Guiding Hidden Layer Representations for Improved Rule Extraction from Neural Networks

Thuan Q. Huynh and James A. Reggia

***Abstract*— The production of relatively large and opaque weight matrices by error backpropagation learning has inspired substantial research on how to extract symbolic human-readable rules from trained networks. While considerable progress has been made, the results at present are still relatively limited, in part due to the large numbers of symbolic rules that can be generated. Most past work to address this issue has focused on progressively more powerful methods for rule extraction (RE) that try to minimize the number of weights and/or improve rule expressiveness. In contrast, here we take a different approach in which we modify the error backpropagation training process so that it learns a different hidden layer representation of input patterns than would normally occur. Using five publicly available datasets, we show via computational experiments that the modified learning method helps to extract fewer rules without increasing individual rule complexity and without decreasing classification accuracy. We conclude that modifying error backpropagation so that it more effectively separates learned pattern encodings in the hidden layer is an effective way to improve contemporary RE methods.**

***Index Terms*— Hidden layer representation, neural networks, penalty function, rule extraction.**

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 (Fig. 1). There are several variants of error backpropagation, usually driven by minimizing the sum of squared error between the actual output values and the target teaching signals. 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 a trained network has learned. This difficulty has inspired substantial past research on how to extract symbolic human-readable rules from a network so that one 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]–[5]), the results obtained are still very limited.

There are three main approaches that have been taken in past work on rule extraction (RE) from neural networks: pedagogical, decompositional, and eclectic (a hybrid of the other two) [1], [4]. *Pedagogical methods* consider a neural network to be a blackbox oracle that provides class labels for any input vectors including the ones that are not in the training set. Notable algorithms include OSRE [6], RE-RX [7], and Minerva [8]. 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 ([9], [10]). *Eclectic methods* are a hybrid of the other two methods. In this paper, we focus on decompositional methods.

Many decompositional approaches, including ours, first extract the rules that explain the mapping between the hidden unit activations and the outputs, and then extract rules governing the input–hidden layer relationship. These rules are then combined to produce the final input–output rules. Such approaches try to produce simpler and fewer rules at the hidden output layers so that they can generate fewer rules overall. Beside good classification accuracy and generalization, having a smaller number of rules is a very important criterion for RE algorithms so that a human can understand their content easier. The number of rules generated depends heavily upon how the hidden unit activation vector encodings of the input patterns are formed. 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 for decompositional methods because sometimes the hidden layer representations of the input patterns are so complex or distributed that many rules are required to explain the hidden output layer mapping.

Research in this area has largely focused on learning networks with fewer weights and better ways to express the rules, but relatively little work has been done on altering training methods to learn a better hidden layer representation so that *any* RE process becomes more effective. Potentially, a better representation might allow the first stage to extract fewer and more compact regions in the hidden activation space, thereby leading to a more concise and easier to understand set of rules. In this paper, we propose new "error terms" to augment the standard sum of squared error. The new error terms encourage backpropagation to learn a better separated encoding over the hidden layer in which vectors of hidden unit activations are farther apart than they would normally

Manuscript received February 26, 2010; revised November 10, 2010; accepted November 15, 2010. Date of publication December 6, 2010; date of current version February 9, 2011. This work was supported in part by the National Science Foundation Award IIS0753845.

The authors are with the Department of Computer Science, University of Maryland, College Park, MD 20742 USA (e-mail: thuan@cs.umd.edu; reggia@cs.umd.edu).

Digital Object Identifier 10.1109/TNN.2010.2094205

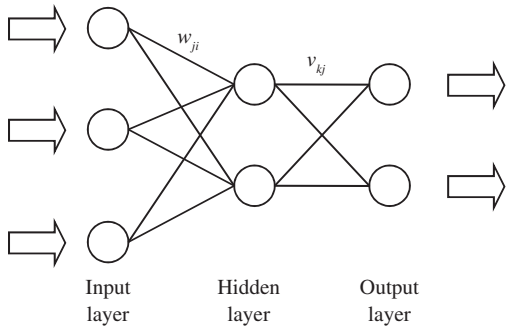

Fig. 1. Typical fully connected feedforward neural network.

be with standard backpropagation. Gradient descent is used to modify error backpropagation when using these new error terms. 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 terms to assess their impact on the number of rules generated.

II. Algorithm

In this section, we first describe a new error term that makes hidden unit activation patterns more separated from one another, and provides an efficient local way to compute the gradient of this new term during 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 E_2

In this paper, we are interested in extracting rules from multilayer feedforward neural networks with one hidden layer as shown in Fig. 1. The activation of the jth hidden unit when the pth input pattern is presented is calculated as the logistic function of the weighted sum of inputs

$$a_{H_j}^p = \sigma\left(\sum_{i \in input} w_{ji} x_i^p\right)$$

where x_i^p is the ith input unit value of the pth instance, w_{ji} is the weight from the ith input unit to the jth hidden unit, and $\sigma()$ is the logistic function $\sigma(x) = 1/(1+e^{-x})$.

Similarly, the activation of the kth output is calculated as the logistic function of the weighted sum of hidden unit activations

$$a_{O_k}^p = \sigma\left(\sum_{j \in hidden} v_{kj} a_{H_j}^p\right)$$

where v_{kj} is the weight from the jth hidden unit to the kth output unit (see Fig. 1).

The usual error function computed over the output units is

$$E_1 = \frac{1}{2}\sum_{p=1}^{N}\sum_{k \in output}\left(t_k^p - a_{O_k}^p\right)^2$$

where t_k^p is the target output for the kth output unit when the pth input pattern is presented, and N is the size of the input dataset (number of input–output pairs).

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

$$E_2 = -\frac{1}{2}\sum_{p=1}^{N}\sum_{q=1}^{N}\sum_{k \in hidden}\left(a_{H_k}^p - a_{H_k}^q\right)^2$$

where $a_{H_k}^p$ is the activation of the kth hidden unit when the pth input sample is presented and $a_{H_k}^q$ is analogous for the qth pattern. Thus E_2 is the sum over all pairs (p, q) of the squared Euclidean distance between two hidden activation vectors for the pth and qth input patterns. The negative sign ensures that when neural network training minimizes the measure E_2, it will maximize the distances between the hidden layer vectors. When $p = q$, only zeroes enter the sum, so no special attention is given to that situation. The new total error function guiding learning is

$$E = \alpha E_1 + \beta E_2$$

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_1, so β must be quite small to scale E_1 and E_2 appropriately.

In order to train the network with error backpropagation, we need to compute $\partial E/\partial w_{ji} = \alpha(\partial E_1/\partial w_{ji}) + \beta(\partial E_2/\partial w_{ji})$ and $\partial E/\partial v_{kj} = \alpha(\partial E_1/\partial v_{kj}) + \beta(\partial E_2/\partial v_{kj})$ where w_{ji} is an input-to-hidden weight, and v_{kj} is an hidden-to-output weight. Of course, the standard terms $\partial E_1/\partial w_{ji}$ and $\partial E_1/\partial v_{kj}$ can be computed efficiently as in [11]. We also have $\partial E_2/\partial v_{kj} = 0\ \forall j, k$ with v_{kj} being the weight to the kth output unit from the jth hidden unit because E_2 does not have any v_{kj} component. As shown in Appendix A, $\partial E_2^p/\partial w_{ji}$ can be computed efficiently as follows:

$$\frac{\partial E_2^p}{\partial w_{ji}} = -N(a_{H_j}^p - \overline{a_{H_j}})a_{H_j}^p(1 - a_{H_j}^p)x_i^p \tag{1}$$

with N being the number of training patterns (a constant) and $\overline{a_{H_j}}$ being the average activation of the jth hidden unit over all training patterns.

