Pruned neural networks for regression

Rudy Setiono and Wee Kheng Leow

School of Computing
National University of Singapore
Singapore 117543

Abstract. Neural networks have been widely used as a tool for regression. They are capable of approximating any function and they do not require any assumption about the distribution of the data. The most commonly used architectures for regression are the feedforward neural networks with one or more hidden layers. In this paper, we present a network pruning algorithm which determines the number of units in the input and hidden layers of the networks. We compare the performance of the pruned networks to four regression methods namely, linear regression (LR), Naive Bayes (NB), k-nearest-neighbor (kNN), and a decision tree predictor $M5'$. On 32 publicly available data sets tested, the neural network method outperforms NB and kNN if the prediction errors are computed in terms of the root mean squared errors. Under this measurement metric, it also performs as well as LR and $M5'$. On the other hand, using the mean absolute error as the measurement metric, the neural network method outperforms all four other regression methods.

1 Introduction

In addition to pattern classification problems, regression or function approximation is the predictive learning problem for which feedforward neural networks have been widely applied. Neural networks have several advantages over statistical regression techniques. First, no assumption about the distribution of the data is required. Second, there is no need to select the regression model *a priori*. And third, neural networks have been shown to be capable of approximating any continuous function with arbitrary precision [3, 7].

Different problems require different network architecture and selecting an appropriate network architecture is the most important step in obtaining an accurate model for regression. Since we restrict ourselves to networks with a single hidden layer, architecture selection boils down to finding appropriate numbers of units in the input and hidden layers. To find an appropriate number of hidden units, constructive algorithms start with a few hidden units and add more units as needed to improve network accuracy [1, 8, 14]. Destructive algorithms, on the other hand, start with a large number of hidden units and remove those that are found to be redundant [11]. The number of useful input units correspond to the number of relevant input attributes of the data. Typical algorithms usually start by assigning one input unit to each attribute, train the network with all input attributes and then remove network input units that correspond to irrelevant data attributes [15, 16]. Various measures of the contribution of an input attribute to the network predictive accuracy have been developed [2, 10, 13, 18].

The purpose of this paper is (1) to present an algorithm for removing redundant or irrelevant input and hidden units from feedforward neural networks for regression and (2) to compare the predictive accuracy of the neural networks with those of other methods for regression on publicly available data sets. Our proposed pruning algorithm removes units from the network by making use of a cross-validating data set. The weights of network connections from a unit that is considered for removal are set to zero and the network is retrained. If the accuracy of the network on the cross-validation set improves or deteriorates within an acceptable level, then the unit is pruned from the network. The same criteria for removal is applied to the input and hidden units. The pruning process is terminated when no unit can be removed without causing the network accuracy on the cross-validation set to drop below the prescribed level.

While there are several papers that propose algorithms for constructing and/or training neural network for regression [6, 8, 9], we have been unable to find a paper that compares the accuracy of neural networks for regression against those of other traditional methods such as statistical regression method. A recent study by Frank et al. [5] on the application of naive Bayes methodology for regression provides us with an excellent opportunity for making comparisons among the various regression methods. Test results from thirty-two problems, all but one are real-world problems, are reported in the study. The data sets are available from their website[1] as part of the WEKA project. The results from our network pruning algorithm show that neural networks perform as well as linear regression if the prediction errors are measured in terms of the root mean squared errors. However, using the mean absolute error as the measurement metric, neural networks outperform linear regression and three other regression methods.

The paper is organized as follows. Section 2 presents the neural network architecture, training and pruning for regression. Section 3 describes our pruning algorithm. Section 4 presents the results from our pruning algorithm and compares them to those of other methods reported in [5]. Finally, Section 5 discusses future works and concludes the paper.

