units and the weights between them and other units contain the network's internal representation of the input. This distributed representation as large matrices of floating point numbers makes it very difficult for a person to understand what the network has learned. This difficulty has inspired substantial past research on how to extract symbolic, human-readable rules from a network so that we can be more confident about its classifications and understand more about what has been learned from the data. In spite of a large amount of work addressing this issue ([1], [2], [3], [4], [5]), the results obtained are still very limited.

There are three main approaches that have been taken in past work on rule extraction from neural networks: pedagogical, decompositional and eclectic (hybrid of the other two). Pedagogical methods consider a neural network as a black-box oracle that provides cleaner and more class labels. They extract input-output rules without looking at the units and weights. Decompositional methods investigate hidden units and weight matrices to produce rules that follow the internal working of the networks. Beside good classification accuracy, having a smaller number of rules is a very important criterion for rule extraction algorithms so that a human can understand their content easier.

Many of the decompositional approaches, including ours, first extract the rules that explain the mapping between the hidden unit activations and the output and then extract the rules governing the input - hidden layer relationship. These rules are then combined to produce the final input - output rules. This approach tries to produce simpler and fewer rules at the hidden - output layer so that it can generate fewer rules overall. The number of rules depends heavily on how the hidden unit activation vectors - encodings of the inputs - are arranged in space. During learning, backpropagation is free to create any encoding scheme over the hidden units as long as the final error at the output layer is minimized. This presents a problem to decompositional methods because sometimes the hidden layer representations of the input patterns are so complex or distributed that a lot of rules are required to explain the hidden - output layer mapping. In this paper we propose a new error term to augment the standard sum of squared error. The new error term encourages backpropagation to learn a sparser encoding over the hidden layer in which vectors of hidden unit activations are further apart than they would normally be with standard backpropagation. Gradient descent is used to modify error backpropagation when using this new error term. Our hypothesis was that this would result in fewer rules. Our computational experiments thus compared the same rule generation procedure using error backpropagation with and without the new error term to assess its impact on the number of rules generated.

The rest of this paper is organized as follows: Section II describes the new error term and the rule extraction algorithm. Experimental results are presented in Section III. Section IV gives conclusion and discussion.

II. The Algorithm

In this section, we first describe the new error term that makes the hidden unit activation representation sparser and an efficient way to compute the derivative of this term for training using error backpropagation. Then we present an algorithm to extract symbolic rules that utilizes the improved representation at the hidden layer.

A. New error term

In this work we are interested in extracting rules from multilayer feedforward neural networks with one hidden layer as shown in Figure 1.

The activation of the j^{th} hidden unit when the p^{th} instance is presented is calculated as the logistic function of the weighted sum of inputs:

$$a^p_{H_j} = \sigma(\sum_{i=1}^{input} w_{ji} x^p_i)$$

where

- x^p_i is the i^{th} input unit value of the p^{th} instance.

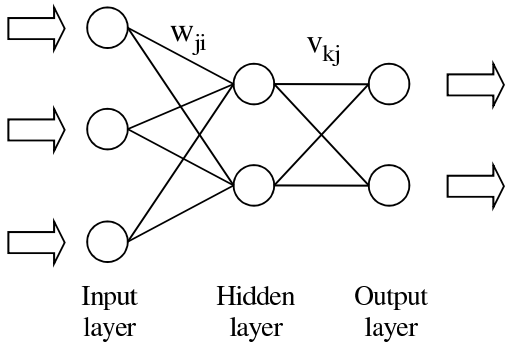

Fig. 1. A typical fully connected feedforward neural network

- w_{ji} is the weight from the i^{th} input unit to the j^{th} hidden unit.
- $\sigma()$ is the logistic function.

The activation of the k^{th} output is calculated as the logistic function of the weighted sum of hidden unit activations:

$$a^p_{O_k} = \sigma(\sum_{j=1}^{hidden} v_{kj} a_{H_j})$$

where v_{kj} is the weight from the j^{th} hidden unit to the k^{th} output unit.

The usual error function computed over the output units is:

$$E = \sum_{p=1}^{N} \sum_{k=1}^{output} (t^p_k - a^p_{O_k})^2$$

where t^p_k is the target output for the k^{th} output unit when the p^{th} input pattern is presented, and N is the size of the input data set (number of input - output pairs).