It is remarkable that when computing $\partial E_2/w_{ji}$ for the pth input sample, besides looking at the activation of the jth hidden unit and the ith 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 jth hidden unit. This local property is highly desired in neural network training.

B. RE Algorithm

The same RE algorithm is used for both the experimental condition ($E = E_1 + E_2$) and the control condition ($E = E_1$, which is basic error backpropagation). The outline of the RE algorithm in both cases is as follows.

Step 1: Train the network.
Step 2: Cluster the individual hidden unit activation values.
Step 3: Extract rules explaining the output in terms 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 inputs 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 the program *C4.5rules* [12].

Steps 1–4 are similar to past RE methods in [13]–[15] but differ in a number of ways. We use: 1) a different error function that puts a strong emphasis on hidden unit activation patterns' separability rather than pruning [16]; 2) a different learning algorithm; and 3) *C4.5rules* for extracting the simplified hidden-output mapping. Regardless of these differences from past work, the same RE procedure is applied in comparing standard backpropagation (E_1) versus the enhanced method ($E_1 + E_2$).

Resilient backpropagation (RPROP) [17] is used in Step 1. It is an improved backpropagation learning algorithm that trains networks faster by adjusting the weight update based on the direction 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 $E_d = \lambda \sum_j \sum_i w_{ji}^2$ to prevent weights from getting too large [18]. Weight decay has been shown to improve the generalization performance of neural networks (regularization). This term is used implicitly in both control and experimental simulations in this paper.

The logistic hidden unit activation values are in the range $(0, 1)$. After training, the values experienced at each hidden unit can be clustered together into disjoint intervals $[0, r_1), [r_1, r_2), \ldots, [r_n, 1]$ 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 [19] to cluster the activation values. This algorithm first makes one interval for each activation value, sorts the intervals in increasing order, and 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 $a_{H_j}^p$ is in the first interval and $a_{H_j}^q$ is in the second interval, the two intervals cannot be merged as we no longer could determine which class label to assign knowing only the interval that the jth hidden unit is in.

In Step 3, we extract rules having the form $(H_{i_1} = l_1, H_{i_2} = l_2, \ldots) \rightarrow class = c_j$, which mean that *if the i_1th hidden unit's activation value is in interval l_1 and the i_2th hidden unit's activation value is in interval l_2 and ..., then classify the sample as class c_j*. RE is done using *C4.5rules*. This extraction step is also a base step in other RE algorithms. It is very important to have fewer rules at this point because the number of rules here strongly affects the final number of rules that ultimately specify the input–output relationship.

The novelty of this method is in the use of the new error term that "pushes" the hidden activation vectors away from each other, so that (as seen below) 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 binary 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 rules with continuous inputs simpler. It also helps in 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 jth hidden unit activation is in an interval $[r_1, r_2)$ stated as follows:

$$\begin{aligned} r_1 &\leq a_{H_j} < r_2 \\ r_1 &\leq \sigma(w_{j1}x_1 + w_{j2}x_2 + \cdots + w_{jn}x_n) < r_2 \\ \sigma^{-1}(r_1) &\leq w_{j1}x_1 + w_{j2}x_2 + \cdots + w_{jn}x_n < \sigma^{-1}(r_2). \end{aligned} \quad (2)$$

Not all x_i are present in each rule since unnecessary weights were already pruned 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 the terms in the conjunctions can easily be pruned with the above inequality to produce rules explaining the output classification directly from the input.

For problems with binary inputs, we use *C4.5rules* [12] 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 jth hidden unit has the form

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

Because the rules in this step are only concerned with which interval a hidden unit activation is in, there are usually very few simple rules. 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.

The important distinction between our RE method and *C4.5rules* is that *C4.5rules* generates a *single* decision tree/rule set directly from the dataset while our method generates *two* intermediate rule sets and combines them. The first rule set captures the relationship between the hidden activation intervals and the output. It is usually very simple with very few rules because of the improved hidden activation patterns. The second rule set explains the relationship between the hidden activation intervals and the input data. These rules are also simple because they are concerned with specific hidden activation intervals. It should also be noted that other methods beside *C4.5rules* could be used to extract these intermediate rules.

C. Illustrative Example

In this section, the hidden unit encodings learned by the neural network for the *waveform* problem [20] are used to

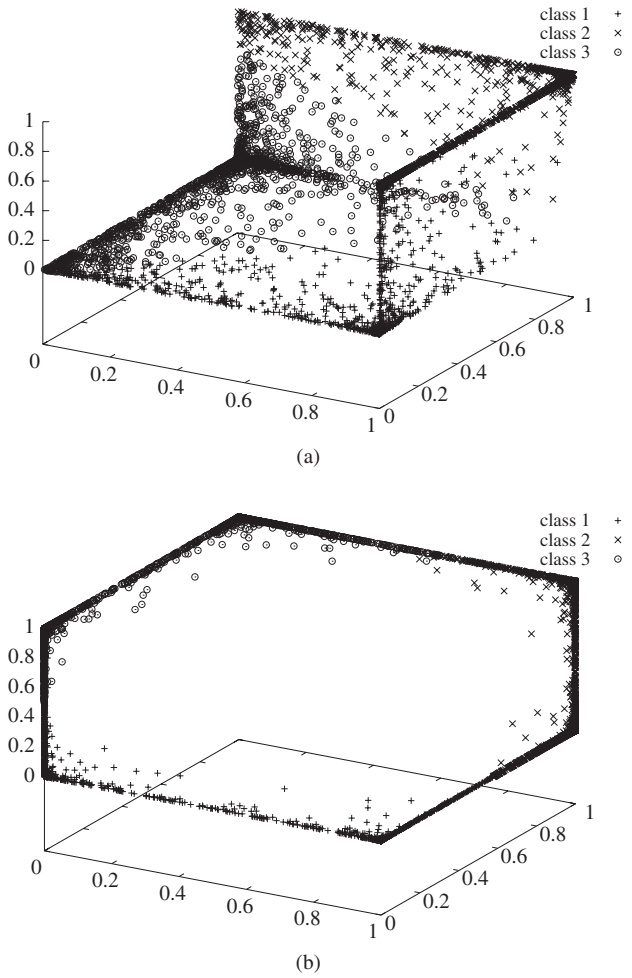

Fig. 2. Input patterns throughout hidden unit activation space for the *waveform* problem after training with (a) regular backpropagation ($E = E_1$) versus (b) same error function but augmented to include the new error term ($E = E_1 + E_2$).

illustrate our approach. The *waveform* dataset 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 the inputs are standardized using z-values [21]. The 5000 instances are divided randomly into three sets: 4000 for training, 500 for testing, and 500 for validation. A three-layer feedforward neural network with four hidden units is trained on the data.

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 instance p. This vector is an encoding of the 21-D vector input. We are interested in how these 4-D vectors are arranged in the 4-D space when the new error term E_2 is used and when it is not. Without losing generality, we chose three of the 4-D arbitrarily in order to visualize the locations of these vectors in the following representative example from one of the runs.

Fig. 2(a) shows the training data patterns plotted in hidden unit activation space after the network has been trained with the regular sum of squared error function E_1 used in standard backpropagation. It can be seen that the vectors are clustered into three groups corresponding to the three classes. While many of them are in the corners or along the edges, quite a number are spread out over the interior instead and close to vectors in other classes. 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 activation 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 Fig. 2(b). The three clusters are more visible as they move closer to the edges and three corners, and fewer vectors are in the interior. Clearly, the augmented learning procedure [Fig. 2(b)] pushes the interior hidden encodings for input patterns in different classes further from each other than with standard backpropagation [Fig. 2(a)] in this example.