2 Network training and pruning

In this section we describe our training and pruning algorithm. The available data samples $(\mathbf{x}^i, y^i), i = 1, 2, \ldots$, where $\mathbf{x}^i \in \mathbb{R}^N$ and $y^i \in \mathbb{R}$, are first randomly divided into 3 subsets: the training, the cross-validation and the test sets. Using the training data set, a network with H hidden units is trained, so as to minimize

[1] http://www.cs.waikato.ac.nz/~ml/weka/index.html

the sum of squared errors $E(\mathbf{w},\mathbf{v})$ augmented with a penalty term $P(\mathbf{w},\mathbf{v})$:

$$E(\mathbf{w},\mathbf{v}) = \sum_{i=1}^{K} \left(\tilde{y}^i - y^i\right)^2 + P(\mathbf{w},\mathbf{v}) \tag{1}$$

$$P(\mathbf{w},\mathbf{v}) = \epsilon_1 \left(\sum_{m=1}^{H} \sum_{\ell=1}^{N} \frac{\beta w_{m\ell}^2}{1+\beta w_{m\ell}^2} + \sum_{m=1}^{H} \frac{\beta v_m^2}{1+\beta v_m^2} \right) + \epsilon_2 \left(\sum_{m=1}^{H} \sum_{\ell=1}^{N} w_{m\ell}^2 + \sum_{m=1}^{H} v_m^2 \right) \tag{2}$$

where K is the number of samples in the training data set, $\epsilon_1, \epsilon_2, \beta$ are positive penalty parameters, and $\tilde{y}^i$ is the predicted function value for input sample $\mathbf{x}^i$

$$\tilde{y}^i = \sum_{m=1}^{H} \delta\left((\mathbf{x}^i)^T \mathbf{w}_m\right) v_m,$$

$\mathbf{w}_m \in \mathbb{R}^N$ is the vector of network weights from the input units to hidden unit m, $v_m \in \mathbb{R}$ is the network weight from hidden unit m to the output unit, $\delta(\xi)$ is the hyperbolic tangent function $(e^{\xi} - e^{-\xi})/(e^{\xi} + e^{-\xi})$, and $(\mathbf{x}^i)^T \mathbf{w}_m$ is the scalar product of $\mathbf{x}^i$ and $\mathbf{w}_m$.

A local minimum of the error function $E(\mathbf{w},\mathbf{v})$ can be obtained by applying any nonlinear optimization methods such as the gradient descent method or the quasi-Newton method. In our implementation, we have used a variant of the quasi-Newton method, namely the BFGS method [4] due to its faster convergence rate than the gradient descent method.

A new pruning algorithm called N2PFA (Neural Network Pruning for Function Approximation) is proposed. In the algorithm, the mean absolute error (MAD) of the network's prediction is used to measure the network's performance. In particular, MAD p on the training set $\mathcal{T}$ and MAD q on the cross-validation set $\mathcal{X}$ are used to determine when pruning should be terminated:

$$p = \frac{1}{|\mathcal{T}|} \sum_{y^i \in \mathcal{T}} |\tilde{y}_i - y^i| \qquad q = \frac{1}{|\mathcal{X}|} \sum_{y^i \in \mathcal{X}} |\tilde{y}_i - y^i| \tag{3}$$

Algorithm N2PFA

Given: Data set $(\mathbf{x}^i, y^i), i = 1, 2, \ldots, K$.
Objective: Find a neural network that fits the data and generalizes well.

Step 1. Split the data into 3 subsets: training, cross-validation, and test sets.
Step 2. Train a network with a relatively large number of hidden units to minimize the error function (1).
Step 3. Compute p and q, and set $pbest = p$, $qbest = q$, $ermax = \max\{pbest, qbest\}$.
Step 4. Remove redundant hidden units:

1. For each $m = 1, 2, \ldots, H$,
   set $v_m = 0$ and compute the prediction errors p_m and q_m.

2. Retrain the network with $v_h = 0$ where $p_h = \min_m p_m$, and compute p and q of the retrained network.
3. If $p \le (1+\alpha) ermax$ and $q \le (1+\alpha) ermax$, then
   - Remove hidden unit h.
   - Set $pbest = \min\{p, pbest\}$, $qbest = \min\{q, qbest\}$ and $ermax = \max\{pbest, qbest\}$.
   - Set $H = H - 1$ and go to Step 4.1.

   Else use the previous setting of network weights.

