ESANN'1999 proceedings - European Symposium on Artificial Neural Networks
Bruges (Belgium), 21-23 April 1999, D-Facto public., ISBN 2-600049-9-X, pp. 307-312

An Integer Recurrent Artificial Neural Network for Classifying Feature Vectors

Roelof K Brouwer PEng, PhD

University College of the Cariboo, Canada

Abstract: The main contribution of this report is the development of an integer recurrent artificial neural network (IRANN) for classification of feature vectors. The network consists both of threshold units or perceptrons and of counters, which are non-threshold units with bi-nary input and integer output. Input and output of the network consists of vectors of natural numbers. For classification representatives of sets are stored by calculating a connection matrix such that all the elements in a training set are attracted to members of the same training set. The class of its attractor then classifies an arbitrary element if the attractor is a member of one of the original training sets. The network is successfully applied to the classification of sugar diabetes data and credit application data.

Introduction

One type of neural network commonly used for classification is a multilayer perceptron (MLP), a feed forward net with one or more layers of nodes between the input and output nodes and with back-propagation for training (Werbos 1974). It has problems of long training time, getting stuck in local minimum and poor generalization.

Another class of neural networks useful for classification is the class of Recurrent Neural Networks an example of which is the Hopfield network (Hopfield, 1986). The main advantages of the Hopfield network and the recurrent network proposed here versus the back-propagation trained multilayer perceptron, MLP (Werbos 1974), is the simplicity of the learning procedure compared to back-propagation and the relative ease with which the recurrent network can be implemented in hardware.

The proposed network is a 3 layer recurrent neural network with two hidden layers whose state vector components are natural numbers rather than binary or bi-polar. The feature vectors are not converted to a vector of binary values before input as in the case of a Hopfield network.

This paper starts with a brief description of what is meant by the standard unary form of representing natural numbers and next describes the architecture and dynamics of the proposed network. It then describes the two ways of using a recurrent network for classification and how the network proposed here may be used for classification.

ESANN'1999 proceedings - European Symposium on Artificial Neural Networks
Bruges (Belgium), 21-23 April 1999, D-Facto public., ISBN 2-600049-9-X, pp. 307-312

Finally the results of using the network for classification are compared to results obtained by using other methods.

Unary Standard Form

For later description of the architecture and dynamics of the network it is useful to define the standard unary representation of natural numbers. The unary representation of a natural number is a string of 1's equal in length to the natural number. If the length of the string is to be fixed then the filler will consist of 0's. For example the string 1 0 1 0 1 1 0 1 represents the value 5. This manner of representing a natural number is called unary code. The unary standard form is the case where the 1's precede the 0's. For example 11100 with a string length of 5 represents the value 3. An array of integers can also be represented by a string of 0's and 1's. For example if the field widths or string lengths for representation of individual integers are 4 and 3 then the standard unary form for the vector 2, 1 would be 1 1 0 0 1 0 0.
Let i(b) be the function returning the number of 1's in a binary sequence. Let u(w,x), return a binary sequence of length w, and with *x* 1's where all the 1's precede the 0's. Let **iv**(**p**,**b**) return an integer vector where **p** is a vector of integers, which partitions the binary sequence so that each part is converted into an integer using the function i(b). Thus **iv**((4,3), (1010110)) is equal to (2,2). Let **uv**(**p**,x) return the catenation of binary strings $u(p_i,x_i)$ i = 0,1,… from a vector of natural numbers. For example **uv**((3,4,2),(2,2,1)) would be 1 1 0 1 1 0 0 1 0.

Description of Architecture

In this network, shown in Figure 1, there are 3 distinct layers consisting of two fully connected hidden layers and one input-output layer (the first layer of counters (i.e. they simply sum their input)) is repeated on the right in the diagram. The state of the network is presented by a vector of natural numbers, which are the activation values of the units in the input/output layer. The i^{th} component of this vector for i = 0,1,..k-1 is equal to the number of neurons in the i^{th} section of the 2^{nd} hidden layer with activation values greater than their thresholds. The function of the 1^{st} hidden layer is to convert natural numbers into their standard unary representation.