TABLE I
DATASETS USED FOR EVALUATION

Dataset	No.attrs	No.class	No.instances	Input
Waveform	21	3	5000	Continuous
Yeast	8	10	1484	Continuous
Imgseg	18	7	2310	Continuous
Nursery	8	5	1296	Discrete
Splice	60	4	3190	Discrete

III. EXPERIMENTAL RESULTS

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 of E_1 alone, i.e., standard backpropagation). We evaluated the effectiveness of our RE method by selecting five arbitrary datasets having more than 1000 instances from the UC Irivne Machine Learning Repository [20]: the *waveform, yeast, image-segmentation, nursery*, and *splice* problems. These are large and difficult datasets with many attributes and classes.

Table I shows the characteristics of the five datasets used in the experiments. Three of these have continuous inputs: the *waveform problem* involves classifying waves into one of three classes based on 21 noisy features, the *yeast problem* is a protein localization site determination problem, and the *image segmentation* problem classifies pixels in images using 17 continuous value features. The other two datasets have discrete/categorical inputs. The dataset *nursery* is an application ranking database for admission to nursery schools. Applications are classified into five classes indicating how strongly the applicant is recommended. The eight 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. A set of 1296 (10%) instances were chosen randomly from 12 961 instances in the original dataset to shorten the running time. The *splice* problem is to recognize the boundary

TABLE II
REGULAR VERSUS MODIFIED BACKPROPAGATION
(AVERAGED OVER 100 RUNS)

Dataset	E_1/N		$-2E_2/N^2$	
	Regular	New	Regular	New
Waveform	0.128	0.149	1.632	1.949 (+19%)
Yeast	0.384	0.393	0.706	1.190 (+69%)
Imgseg	0.085	0.107	1.979	2.684 (+35%)
Nursery	0.075	0.095	1.050	1.125 (+7%)
Splice	0.068	0.073	0.971	1.175 (+21%)

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 both the experimental and control runs are as follows.

1) Ten-fold cross validation scheme: we split each dataset randomly into 10 subsets of approximately equal size. Eight subsets were used for training, one for validation, and one for measuring the accuracy of the extracted rules. We repeated the procedure 10 times, where 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. For the *image-segmentation* dataset, which was divided into training and test set by the data donor, we first merged the original training and test data, and then did 10-fold cross validation as we did with other datasets.
2) In each run, the experiments with and without our new error terms have the same starting point, i.e., matched initial weights and dataset division to make comparison maximally compatible. Paired t-tests are used to evaluate the results.
3) Weight decay rate was set to 0.00001.
4) β was set to 0.00001 for *waveform* and *nursery*, 0.00005 for *yeast*, 0.0003 for the *splice* problem, and 0.00007 for the *image-segmentation* problem. To determine these values, we did a few pilot runs with each dataset where we started with $\beta = 0$ (the control case of using E_1 alone) and slowly increased β until the accuracy rate dropped more than 5% compared to the control case. This determined the values of β that we used for the 100 runs reported in our experimental results (and also for α since $\alpha = 1-\beta$). The contribution of βE_2 is much more significant than it looks. At the end of training, E_1 is of the order of 10^2 because it is a sum of over 1000 squared errors from all output units. E_2 is of 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 $5\% \sim 70\%$ of E for the five problems.
5) The number of output units corresponds to the number of classes in the data. When doing a classification, the class whose output unit has the highest activation value is chosen as the class for the instance.
6) Continuous input attribute values were standardized with z-value scores [21].
7) RPROP with weight backtracking was set up to run for a maximum of 400 epochs or until validation error goes up for 10 consecutive epochs. The network with highest validation accuracy was saved for subsequent RE.

Table II shows the effect of the new error term E_2 on the network's average testing error (E_1/N) and the average distances among hidden unit activation vectors ($-2E_2/N^2$) where N is the number of data instances. A network's average testing error and average squared distance between hidden activation vectors are shown after training with regular backpropagation versus the new combined error term. Since E_2 is $1/2$ of the sum of squared distances between each pair (total N^2 pairs) of activations, $-2E_2/N^2$ is the average squared distance between each pair. The *regular* columns show results when we train with "regular" backpropagation using $E = E_1$. The *new* columns show the results with $E = E_1 + E_2$. These data show that activation pattern distances were increased up to 69% with only a small change in network errors. The small change in E_1 backs our hypothesis that it is possible to make error backpropagation learn a different encoding that satisfies other criteria (smaller E_2) while still maintaining the network's accuracy. Modified backpropagation was able to learn an encoding with increased pattern separation at the hidden layer that had 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.

Fig. 3 shows the values of (a) E_1/N and (b) $-2E_2/N^2$ during one training run on the *waveform* dataset using error backpropagation with the regular error function E_1 and with the new error term E_2. The training error was slightly higher when trained with $E = \alpha E_1 + \beta E_2$. This is expected because error backpropagation has to minimize both terms in this latter case. But the change is very small and not enough to affect the overall classification accuracy significantly. Fig. 3(b) shows the average squared distance between hidden unit activation pairs $-2E_2/N^2$. Training quickly increases the distance in both cases, but significantly more when trained with $E = \alpha E_1 + \beta E_2$.

Table III presents the experimental results concerning RE. The new error term helped to reduce the number of rules significantly, at least 13% for the *splice* problem and up to 80% for the *waveform* problem. The new smaller sets of rules also have roughly the same classification rates as the ones produced without the new error term (rightmost columns of Table III). Note that the accuracy rate for the *yeast* problem is quite low, but it is still comparable to the best published results of 54% in [22]. The reason for such a low accuracy rate is that the dataset is very difficult with 10 classes unevenly distributed.

We used paired t-tests to determine whether the reduction in numbers of rules and the change in accuracy caused by training with E_1 versus $E_1 + E_2$ are significant, using a standard significance level 0.05. Bonferroni correction [23] for 10 tests requires the significance to be defined as $p < 0.05/10 = 0.005$. Statistically significant changes are printed

TABLE III
EXPERIMENTAL RESULTS: MEAN (STANDARD DEVIATION) OVER 100 RUNS

Dataset	No. of rules			Rule accuracy	
	E_1	$E_1 + E_2$	Reduced	E_1	$E_1 + E_2$
Waveform	70.12 (26.89)	*14.30 (13.56)*	80%	85.08% (1.96)	85.19% (2.04)
Yeast	90.17 (23.09)	*51.37 (18.23)*	43%	51.55% (4.21)	51.40% (4.37)
Imgseg	38.34 (9.27)	*32.02 (7.37)*	16%	91.58% (2.33)	*89.29% (3.06)*
Nursery	192.42 (95.34)	*41.85 (43.85)*	78%	88.55% (4.01)	89.33% (2.86)
Splice	90.21 (84.42)	78.66 (54.49)	13%	90.19% (3.81)	89.85% (4.09)

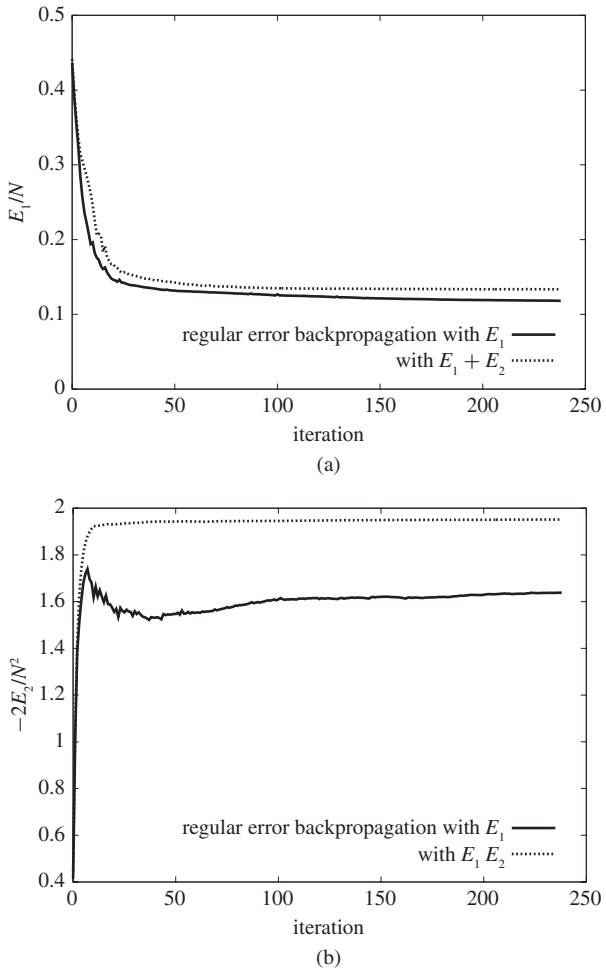

Fig. 3. Typical plot of (a) average network error E_1/N and (b) average hidden layer activation pattern separation $-2E_2/N^2$ during network training.

