ESANN'1996 proceedings - European Symposium on Artificial Neural Networks
Bruges (Belgium), 24-25-26 April 1996, D-Facto public., ISBN 2-9600049-6-5, pp. 149-154

A Novel Two-layer Neural Network Classifier

Guoping Qiu

School of Computing & Mathematics
University of Derby, Derby DE22 1GB, United Kingdom

Abstract: In this paper, a novel neural network architecture designed for pattern classification purposes is presented. The classifier is a two-layer neural network. The first layer classifies the input vectors into a number of clusters using a stochastic competitive learning algorithm. The output of this layer is a Gibbs probability distribution for the association of the input with these clusters. The output of the first layer is used as input to the second layer that implements a classifier similar to the Bayes minimum risk classifier. A new complementary Hebbian learning algorithm is proposed to train the second layer. Computer simulations have been performed and the results demonstrate that the new classifier consistently provides high correct recognition rates and is competitive to other similar systems.

1. Introduction

One of the most popular application areas of neural networks has been pattern recognition. Many researchers have shown the effectiveness of the multilayer feedforward networks in many pattern recognition tasks. However, the learning process of backpropagation algorithm is very slow, in addition, it is very difficult to interpret the outputs of the hidden layer units. In this paper, we present a novel neural network architecture. Experimental results are used to show that the new system has a fast convergence rate and can provide high recognition performance. It is also shown that the new system is competitive to other neural models.

2. Minimum risk classifier neural network

This is a two-layer neural network. The task of the first layer is to classify the input into a number of clusters and produce as its output a Gibbs probability distribution for the association of the input with these clusters. The task of the second layer is to decide the class of the input vector according to a minimum risk principle similar to that of Bayes classifier. A schematic diagram of the network is shown in Fig. 1.

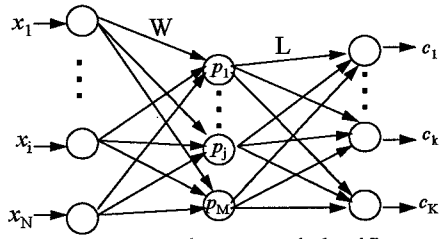

Fig. 1. A two-layer neural classifier

ESANN'1996 proceedings - European Symposium on Artificial Neural Networks
Bruges (Belgium), 24-25-26 April 1996, D-Facto public., ISBN 2-9600049-6-5, pp. 149-154

Let $\bar{w}_i = (w_{1i}, w_{2i}, \cdots, w_{Ni})$ be the connection weight vector between input layer and the *i*th unit in the hidden layer, $\bar{l}_i = (l_{1i}, l_{2i}, \cdots, l_{Mi})$ be the connection weight vector between hidden layer and the *i*th unit in the output layer. Let $\bar{p} = (p_1, p_2, \cdots, p_M)$ be the hidden layer output vector and $\bar{c} = (c_1, c_2, \cdots, c_K)$ be the output vector. The operational equations of the network are as follows:

$$p_i = \frac{e^{-\beta \|\bar{x} - \bar{w}_i\|^2}}{\sum_{j=1}^{M} e^{-\beta \|\bar{x} - \bar{w}_j\|^2}} \qquad \forall i \tag{1}$$

where $\|\cdots\|$ is the Euclidean norm, $\beta > 0$ is an "annealing factor".

$$c_i = \begin{cases} 1 & if\ \sigma(\bar{l}_i \bar{p}^T) < \sigma(\bar{l}_j \bar{p}^T),\ for\ j = 1,2,\cdots,K\ and\ j \neq i \\ 0 & \text{otherwise} \end{cases} \quad \forall i \tag{2}$$

where $\sigma(z) = \frac{1}{1+e^{-z}}$ is the transfer function, T represents transposition. The network classifies the input vector belongs to class *i* if and only if $c_i = 1$.