We introduce a penalty term E_2 that decreases as the hidden unit activation vectors are further apart.

$$E_2 = -\frac{1}{2} \sum_{p=1}^{N} \sum_{q=1}^{N} \sum_{k=1}^{hidden} (a^p_{H_k} - a^q_{H_k})^2$$

where $a^p_{H_k}$ is the activation of the k^{th} hidden unit when the p^{th} input sample is presented. E_2 is the sum over all pairs (p, q) of the squared Euclidean distance between two hidden activation vectors for the p^{th} and q^{th} input patterns. The negative sign ensures that when neural network training minimizes the error, it will maximize the distances between the hidden layer vectors.

The new total error function guiding learning is:

$$E' = \alpha E + \beta E_2$$

where $\alpha, \beta > 0, \alpha + \beta = 1$. Note that the double sum over p and q can make E_2 quite large relative to E, so β must be quite small to scale E and E_2 appropriately.

In order to train the network with error backpropagation, we need to compute $\frac{\partial E'}{\partial w} = \alpha \frac{\partial E}{\partial w} + \beta \frac{\partial E_2}{\partial w}$. First, the standard term $\frac{\partial E}{\partial w}$ can be computed efficiently as in [6].

We have $\frac{\partial E_2}{\partial v_{kj}} = 0$ with v_{kj} being the weight to the k^{th} output unit from the j^{th} hidden unit because E_2 does not have any v_{kj} component.

With w_{ji} being the weight from the i^{th} input unit to the j^{th} hidden unit, we have derived an efficient way to compute the derivative $\frac{\partial E_2}{\partial w_{ji}}$ as follows:

Let $E_2 = \sum_p E^p_2$ where:

$$E^p_2 = -\frac{1}{2} \sum_{q=1}^{N} \sum_{k=1}^{hidden} (a^p_{H_k} - a^q_{H_k})^2$$

Then we can calculate the gradient of the error E^p_2 with respect to weight w_{ji}:

$$\frac{\partial E^p_2}{\partial w_{ji}} = \sum_{k=1}^{hidden} \frac{\partial E^p_2}{\partial a^p_{H_k}} \frac{\partial a^p_{H_k}}{\partial w_{ji}}$$

Because $\frac{\partial a^p_{H_k}}{\partial w_{ji}} = 0$ with $k \neq j$, we have:

$$\begin{aligned}
\frac{\partial E^p_2}{\partial w_{ji}} &= \frac{\partial E^p_2}{\partial a^p_{H_j}} \frac{\partial a^p_{H_j}}{\partial w_{ji}} \\
&= -\frac{1}{2} \sum_{q=1}^{N} \sum_{k=1}^{hidden} \frac{\partial (a^p_{H_k} - a^q_{H_k})^2}{\partial a^p_{H_j}} \frac{\partial a^p_{H_j}}{\partial w_{ji}} \\
&= -\frac{1}{2} \sum_{q=1}^{N} \sum_{k=1}^{hidden} 2(a^p_{H_k} - a^q_{H_k}) \left(\frac{\partial (a^p_{H_k} - a^q_{H_k})}{\partial a^p_{H_j}} \right) \frac{\partial a^p_{H_j}}{\partial w_{ji}} \\
&= -\sum_{q=1}^{N} \sum_{k=1}^{hidden} (a^p_{H_k} - a^q_{H_k}) \left(\frac{\partial a^p_{H_k}}{\partial a^p_{H_j}} - \frac{\partial a^q_{H_k}}{\partial a^p_{H_j}} \right) \frac{\partial a^p_{H_j}}{\partial w_{ji}}
\end{aligned}$$