Step 5. Remove irrelevant inputs:

1. For each $l = 1, 2, \ldots, N$,
   set $w_{ml} = 0$ for all m and compute the prediction errors p_l and q_l.
2. Retrain the network with $w_{mn} = 0$ for all m where $p_n = \min_l p_l$, and compute p and q of the retrained network.
3. If $p \le (1+\alpha) ermax$ and $q \le (1+\alpha) ermax$, then
   - Remove input unit n.
   - Set $pbest = \min\{p, pbest\}$, $qbest = \min\{q, qbest\}$ and $ermax = \max\{pbest, qbest\}$.
   - Set $N = N - 1$ and go to Step 5.1.

   Else use the previous setting of network weights.

Step 6. Report the accuracy of the network on the test data set.

The parameter $ermax$ is used to determine if a unit can be removed. Typically, at the beginning of the algorithm when there are many hidden units in the network, the training error p will be much smaller than the cross-validation error q. The value of p increases as more and more units are removed. As the network approaches its optimal structure, we expect q to decrease. As a result, if only $pbest$ is used to determine if a unit can be removed, many redundant units can be expected to remain in the network when the algorithm terminates because $pbest$ tends to be small at the beginning of the algorithm. On the other hand, if only $qbest$ is used, then the network would perform well on the cross-validation set but may not necessarily generalizes well on the test set. This could be caused by the small number of samples available for cross-validation or the uneven distribution of the data in the training and cross-validation sets. Therefore, $ermax$ is assigned the larger of $pbest$ and $qbest$ so as to remove as many redundant units as possible without sacrificing generalization accuracy. The parameter α is introduced to control the chances that a unit will be removed. With a larger value of α, units are more likely to be removed. However, the accuracy of the resulting network on the test data set may deteriorate. We have conducted extensive experiments to find a value for this parameter that works well for all of our test problems. We report our experimental results in the next section.

3 Experimental results

3.1 Experimental methodology

The data sets used in the experiment and the summary of their attribute features are listed in Table 1. They are shown in increasing order of the number of samples. Most of the data sets consist of both numeric and discrete attributes. The total number of attributes ranges from 2 to 25. Except for problem no. 19 pwLinear, all of the problems are from real world domains.

Table 1. Characteristics of the datasets used for experiments.

No.	Dataset	Instances	Missing values (%)	Numeric attributes	Discrete attributes	Neural network inputs
1	schlvote	38	0.4	4	1	6
2	bolts	40	0.0	7	0	8
3	vineyard	52	0.0	3	0	4
4	elusage	55	0.0	1	1	14
5	pollution	60	0.0	15	0	16
6	mbagrade	61	0.0	1	1	3
7	sleep	62	2.4	7	0	8
8	auto93	93	0.7	16	6	62
9	baskball	96	0.0	4	0	5
10	cloud	108	0.0	4	2	10
11	fruitfly	125	0.0	2	2	9
12	echoMonths	131	7.5	6	3	11
13	veteran	137	0.0	3	4	11
14	fishcatch	158	6.9	5	2	15
15	autoPrice	159	0.0	15	0	16
16	servo	167	0.0	0	4	20
17	lowbwt	189	0.0	2	7	20
18	pharynx	195	1.1	1	10	37
19	pwLinear	200	0.0	10	0	11
20	autoHorse	205	1.1	17	8	69
21	cpu	209	0.0	6	1	37
22	bodyfat	252	0.0	14	0	15
23	breastTumor	286	0.3	1	8	37
24	hungarian	294	19.0	6	7	23
25	cholesterol	303	0.1	6	7	23
26	cleveland	303	0.1	6	7	23
27	autoMpg	398	0.2	4	3	26
28	pbc	418	15.6	10	8	30
29	housing	506	0.0	12	1	14
30	meta	528	4.3	19	2	66
31	sensory	576	0.0	0	11	33
32	strike	625	0.0	5	1	24

The following experimental setting were used to obtain the statistics from our network pruning algorithm:

– Ten-fold cross-validation scheme: We divided each data set randomly into 10 subsets of equal size. Eight subsets were used for training, one subset was used for cross validating, and one subset for measuring the predictive accuracy of the pruned network. This procedure was performed 10 times so that each subset was tested once. Test results were averaged over 20 ten-fold cross-validation runs.