2.1 Gibbs distribution clustering sublayer

We can view Eq. (1) as the Gibbs probability distribution [1] for the association of the input vector with the M clusters. Therefore, we call the first layer the Gibbs distribution clustering sublayer. The cluster centre vector of the *i*th cluster is $\bar{w}_i$. The training task for this layer is to estimate these M cluster centres. Whilst there are many clustering techniques , such as the *K*-means algorithm [2] and Kohonen's learning vector quantization (LVQ) [3], can be used to calculate the cluster centres, we present in this section a robust clustering algorithm [4]. The method is called *Stochastic Competitive Learning Algorithm* (*SCLA*) and is briefly described as follow

1. Initialise $\bar{w}_i(0)$, for *i* = 1, 2, ···, M, set $\beta = \beta(0)$.
2. Update all weight vectors according the following equation

$$\bar{w}_i(t+1) = \bar{w}_i(t) + \gamma_i(t) \sum_{k=1}^{J} p_i(k)(\bar{x}(k) - \bar{w}_i(t)) \quad \forall i \tag{3}$$

   where $p_i(k)$ is the *i*th hidden unit's output when the *k*th training vector $\bar{x}(k)$ is presented to the network. J is the total number of training samples.
3. Increase β, if $\beta(t) < \beta_{max}$, go to 2, otherwise stop.

The algorithm starts at a small value of β, for which $p_i(k)$ is approximately uniform. That is, at the early stage of the learning process, there is not a definite winner. As learning progress, β is gradually increases and the output of the units

ESANN'1996 proceedings - European Symposium on Artificial Neural Networks
Bruges (Belgium), 24-25-26 April 1996, D-Facto public., ISBN 2-9600049-6-5, pp. 149-154

$p_i(k)$ begins peaking up around the Euclidean winner. In time, the units which are closer to the input will have a larger output, whilst those units which are far away from the input vector will have a smaller output. In the limit $\beta \to \infty$, for each input vector, only one processing unit will have its output equal to one, all the others will have their outputs equal to zero, i.e. the network becomes a winner-take-all network.

In SCLA, the weight vectors are updated in a *batch mode*, that is, the weights are updated after the entire set of training samples are presented to the network; this operation, as opposed to the *on-line* updating mode, can eliminate the effects of different orders of input sequence on the final results.

Before implementing the SCLA, there are several parameters have to be decided. First, the initial annealing factor $\beta(0)$. There exists no theoretical guidance of choosing this value. This value can not be set too high or too low. If it is too low, all the processing unit outputs will be equal to one another, the weight vectors may group together, thus resulting in poor solutions. Conversely, if it is too high, the annealing process may not be effective. Through experiment, it was found that the average squared Euclidean distance between all the sample points and their centre was a good candidate.

$$\frac{1}{\beta(0)} = \frac{1}{J}\sum_{k=1}^{J}\|\bar{x}(k) - C\|^2 \qquad (4)$$

where $C = \frac{1}{J}\sum_{k=1}^{J}\bar{x}(k)$ is the centre vector of all the training samples

Secondly, we have to decide a scheme for increasing β. For simplicity, a so called "conceptually simple" cooling schedule given by: $\beta(t+1) = 1.05\beta(t)$ is adopted [4].

Another important issue is to choose a scheme for updating the learning rates. A commonly used approach [1] is to start from some chosen values of the learning rate, and gradually reducing them during the learning process. One suggestion of choosing the learning rate values by Grossberg is as follows: if a given unit wins the competition frequently, its learning rate should be reduced; conversely, if a given unit wins less frequently, its learning rate should be increased. Following this suggestion, a choice of the learning rates for the SCLA can be set as follow [4]:

$$\frac{1}{\gamma_i(t)} = \sum_{k=1}^{J} p_i(k,t) \qquad \text{for } j = 1,2,\cdots,n \qquad (5)$$

where $p_i(k,t)$ is the output of the hidden unit *i* at time *t*.

ESANN'1996 proceedings - European Symposium on Artificial Neural Networks
Bruges (Belgium), 24-25-26 April 1996, D-Facto public., ISBN 2-9600049-6-5, pp. 149-154

SCLA is a robust clustering algorithm in the sense that the final cluster centres obtained are not dependent on the initial weight vectors. In our present application, the stopping criteria β_{max} determines the "fuzziness" among the clusters.