With $k \neq j$, we have $\frac{\partial a^p_{H_k}}{\partial a^p_{H_j}} \frac{\partial a^p_{H_j}}{\partial w_{ji}} = 0$ and $\frac{\partial a^q_{H_k}}{\partial a^p_{H_j}} \frac{\partial a^p_{H_j}}{\partial w_{ji}} = 0$ because $a^p_{H_k}$ and $a^q_{H_k}$ do not have w_{ji} component. So:

$$\begin{aligned}
\frac{\partial E^p_2}{\partial w_{ji}} &= -\sum_{q=1}^{N} \left[(a^p_{H_j} - a^q_{H_j}) \left(\frac{\partial a^p_{H_j}}{\partial a^p_{H_j}} - \frac{\partial a^q_{H_j}}{\partial a^p_{H_j}} \right) \right] \times \frac{\partial a^p_{H_j}}{\partial w_{ji}} \\
&= -\sum_{q=1}^{N} \left[(a^p_{H_j} - a^q_{H_j}) \left(1 - \frac{\partial a^q_{H_j}}{\partial a^p_{H_j}} \right) \right] \times \frac{\partial a^p_{H_j}}{\partial w_{ji}}
\end{aligned}$$

For $p \neq q$, we can assume that $a^q_{H_j}$ does not change when we process pattern p. This leads to $\frac{\partial a^q_{H_j}}{\partial a^p_{H_j}} = 0$ or $(1 - \frac{\partial a^q_{H_j}}{\partial a^p_{H_j}}) = 1$. When $p = q$, we have $(a^p_{H_j} - a^q_{H_j}) = 0$. So:

$$\begin{aligned}
\frac{\partial E^p_2}{\partial w_{ji}} &= -\sum_{q=1, q \neq p}^{N} (a^p_{H_j} - a^q_{H_j}) \times \frac{\partial a^p_{H_j}}{\partial w_{ji}} \\
&= -((N-1) a^p_{H_j} - \sum_{q=1, q \neq p}^{N} a^q_{H_j}) \frac{\partial a^p_{H_j}}{\partial w_{ji}} \\
&= -(N a^p_{H_j} - \sum_{q=1}^{N} a^q_{H_j}) \frac{\partial a^p_{H_j}}{\partial w_{ji}} \\
&= -N(a^p_{H_j} - \overline{a_{H_j}}) \frac{\partial a^p_{H_j}}{\partial w_{ji}}
\end{aligned}$$

with N being the number of training patterns and $\overline{a_{H_j}}$ is the average activation of the j^{th} hidden unit over all input samples. As with the usual backpropagation derivation, we have $\frac{\partial a^p_{H_j}}{\partial w_{ji}} = a^p_{H_j}(1 - a^p_{H_j})x^p_i$. Thus,

$$\frac{\partial E^p_2}{\partial w_{ji}} = -N(a^p_{H_j} - \overline{a_{H_j}})a^p_{H_j}(1 - a^p_{H_j})x^p_i$$

It is interesting to note that when computing $\frac{\partial E'}{w_{ji}}$ for the p^{th} input sample, besides looking at the activation of the j^{th} hidden unit and the i^{th} input unit as is done with the usual backpropagation training, we only need one more value $\overline{a_{H_j}}$ which can be computed and stored locally at the j^{th} hidden unit. This local property is highly desired in neural network training.

B. Rule extraction algorithm

The outline of our rule extraction algorithm is as follows

- Step 1: Train the network
- Step 2: Cluster the hidden unit activation values
- Step 3: Extract rules explaining the output in term of clustered hidden unit activation values
- Step 4: Prune unnecessary weights connecting the input layer to the hidden layer
- Step 5a: If the data consists of continuous attributes: generate rules in the form of linear inequalities on input for hidden unit activation cluster values.
- Step 5b: If the data consists of binary attributes: generate decision tree rules for each hidden unit activation value cluster using C4.5 [7]

Step 1 to 4 are similar to [8], [9], [10] but differ in a number of ways: we use (1) a different error function that puts a strong emphasis on hidden unit activations' sparsity rather than pruning (2) a different learning algorithm and (3) C4.5 for extracting the simplified hidden - output mapping.

In step 1, we use RPROP [11] (resilient backpropagation) an improved backpropagation learning algorithm that trains networks faster by adjusting the weight update based on the behavior of the gradient instead of the magnitude of the derivatives. It also requires few training parameters. We augment the error function with the popular *weight decay* term $E_d = \lambda \sum_j \sum_j w^2_{ji}$ to prevents weights from getting too large [12]. Weight decay has been shown to improve the generalization performance of neural networks (regularization).

