ESANN'2001 proceedings - European Symposium on Artificial Neural Networks
Bruges (Belgium), 25-27 April 2001, D-Facto public., ISBN 2-930307-01-3, pp. 365-370

One-to-Many Mappings represented on Feed-forward Networks

Roelof K. Brouwer[1] and Witold Pedrycz[2]

[1]University College of the Cariboo [2]University of Alberta

Abstract

Multiplayer perceptrons or feed-forward networks are generally trained to represent functions or many-to-one (m-o) mappings. This creates a problem if the training data exhibits the property of many-to-many or almost many-many valued-ness because the model, which generated the data, was many-to-many. Therefore in this paper a modified feed-forward network and training algorithm is considered to represent a multi-valued mappings. The solution consists of adding another input to the standard feed-forward network and of modifying the training algorithm. This additional input will generally have no training values provided and an amended training algorithm is used to find its values.
The modified feed-forward network and training method has been successfully applied both in representing the mapping implied by data generated by multi-valued functions and in representing the mapping implied by data obtained from benchmark databases.

Keywords feed forward neural networks, relations, one-to-many mappings, data segmentation, clustering, many-valued functions

1 Introduction: relations versus functions in neurocomputing

Multiplayer perceptrons or feed-forward networks have been trained to represent functions or many-to-one (m-o) mappings (Hecht-Nielsen ,1989; Lippman, 1987; Rumelhart, 1986; Smith, 1993; Werbos 1974). A function is a special kind of relation and since many-valued-ness corresponds to relations it is worthwhile to consider relations in general. A classic example of many-valued-ness in AI research is the Necker cube in which case two different object orientations are obtained from the same 2-dimensional image. Also the data stored in databases generally represents relations or associations between variables that are many to many (m-m).
Given some of the data for a mapping between two data sets it may not be known whether the model which generated the data is m-o or m-m even though technically the training data provided is m-o since two unique outputs associated with two very similar inputs may actually be associated with the same input. (i.e. the two inputs may actually be two approximations of the same input). The model may be m-m and the training data produced by the model m-o and almost m-m. This aspect will not be known in advance.
In (Uno, 1995; Hiraoka, 1998) it was demonstrated that the bottleneck network with feedback can learn many-valued functions. The authors studied the problem of

recalling the hand configurations required to grasp an object based upon its image. In (Hiraoka, 1998,1999) the relaxation method for recall described in (Uno 1995) is replaced by successive iteration. The authors of another paper (Tomikawa 1998) discuss a very similar method for using a recurrent neural network for approximating a certain version of o-m mappings.
This paper is organized in the following manner. First comes an illustration of multi-valuedness and the things that give rise to many-valuedness. Next is an overview of the structure and learning involved in a particular approach and also a detailed description of the learning algorithm. Experiments describing the results of applying this approach are then described. The paper ends with a description of some of the problems with the method proposed here.

2 An Illustration of Multivaluedness and its Sources

It is possible, given certain one-dimensional training data, that the many or almost many-valued ness is due to the fact that the given data is produced by a model with say two or more independent variables while the data available for training is for only some of the independent variables and for the dependent variable. The training data may be o-m or almost o-m even though the source of the data with additional input was m-o. Figure 1 shows a model of many-valuedness which was due to the fact that the output is produced by three different functions.

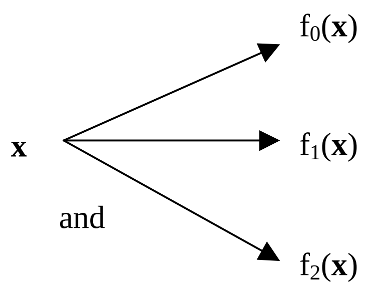

FIGURE 1 Demonstration of m-m property due to having several functions

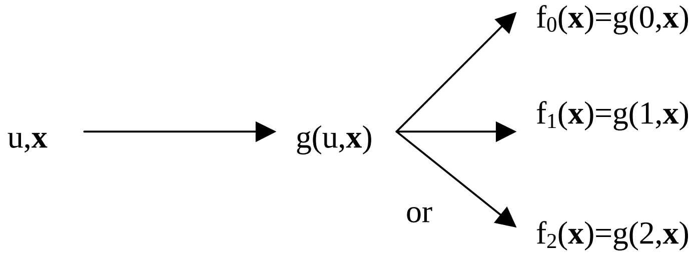

FIGURE 2 Changing m-m mapping to m-o mapping

In the demonstration in Figure 2 we turn the m-m function into a m-o function by considering the subscript function identifier that plays the role of function identifier to be another independent variable with values from {0,1,2}. The 3 mappings $\Re^n \xrightarrow{f_0} \Re$, $\Re^n \xrightarrow{f_1} \Re$, and $\Re^n \xrightarrow{f_2} \Re$ giving rise to a m-3 mapping become the single mapping $\{0,1,2\}\times\Re^n \xrightarrow{g_0} \Re$.
A specific example of the preceding is the case where one or more of the independent variables used for prediction is a nominal variable. Consider for example the data for predicting the age of abalone. (source: abalone data set in the machine learning data base found at http://www.ics.uci.edu/~mlearn/ MLRepository.html). The data generally is used to develop a function for predicting the age of abalone from physical measurements. The number of attributes is 8 with all but one being continuously valued. The remaining variable is nominal with values “male”, “female”, and “Infantile”. We can think of three prediction functions in this case, one for each of the values of the nominal variable. The nominal values can be thought of as function identifiers with a different function for each type of abalone. Without the nominal variable we have a model which is m-3 and training data which is almost m-3. There are three ages in the model for each combination of values of the remaining continuously valued attributes.

