Adding Input Nodes to a Trained Neural Network

Carrie Gates

31 January 2000

Abstract

Neural networks suffer from the requirement of needing to be completely retrained in order to add a new input to the network. The most severe impediment to this retraining is that if an additional input is found to be required, often the accompanying data required for retraining is not available in sufficient quantity. This paper investigates this problem via experimental results. These results indicate that further attention to this problem may result in a satisfactory algorithm.

1 Introduction

Neural networks are deployed across a diverse set of problems, ranging from time-series problems such as predicting changes in the stock exchange[1], to controlling a robot[2], to predicting the presence of medical diseases[3]. Additionally, neural networks can take on many different forms, such as feed-forward, recurrent and Kohonen networks. Feed-forward networks are generally concerned making a prediction, such as what letter a written character might be, based on a series of inputs. Recurrent networks are use their outputs as more inputs, thus feeding back upon itself. Kohonen networks, by contrast, are self-organizing networks, where a network learns on its own instead of being trained.[4] These networks are typically used to classify data into subsets where relational patterns are not already known.

The most common form of problem requires predicting an output based on a set of inputs. For example, given the concentrations of various ions in a volume of water, predict its conductivity. This lends itself naturally to a feed forward network. In this form, a network is trained with a series of inputs whose outputs are known. From this, the network can “predict” the output for inputs which it has not seen before. This is especially useful in situations where an equation relating the inputs and output is not known. A neural network is therefore a black box which provides a relationship between inputs and output.

2 Analysis of the Problem

While neural networks have been adopted for many applications, there are still many disadvantages to their use. The results from a neural network are only as good as their training. To train a neural network requires much time and experimentation to determine the best network configuration, such as the optimum number of hidden nodes, which inputs to present and how, which learning algorithm to use, the optimum learning rate, etc. Beyond basic configuration, the training period itself can often last days. Additionally, the number of training examples required needs to be determined — too few and the network can not generalize sufficiently well to be useful, too many and the network might over-generalize, hence producing sub-optimal results.

As neural network programs tend to deal with problems for which there are no known equations, one can not even determine if the appropriate inputs are being used. For example, one might use ten particular ion concentrations to predict the conductivity of water. The results for these predictions might be satisfactory, such as a correct prediction in 75% of all cases, but not ideal. It may later be discovered that an eleventh ion has a strong effect, even at small concentrations, on the conductivity. It would then be desirable to add this input to the neural network, potentially then increasing its predictive accuracy above 75%.

Adding an input, however, is not without its own problems. Least of these problems is the time required for the network to be retrained with the new input. Of even greater concern is that, in order to learn, a neural network requires some minimum number of training sets upon which to learn. If a new input is suddenly required, it is likely that there is little or no data with this additional input. As a result, there is little to no data upon which to train the network.

This paper investigates the possibility of adding a new input node to an already trained neural network with little or no retraining. The initial approach involved simulating the problem with a known data set where a significant input was not added to the first network. It was then added to the network with a minimal number of training examples and the results were examined. The theoretical basis for the results is explored and directions for future research is given.

3 The Data Set

The data used for training a neural network is the most important aspect of neural networks. As such, it must be clean and reliable, and available in sufficient quantity with sufficient variations for training the network.[5] These qualities are especially applicable to medical data, which is often sparse and noisy.[6] The data set used in this paper is no exception.

The data set used for this problem is a medical data set provided by Dr. Peltekian at the Victoria General Hospital. It contains 218 patients, 109 of whom had alcoholic hepatitis as determined by a liver biopsy. Of this data set, 25.7% of patients were originally misdiagnosed as having a disease other than alcoholic hepatitis, with alcoholic hepatitis not even being suspected. Thus having a network which predicts the presence of this disease is important for both the patient and the hospital.

For each patient there were 25 laboratory and clinical tests available. However, not all test results were available for all patients. Only 63 patients had complete data sets, while 67 patients were missing seven or more test results.