- The same set of values for the penalty parameters in the penalty term (2) were used: $\epsilon_1 = 0.5, \epsilon_2 = 0.05$ and $\beta = 0.1$.
- During pruning, the value of α was set to 0.025.
- The starting number of hidden units for all problems was 8. The number of input units are shown in Table 1. The number of input units includes one unit with a constant input value of 1 to implement hidden unit bias.
- One input unit was assigned to each continuous attribute in the data set. Discrete attributes were binary coded. A discrete attribute with D possible values was assigned D network inputs.
- Continuous attribute values were scaled to range in the interval $[0, 1]$, while binary-encoded attribute values were either 0 or 0.2. We found that the 0/0.2 encoding produced better generalization than the usual 0/1 encoding.
- A missing continuous attribute value was replaced by the average of the non-missing values. A missing discrete attribute value was assigned the value "unknown" and the corresponding input $\mathbf{x}$ was set to the zero vector $\mathbf{0}$.
- Target output values were linearly scaled to range in the interval $[0, U]$, where U was 32 for bolts; 16 for auto93, fishcatch, autoPrice, servo, autoHorse, cpu, bodyfat and housing; and 4 for all other problems.

3.2 Results and comparison to other methods

The predictive accuracy of various regression methods have been measured in terms of the relative root mean squared error (RRMSE) and the relative mean absolute error (RMAE):

$$\mathrm{RRMSE} = 100 \times \sqrt{\sum_i (\tilde{y}_i - y_i)^2 / \sum_i (\overline{y} - y_i)^2}$$

$$\mathrm{RMAE} = 100 \times \sum_i |\tilde{y}_i - y_i| / \sum_i |\overline{y} - y_i|$$

where the summation is computed over the samples in the test set and $\overline{y}$ is the average value of y_i in the test set. These relative errors are preferred over the usual sum of squared errors because they normalize the differences in the output ranges of different data sets.

Our results are summarized in Tables 2 and 3. For comparison purpose, we also reproduce the statistics from Frank et al. [5] for four other regression methods. Naive Bayes (NB) method [5] applies Bayes' theorem to estimate the probability density function of the target value y given a sample $\mathbf{x}$. A crucial assumption is that given the predicted value y, the attributes of $\mathbf{x}$ are independent of each other. LR is the standard linear regression method. Attribute selection was accomplished by backward elimination. The k-nearest-neighbor (kNN) is a distance-weighted k-nearest-neighbor method. The value of k varied from 1 to 20 and the optimal value of k was chosen using leave-one-out cross-validation on the training data. The model-tree predictor method $M5'$ generates binary decision trees with linear regression functions at the leaf nodes [17]. This method is an improved re-implementation of Quinlan's M5 [12].

To compare the neural network accuracy on a test problem with that of another method, we computed the estimated standard error of the difference between the two averages. The t statistic for testing the null hypothesis that the two means are equal was then obtained. We conducted a two-tailed test with significance level $\alpha = 0.01$. If the null hypothesis is rejected and the network's testing error is smaller than that of an existing method, the neural network wins; otherwise it loses. Neural network wins are marked by bullets ($\bullet$), while losses are marked by diamonds ($\diamond$). Cases with no significant difference in the average accuracy (i.e., ties) are left unmarked.

Table 4 summarizes the wins and losses of the various methods. NN outperforms NB and kNN regardless of the performance measure. When measured using the relative root mean squared errors, NN is as accurate as or more accurate than NB and kNN for all the problems tested. NN is more accurate than LR on 13 data sets and is less accurate on 8 data sets. NN's performance is comparable to that $M5'$, winning and losing in about the same number of problems.

