Recent Research in Science and Technology 2012, 4(1): 24-32
ISSN: 2076-5061
Available Online: http://recent-science.com/

Design of an optimal multi-layer neural network for eigenfaces based face recognition

Ravindra Pal Singh[1]*, Vinay Rawat[1], Mayank Pawar[2] and Raj K. Mishra[1]

[1]Dev Bhoomi Group of Institutions (DBGI), Dehradun (U.K), India.
[2]Teerthankar Mahaveer University (TMU), Moradabad (U.P), India.

Abstract
Face recognition is one of the most popular problems in the field of image analysis. In this paper, we discuss the design of an optimal multi-layer neural network for the task of face recognition. There are many issues while designing the neural network like number of nodes in input layer, output layer and hidden layer(s), setting the values of learning rate and momentum, updating of weights. Lastly, the criteria for evaluating the performance of the neural network and stopping the learning are to be decided. We discuss all these design issues in the light of the eigenfaces based face recognition. We report the effects of variations of these parameters on number of training cycles required to get optimal results. We also list the optimized values for these parameters. In our experiments, we use two face databases namely ORL and UMIST. These databases are used to construct the eigenfaces. The original faces are reconstructed using the top eigenfaces. The factors used in the reconstruction of the faces are used as the inputs to the neural network.

Keywords: Neural network, eigenfaces, face recognition, hidden layer, back propagation.

INTRODUCTION

Face recognition is one of the well-known problems in the field of image processing. In face recognition problem, a given face is compared with the faces stored in a face database in order to identify the person, who have the given face. The purpose is to find a face in the database, which has the highest similarity with the given face. One of the important algorithms for face recognition is the eigenface algorithm [1]. Since, face recognition is a high-dimensional pattern recognition problem, eigenface algorithm, which reduces the dimensionality of the input face space, is found to be one of the most successful methodologies. Eigenface algorithm uses the Principal Component Analysis (PCA) for dimensionality reduction to find the vectors which best account for the distribution of face images within the entire image space. These vectors define the subspace of face images and the subspace is called face space. All faces in the training set are projected onto the face space to find a set of weights that describes the contribution of each vector in the face space. To identify a test image, it requires the projection of the test image onto the face space to obtain the corresponding set of weights. By comparing the weights of the test image with the set of weights of the faces in the training set, the face in the test image can be identified. A multi-layer perceptron (MLP), a multi-layer neural network that was first proposed by Frank Rosenblatt [2], has also been widely used for the task of face recognition [3, 4].

In this paper, the design of an optimal multi-layer perceptron for eigenfaces based face recognition. There are many issues while designing the multi-layer neural network like number of nodes in input layer, output layer and hidden layer(s), setting the values of learning rate and momentum, updating of weights. Lastly, the criteria for evaluating the performance of the neural network and stopping the learning are to be decided. We discuss all these design issues in the light of the eigenfaces based face recognition. We try to get optimal value for all these parameters.

The paper is organized as follows. In Section 2, we discuss the suitability of multi-layer perceptron for task of face recognition. In Section 3, we discuss all the design parameters in detail. We briefly look at the related work in Section 4. We present our results in Section 5. Finally, we conclude in Section 6.

THEORY

Multilayer Perceptron

The single layer perceptrons have two layers consisting of neurons, an input layer and an output layer. The output of a discrete neuron can only have the values zero (non firing) and one (firing). Each neuron has a real-valued threshold and fires if and only if its accumulated input exceeds that threshold. Each connection from an input node j to an output neuron i has a real-valued weight w_{ij}. For some problems, like the famous XOR problem, the single layer perceptrons fail to perform. This paves the way for the more advanced multi-layer perceptrons. MLPs are feed forward neural networks trained with the standard back propagation algorithm and have one input layer, one output layer and one or more hidden layers.

By training a MLP, the output space is separated into regions. The ability of a MLP to correctly classify input data patterns, which has not been used for training the MLP, is termed as generalization [5]. The multi-layer perceptrons are highly suitable for any classification task e.g. pattern recognition, character recognition and face recognition. This is due to the reason that the MLPs build hyper

Received: Nov 13, 2011; Revised: Dec 20, 2011; Accepted: Jan 13, 2012.

*Corresponding Author

Ravindra Pal Singh
Dev Bhoomi Group of Institutions (DBGI), Dehradun (U.K), India.

Tel: +91-9456667003;
Email: ravindra_rajeev@yahoo.com

surfaces that divide the output space into different classes that have dissimilar properties. In the face recognition process it is desired to have all the faces of the same person to belong to the same class and the faces of different persons to be classified as different classes. A remarkable thing about MLP is that it has good extrapolative and interpolative properties and is thus able to correctly classify faces that it has not been trained with as belonging to the correct class. The MLPs are trained with the back-propagation algorithm which is an error-minimizing and optimization approach.