3 Structure and Learning for a modified feed-forward neural network

Since a feed-forward network requires m-o data we have to define a way of dealing with the many-valued relational property of the data. The approach used here is to turn the m-m mapping into a m-o mapping by augmenting the inputs by one or more additional components. The additional component(s) are used to contain values to distinguish between the many values mapped to by a particular input combination.
The network used for training consists of an input layer, an output layer and one hidden layer. The method used for training is the gradient descent method. The gradient of the output with respect to all unknowns is found by feeding the gradients forward as described in (Brouwer, 1997). This method is used rather than back-

propagation (Werbos, 1974; Rumelhart, 1986) because it is very readily expressed in high level notation without requiring subscripts and is readily appended for use in determining the values for the unknown vector variable, **u** by gradient descent. The algorithm, which is easily generalized to many hidden layers is summarized below. “+/” is the summation operator which sums values over the first dimension. $\mathbf{out}^{(0)}$ is the output of the input layer of the network including **u** and $\mathbf{out}^{(2)}$ is the actual output of the network. ∇ stands for gradient. Following is the training algorithm.

Algorithm for finding values for the connection matrices and the values for the unknown variable using gradient descent

```
1. initialize
$\mathbf{W}^{(k)}$      k = 1, 2 and $\mathbf{u}$
2. determine gradients
$\nabla_{\mathbf{W}}^{(1)} C$ =.+/($\mathbf{out}^{(2)}$ - $\mathbf{d}$) * f ` ($\mathbf{W}^{(2)'}$ . ($\mathbf{out}^{(1)}$,-1))*$\mathbf{W}^{(2)}$ * “1_ f ` ($\mathbf{W}^{(1)'}$ . ($\mathbf{out}^{(0)}$,-1)) * / ($\mathbf{out}^{(0)}$,-1)
$\nabla_{\mathbf{W}}^{(2)} C$=.($\mathbf{out}^{(2)}$ - $\mathbf{d}$) * f ` ($\mathbf{W}^{(2)'}$ . ($\mathbf{out}^{(1)}$,-1)) * / ($\mathbf{out}^{(1)}$,-1)
$\nabla_{\mathbf{u}} C$ =. +/($\mathbf{out}^{(2)}$ - $\mathbf{d}$) * f ` ($\mathbf{W}^{(2)'}$ . ($\mathbf{out}^{(1)}$,-1)) * $\mathbf{W}^{(2)}$. f ` ($\mathbf{W}^{(1)'}$ . ($\mathbf{out}^{(0)}$,-1)) * k {. “ 1 $\mathbf{W}^{(1)}$
3. modify
$\Delta \mathbf{W}^{(k)}$ =. - $\mu_1 \nabla_{\mathbf{W}}^{(k)} C$      k = 1, 2
$\Delta \mathbf{u}$ =.   - $\mu_2 \nabla_{\mathbf{u}} C$
```

$\mathbf{W}^{(i)'}$ is $\mathbf{W}^{(i)}$ i=1,2 with an additional column to include the bias terms. $\mathbf{out}^{(i)}$, i=1,2 are appended with –1 to multiply the bias terms. $\mathbf{W}^{(2)}$ * “1_ f ` ($\mathbf{W}^{(1)'}$ . $\mathbf{out}^{(0)}$,-1) means that each row of $\mathbf{W}^{(2)}$ is multiplied by the vector f ` ($\mathbf{W}^{(1)'}$. $\mathbf{out}^{(0)}$,-1) . k {. “ 1 $\mathbf{W}^{(1)}$ is made up of the first k columns of $\mathbf{W}^{(1)}$.
The gradient of **u** is used to determine **u** if it is desirable to consider all possible real values. If we think that there is only one additional variable with a small number of possible values, as in the case following, we may select the value out of a set of discrete values that gives the smallest absolute value of error for the training element given the connection matrices learned so far.

4 Experimental studies

In this section, we show the results of carrying out a series of numeric simulations to show how the network can learn and also show the impact on learning performance of having the separation variable. The performance measure, Q, is the root mean square or square root of the average of the squares of the individual errors.
The source of the data is the abalone data set mentioned previously. For training, 5 units in the hidden layer were used. The training performance in terms of rmse versus training time in epochs is as shown in Figure 3. The nominal feature is purposely left out of the training data to make it almost m-3. Three values from {0,1,2} were used for the unknown input.