In terms of the relative mean absolute prediction errors, NN clearly outperforms the statistical methods in most of the problems tested. NN predictions are more accurate than those of NB, LR and kNN on 2 out of 3 problems tested. Only on 2 problems are the predictions of the neural networks significantly worse than those of NB and kNN. For all problems, the relative mean absolute errors of the neural networks are as good as or better than those of linear regression. Compared to $M5'$, neural networks are more accurate on 12 problems and less accurate only on 5 problems. For the remaining 15 problems, there is no significant difference in the performance of the two methods.

4 Conclusion and future work

A simple method for removing redundant hidden units and irrelevant input units from feedforward neural networks has been presented. We have shown the effectiveness of the proposed method on 32 publicly available data sets. With respect to the relative root mean squared errors, NN predicts as well as or better than Naive Bayes and k-nearest-neighbors on all the problems. Its performance is comparable to those of linear regression and $M5'$. Using the relative mean absolute error as the performance measure, NN outperforms all four regression methods.

Acknowledgments This work was done while the first author was spending his sabbatical leave at the Computational Intelligence Lab, University of Louisville, Kentucky. He is grateful to Professor J. M. Zurada for providing him with office space and computing facilities.

Table 2. Relative root mean square error $\pm$ the standard deviation from five regression methods.

No.	NN	NB	LR	kNN	$M5^{'}$
1	193.42 ± 58.02	263.70 ± 87.2	233.31 ± 93.3	267.91 ± 97.1	164.24 ± 70.8
2	42.45 ± 20.41	57.61 ± 10.7	53.37 ± 10.6	$77.95 \pm 10.9 \bullet$	32.28 ± 7.6
3	80.65 ± 10.01	82.90 ± 9.6	77.37 ± 9.7	75.52 ± 8.1	77.21 ± 9.9
4	60.39 ± 8.18	65.51 ± 6.3	53.21 ± 5.9	$70.94 \pm 7.0 \bullet$	$49.36 \pm 3.9 \diamond$
5	73.50 ± 9.70	$97.03 \pm 11.1 \bullet$	$98.86 \pm 9.2 \bullet$	$85.99 \pm 8.4 \bullet$	79.09 ± 8.5
6	105.00 ± 10.21	103.84 ± 6.8	$85.63 \pm 4.0 \diamond$	$118.54 \pm 8.3 \bullet$	$85.63 \pm 4.0 \diamond$
7	89.87 ± 7.62	95.01 ± 10.5	82.08 ± 7.6	90.88 ± 8.5	92.48 ± 25.8
8	55.80 ± 5.67	$66.62 \pm 6.4 \bullet$	$67.25 \pm 5.7 \bullet$	$71.77 \pm 4.2 \bullet$	$61.79 \pm 4.8 \bullet$
9	86.94 ± 3.53	$92.03 \pm 4.0 \bullet$	$80.11 \pm 2.1 \diamond$	90.02 ± 3.6	$81.35 \pm 2.3 \diamond$