The logistic hidden unit activation values are in the range $[0, 1]$. After training, the values at each hidden units can be clustered together into disjoint intervals $[0, r_1), [r_1, r_2), ..., [r_n, 1]$ such that we only need to know which interval the hidden activation values are in to determine the class label of training instances. We use the Chi2 discretization algorithm [13] to cluster the activation values. The algorithm first makes one interval for each activation value, sorts the intervals in increasing order, then uses χ^2 statistics to determine which pair of adjacent intervals should be merged next. Some pairs of intervals are not allowed to be merged because that would affect the classification accuracy. For example, when there are two training examples p and q with different class labels such that $a^p_{H_j}$ is in the first interval and $a^q_{H_j}$ is in the second interval, the two intervals cannot be merged as we no longer can determine which class label to assign knowing only the interval that the j^{th} hidden unit is in.

In step 3, we extract rules having the form $(H_{i_1} = l_1, H_{i_2} = l_2, ...) \rightarrow class = c_j$ which means *if the i^{th}_1 hidden unit's activation value is in interval l_1 and the i^{th}_2 hidden unit's activation value is in interval l_2 and ... then classify the sample as class c_j*. Rule extraction is done by C4.5. This extraction step is also a base step in other rule extraction algorithms. It is very important to have fewer rules here because the number of rules here strongly affects the final number of rules that ultimately specify the input - output relationship.

The novelty of our method is in the use of the new error term that "pushes" the hidden activation vectors away from each other, so that their component values tend to cluster toward the two ends of the interval $[0, 1]$. This in turn results in many hidden units having values clustered into only two intervals $[0, r), (r, 1]$. Having such simple splits is highly desirable for making fewer and simpler rules.

Step 4 prunes the network by removing unnecessary connections from the input units to the hidden units. Pruning reduces the number of weights, thus making the rules with continuous inputs simpler. It also helps extracting simpler rules for binary inputs. We use a simple pruning scheme that greedily removes weights in increasing order of their magnitudes and stops when the accuracy in the validation set drops below a specified threshold.

Step 5 is different for continuous and discrete attributes. If the inputs consist of continuous attributes, we can generate directly rules that depend upon when the j^{th} hidden unit activation is in an interval $[r_1, r_2)$ stated as follows:

$$\begin{aligned} r_1 &\leq a_{H_j} < r_2 \\ r_1 &\leq \sigma(w_{j1}x_1 + w_{j2}x_2 + ... + w_{jn}x_n) < r_2 \\ \sigma^{-1}(r_1) &\leq w_{j1}x_1 + w_{j2}x_2 + ... + w_{jn}x_n < \sigma^{-1}(r_2) \end{aligned}$$

Not all x_i are present in each rule because we have already pruned unnecessary weights in step 4. Every hidden-output rule produced in step 3 is a conjunction of which interval each hidden unit value must be in, so we can easily replace the terms in the conjunctions with the above inequality to produce rules explaining the output classification directly from the input.

For problems with binary inputs, we use C4.5 to generate one set of rules for each hidden unit's activation. The rules tell the conditions on inputs that would make a hidden unit activation value fall into one interval. For example, a rule for the j^{th} hidden unit has the form:

$(x_{i_1} = b_1, x_{i_2} = b_2, ...) \rightarrow a_{H_j} \in k^{th} interval$

Because the rules in this step are only concerned with which interval a hidden unit activation is in, they are much simpler.

We then replace each term $H_i = l_i$ in step 3 with the input-hidden layer rules, simplify the boolean expressions, and remove the duplicates to have the final rules explaining the classification directly from the input values.

C. Illustrative example

We consider the hidden unit encodings learned by the neural network for the *waveform* problem [14] to illustrate our approach. The *waveform* data set consists of 5000 instances of waves. Each wave is characterized by 21 continuous inputs with noise. The problem is to classify these waves into one of three classes.

First we standardized the inputs using z-values [15], and used a three layer feedforward neural network with 4 hidden units to train on 4000 instances. Another 500 instances were reserved for testing and 500 others were used for validation.

After training, we can compute the hidden unit activations $(a^p_{H_1}, a^p_{H_2}, a^p_{H_3}, a^p_{H_4})$ of the four hidden units for each p^{th} instance. This vector is an encoding of the 21-dimension vector input. We are interested in how these 4 dimensional vectors are arranged in the four dimensional space. Without losing generality, we chose 3 of the 4 dimensions arbitrarily in order to plot the locations of these vectors in the following.