¿From previous work[7],[8],[9], it was agreed using three different methods that the presence of hepatomegaly was significant when predicting the presence of alcoholic hepatitis. Also generally found as important was mean corpuscular volume and the ratio of aspartate aminotransferase (AST) to alanine aminotransferase (ALT). Chosen as insignificant inputs were age, gender, aspartate aminotransferase levels, bilirubin, heamoglobin and white blood cell count.

Using the seven tests identified, the data set was cleansed so that only those patients who had results for all seven tests were retained. This resulted in a data set containing 185 patients, 99 of whom had alcoholic hepatitis and 86 of whom did not.

4 Initial Network

A neural network was constructed using only the six inputs identified as contributing little to the prediction of hte presence of alcoholic hepatitis. These inputs were normalized to a value between 0 and 1. Forty-one nodes were used in the hidden layer. The learning algorithm utilized was the standard quadratic error method with a learning rate $\eta = 0.1$.

The data set was divided into two sets — A and B. A contained 150 patients, 89 of whom had alcoholic hepatitis, while set B contained 29 patients, 10 of whom had alcoholic hepatitis. Set A was used to train the neural network using leave-one-out cross-validation.[10] Set B was then used both as a validation set and to train the additional input node as described later in this paper.

Five different neural networks were trained to provide a baseline against which comparisons could be made. The first network contained all of the possible inputs — both the set of six insignificant inputs plus all three significant inputs. The second network contained only the six insignificant inputs. The remaining three networks each contained one of the three significant inputs. As leave-one-out cross-validation was used, the results for the prediction set were calculated based on a summary over all 150 trials. Three calculations are given: accuracy, sensitivity and specificity. Accuracy was the number of correct predictions over the number of total predictions. Sensitivity was the number of patients for whom a correct prediction of the presence of alcoholic hepatitis was given over the total number of patients with this disease, or the percentage of correct positives. Similarly, specificity was the number of patients for whom alcoholic hepatitis was correctly predicted as not being present over the number of patients whom did not have this disease, or the percentage of correct negatives. Similar results are shown for the prediction set, set B. As the calibration phase resulted in 150 different networks, the validation set was calculated as an average over all 150 networks.

The results are summarized in Tables 1 and 2. The expected results were that the network with all the inputs would perform best, while the network with only the insignificant inputs

	Accuracy	Sensitivity	Specificity
All inputs	83.333	89.899	70.588
Insignificant inputs only	74.000	80.808	60.784
Insignificant + mcv	82.000	83.838	78.431
Insignificant + hepatomegaly	79.333	84.849	68.628
Insignificant + ast/alt	71.333	76.768	60.784

Table 1: Baseline prediction set errors for networks with different inputs

	Accuracy	Sensitivity	Specificity
All inputs	99.241	99.867	98.912
Insignificant inputs only	84.621	96.000	78.632
Insignificant + mcv	95.770	99.533	93.790
Insignificant + hepatomegaly	99.862	99.867	99.859
Insignificant + ast/alt	88.943	96.667	84.877

Table 2: Baseline validation set errors for networks with different inputs

would perform the worst. The remaining three networks would fall someplace in between these two extremes. As can be seen from the Tables, the results are similar to what was expected, though not exact. The results from Table 1 indicate that effect of adding the ast/alt ratio to the insignificant inputs does not improve accuracy, but merely decreases sensitivity and increases specificity. These results are supported by Table 2. However, adding either hepatomegaly or mean corpuscular volume (mcv) to the inputs increased the accuracy to one similar to that of the network using all inputs. However, Table 2 shows that mcv is not as good a predicter as hepatomegaly, which even slightly outperforms all inputs. Hepatomegaly was therefore chosen as the test input to be added to a trained network.