10	42.76 ± 2.58	$56.55 \pm 2.9 \bullet$	$39.71 \pm 2.2 \diamond$	$75.46 \pm 2.8 \bullet$	$40.07 \pm 2.0 \diamond$
11	106.57 ± 2.82	$123.54 \pm 4.6 \bullet$	$100.16 \pm 0.5 \diamond$	$120.30 \pm 4.2 \bullet$	$100.38 \pm 0.6 \diamond$
12	74.92 ± 3.55	$81.63 \pm 2.2 \bullet$	$68.93 \pm 1.2 \diamond$	72.61 ± 2.2	$68.50 \pm 1.4 \diamond$
13	97.80 ± 4.83	93.66 ± 5.7	97.39 ± 7.0	$106.15 \pm 8.5 \bullet$	93.70 ± 6.0
14	15.46 ± 0.84	$32.14 \pm 2.2 \bullet$	$30.76 \pm 2.3 \bullet$	$34.82 \pm 4.2 \bullet$	$16.61 \pm 0.7 \bullet$
15	44.85 ± 3.92	43.63 ± 1.6	$49.28 \pm 3.4 \bullet$	46.05 ± 2.2	$37.95 \pm 2.5 \diamond$
16	32.27 ± 6.18	$75.07 \pm 1.8 \bullet$	$62.71 \pm 9.6 \bullet$	$57.21 \pm 7.0 \bullet$	$38.35 \pm 2.2 \bullet$
17	66.64 ± 2.81	64.32 ± 1.3	$62.77 \pm 2.0 \diamond$	$70.02 \pm 1.8 \bullet$	$62.27 \pm 1.1 \diamond$
18	67.48 ± 3.29	$87.38 \pm 2.9 \bullet$	$79.33 \pm 2.6 \bullet$	$80.23 \pm 1.6 \bullet$	$72.86 \pm 2.1 \bullet$
19	37.99 ± 1.31	$53.53 \pm 1.1 \bullet$	$51.08 \pm 1.4 \bullet$	$54.80 \pm 1.6 \bullet$	$33.19 \pm 0.8 \diamond$
20	26.79 ± 2.43	$39.11 \pm 2.0 \bullet$	$54.20 \pm 5.4 \bullet$	$44.81 \pm 1.4 \bullet$	$31.72 \pm 2.5 \bullet$
21	13.99 ± 2.66	$35.91 \pm 4.4 \bullet$	$52.17 \pm 8.4 \bullet$	$38.57 \pm 5.9 \bullet$	$18.99 \pm 2.6 \bullet$
22	12.66 ± 1.17	$26.73 \pm 0.6 \bullet$	12.50 ± 0.7	$36.72 \pm 0.9 \bullet$	$10.49 \pm 0.9 \diamond$
23	99.61 ± 1.59	$103.04 \pm 1.5 \bullet$	$97.23 \pm 1.2 \diamond$	$105.98 \pm 1.2 \bullet$	$97.03 \pm 1.1 \diamond$
24	73.97 ± 2.06	73.04 ± 2.3	74.16 ± 0.7	72.67 ± 1.7	$76.95 \pm 2.1 \bullet$
25	100.78 ± 1.07	$103.90 \pm 1.5 \bullet$	99.68 ± 1.7	$102.69 \pm 1.1 \bullet$	$103.54 \pm 1.9 \bullet$
26	70.36 ± 1.83	$76.00 \pm 2.0 \bullet$	70.54 ± 1.0	$73.49 \pm 1.4 \bullet$	$74.51 \pm 2.1 \bullet$
27	34.92 ± 0.54	$42.44 \pm 0.7 \bullet$	$38.43 \pm 0.9 \bullet$	$42.62 \pm 1.1 \bullet$	$36.37 \pm 0.6 \bullet$
28	80.99 ± 0.95	$87.33 \pm 1.1 \bullet$	80.48 ± 0.7	$87.73 \pm 1.2 \bullet$	$86.22 \pm 1.4 \bullet$
29	36.96 ± 1.10	$61.00 \pm 1.6 \bullet$	$52.78 \pm 1.1 \bullet$	$45.14 \pm 1.5 \bullet$	$39.20 \pm 1.9 \bullet$
30	165.09 ± 34.30	$238.24 \pm 71.7 \bullet$	$281.85 \pm 51.1 \bullet$	$240.99 \pm 47.0 \bullet$	200.17 ± 75.3
31	88.86 ± 1.22	$93.11 \pm 0.9 \bullet$	$94.58 \pm 1.6 \bullet$	$90.07 \pm 0.7 \bullet$	$86.10 \pm 0.9 \diamond$
32	86.47 ± 1.38	$162.37 \pm 12.9 \bullet$	$84.55 \pm 1.7 \diamond$	86.59 ± 2.3	$84.47 \pm 1.2 \diamond$

Table 3. Relative mean absolute error ± the standard deviation from five regression methods.