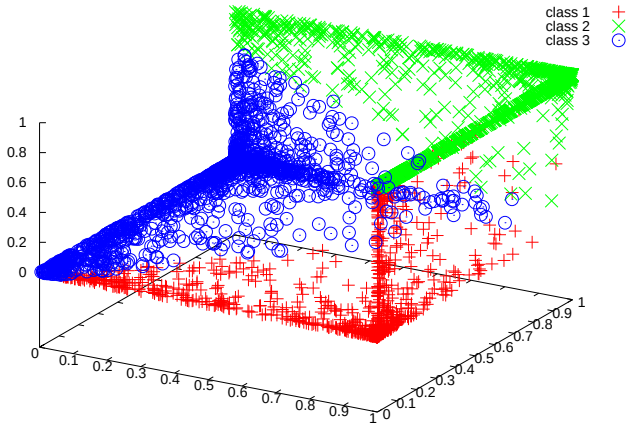

Fig. 2. Input patterns throughout hidden unit activation space for the *waveform* problem after training with regular backpropagation

Figure 2 shows the plot after the network has been trained with the regular sum of squared error function used in standard backpropagation. It can be seen that the vectors are clustered into 3 groups corresponding to the 3 classes. While many of them are in the corners or along the edges, quite a number are spread out instead. These vectors make it hard to draw planes separating the clusters; in other words, more rules would be expected to be needed to explain the hidden activation - output relationship.

What we want to do is to push these vectors further away from each other during learning so that it is easier to separate them. This is done with the help of the new error term E_2 that penalizes having vectors close together. The effect of the training with this new error term is shown in Figure 3. The

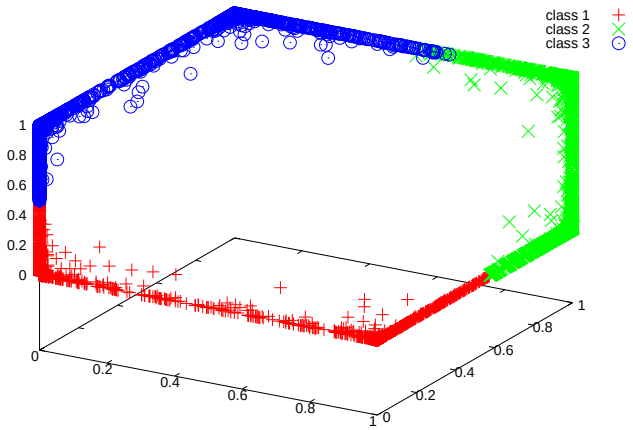

Fig. 3. Input patterns throughout hidden unit activation spae for the *waveform* problem after backpropagation training that incorporates the new error term

three clusters are more visible as they move closer to the edges and three corners and fewer vectors are in the middle.

III. Experimental Results

TABLE I

DATA SETS USED FOR EVALUATION

data set	no.attrs	no.class	no.instances	input
waveform	21	3	5000	continuous
yeast	8	10	1484	continuous
nursery	8	5	1296	discrete
splice	60	4	3190	discrete

TABLE II

EFFECT OF E_2 ON NEURAL NETWORK TRAINING

data set	$-E_2/N^2$		E/N	
	regular	new	regular	new
waveform	1.631	1.769 (+9%)	0.357	0.361
yeast	0.696	0.954 (+36%)	0.618	0.620
nursery	1.049	1.233 (+18%)	0.271	0.279
splice	0.616	0.878 (+42%)	0.229	0.275

The goal of this evaluation was to compare the number of rules extracted from a trained error backpropagation network when E_2 was included in the error function (experimental condition) versus the number when E_2 was not included (control condition, i.e., standard backpropagation). We evaluated the effectiveness of our rule extraction method by selecting five arbitrary data sets having more than 1000 instances from the UCI Machine Learning Repository [14]: the *waveform*, *yeast*, *image-segmentation*, *nursery* and *splice* problems. These are large and difficult datasets with many attributes and classes. The method performed well on four out of five of these data sets: *waveform*, *yeast*, *nursery* and *splice*. For *image-segmentation*, the classification accuracy went down as we increased β for reducing the number of rules so our method was ineffective. We have therefore omitted it from further consideration in the following.