5 Adding an Input Node

In an effort to improve the predictive accuracy of the neural network with only insignificant inputs as described in the previous section, an input node was added for hepatomegaly. The first difficulty came when determining what initial weights to assign to the connections from the new input node to each hidden node. The options available included:

1. randomly assigned
2. largest weights already present in network
3. smallest weights already present in network

		error (set A)	error (set B)
random	accuracy	86.480	84.529
	sensitivity	91.670	96.267
	specificity	76.405	78.351
largest	accuracy	75.956	80.000
	sensitivity	81.899	87.133
	specificity	64.418	76.246
average	accuracy	72.702	78.023
	sensitivity	76.047	79.267
	specificity	66.209	77.368
smallest	accuracy	56.964	69.195
	sensitivity	47.051	51.667
	specificity	76.209	78.421

Table 3: Errors for each weighting method when adding an additional input node

4. average of weights already present in network

5. some user-decided percentage of the largest weights already present in the network

The results for adding an input node with no training using options one through four is given in Table 3. These results were arrived at by using the 150 networks already generated using only insignificant inputs. For the randomly-assigned additional node weights, 150 trials were performed — one per network created during the calibration phase. Each patient in set A was then tested against each of the 150 networks, followed by the same procedure for set B. The results from the multiple networks were then averaged to provide the values given in Table 3. The values used for the "largest" weights were arrived at by determining which input had the largest overall weights to the hidden layer. The weight to each hidden node was then copied to the equivalent connection for the new input node. Thus, if the input with the largest overall weights had a very small weight to a particular hidden node, the new input node would have that same weight to that same hidden node. The assumption here is that the new input is as important as the input with the largest weights and should thus have the same impact on the network. Using the smallest weight values was calculated in a similar fashion. Finally, the average weight values were calculated by averaging all weights going to a particular hidden node and then using that value for the weight from the new input node to that hidden node.

Five patients were then chosen randomly from set B to use for training the new neural network. Only five patients were used based on the premise that in a real-world application very little training data would be available. Two of the patients chosen had alcoholic hepatitis, while the remaining three did not.

Three trials were then performed using three different learning algorithms. The first algorithm changed the weights on the new connections only, and made this change after only

		Error (Percent)		
		set A	5 new inputs only	rest of set B
	accuracy	48.658	91.467	26.639
new connections	sensitivity	59.024	100.000	60.083
only	specificity	28.536	85.778	9.917
	accuracy	45.311	96.267	25.194
whole network	sensitivity	52.788	91.000	53.750
1 epoch only	specificity	30.797	99.778	10.917
	accuracy	47.138	100.000	25.806
whole network	sensitivity	56.094	100.000	56.750
complete training	specificity	29.752	100.000	10.333

Table 4: Error rates after training with new input

one training epoch regardless of error. The second method adjusted the weights on the whole network, again based on one training epoch per patient only. The third method performed a complete training on all five patients until the error was less than an acceptable epsilon. The results for each method are given in Table 4. The original weights chosen for the new input before training were random in all cases. Again, all 150 networks generated above were used for each test, retrained for the new input.

6 Discussion of Results

The results from Table 3 are interesting. This Table shows that the best results, before any training is performed, occurs when the weights for the new input are set randomly. After this, the best results are obtained from setting the weights the same as the input with the largest overall weight vector, with the results becoming poorer as the weights for the new input become smaller. This later is not particularly surprising, as the premise for adding a new input is that this input is significant to solving the problem. What is surprising, however, is how well choosing random weights performed. One possible explanation for this is that the majority of weights in this particular network are small, and that the randomly-chosen weights were relatively large, thus giving more importance to the new input over the other inputs.

However, as can be seen from Table 4, the results from this approach after training are not promising. With all algorithms, the accuracy dropped to less than 50% on the original data set, and to approximately 25% for the validation set (after removing the five samples used for retraining). It is believed that the training of the network with the five new patients with the new input optimized for those five patients only, driving the network away from the global minimum and into a very localized minimum. However, given the promising results when just a random number was used, further investigation into an algorithm which changes only those randomly-chosen weights needs to be performed.

At this point some analysis must be performed from a more theoretical perspective. Is there any reason to believe that it is possible to add an input node with little or no training?

Information theory represents a neural network as an n-dimensional error surface where the goal is to find the global minimum in the space, which represents the lowest error. The number of dimensions is determined by the number of weighted connections in the network. For example, in a network with n inputs, m hidden nodes and k outputs, the dimensionality of the error surface is $n \times m + m \times k$. By adding an input node the dimensionality increases by m to $(n+1) \times m + m \times k$.