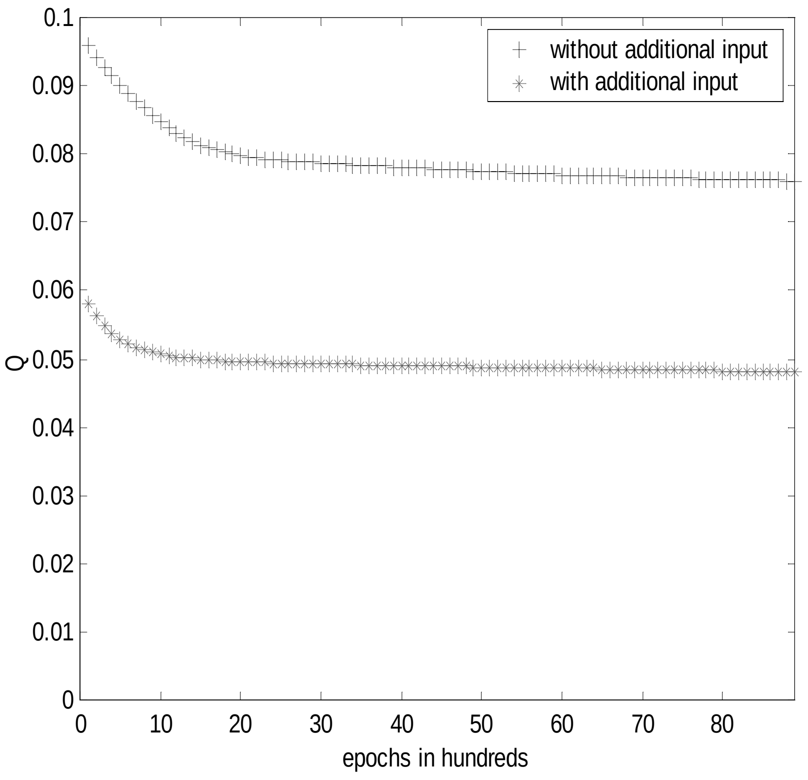

FIGURE 3 Two learning curves with and without additional input

Figure 3 above clearly shows a marked improvement in training due to adding an input variable using 3 values. Further learning was tried on artificially generated data including bi-variate normal distributed data, data generated from a two variable exponential function, and data generated from a 3 variable length function. In these cases the function was made almost m-m by dropping a variable. In case of the length function gradient descent was used to obtain values for the unknown input variable. The results obtained again were excellent. Lack of space prevents their coverage here.

5 Concluding remarks

Data presented to a feed-forward network for learning may not be learnable without some attention given to the number of inputs for the feed-forward network. The data as presented could be m-m, which is not appropriate for a feed-forward network that is intended to represent an m-o mapping. It is quite possible that the data was generated by a model consisting of two input variables with the data presented for training consisting only of the values for one of the dependent variables. This paper has shown that the feed-forward network and its training algorithm can be modified to

include another input variable with values restricted to {0,1} so that the network can distinguish between two different output values corresponding to one value for the input provided. Thus the many-valued-ness is reduced to single-valued-ness.
We may now have a network to represent an m-m mapping but it only represents a mapping and not truly a relation because the mapping represented is uni-directional. We should be able to go into both directions during retrieval. This suggests development possibly of another network.

6 References

R. K. Brouwer "Training a feed-forward network by feeding gradients forward rather than by back-propagation of errors." Neurocomputing 16, 1997, 117-126
R. Hecht-Nielsen *Neurocomputing*. New York: Addison-Wesley, 1989.
Kazuyuki Hiraoka and Yoshizawa Shuji. "Recalling of Many-Valued Functions by Successive Iteration on Bottleneck Networks." ICONIP'98 5th International Conference on Neural Information Processing
K. Hiraoka , T. Shigehara, H. Mizoguchi, T. Mishima, and S. Yoshizawa "On the overfitting of the five-layered bottleneck network." Proceedings. ICONIP '99. 6th International Conference on Neural Information Processing, Volume: 1,1999 Page(s): 234 -239 vol.1
K. E. Iverson J Introduction and Dictionary Iverson Software Inc., 33 Major Street, Toronto, Ontario, Canada M5S 2K9. 1991. (see also http://www.jsoftware.com).
R. P. Lippmann, () "An Introduction to computing with Neural Nets", *IEEE ASSP Magazine* **4**, 1987, pp4-22.
D. Pountain "The Joy of J", Byte September 1995.
D. E. Rumelhart, and J. L. McClelland "Learning internal representations by error propagation." In: *Parallel Distributed Processing*, MIT Press, Boston, 1986, pp 318-364.
M.Smith *Neural Networks for Statistical Modeling*. Van Nostrand Reinhold, New York 1993.
Y. Tomikawa and Kenji Nakayama. "Approximating Many Valued Mappings Using a Recurrent Neural Network" Proceedings of 1998. IEEE World Congress on Computational Intelligence.
Y. Uno, N. Fukumura, R. Suzuki and M. Kawato "A computational model for recognizing objects and planning hand shapes in grasping movements" Neural Networks, Vol. 8, No. 6, 1995,pp. 839-852.
P. J. Werbos "Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences." *Doctoral Dissertation*, Appl. Math., 1974 Harvard University, Mass.