in italics in Table III. Corresponding p-values are shown in Table IV. The reduction in numbers of rules is significant in all cases (p from 5.4×10^{-35} to 1.9×10^{-6}) except for the *splice* problem ($p = 0.243$). The change in accuracy is *not* significant in all cases (p from 0.45 to 0.71) except for the *image-segmentation* problem ($p = 6.4 \times 10^{-9}$). The tests confirmed that E_2 reduced the number of rules without degrading accuracy.

More significantly, the best among 100 runs in the experiments were able to extract even smaller rule sets than the averages described above. RE using E_2 extracted only 5 rules explaining the classification of 5000 *waveform* data instances

TABLE IV
P-VALUE OF t-TESTS COMPARING E_1 AND $E_1 + E_2$

Dataset	Avg. no. of rules	Accuracy rates
Waveform	5.4×10^{-35}	0.4818
Yeast	4.7×10^{-25}	0.7179
Imgseg	1.9×10^{-06}	6.4×10^{-9}
Nursery	4.3×10^{-28}	0.0456
Splice	0.243	0.5574

with 88% accuracy rate, 19 rules for the *yeast* dataset with 57% accuracy, 17 rules for the *image-segmentation* dataset with 90% accuracy, 15 rules for the *nursery* dataset with 93% accuracy, and 14 rules for the *splice* dataset with 94% accuracy. These are better than the best numbers of rules using E_1: 14, 49, 19, 15, and 19, respectively.

IV. CLASS LABEL-AWARE SEPARATION

Since E_2 incorporates the sum of distances between all pairs of hidden unit activations, its effect is to push every activation pattern away from the rest. Such an approach ignores class labels, and this omission suggests another more targeted strategy. If one could take into account the class labels of the training data, then just the activation patterns of instances from *different* classes could be pushed apart, while instead the activations of instances from the *same* class could be treated differently, i.e., they could be pushed closer to one another. Potentially, such an approach could be even more effective in lowering the number of rules generated. Therefore, we examined two new penalty terms E_3 and E_4: E_3 penalizes hidden unit activation vectors *from different classes* having small Euclidean distances, while E_4 penalizes hidden unit activation vectors *from the same class* having big Euclidean distances. More specifically, E_3 and E_4 are given by

$$E_3 = -\frac{1}{2}\sum_{p=1}^{N} \sum_{\substack{q=1 \\ class(q)\neq class(p)}}^{N} \sum_{k\in hidden} (a_{H_k}^p - a_{H_k}^q)^2$$

$$E_4 = \frac{1}{2}\sum_{p=1}^{N} \sum_{\substack{q=1 \\ class(q)=class(p)}}^{N} \sum_{k\in hidden} (a_{H_k}^p - a_{H_k}^q)^2.$$

It is important to note the negative sign in E_3 and its absence in E_4. Minimization of E_3 and E_4 *increases* the distances of hidden activation vectors from different classes and *decreases* the distances of activations from the same class.

TABLE V
EXPERIMENTAL RESULTS: MEAN (STANDARD DEVIATION) OVER 100 RUNS

Dataset	Waveform		Yeast		Imgseg	
	# rules	Accuracy	# rules	Accuracy	# rules	Accuracy
E_1	70.12 (26.89)	85.08% (1.96)	90.17 (23.09)	51.55% (4.21)	38.34 (9.27)	91.58% (2.33)
$E_1\&E_2$	*14.30 (13.56)*	85.19% (2.04)	*51.37 (18.23)*	51.40% (4.37)	*32.02 (7.37)*	89.29% (3.06)
$E_1\&E_3\&E_4$	*8.79 (3.77)*	*85.71% (2.34)*	52.52 (17.67)	51.75% (4.43)	*26.12 (8.18)*	*91.86% (2.11)*
C4.5rules	77.50 (8.50)	77.30% (1.63)	36.50 (4.32)	59.22% (5.15)	30.00 (1.80)	95.70% (1.00)

Dataset	Nursery		Splice	
	# rules	Accuracy	# rules	Accuracy
E_1	192.42 (95.34)	88.55% (4.01)	90.21 (84.42)	90.19% (3.81)
$E_1\&E_2$	*41.85 (43.85)*	89.33% (2.86)	78.66 (54.49)	89.85% (4.09)
$E_1\&E_3\&E_4$	29.15 (21.08)	*89.93% (2.41)*	*26.92 (14.63)*	90.93% (4.19)
C4.5rules	71.49 (6.03)	91.29% (2.74)	39.90 (3.67)	94.33% (1.18)

TABLE VI
P-VALUE OF t-TESTS ON THE NUMBER OF RULES

Dataset	E_1 versus $E_1+E_3+E_4$	E_1+E_2 versus $E_1+E_3+E_4$
Waveform	6.3×10^{-42}	0.0001
Yeast	2.6×10^{-22}	0.6524
Imgseg	3.3×10^{-15}	5.3×10^{-6}
Nursery	7.4×10^{-31}	0.0063
Splice	1.4×10^{-11}	1.3×10^{-14}

TABLE VII
P-VALUE OF t-TESTS ON ACCURACY RATES

Dataset	E_1 versus $E_1+E_3+E_4$	E_1+E_2 vs $E_1+E_3+E_4$
waveform	0.0005	0.0007
Yeast	0.6262	0.4305
Imgseg	0.2994	4.3×10^{-11}
Nursery	0.0002	0.0103
Splice	0.2125	0.0680

A. Learning Rules

As with E_2, we need $\partial E_3^p/\partial w_{ji}$ and $\partial E_4^p/\partial w_{ji}$ in order to compute the weight change for gradient descent. Appendix VII shows that we can compute both terms efficiently as follows:

$$\frac{\partial E_3^p}{\partial w_{ji}} = \left(N(a_{H_j}^p - \overline{a_{H_j}}) - N_{C(p)}(a_{H_j}^p - \overline{a_{H_j}^{C(p)}})\right) \times a_{H_j}^p(1 - a_{H_j}^p)x_i^p$$

$$\frac{\partial E_4^p}{\partial w_{ji}} = \left(N_{C(p)}(a_{H_j}^p - \overline{a_{H_j}^{C(p)}})\right) a_{H_j}^p(1 - a_{H_j}^p)x_i^p$$

with $N_{C(p)}$ being the number of training patterns in the same class as p and $\overline{a_{H_j}^{C(p)}}$ is the average activation of the jth hidden unit when patterns in $C(p)$ are presented at the input layer.

Computing $\partial E_3^p/\partial w_{ji}$ and $\partial E_4^p/\partial w_{ji}$ is also local because it requires only local information to be stored at each hidden unit, the average activation and number of examples for each class. However, unlike with E_2, the hidden units must also know the target class of each instance. This can be done by backpropagating the class label from the output layer to the hidden layer together with the error signal.