No.	NN	NB	LR	kNN	$M5'$
1	180.44 ± 53.81	249.71 ± 88.6	347.09 ± 130.1 •	225.52 ± 83.4	223.48 ± 100.3
2	36.15 ± 17.42	53.63 ± 9.0 •	67.27 ± 16.7 •	67.80 ± 9.4 •	36.19 ± 10.0
3	78.52 ± 9.43	80.87 ± 8.7	89.15 ± 10.2 •	74.09 ± 7.4	86.47 ± 11.6
4	58.03 ± 7.54	63.57 ± 5.6	58.61 ± 6.6	66.00 ± 6.0 •	53.67 ± 3.4
5	69.58 ± 9.49	98.86 ± 14.0 •	110.73 ± 13.0 •	85.76 ± 13.6 •	85.40 ± 12.4 •
6	102.86 ± 11.84	101.90 ± 7.9	93.51 ± 8.3	112.10 ± 10.6	93.51 ± 8.3
7	92.82 ± 7.19	96.65 ± 12.3	92.83 ± 11.1	95.17 ± 14.1	104.61 ± 24.1
8	51.21 ± 4.86	61.15 ± 4.5 •	70.66 ± 5.8 •	65.08 ± 4.1 •	59.98 ± 4.2 •
9	86.82 ± 4.22	89.96 ± 3.3	83.27 ± 2.5	88.83 ± 2.5	84.50 ± 2.4
10	39.31 ± 2.42	49.60 ± 2.5 •	39.18 ± 2.4	68.10 ± 2.7 •	38.84 ± 2.2
11	104.97 ± 3.47	122.10 ± 4.4 •	105.39 ± 3.4	122.67 ± 4.1 •	105.90 ± 3.2
12	70.93 ± 3.96	75.04 ± 2.7 •	72.27 ± 2.1	68.97 ± 2.5	66.54 ± 2.3 ◇
13	89.34 ± 5.43	82.56 ± 3.7 ◇	99.02 ± 6.4 •	101.56 ± 6.5 •	93.13 ± 6.1
14	11.38 ± 0.60	23.28 ± 1.2 •	30.05 ± 2.6 •	21.31 ± 1.6 •	15.31 ± 0.6 •
15	39.54 ± 3.05	40.50 ± 2.4	46.04 ± 3.2 •	38.56 ± 2.2	34.30 ± 2.5 ◇
16	26.27 ± 3.40	55.77 ± 1.6 •	66.42 ± 8.2 •	53.40 ± 4.7 •	30.23 ± 1.6 •
17	66.15 ± 3.35	63.40 ± 1.4	65.31 ± 2.3	65.61 ± 1.7	63.88 ± 1.4
18	65.17 ± 2.73	80.05 ± 2.2 •	78.20 ± 2.6 •	74.76 ± 1.6 •	71.88 ± 2.2 •
19	37.72 ± 1.08	52.35 ± 0.9 •	52.38 ± 1.4 •	53.11 ± 1.4 •	34.03 ± 1.0 ◇
20	21.61 ± 1.64	29.37 ± 1.4 •	50.71 ± 6.2 •	29.53 ± 1.3 •	26.87 ± 1.6 •
21	10.25 ± 1.46	31.29 ± 3.0 •	56.63 ± 7.4 •	27.22 ± 2.5 •	17.34 ± 1.7 •
22	7.89 ± 0.46	21.92 ± 0.3 •	7.74 ± 0.1	34.95 ± 1.0 •	5.51 ± 0.1 ◇
23	100.17 ± 1.75	104.88 ± 1.2 •	99.61 ± 1.8	106.57 ± 1.6 •	100.13 ± 1.8
24	55.36 ± 3.09	39.81 ± 1.0 ◇	93.72 ± 3.2 •	49.22 ± 2.9 ◇	57.72 ± 3.2
25	99.92 ± 1.24	101.91 ± 0.9 •	101.83 ± 2.2 •	102.37 ± 1.4 •	105.57 ± 2.3 •
26	59.77 ± 1.47	59.37 ± 1.7	66.21 ± 1.1 •	62.91 ± 1.6 •	64.89 ± 1.2 •
27	30.89 ± 0.64	37.55 ± 0.6 •	35.64 ± 0.9 •	36.82 ± 0.8 •	32.07 ± 0.7 •
28	78.48 ± 0.92	81.55 ± 1.4 •	80.88 ± 1.1 •	86.22 ± 1.5 •	83.66 ± 1.3 •
29	34.94 ± 0.92	56.74 ± 0.7 •	51.76 ± 0.5 •	40.76 ± 1.0 •	36.43 ± 1.2 •
30	120.96 ± 19.12	134.28 ± 23.0	236.82 ± 35.2 •	163.12 ± 21.4 •	119.84 ± 20.4
31	89.64 ± 1.30	94.02 ± 1.0 •	96.25 ± 1.5 •	91.21 ± 0.8 •	88.24 ± 1.1 ◇
32	71.52 ± 1.49	93.63 ± 2.9 •	75.46 ± 1.6 •	68.42 ± 1.0 ◇	70.83 ± 1.5

Table 4. Summary of the results from neural networks compared to those from other methods.