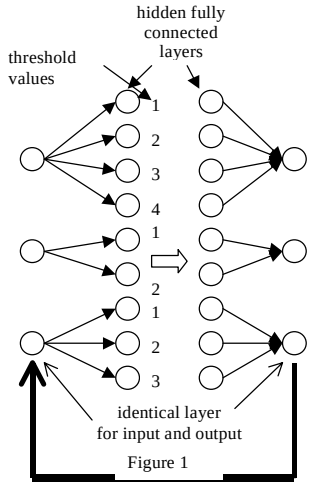

Figure 1

During iterations or state changes, the non-threshold units in the input/output layer simply sum their binary input and carry out the operation i(p,a). Each unit in the 1^{st} hidden layer has a fixed non-adaptive threshold and exactly one input. The threshold values for these units are 1,2, .. corresponding to the position in the

ESANN'1999 proceedings - European Symposium on Artificial Neural Networks
Bruges (Belgium), 21-23 April 1999, D-Facto public., ISBN 2-600049-9-X, pp. 307-312

section to which the unit belongs. Only the connection matrix between the hidden layers and the threshold values for the 2nd hidden layer are adaptive.

Network Dynamics

Transitions are assumed to be synchronous and therefore matrices can pre-multiply state vectors to represent weighting of inputs to units. The dynamics are such that a state transition is produced by $\mathbf{T}(\mathbf{x}) \leftarrow \mathbf{iv}(\mathbf{p},(\ (\mathbf{W},\mathbf{t}).(\ \mathbf{uv}(\mathbf{p},\mathbf{x},), -1)) \geq 0)$. The function $\mathbf{uv}(\mathbf{p},\mathbf{x})$ converts the vector $\mathbf{x}$ into a binary vector using partition vector $\mathbf{p}$. After catenation with a -1 the result is premultiplied by the appended connection matrix $\mathbf{W},\mathbf{t}$ where $\mathbf{t}$ is a vector of thresholds. This result is thresholded using 0 to produce another binary vector (the function $\mathbf{y} \leftarrow \mathbf{x} \geq 0$ is such that $\mathbf{y}_I = 0$ if $\mathbf{x}_I < 0$ and 1 otherwise). This new result is converted back to a vector of natural numbers using the function $\mathbf{iv}(\mathbf{x})$. The purpose of the conversions is that internal states with the same number of 1's in each partition be equivalent.

Storing Fixed Points

If there is a sequence of states $\mathbf{y}^{(i)}$ i = 0,.. k such that

$$\mathbf{y}^{(i)} = T(\mathbf{y}^{(i-1)}) \quad i = 1, 2, .. k$$

and

$$\mathbf{y}^{(k)} = T(\mathbf{y}^{(k)})$$

then $\mathbf{y}^{(k)}$ is an attractor of $\mathbf{y}^{(0)}$ and $\mathbf{y}^{(k)}$ is called a fixed point (Kamp, 1990). $\mathbf{y}^{(0)}$ will be called an attractee. Note that an attractor is an attractee which is attracted to itself and that all the states $\mathbf{y}^{(i)}$ i=0,1..k are attractees for $\mathbf{y}^{(k)}$.

An auto-associative memory system in which fixed points are stored may be used for classification by computing a connection matrix such that prototypes, representing the classes involved in classification, become fixed points of the corresponding network. The class of an element to be classified is taken to be the class of the prototype state to which it is attracted. The training algorithm for the fixed point mapping considered here is based on the perceptron learning rule. Of course only the connection matrix between the two perceptron layers has to be determined.