B. Experimental Results

The purpose of this second set of experiments is to evaluate the effectiveness of the new error terms E_3 and E_4 on the same five large datasets described earlier. The results are compared with regular error backpropagation, error backpropagation with E_2, and *C4.5rules*.

As described earlier in Section III, the results of RE using $E = \alpha E_1 + \beta E_2$ already show a clear improvement over $E = E_1$ (summarized in rows E_1 and $E_1 + E_2$ of Table V). The results with E_3 and E_4 are even better for four out of five datasets. The numbers of rules for the *waveform* and *splice* datasets are reduced further by 40%, the number of rules for the *nursery* dataset is reduced further by 31%, and the number of rules for the *image-segmentation* dataset is reduced further by 19%, with no significant change in classification accuracy rates. For the fifth dataset *yeast*, the result is the same as with E_2.

We used paired t-tests to determine whether the reduction in numbers of rules and the change in accuracy caused by training with E_1 versus $E_1 + E_2$ and $E_1 + E_2$ versus $E_1 + E_3 + E_4$ are significant. Bonferroni correction for 20 tests (5 datasets $\times$ 2 settings $\times$ 2 criteria) requires $p < 0.05/20 = 0.0025$. Statistically significant changes are printed in italics in Table V. Corresponding p-values are shown in Tables VI and VII. The tests showed that $E_1 + E_3 + E_4$ further reduced the numbers of rules significantly compared to training with $E_1 + E_2$ in three datasets *waveform, image-segmentation, splice* with $p < 0.0001$. For the other two, the reduction is still significant compared to training with E_1 (regular error backpropagation) with $p < 1.4 \times 10^{-11}$. These tests also showed that the changes in accuracy rates are not significant with the exception of *waveform* and *image-segmentation* using $E_1+E_3+E_4$ versus E_1+E_2, and *nursery* using E_1 versus $E_1 + E_3 + E_4$. Interestingly, in these three cases, the accuracy rates actually *increased* with the use of the newer penalty terms. Overall, training with new error terms E_2, E_3, and E_4 showed a significant reduction in number of rules with insignificant change in accuracy over regular error backpropagation.

When extracting rules from datasets having all discrete input attributes such as *nursery* and *splice*, it is important

TABLE VIII
MEAN (STANDARD DEVIATION) OF NUMBER OF ANTECEDENTS OVER 100 RUNS

Dataset	Nursery	Splice
E_1	5.6 (1.1)	6.3 (1.4)
E_1&E_2	3.4 (1.1)	6.6 (1.1)
E_1&E_3&E_4	3.1 (0.8)	6.1 (1.2)
C4.5rules	3.5 (0.2)	4.5 (0.1)

to have small numbers of antecedents per rule to keep the rules easier to understand. Further, it is conceivable that, when lowering the number of rules when using E_2 or $E_3 + E_4$, one might simultaneously be increasing the number of antecedents per rule, thereby compromising the parsimony gained by the modified hidden layer presentation. Table VIII, which shows the average number of antecedents per rule for each method, indicates that this problem did not occur. These averages are either lower or almost the same. It shows that our method was able to reduce the number of rules without making the rules more complex. With the *nursery* dataset, the rules extracted using the new error terms actually have fewer antecendents per rule. The average is about the same as with *C4.5rules*, yet there are fewer rules compared to the *C4.5rules* result. Fig. 4 shows one rule set extracted from the dataset *splice* with 17 rules where the average number of antecendents are higher than with *C4.5rules* but still not overly complex to understand.

When extracting rules from datasets having continuous attributes such as *waveform*, *yeast*, and *image-segmentation*, the parsimony of rules is measured by the number of terms left in 2, which also indicates the number of input-hidden weights after pruning. The average numbers of these weights are increased insignificantly from 49.8 to 54.2 for the *waveform* dataset and from 17.3 to 17.6 for the *yeast* dataset, but dropped slightly from 65.66 to 63.67 for both E_2 and E_3+E_4 compared to E_1. It should be noted that, because the format of our rules (inequalities) for these two continuous input datasets are different from *C4.5's*, the *number of rules* are not directly comparable. Pruning has a side effect that reduces the *fidelity* of the RE method. Fidelity is a measure of how closely the extracted rules follow the network's behavior. In all of our experiments, the average accuracies of the rules and the networks differed by no more than 3%.

More significantly, the best among 100 runs in the experiments were able to extract even smaller rule sets than the averages described above. RE using $E_3 + E_4$ extracted only 5 rules explaining the classification of 5000 *waveform* data instances with 89% accuracy rate, 23 rules for the *yeast* dataset with 52% accuracy, 12 rules for the *image-segmentation* dataset with 92.21% accuracy, 13 rules for the *nursery* dataset with 93.8% accuracy, and 17 rules for the *splice* dataset with 94% accuracy. RE with new error terms clearly outperformed the popular rule-based system *C4.5rules* in extracting rules for four out of five datasets. For the remaining *yeast* dataset, it also helped to reduce the number of rules, although not enough to surpass *C4.5rules*.

C4.5rules' rules usually overlap, have to be applied in order, and a *default class* is assigned if no rule matches the input data.

$a_{28} \neq A, a_{31} \neq C \rightarrow N$
$a_{29} \neq G, a_{34} \neq T \rightarrow N$
$a_{28} \neq A, a_{30} \neq T \rightarrow N$
$a_{29} \neq G, a_{30} \neq T \rightarrow N$
$a_{30} = T, a_{31} = C, a_{34} = T \rightarrow$ E
$a_{17} \neq G, a_{20} \neq A, a_{23} \neq C, a_{27} \neq C, a_{27} \neq C, a_{28} = A, a_{29} = G, a_{31} \neq C \rightarrow I$
$a_{28} \neq A, a_{32} = A \rightarrow N$
$a_4 \neq A, a_{22} \neq T, a_{27} \neq C, a_{27} \neq C, a_{28} = A, a_{29} = G, a_{31} \neq C \rightarrow I$
$a_{27} = C, a_{31} \neq C \rightarrow N$
$a_{17} \neq G, a_{20} \neq A, a_{21} \neq G, a_{23} \neq C, a_{27} \neq C, a_{27} \neq C, a_{28} = A, a_{29} = G, a_{32} \neq A, a_{34} \neq T \rightarrow I$
$a_{29} \neq G, a_{30} = T, a_{31} = C, a_{32} = A \rightarrow E$
$a_4 \neq A, a_{21} \neq G, a_{22} \neq T, a_{27} \neq C, a_{27} \neq C, a_{28} = A, a_{29} = G, a_{32} \neq A, a_{34} \neq T \rightarrow I$
$a_{29} \neq G, a_{33} = G \rightarrow N$
$a_4 \neq A, a_{22} \neq T, a_{27} \neq C, a_{27} \neq C, a_{28} = A, a_{29} = G, a_{30} \neq T \rightarrow I$
$a_{17} \neq G, a_{20} \neq A, a_{23} \neq C, a_{27} \neq C, a_{27} \neq C, a_{28} = A, a_{29} = G, a_{30} \neq T \rightarrow I$
$a_{13} \neq G, a_{27} = C, a_{31} \neq C \rightarrow N$
default: N

Fig. 4. Rule set extracted from a neural network trained on the *splice* dataset. Here a_i denotes the nucleotide at position i, where a_i can be either A, T, G, or C. I (intron), E (exon), and N (neither) are the three classes of the DNA sequences to be predicted.

Some of the rules have the same outcome as the *default class*. Since our method uses *C4.5rules*, our final combined rules also have the same properties. When we removed the rules that have the same outcome as the *default*, we expected that the accuracy rates could decrease significantly, but they did not. Table IX shows the results with these rules removed. The average numbers of rules were significantly lower, while there are insignificant changes to the accuracy rates. The reason could be that each intermediate rule set is responsible for an interval of hidden unit activation so its rules are simple and not overlapping. Such rule sets can be simplified by removing rules having the same outcome as the default class. The combination of these simple rules thus does not need these extra rules either.