TABLE III
EXPERIMENTAL SIMULATION RESULTS (AVERAGES OVER 100 RUNS)

data set	$-E_2/N^2$		No. of *I-H* weights			No. of rules			Rule accuracy	
	regular	new	initial	regular	new	regular	new	reduced	regular	new
waveform	1.631	1.769	86	53.14	55.31	70.12	41.71	41%	85.17%	85.51%
yeast	0.696	0.954	27	16.94	18.14	90.51	74.91	18%	51.88%	51.95%
nursery	1.049	1.233	78	42.55	45.25	191.72	136.83	29%	87.89%	87.45%
splice	0.616	0.878	482	239.67	440.67	567.43	267.77	53%	91.97%	90.36%

Table I shows the other four data sets used in the experiments. Two of these have continuous inputs: the *waveform problem* involves classifying waves into one of three classes based on 21 noisy features and the *yeast problem* is a protein localization site determination problem.

The other two data sets have discrete/categorical inputs. Data set *nursery* is an application ranking database for admission to nursery schools. Applications are classified into 5 classes indicating how strongly the applicant is recommended. The 8 categorical attributes are encoded into 25 binary input units using nominal encodings: a category with m unique values is encoded as m binary input units, with only one bit corresponding to the value being on. We only used 10% of the available instances in this case. The *splice* problem is to recognize the boundary between exons and introns giving a DNA sequence. The 60 attributes, each representing one nucleotide $\{$A,T,G,C$\}$, are encoded into 240 binary input units.

For each dataset, the settings for the experiments are as follows:

1) Ten-fold cross validation scheme: We split each data set randomly into 10 subsets of equal size. Eight subsets were used for training, one was used for validation and one for measuring the accuracy of the extracted rules. We repeated the procedure 10 times, each time one different subset was used as the testing set. We also ran each experiment 10 times with different random initial weights. The reported number of rules and accuracy are averages over all 100 runs. Having so many runs ensures that any improvement comes from the method and not just by chance.
2) The initial weights and test set division were held the same for the experiments with and without our new error term. This gave the same starting point for both.
3) Weight decay rate was set to 0.00001.
4) β was set to 0.00001 for *waveform* and *nursery*, and 0.00005 for *yeast* and *splice* problems. The contribution of βE_2 is much more significant than it looks. At the end of training, E is in the order of 10^2 because it is a sum of over 1000 squared errors from all output units. E_2 is in the order of 10^6 because it is the sum over *all pairs* of Euclidean distances. These choices of β make the contribution of E_2 about $5\% \sim 30\%$ of E' for the four problems.
5) The number of output units corresponds to the number of classes in the data. When doing classification, the class whose output unit has the highest activation value is chosen as the class for the instance.
6) Continuous attribute values were standardized with z-value scores [15].
7) RPROP with weight backtracking was set up to run with a maximum 400 epochs or until validation error goes up for 10 consecutive epochs. The network with highest validation accuracy was saved for subsequent rule extraction.

Table II shows the effect of the new error term on E and E_2. The *regular* columns show results when we train with regular backpropagation and E. The *new* columns show the improved results with E'. They show that $-E_2$ was increased up to 42% with only a small change in E. So backpropagation was able to learn an improved encoding at the hidden layer that has higher total squared distances between the hidden unit activation vectors while still maintaining near minimum error at the output layer. The choice of β has a strong impact on the accuracy and E_2. If β is too high, backpropagation would focus too much on E_2 and push all activation vectors to the corners resulting in low classifying accuracy. The smaller β is, the less increase in sum of squared distances is observed, and the number of rules will stay the same.

Table III presents the experimental results concerning rule extraction. The new error term helped reduce the number of rules significantly, at least 18% for the *yeast* problem and up to 53% for the *splice* problem. The new, smaller sets of rules also have roughly the same classification rates as the ones produced without the new error term (right most columns of table III). The largest accuracy rate drop is only 1.8% for the *splice* result. We may note that the accuracy rate for *yeast* is quite low, but it is still comparable to the published results 54% in [16]. The reason for such allow accuracy rate is that the data set is very difficult with 10 classes unevenly distributed.