	Relative RMSE			Relative MAE		
NN versus	Wins (•)	Ties	Losses (◇)	Wins (•)	Ties	Losses (◇)
Naive Bayes	22	10	0	20	10	2
LR	13	11	8	22	10	0
kNN-RMSE/MAE	24	8	0	22	8	2
$M5'$	12	7	13	12	15	5

References

1. Ash, T. (1989) Dynamic node creation in backpropagation networks. *Connection Science*, 1 (4), 365-375.
2. Belue, L.M. and Bauer, Jr. K.W. (1995) Determining input features for multilayer perceptrons. *Neurocomputing*, 7 (2) 111-121.
3. Cybenko, G. (1989) Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals, and Systems*, 2, 303-314.
4. Dennis Jr. J.E. and Schnabel, R.E. (1983) Numerical methods for unconstrained optimization and nonlinear equations. Englewood Cliffs, New Jersey: Prentice Halls.
5. Frank, E., Trigg, L., Holmes, G. and Witten, I.H. (1998) Native Bayes for regression. Working Paper 98/15, Dept. of Computer Science, University of Waikato, New Zealand.
6. Gelenbe, E., Mao, Z.-H., Li. Y.-D. (1999) Function approximation with spike random networks. *IEEE Trans. on Neural Networks*, 10 (1), 3-9.
7. Hornik, K. (1991) Approximation capabilities of multilayer feedforward networks. *Neural Networks*, 4, 251-257.
8. Kwok, T.Y. and Yeung, D.Y. (1997) Constructive algorithms for structure learning in feedforward neural *IEEE Trans. on Neural Networks*, 8 (3),630-645, May 1997.
9. Kwok, T.Y. and Yeung, D.Y. (1997) Objective functions for training new hidden units in constructive neural networks. *IEEE Trans. on Neural Networks*, 8 (5) 1131-1148.
10. Mak, B. and Blanning, R.W. (1998) An empirical measure of element contribution in neural networks. *IEEE Trans. on Systems, Man, and Cybernetics - Part C*, 28 (4) 561-564.
11. Mozer, M.C. and Smolensky, P. (1989) Using relevance to reduce network size automatically. *Connection Science*, 1 (1), 3-16.
12. Quinlan, R. (1992) Learning with continuous classes. In *Proc. of the Australian Joint Conference on Artificial Intelligence*, 343-348, Singapore.
13. Steppe, J.M. and Bauer, Jr. K.W. (1996) Improved feature screening in feedforward neural networks. *Neurocomputing*, 13 (1) 47-58.
14. Setiono, R. and Hui, L.C.K. (1995) Use of a quasi-Newton method in a feedforward neural network construction algorithm. *IEEE Trans. on Neural Networks*, 6 (1), 273-277.
15. Setiono, R. and Liu, H. (1997) Neural network feature selector. *IEEE Trans. on Neural Networks*, 8 (3), 654-662.
16. Zurada, J.M., Malinowski A. and Usui, S. (1997) Perturbation method for deleting redundant inputs of perceptron networks. *Neurocomputing*, 14 (2) 177-193.
17. Wang, Y. and Witten, I.H. (1997) Induction of model trees for predicting continuous classes. In *Proc. of the Poster Papers of the European Conference on Machine Learning.* Prague: University of Economics, Faculty of Informatics and Statistics.
18. Yoon, Y., Guimaraes, T. and Swales, G. (1994) Integrating artificial neural networks with rule-based expert systems. *Decision Support Systems*, 11, 497-507.