Table X shows the average running time in seconds spent on RE and the total running time including training the networks. All experiments were run on an Intel Core2 2.4-GHz system. We found that the new penalty terms increased the running time due to more computation, as would be expected. The most increase in running time was recorded with the *splice* dataset and it was only 4.4 times. In their original forms, E_2, E_3, and E_4 require N times more computation to compute than E_1 where N takes values from 1296 to 5000. But our derivation using local computations made it possible to keep the increase in running time surprisingly low even though $N = 3190$ for the *splice* dataset.

C. Experiment with an Artificial Dataset

We also evaluated our RE algorithm on the three MONKS problems [20]. These small artificial datasets are generated from known rules so they provide another way to verify our RE algorithm. Table XI shows the results for these three problems.

TABLE IX
EXPERIMENTAL RESULTS: MEAN (STD DEV.) OVER 100 RUNS WITH RULES RESULTING IN DEFAULT CLASS REMOVED

Dataset	Waveform		Yeast		Imgseg	
	# rules	Accuracy	# rules	Accuracy	# rules	Accuracy
E_1	44.34 (18.18)	85.08% (1.94)	61.29 (19.04)	51.61% (4.40)	38.34 (9.27)	91.58% (2.33)
$E_1\&E_2$	9.33 (8.24)	85.20% (2.08)	35.86 (12.94)	51.42% (4.41)	32.02 (7.37)	89.29% (3.06)
$E_1\&E_3\&E_4$	6.20 (2.40)	85.73% (2.32)	37.52 (12.86)	51.86% (4.19)	26.12 (8.18)	91.86% (2.11)

Dataset	Nursery		Splice	
	# rules	Accuracy	# rules	Accuracy
E_1	113.04 (69.74)	88.57% (4.04)	28.73 (31.61)	88.55% (4.45)
$E_1\&E_2$	25.25 (29.78)	89.37% (2.81)	31.19 (31.26)	89.18% (4.32)
$E_1\&E_3\&E_4$	18.85 (13.56)	89.94% (2.43)	15.88 (6.25)	90.12% (4.57)

TABLE X
AVERAGE TIME FOR RE AND TOTAL RUNNING TIME (S)

Dataset	Waveform		Yeast		Imgseg		Nursery		Splice	
	RE	total	RE	total	RE	total	RE	total	RE	total
E_1	7.3	12.1	1.1	2.5	3.6	7.5	0.8	1.8	0.8	3.3
$E_1\&E_2$	4.5	11.3	1.0	2.6	3.6	9.0	0.7	2.2	0.8	7.7
$E_1\&E_3\&E_4$	6.9	17.7	1.0	2.9	3.6	11.9	0.7	2.8	0.9	14.5

TABLE XI
RESULTS WITH MONKS DATASETS (AVERAGE OVER 100 RUNS)

Dataset	our methods		*C4.5rules*		Source
	# rules	Accuracy	# rules	Accuracy	# rules
MONKS 1	6.8	93.11%	14	100%	5
MONKS 2	7.53	77.16%	15	66.2%	16
MONKS 3	2.65	97.15%	14	96.3%	3

The results are the same for E_1, $E_1 + E_2$, and $E_1 + E_3 + E_4$ probably because the results of the control experiment already approached the best obtainable results from this method, so only one row is given per dataset.

Results with the MONKS 3 problem are very interesting in that our method found even *fewer* rules than the rule set that produced the data (rightmost column of Table XI) while retaining very high accuracy. The reason is that 5% noise was added to the training set but not the testing set. Our method found a rule set with size 2, which has an accuracy rate of 97.22% on test data. This rule set is consider acceptable because only 95% of the training data have correct labels. In order to have 100% accuracy on the test set, a bigger rule set with three rules is required, but its accuracy rate on the training data is only 95%. Noise was not added to the MONKS 1 and MONKS 2 datasets.

In the MONKS 1 problem, *C4.5rules* tended to overprune trees and produce hidden-output rules that have low accuracy in Step 3 of our algorithm. We believe the reason for this is that it usually has 6–8 unique data instances in this step (there is usually much less data for the hidden-output RE step), and that is too few to reliably prune its decision tree. Using this output from *C4.5rules*, we obtained a slightly low accuracy rate although the trained neural networks had 99.79% accuracy rates. The average 6.8 is also very close to the five rules used to generate the data so it is almost impossible for the penalty terms to find a better rule set in this case. The MONKS 2 dataset was generated using rules of the type N-of-M, which is very difficult to express using our rule type. Our algorithm and *C4.5rules* could not discover enough rules to raise the accuracy rates. *C4.5rules* by itself found 15 rules on average with 66.2% accuracy.

V. DISCUSSION AND CONCLUSION

In this paper, we presented a method to improve the extraction of symbolic rules from multilayer feedforward neural networks by adding additional terms to the error function. These terms encourage the formation of a more separable internal representation at the hidden layer. We also derived and implemented efficient ways to incorporate these new error terms into the training process *while retaining local computations*. Unlike past RE methods, our approach focuses on modifying training so that existing RE methods work more effectively. The three introduced penalty terms E_2, E_3, and E_4 share the same purpose of making the hidden unit activations of different classes more separable. While E_2 is simple and does not rely on class label, E_3 and E_4 are more complex and employ class labels to increase and decrease the activation distances discretionarily.

Extensive experiments with five large publicly available datasets showed that our approach helped to reduce the number of rules significantly without sacrificing classification accuracy. Rule sets extracted from networks trained with E_2 are smaller than with regular error backpropagation. Even fewer rules can be extracted from networks trained with E_3 and E_4.

These results showed that our method outperforms the popular *C4.5rules* program in four out of five of these datasets. An important future research direction will be to compare these results with those of other RE methods in the literature.

Our aim was not to produce a new RE algorithm but to provide an improved way to train networks that might help most RE algorithms using the decompositional approach. The RE process that we used is fairly standard and similar to those used in many other algorithms. Those algorithms use different approaches to extract the intermediate rules, from exhaustive search to complicated heuristics. Here we used *C4.5rules* as a standard and straightforward algorithm in order to generate the intermediate rules. This should not be confused with using *C4.5rules* to generate rules directly from the whole dataset. We used the same RE algorithm for both the control condition (regular backpropagation using E_1 alone) and experimental learning (E_2, E_3, and E_4) to show the effectiveness of the new terms on RE algorithms using the hidden activation intervals. It is hypothesized that *any* RE method using a similar approach will benefit from the use of these terms.

The surprising result with the default class in Table IX demonstrated another advantage of neural-network-based RE. The extracted rules often appear to be nonoverlapping, so that rules resulting in the default class can be removed. Such rule sets are easier to apply and also easier for a person to use to study properties of datasets. Although it is not clear what caused the rules to be nonoverlapping, the answer is likely to be the way neural networks using regular backpropagation divide the input space using hidden unit activation intervals.

Experimental results showed that accuracy rates and E_1 changed very little when networks were trained with the augmented penalty terms. This demonstrates a well-known property of neural networks, there are many possible encodings at the hidden layer that can provide correct outputs. These encodings are biased toward more separation of activity patterns with our approach. Based on the promising results of this approach, an important future research direction will be to study other ways to bias the encodings beyond the sum of squared distances. Presumably, our approach can also be applied to other error backpropagation learning rules with different error functions such as in [24]–[26]. An especially important learning method for examination in this context is the Levenberg–Marquardt algorithm [27], [28] due to its effectiveness.