2.2 Complementary Hebbian leaning for minimum risk classification

In the classical Bayesian approach to pattern recognition, the starting point is the definition of a decision function for each of the all possible classes. Let $d_i(\bar{x}(k))$ be the decision function for class c_i. The Bayes' decision rule decides $\bar{x}(k)$ belongs to class c_j if and only if $d_j(\bar{x}(k)) < d_i(\bar{x}(k))$, for all $i \neq j$. In principle and in practice, the decision functions $d_i(\bar{x}(k))$ is some functions of the a posteriori probability measures, $p(c_i|\bar{x}(k)$, for i = 1, 2, ..., K. Such a classifier is known as the Bayes minimum risk classifier [5].

The task of the second layer of the network is to decide the class of the input vector. To achieve this aim, we implement this layer following a similar approach to that of Bayes minimum risk classifier. We view the output of the second layer as an estimation of the a posteriori probability distribution measures. However, in our present system, the number of hidden units and therefore the number of the a posteriori probability distribution measures does not have to be identical to the number of classes, whilst in Bayes classifier each class has a corresponding the a posteriori probability distribution measure. In addition, we do not have to know explicitly the relation between the hidden layer units and the class patterns. We view $\sigma(\bar{l}_j\bar{p}^T)$ as the risk undertaken in deciding the input belongs to class j. The learning process has to achieve the aim that $\sigma(\bar{l}_j\bar{p}^T)$ is the smallest when the input comes from class j. To do this, we propose in this session a complementary Hebbian learning algorithm as described below: For all j = 1, 2, ..., K and i = 1, 2, ..., M, we set the initial weights $l_{ij}(0) = 0$, the connection matrix is updated as

If $\bar{x}(t) \in class\, j$: $l_{ij}(t+1) = l_{ij}(t) - \lambda(p_j - \sigma(\bar{l}_j\bar{p}^T)l_{ij}(t))\sigma(\bar{l}_j\bar{p}^T)$ (6)

If $\bar{x}(t) \notin class\, j$: $l_{ij}(t+1) = l_{ij}(t) + \xi(p_j - \sigma(\bar{l}_j\bar{p}^T)l_{ij}(t))\sigma(\bar{l}_j\bar{p}^T)$ (7)

where λ and ξ are learning constants and $\lambda > \xi$. Clearly, equation (7) is a modified version of Oja's Hebbian rule [1] (non-linear output units are used here whilst the original Oja's Hebbian rule uses linear output units). Equation (6) is the anti-Hebbian learning counterpart of (7). We therefore call the algorithm complementary Hebbian learning.

The basic idea behind the algorithm is: If $x(t) \in c_j$, we want to make $\sigma(\bar{l}_j\bar{p}^T)$ the smallest. The learning algorithm uses (6) to decrease $\sigma(\bar{l}_j\bar{p}^T)$ and (7) to

ESANN'1996 proceedings - European Symposium on Artificial Neural Networks
Bruges (Belgium), 24-25-26 April 1996, D-Facto public., ISBN 2-9600049-6-5, pp. 149-154

increase $\sigma(\bar{l}_i\bar{p}^T)$, *for* $i = 1,2,\cdots,K,\ i \neq j$. The training is completed when no improvements in the number of correctly classified vectors are achieved by further iterations. Theoretical analysis of the properties of the algorithm has yet to be conducted at this stage. However, extensive simulations have shown that the algorithm always converges within a small number of epochs.

3. Simulation results

The experiment is performed on a non-separable problem taken from Kohonen [6]. The data are drawn from two concentric two-dimensional Gaussian distributions, each with zero mean. The co-ordinates are independent for both classes with a variance of 1 for class 1 and a variance of 4 for class 2. These distributions overlap, so no classifier provides perfect classification. The optimal Bayes minimum risk classifier will classify 73.624 % of vector correctly on average.