Let us consider a single field of perceptrons in the 2nd hidden layer. The goal is that a certain number of perceptrons must be on. If the desired number is less than the actual number then certain of the on-neurons must be turned off by adjusting their weights. If the desired number is greater than the actual number then certain of the off-neurons must be turned on. Several strategies for determining which additional units should be turned off or on are possible. The strategy chosen here is to modify the weights of the neurons whose activation values are closest to their thresholds. We may call this the "least change" algorithm. For a particular field let n_d be the desired number of 1's, let n_a be the actual number of 1's, and let k be $n_a - n_d$. If $k > 0$ then the activation values of k of the neurons must be turned negative. If the neurons in the field are arranged in descending order of activation values then the ideal neurons to change are neurons n_d, $n_d + 1$, ... $n_a - 1$. Note that neurons 0, 1, .. $n_a - 1$ are positive and

ESANN'1999 proceedings - European Symposium on Artificial Neural Networks
Bruges (Belgium), 21-23 April 1999, D-Facto public., ISBN 2-600049-9-X, pp. 307-312

the rest are negative. Neuron 0 is the first neuron in the field under discussion after the neurons have been relabeled for ordering. If $k < 0$ then the activation values of k of the neurons must be turned positive. Again if the neurons in the field are arranged in descending order of activation values then the ideal neurons to change are neurons n_a, $n_a + 1$, .. $n_d - 1$.
For example let the desired integer value for a field be 4 in a field width of 7 and let the actual pattern be 1001010. This means that one additional neuron needs to be turned on. Also assume the following.

Indices of neurons	8	9	10	11	12	13	14
Pre-threshold values	4.0	-0.1	-0.3	3.1	-0.05	1.2	-8.1
Post-threshold values	1	0	0	1	0	1	1
Ordering of neurons (by pre-threshold values)	8	11	13	12	9	10	14
Post-threshold values	1	1	1	0	0	0	0

Based on the new ordering and for "least change" strategy the 12th element should be modified to give a positive activation value. On the other hand if the desired integer value is 2 then the 13th element should be modified.
Once it has been determined in each field which neurons must have their post-threshold values reversed a target binary vector, **d**, has been established which can be used in modifying the connection matrix using the expression following. For n perceptrons, each with the same n inputs, the equations for learning become in matrix form (Brouwer 1995)

$$\mathbf{e} = \mathbf{d}-\mathbf{a}$$
$$\mathbf{W}' = \mathbf{W} + \mathbf{e} \otimes \mathbf{x}^t$$
$$\mathbf{t}' = \mathbf{t} - \mathbf{e}$$

e is the error vector, **a** is the actual vector of neuron states for the 2nd hidden layer, **d** is the desired output vector and **x** is the vector of outputs of the 1st hidden layer. The symbol $\otimes$ denotes the vector cross product. This is the perceptron learning rule essentially.

Hetero-associative Memories for Classification

(Incorporating the attractee-attractor relation in the Network Explicitly)

Brouwer (1995) describes a method of training a Hopfield style network, a GDHN, for classification which focuses on the attractees rather than on the attractors. Rather than training the network so that the class exemplars become fixed points or attractors, training is done so that the class exemplars are attracted to particular attractors.

This approach is based on the point of view that (1) a distinction should be drawn between exemplars and prototypes in incorporating sample data in the network and (2) training should explicitly include the storing of the relationship between attractees and their attractors.

ESANN'1999 proceedings - European Symposium on Artificial Neural Networks
Bruges (Belgium), 21-23 April 1999, D-Facto public., ISBN 2-600049-9-X, pp. 307-312

Algorithms for storing the attractee-attractor(Ae-Ar) relation are based on the transformation $\mathbf{y} = \mathbf{AT}(\mathbf{x})$ (Brouwer, 1995). $\mathbf{x}$ here is the attractee that is attracted to attractor y. AT(x) is defined in terms of $\mathbf{T}(\mathbf{x})$ by

$$\exists \text{ an integer } k > 0 \quad \ni \quad \mathbf{AT}(\mathbf{x}) = \mathbf{T}^{(k)}(\mathbf{x}) \quad \Leftrightarrow \quad \mathbf{T}^{(k)}(\mathbf{x}) = \mathbf{T}^{(k-1)}(\mathbf{x})$$

$T^{(k)}(\mathbf{x})$ is defined by the recursive expression

$$\mathbf{T}^{(0)}(\mathbf{x}) = \mathbf{x}$$
$$\mathbf{T}^{(k)}(\mathbf{x}) = \mathbf{T}(\mathbf{T}^{(k-1)}(\mathbf{x}))$$

To permit the generalization required for classification of unclassified vectors, the attractors should serve as attractors not only for elements in the training set but also for elements similar to elements in the training set. The class of an arbitrary input is found if the input is attracted to an element of one of the attractor sets for the training sets.