As with many approaches using penalty terms, there is a tradeoff in terms of parameter adjustments, a small value of α will make the activation patterns very separated and good for RE, but it cannot keep the training error low enough. The opposite holds for small β. The problem is to influence the training enough to produce the desired separation without compromising E_1 or accuracy. Since we found that there is no single value of β that works best across all datasets, an important future direction for research is to study different schemes to adapt α and β automatically as training progresses. Such work might try to establish properties of a dataset that predict reasonable values for β, or investigate how varying β during training as a function of error rate influences the rule acquisition process.

APPENDIX A
COMPUTING $\partial E_2^p/\partial w_{ji}$

Let $E_2 = \sum_p E_2^p$ where

$$E_2^p = -\frac{1}{2}\sum_{q=1}^{N}\sum_{k\in hidden}(a_{H_k}^p - a_{H_k}^q)^2.$$

The gradient of the error E_2^p with respect to weight w_{ji} can be calculated as [29]

$$\frac{\partial E_2^p}{\partial w_{ji}} = \sum_{k\in hidden}\frac{\partial E_2^p}{\partial a_{H_k}^p}\frac{\partial a_{H_k}^p}{\partial w_{ji}}.$$

Because $\partial a_{H_k}^p/\partial w_{ji} = 0$ with $k \neq j$, we have

$$\begin{aligned}\frac{\partial E_2^p}{\partial w_{ji}} &= \frac{\partial E_2^p}{\partial a_{H_j}^p}\frac{\partial a_{H_j}^p}{\partial w_{ji}}\\ &= -\frac{1}{2}\sum_{q=1}^{N}\sum_{k\in hidden}\frac{\partial (a_{H_k}^p - a_{H_k}^q)^2}{\partial a_{H_j}^p}\frac{\partial a_{H_j}^p}{\partial w_{ji}}\\ &= -\sum_{q=1}^{N}\sum_{k\in hidden}(a_{H_k}^p - a_{H_k}^q)\left(\frac{\partial a_{H_k}^p}{\partial a_{H_j}^p} - \frac{\partial a_{H_k}^q}{\partial a_{H_j}^p}\right)\frac{\partial a_{H_j}^p}{\partial w_{ji}}.\end{aligned}$$

With $k \neq j$, we have $(\partial a_{H_k}^p/\partial a_{H_j}^p)(\partial a_{H_j}^p/\partial w_{ji}) = 0$ and $(\partial a_{H_k}^q/\partial a_{H_j}^p)(\partial a_{H_j}^p/\partial w_{ji}) = 0$ because $a_{H_k}^p$ and $a_{H_k}^q$ do not have w_{ji} components. So

$$\begin{aligned}\frac{\partial E_2^p}{\partial w_{ji}} &= -\sum_{q=1}^{N}\left[(a_{H_j}^p - a_{H_j}^q)\left(\frac{\partial a_{H_j}^p}{\partial a_{H_j}^p} - \frac{\partial a_{H_j}^q}{\partial a_{H_j}^p}\right)\right]\times\frac{\partial a_{H_j}^p}{\partial w_{ji}}\\ &= -\sum_{q=1}^{N}\left[(a_{H_j}^p - a_{H_j}^q)\left(1 - \frac{\partial a_{H_j}^q}{\partial a_{H_j}^p}\right)\right]\times\frac{\partial a_{H_j}^p}{\partial w_{ji}}.\end{aligned}$$

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

$$\begin{aligned}\frac{\partial E_2^p}{\partial w_{ji}} &= -\sum_{q=1,q\neq p}^{N}(a_{H_j}^p - a_{H_j}^q)\times\frac{\partial a_{H_j}^p}{\partial w_{ji}}\\ &= -\left((N-1)a_{H_j}^p - \sum_{q=1,q\neq p}^{N}a_{H_j}^q\right)\frac{\partial a_{H_j}^p}{\partial w_{ji}}\\ &= -\left(Na_{H_j}^p - \sum_{q=1}^{N}a_{H_j}^q\right)\frac{\partial a_{H_j}^p}{\partial w_{ji}}\\ &= -N(a_{H_j}^p - \overline{a_{H_j}})\frac{\partial a_{H_j}^p}{\partial w_{ji}} \qquad (3)\end{aligned}$$

with N being the number of training patterns (a constant) and $\overline{a_{H_j}}$ being the average activation of the jth hidden unit over all input samples. As with the usual backpropagation derivation using logistic transfer function, we have $\partial a_{H_j}^p/\partial w_{ji} = a_{H_j}^p(1 - a_{H_j}^p)x_i^p$. Thus

$$\frac{\partial E_2^p}{\partial w_{ji}} = -N(a_{H_j}^p - \overline{a_{H_j}})a_{H_j}^p(1 - a_{H_j}^p)x_i^p.$$

APPENDIX B
COMPUTING $\partial E_3^p/\partial w_{ji}$ AND $\partial E_4^p/\partial w_{ji}$

Computing $\partial E_3^p/\partial w_{ji}$ follows the same system as with $\partial E_2^p/\partial w_{ji}$, until (3). The only difference is that the sum is only over q's that are in a different class from p. Let $C(p)$ be the set of training patterns having the same class as p. We have

$$\frac{\partial E_3^p}{\partial w_{ji}} = -\sum_{q\notin C(p)} (a_{H_j}^p - a_{H_j}^q) \times \frac{\partial a_{H_j}^p}{\partial w_{ji}} = -\left(\sum_{q\notin C(p)} a_{H_j}^p - \sum_{q\notin C(p)} a_{H_j}^q\right) \times \frac{\partial a_{H_j}^p}{\partial w_{ji}}. \quad (4)$$

Consider the left factor of (4)

$$\begin{aligned}
&\left(\sum_{q\notin C(p)} a_{H_j}^p - \sum_{q\notin C(p)} a_{H_j}^q\right) \\
&= (N - N_{C(p)})a_{H_j}^p - \left(\sum_{q=1}^{N} a_{H_j}^q - \sum_{q\in C(p)} a_{H_j}^q\right) \\
&= Na_{H_j}^p - N_{C(p)}a_{H_j}^p - N\overline{a_{H_j}} + N_{C(p)}\overline{a_{H_j}^{C(p)}} \\
&= N(a_{H_j}^p - \overline{a_{H_j}}) - N_{C(p)}(a_{H_j}^p - \overline{a_{H_j}^{C(p)}}) \\
&= N(a_{H_j}^p - \overline{a_{H_j}}) - N_{C(p)}(a_{H_j}^p - \overline{a_{H_j}^{C(p)}}) \quad (5)
\end{aligned}$$

with $N_{C(p)}$ being the number of training patterns in the same class as p and $\overline{a_{H_j}^{C(p)}}$ is the average activation of the jth hidden unit when patterns in $C(p)$ are presented at the input layer. Substituting (5) into (4) gives

$$\frac{\partial E_3^p}{\partial w_{ji}} = \left(N(a_{H_j}^p - \overline{a_{H_j}}) - N_{C(p)}(a_{H_j}^p - \overline{a_{H_j}^{C(p)}})\right) \times a_{H_j}^p(1 - a_{H_j}^p)x_i^p.$$

Similarly, we can compute $\partial E_4^p/\partial w_{ji}$ as

$$\begin{aligned}
\frac{\partial E_4^p}{\partial w_{ji}} &= \sum_{q\in C(p)} (a_{H_j}^p - a_{H_j}^q) \times \frac{\partial a_{H_j}^p}{\partial w_{ji}} \\
&= \left(\sum_{q\in C(p)} a_{H_j}^p - \sum_{q\in C(p)} a_{H_j}^q\right) \times \frac{\partial a_{H_j}^p}{\partial w_{ji}} \\
&= \left(N_{C(p)}a_{H_j}^p - N_{C(p)}\overline{a_{H_j}^{C(p)}}\right) \times \frac{\partial a_{H_j}^p}{\partial w_{ji}} \\
&= \left(N_{C(p)}(a_{H_j}^p - \overline{a_{H_j}^{C(p)}})\right) \times \frac{\partial a_{H_j}^p}{\partial w_{ji}} \\
&= \left(N_{C(p)}(a_{H_j}^p - \overline{a_{H_j}^{C(p)}})\right) a_{H_j}^p(1 - a_{H_j}^p)x_i^p.
\end{aligned}$$

REFERENCES

[1] R. Andrews, A. Tickle, and J. Diederich, "A review of techniques for extracting rules from trained artificial neural networks," in *Clinical Applications of Artificial Neural Networks*. Cambridge, U.K.: Cambridge Univ. Press, 2001, pp. 256–297.