Assumption: Given that a trained network is already at or near a global minimum, it is assumed that it will be near the global minimum when the dimensionality is increased and that the network will only need to be trained in these new dimensions.

At first glance this assumption appears false. This would certainly be the case were the inputs random. However, the inputs can be considered to be related to one another in the case of neural networks. From this one can intuit that a trained neural network would still be near a global minimum when a new input node is added — that the knowledge learned to date would not suddenly become irrelevent.

There is some mathematical basis for this assumption, which can be found from sensitivity analysis in linear programming.[11] This theory essentially states that one can make assumptions regarding how near one is to a minimum by placing increasing contrainsts on your dimensions. For example, one can assume that the dimensions added by increasing the number of input nodes were already present in the problem, however they were unconstrained. Therefore, by placing weights on these dimensions, you are further constraining the problem and can then make assumptions regarding your position relative to the global minimum. This approach, however, requires further investigation regarding its applicability to neural networks and information theory.

7 Conclusions and Future Directions

Adding an additional input node without retraining the entire network is an interesting area to investigate. While the initial results were not promising, as shown in Table 4, the results from before training are extremely promising (see Table 3). Reconsidering the training methods used, and perhaps modifying the traditional training model to account for updating for the additional input only, warrants further investigation.

Concurrent with this, the mathematical basis for adding an input needs to be addressed. The assumption made in this paper and discussed in the results section needs to be either proven or disproven, or at least have a stronger mathematical basis, as opposed to intuition. This can be further investigated from the viewpoint of Linear Programming, and also from examining neural networks as a series of matrices with non-linear transformations.

With regards to this particular data set, the tests should be rerun with a calibration set of only 100 patients, leaving a validation set of 79 patients. Using a validation set of only 29 patients does not allow enough data for a proper analysis of the problem.

Finally, once a method has been found for adding an additional input, this should be tested with other well-known and well-understood data sets to determine if the theories and algorithms developed from the one set will generalize across data sets.

References

[1] Vijay S. Desai and Rakesh Bharati, (1998), "Concepts, Theories and Techniques — the Efficacy of Neural Networks in Predicting Returns on Stock and Bond Indices", *Decision Sciences — Journal for the Decision Sciences Institute*, **29**, no. 2, 405-427.

[2] Dean A. Pomerleau, (1994), "Neural Network-Based Vision for Precise Control of a Walking Robot", *Machine Learning*, **15**, no. 2, 125-137.

[3] Erwin Tafeit and Gilbert Reibnegger, (1999), "Reviews — Artificial Neural Networks in Laboratory Medicine and Medical Outcome Prediction", *Clinical Chemistry and Laboratory Medicine*, **37**, no. 9, 845-855.

[4] John Hertz, Anders Krogh & Richard G. Palmer, (1991), *Introduction to the Theory of Neural Computation*, Addison-Wesley Publishing Company.

[5] Joseph P. Bigus, (1996), *Data Mining with Neural Networks: Solving Business Problems - from Application Development to Decision*, McGraw-Hill, pages 43-59.

[6] Georgia D. Tourassi & Carey E. Floyd, (1997), "The Effect of Data Sampling on the Performance Evaluation of Artificial Neural Networks in Medical Diagnosis", *Medical Decision Making*, **17**, 186-192.

[7] Carrie Gates, (1998), "Diagnosis of Alcoholic Hepatitis Using Neural Networks", unpublished.

[8] Carrie Gates, (1999), "EComm 6014 — Assignment 2", unpublished.

[9] Kevork M. Peltekian, Dickrann A. Malatjalian & Joseph J. Sidorov, (1991), "New Clinical Prediction Rule for Alcoholic Hepatitis", Unpublished.

[10] A. C. Davison & D. V. Hinkley, (1997), *Bootstrap Methods and Their Application*, 293.

[11] Vašek Chvátal, (1983), *Linear Programming*, pages 148-168.