Generally in the classification of feature vectors prototypes are not available as potential attractors. In that case a set of all allowable attractors (which may be the set of training elements to which the attractee belongs) is provided for each attractee or independent state and through training actual attractors are found. The network thus may be trained such that all elements in a training set are attracted to elements in the same training set. The target attractor for an attractee is the training element in the same set as the attractee that is closest to the existing attractor for the attractee. Once a target integer vector for an attractee has been selected the training method of the previous section can be applied. For distance between two elements we may use the Euclidian distance.

The available data is broken up into purely training elements (for adapting the connection matrix), validation elements (to check generalization) and testing elements. At the end of each epoch the best connection matrix so far (in terms of generalization ability) is replaced (Gallant 1990) if a connection matrix is found which generalizes better. The training is discontinued when the percent correctly classified reaches a predetermined value or after a certain number of epochs have passed.

Simulation Results - Classification of Sugar Diabetes Patients and Credit Applicants

Following are the results of applying the method described in this paper to sugar diabetes data and credit data. This is a two class problem where the objective is to predict whether a patient would test positive for diabetes according to World Health Organization criteria (Michie, 1994). This dataset was originally donated by Vincent Sigillito, Applied Physics Laboratory, Johns Hopkins University, Laurel, MD 20707 and was constructed from a larger database. Several statistical and neural networks were tested on this data (Michie, 1994). Logdisc provided the best test accuracy at 77.7 % and the worst of 67.6 % was provided with k-NN. Another data set to which the method has been applied is the Australian credit data set consisting of two classes,

14 attributes or features and 690 observations (Michie 1994). Of 22 other methods applied to this according to Michie(1994) the best accuracy achieved was 86.9 % with Cal5 and the worst was 80% with Quadisc. Backprop produced an accuracy of 84.6 %.
The results obtained by the author's method are as follows. In case of the diabetes data 75.6 % of 214 test elements were correctly classified using 4 hidden perceptrons per feature and a training time of 16 epochs. In case of the credit data 85% of 244 test elements were correctly classified using 2 hidden perceptrons per feature and a training time of 32 epochs. Note that a real valued feature value is represented by a natural number according to the "thermometer code"(Gallant 1990). The range of the real variable to be represented is divided up into intervals which are usually chosen to be of equal size. The intervals may be numbered from 0 to n-1. The integer corresponding to a real value is then the interval to which the real value belongs.

Conclusion

A recurrent network has been proposed which has been shown to be useful in classification of feature vectors. Its advantages are ease of training and accuracy for classification. The network consists of two kinds of units. One counting unit is required for each feature in the feature vectors. The other units are perceptrons. The number of perceptrons required per feature is small. Further study may include improvements or fine-tuning of the algorithm.

References

Brouwer, R. K. (1995). A Method for Training Recurrent Neural Networks for Classification by Building Basins of Attraction. *Neural Networks*, **8**/4, 597-603.
Gallant, S.I. (1990) *Perceptron-Based Learning Algorithms.* I.E.E.E. Transactions on Neural Networks. Vol. 1. No. 2. June 1990 pp. 179-91.
Hecht-Nielsen, R. (1989). *Neurocomputing*. New York: Addison-Wesley.
Hopfield, J. J. & Tank, D. W. (1986). Computing with Neural Circuits: A Model. *Science* 233: 625-633.
Kamp, Y., Hasler, M.(1990) *Recursive Neural Networks For Associative Memory*, John Wiley and Sons.
Michie, D., Spiegelhalter, D. J. & Taylor, C. C. (1994). *Machine Learning, Neural and Statistical Classification* Ellis Norwood.
Werbos, P.J. (1974). Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences. *Doctoral Dissertation*, Appl. Math., Harvard University, Mass.