The numbers of initial and pruned weights of the connections from the input layer to the hidden layer are given in the columns *No. of I-H weights*. For the two problems *waveform* and *yeast* with continuous attributes, the numbers of weights show how simple the final rules are. Although they are not as important as the number of rules, a smaller number of weights is desirable because there will be fewer terms in the rule's inequalities produced in Step 5a. In these

problems, our simple pruning scheme was able to reduce the number of weights about 35%. The numbers of weights left do not affect the readability of rules for the two problems *nursery* and *splice* with discrete attributes because they do not translate directly to the complexity of boolean rules.

IV. Conclusion and discussion

We have presented an improved method to extract symbolic rules from multilayer feedforward neural networks. Experiments with four large publicly available data sets showed that the method helped reduce the number of rules significantly without sacrificing classification accuracy. Our approach was not successful on a fifth data set as explained above. The main contribution is the new error term that encourages backpropagation to learn a better encoding at the hidden layer and an efficient way to compute its derivative for fast network training.

The method only performed well on four out of the five large data sets we have tested. Although the method has reduced the number of rules significantly, it still cannot do as well as rule-based system such as C4.5. We plan to investigate the method on more data sets and explore ways to reduce the number of rules further.

References

[1] A. Darbari, "Rule extraction from trained ANN: A survey", *Technical Report*, Institute of Artificial Iintelligence, Department of Computer Science, TU Dresden, Germany, 2000.

[2] H. Jacobsson, "Rule extraction from recurrent neural networks: A taxonomy and review", *Neural Computation*, volume 17, pages 1223-1263, 2005.

[3] I.A. Taha and J. Ghosh, "Symbolic interpretation of artificial neural networks.", *IEEE Transactions on Knowledge and Data Engineering*, 11(3):448-463, 1999.

[4] A.B. Tickle, R. Andres, M. Golea and J. Diederich, "The truth will come to light: directions and challenges in extracting the knowledge embedded within trained artifical neural networks", *IEEE Transactions on Neural Networks*, volume 9, pages 1057-1068, 1998.

[5] R. Andres, J. Diederich and A.B. Tickle, "A survey and critique of techniques for extracting rules from trained artificial neural networks", *Knowledge-Based Systems*, volume 8, pages 373-389, 1995.

[6] S. Haykin, *Neural Networks: A Comprehensive Foundation*, Prentice Hall PTR Upper Saddle River, NJ, USA, 1998

[7] R. Quinlan, *C4.5: Programs for Machine Learning*, Morgan Kaufman, San Mateo, CA, 1993.

[8] G.G. Towell and J.W. Shavlik, "Extracting refined rules from knowledge-based neural networks", *Machine Learning*, volume 13, number 1, pages 71-101, Springer, 1993

[9] R. Setiono and W.K. Leow, "FERNN: An algorithm for fast extraction of rules from neural networks", *Applied Intelligence*, volume 12, number 1, pages 15-25, Springer, 2000.

[10] H. Lu, R. Setiono and H. Liu, "NeuroRule: A connectionist approach to data mining", *Proceedings of the International Conference on Very Large Data Bases*, pages 478-489, Institute of electrical and electronic engineers (IEEE), 1995.

[11] M. Riedmiller and H. Braun, "Rprop: A fast adaptive learning algorithm", *Proc. of the Int. Symposium on Computer and Information Science VII*, 1992.

[12] A. Krogh and J. Hertz, "A simple weight decay can improve generalization.", *Advances in Neural Information Processing Systems*, 4, 1992.

[13] H. Liu and R. Setiono, "Chi2: Feature selection and discretization of numeric attributes", *Proceedings of the Seventh International Conference on Tools with Artificial Intelligence*, IEEE Computer Society Washington, DC, USA, 1995.

[14] A. Asuncion and D.J. Newman, "UCI Machine Learning Repository", *http://www.ics.uci.edu/∼mlearn/MLRepository.html*, University of California, School of Information and Computer Science, Irvine, CA.

[15] R.O. Duda, P.E. Hart and D.G. Stork, *Pattern Classification*, Wiley New York, 2001.

[16] P. Horton and K. Nakai, "A Probablistic Classification System for Predicting the Cellular Localization Sites of Proteins" *Intelligent Systems in Molecular Biology.* 109-115, St. Louis, USA, 1996.