We use a set of 10,000 random vectors as training samples and another set of 10,000 random vectors as test samples. A network with two input, 32 hidden and two output units is trained. Different values of β_{max} have been tried in the training of the first layer. Table 1 shows the results of ten simulations. Also shown in the table are the number of epochs used to train the first and second layer. The training rates are set as $\lambda = 0.005$ and $\xi = 0.0005$.

TABLE 1 PERCENTAGE OF VECTORS CORRECTLY CLASSIFIED

β_{max}	Training set (% correct)	Testing set (% correct)	No. of epochs (first layer)	No. of epochs (Second layer)
1.71	73.68	73.02	46	6
1.80	73.58	73.13	47	6
1.89	73.53	73.05	47	6
1.98	73.60	73.07	48	6
2.08	73.53	73.15	49	7
2.18	73.50	73.11	50	6
2.41	73.44	73.18	52	7
2.52	73.36	73.18	53	7
2.79	73.36	73.08	55	5
3.23	73.46	72.95	58	5
average	73.504	73.092	51	7

The values of β_{max} are determined through experiment. The overall trends are, for smaller β_{max} (the clusters in the first layer are "fuzzier"), the correct recognition rate is higher for the training data, whilst for larger β_{max} (the clusters in the first layer are "less fuzzy"), the correct recognition rate is lower for the training data. The best generalisation capability (the highest correct recognition rate for the testing data) is achieved at a compromise, i.e. at a middle β_{max} value.

ESANN'1996 proceedings - European Symposium on Artificial Neural Networks
Bruges (Belgium), 24-25-26 April 1996, D-Facto public., ISBN 2-9600049-6-5, pp. 149-154

A neural network model proposed by Spetch [7], the so called probabilistic neural network (PNN) follows closely the principle of Bayes classifier. In fact it is more appropriate to consider PNN as a method of estimating the a posteriori probability distribution measures instead of a neural network. We have implemented PNN on the same data sets. Table 2 shows ten best results obtained on the testing set, where σ is the standard deviation of the Gaussian function used to estimate the probability measures [7]. The average correct rate is 73.116% which is 0.0018% higher than that obtained by the new method. Although training of PNN is one pass process, the computational load in the operation stage and memory requirements are proportional to the size of training samples. Therefore, PNN is considerably more computationally intensive and requires more memory space than the new method.

TABLE 2 PERCENTAGE CORRECTLY CLASSIFIED VECTORS BY PNN

σ	0.187	0.2	0.212	0.224	0.316	0.387	0.447	0.5	0.548	0.592
%	73.02	73.11	73.14	73.17	73.27	73.13	73.20	73.06	73.05	73.01

We have also implemented Kohonen's learning vector quantization classifier on the same sets of data. Different network size have been tried many times. The best results obtained are 73.05% correct recognition rate for the training data and 72.79% correct recognition rate for the testing data.

4. Conclusion

In this paper, a novel neural network classifier designed based on two learning algorithms, an unsupervised competitive learning and a supervised Hebbian leaning has been described. Simulation results are presented which indicate the network has a fast convergence rate and is competitive to other similar models.

5. References

1. J. Hertz, A Krogh and R Palmer: Introduction to the theory of neural computation, Addison-Wesley Publishing Company (1991)
2. R. Duda and P Hart: Pattern classification and scene analysis, Wiley (1973)
3. T. Kohonen: Self-organization and associative memory, Springer-Verlag (1989)
4. G. Qiu: An investigation of neural networks for image processing application, PhD thesis, University of Central Lancashire, UK (1993)
5. Y. Pao: Adaptive pattern recognition and neural networks, Addison-Wesley publishing Company (1989)
6. T. Kohonen, G. Barna and R. Chrisley: Statistical pattern recognition with neural network: Benchmarking studies. IEEE Int'l conf. neural networks, 595-645 (1988)
7. L. Fausett: Fundamentals of neural networks - Architecture, algorithms and application, pp. 385-389, Prentice Hall International, Inc. (1994)