[2] S. Bader, S. Holldobler, and V. Mayer-Eichberger, "Extracting propositional rules from feed-forward neural networks a new decompositional approach," in *Proc. 3rd Int. Workshop Neural-Symbolic Learn. Reason.*, 2007, pp. 1–6.

[3] W. Duch, R. Adamczak, and K. Grabczewski, "A new methodology of extraction, optimization and application of crisp and fuzzy logical rules," *IEEE Trans. Neural Netw.*, vol. 12, no. 2, pp. 277–306, Mar. 2001.

[4] H. Jacobsson, "Rule extraction from recurrent neural networks: A taxonomy and review," *Neural Comput.*, vol. 17, no. 6, pp. 1223–1263, Jun. 2005.

[5] R. Nayak, "Generating rules with predicates, terms and variables from the pruned neural networks," *Neural Netw.*, vol. 22, no. 4, pp. 405–414, May 2009.

[6] T. Etchells and P. Lisboa, "Orthogonal search-based rule extraction (OSRE) for trained neural networks: A practical and efficient approach," *IEEE Trans. Neural Netw.*, vol. 17, no. 2, pp. 374–384, Mar. 2006.

[7] R. Setiono, B. Baesens, and C. Mues, "Recursive neural network rule extraction for data with mixed attributes," *IEEE Trans. Neural Netw.*, vol. 19, no. 2, pp. 299–307, Feb. 2008.

[8] J. Huysmans, R. Setiono, B. Baesens, and J. Vanthienen, "Minerva: Sequential covering for rule extraction," *IEEE Trans. Syst., Man, Cybern., Part B: Cybern.*, vol. 38, no. 2, pp. 299–309, Apr. 2008.

[9] H. Tsukimoto, "Extracting rules from trained neural networks," *IEEE Trans. Neural Netw.*, vol. 11, no. 2, pp. 377–389, Mar. 2000.

[10] R. Setiono, W. Leow, and J. Zurada, "Extraction of rules from artificial neural networks for nonlinear regression," *IEEE Trans. Neural Netw.*, vol. 13, no. 3, pp. 564–577, May 2002.

[11] S. Haykin, *Neural Networks: A Comprehensive Foundation*. Upper Saddle River, NJ: Prentice-Hall, 1994.

[12] J. Quinlan, *C4.5: Programs for Machine Learning*. San Mateo, CA: Morgan Kaufmann, 1993.

[13] H. Lu, R. Setiono, and H. Liu, "Neurorule: A connectionist approach to data mining," in *Proc. 21st Int. Conf. Very Large Data Bases*, 1995, pp. 478–489.

[14] R. Setiono and H. Liu, "Neurolinear: From neural networks to oblique decision rules," *Neurocomputing*, vol. 17, no. 1, pp. 1–24, Sep. 1997.

[15] G. G. Towell and J. W. Shavlik, "Extracting refined rules from knowledge-based neural networks," *Mach. Learn.*, vol. 13, no. 1, pp. 71–101, Oct. 1993.

[16] T. Huynh and R. Setiono, "Effective neural network pruning using cross-validation," in *Proc. Int. Joint Conf. Neural Netw.*, vol. 2. Jul.–Aug. 2005, pp. 972–977.

[17] M. Riedmiller and H. Braun, "RPROP - A fast adaptive learning algorithm," in *Proc. Int. Symp. Comput. Inform. Sci.*, Antalya, Turkey, 1992, pp. 279–286.

[18] A. Krogh and J. Hertz, "A simple weight decay can improve generalization," in *Advances in Neural Information Processing Systems*. San Mateo, CA: Morgan Kaufmann, 1992, pp. 950–957.

[19] H. Liu and R. Setiono, "Chi2: Feature selection and discretization of numeric attributes," in *Proc. 7th IEEE Int. Conf. Tools Artif. Intell.*, Herndon, VA, Nov. 1995, pp. 388–391.

[20] A. Asuncion and D. Newman. (2007). *UCI Machine Learning Repository* [Online]. Available: http://www.ics.uci.edu/~mlearn/MLRepository.html

[21] R. Duda, P. Hart, and D. Stock, *Pattern Classification and Scene Analysis*. New York: Wiley, 2001.

[22] P. Horton and K. Nakai, "A probabilistic classification system for predicting the cellular localization sites of proteins," in *Proc. Int. Conf. Intell. Syst. Mol. Biol.*, vol. 4. 1996, pp. 109–115.

[23] J. Foster, E. Barkus, and C. Yavorsky, *Understanding and Using Advanced Statistics*. Newbury Park, CA: Sage, 2006.

[24] S. Cho and J. Reggia, "Learning competition and cooperation," *Neural Comput.*, vol. 5, no. 2, pp. 242–259, Mar. 1993.

[25] Z. Xu, R. Zhang, and W. Jing, "When does online BP training converge?" *IEEE Trans. Neural Netw.*, vol. 20, no. 10, pp. 1529–1539, Oct. 2009.

[26] O. Ludwig and U. Nunes, "Novel maximum-margin training algorithms for supervised neural networks," *IEEE Trans. Neural Netw.*, vol. 21, no. 6, pp. 972–984, Jun. 2010.

[27] M. Hagan and M. Menhaj, "Training feedforward networks with the Marquardt algorithm," *IEEE Trans. Neural Netw.*, vol. 5, no. 6, pp. 989–993, Nov. 1994.

[28] B. Wilamowski and H. Yu, "Improved computation for Levenberg–Marquardt training," *IEEE Trans. Neural Netw.*, vol. 21, no. 6, pp. 930–937, Jun. 2010.

[29] T. Huynh and J. Reggia, "Improving rule extraction from neural networks by modifying hidden layer representations," in *Proc. Int. Joint Conf. Neural Netw.*, Atlanta, GA, Jun. 2009, pp. 1316–1321.

Thuan Q. Huynh received the Bachelors degree in computing from the National University of Singapore, Singapore, in 2005, and the Master of Science degree from the Department of Computer Science, University of Maryland, College Park, in 2005. He is currently pursuing the Ph.D. degree in the Department of Computer Science, University of Maryland.

His current research interests include neural networks, bio-inspired computing, and artificial intelligence.

James A. Reggia received the M.D. and Ph.D. degrees in computer science from the University of Maryland, College Park, in 1975 and 1981, respectively.

He is a Professor of computer science with joint appointments in the Institute for Advanced Computer Studies, University of Maryland, and in the Department of Neurology, University of Maryland School of Medicine, Baltimore. He has focused on developing new methods for creating large-scale integrated neuro-computational systems as a basis for machine intelligence and neurocognitive modeling. This work has included simulating specific cortical regions for visual and language processing, simulation of basic cortical mechanisms, such as self-organizing map formation, learning to process temporal sequences, emergence of hemispheric asymmetries and specialization, and working memory. He has published many research papers in refereed journals and proceedings of conferences in the below research areas. His current research interests include span neural computation, cause–effect/abductive reasoning, artificial life, and genetic programming.