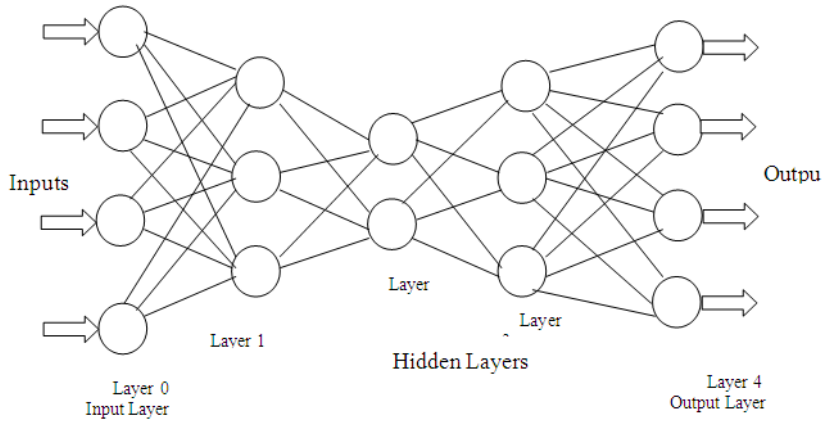

Fig1. A Multi-layer Perceptron

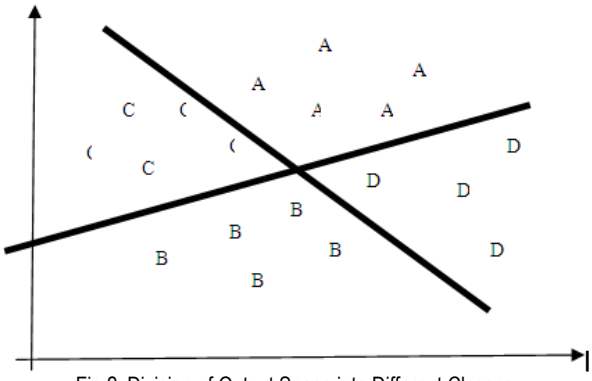

Fig 2. Division of Output Space into Different Classes

Designing an Optimal Multi-layer Neural Network

While designing an optimal multi-layer Neural Network, we have to decide upon a number of parameters. In this section, we discuss the different parameters and the way in which, we determine the optimal values for these parameters.

Number of Nodes in the Input Layer

The number of input nodes can generally be easily determined because the number of nodes in the input layer is equal to the number of inputs that we want to feed into the network. In the typical neural networks for the recognition of the faces, the number of input nodes is equal to the number of pixels in the face image. It leads to the huge complexity of the neural network architecture. But in the Principal Component Analysis, rather than applying the pixel values as the input, only the multipliers of the eigenfaces are used as the input. This approach greatly reduces the complexity of the neural network architecture. The original faces are represented as the sum of products of the eigenfaces and these multipliers. So these multipliers can be used to differentiate between the faces of different persons. The main hurdle is the determination of the number of eigenfaces that are sufficient to correctly represent the variation in the faces. The number of input nodes is then equal to the number of eigenfaces and the input values are the multipliers with which these eigenfaces should be multiplied to correctly reconstruct the original faces.

Number of Nodes in the Output Layer

The number of nodes in the output layer is equal to the number of persons that are to be recognized. The value of the output node corresponding to the correct face will be highest while the other output nodes will have very less values. The output nodes have competition among themselves for the highest output value and the node having the highest value is the winner that decides the identity of the person. It is also possible to have one extra output node for the faces of persons that are outside the training set of faces.

Number of Hidden Layers

Generally one hidden layer with sufficient number of nodes is enough for most of the problems that use neural networks. It is advisable to use as few hidden layers as possible because the addition of each hidden layer significantly increases the network complexity,

increases the number of weighted connections between the nodes and unnecessary addition of the hidden layers will lead to slower learning. However, having more than one hidden layer has few advantages when they are used in the networks where their use is essential, like better learning of relationship between inputs and outputs, faster learning and at times having more than one hidden layer can help in avoiding the pitfalls of the local minimums. For the design of an optimal multi-layer perceptron, we may vary the number of hidden layers and study the effect of these variations. We may also vary the number of nodes in the first hidden layer and see the effects of these variations. Then, we may add the second hidden layer and vary the number of nodes in both the hidden layers and note the effects of these variations. We may repeat the process until we decide upon an optimal value for the number of hidden layers.

Number of Hidden Nodes

Determining the number of hidden nodes is a tricky problem and there are no absolutely correct guidelines for the number of nodes. The number of hidden nodes determines the mapping ability of the network. In other words, larger the number of hidden nodes, more powerful is the network. However, if this number is too large, the generalization may get worse. This is due to over-fitting the training set, which can be solved by using cross-validation. If there are too many hidden nodes in the network, many problems may occur e.g. too much training time, the network may fail to generalize the input data and it may instead memorize the correct response to each input pattern. On the other hand, if there are too few hidden layer units, the network may fail to train correctly because this may result in insufficient and incorrect mapping between the inputs and the outputs. If we examine the weight values on the hidden nodes periodically as the network trains, we can see that weights on certain nodes change very little from their starting values. These nodes may not be participating in the learning process, and fewer hidden nodes may suffice. Some rules that may be used for determining the number of hidden nodes are given as

$$N_{hid} \approx \sqrt{N_{inp} N_{out}}, \quad N_{hid} \leq N_{inp} \text{ and } N_{hid} \leq N_{out}$$

where, N_{hid} is number of hidden nodes, N_{inp} is number of input nodes and N_{out} is the number of output nodes.

Number of Training Examples

From the available training data, a subset of data is needed to train the network successfully. The remaining data can be used to test the network to verify that the network can perform the desired mapping on input vectors it has never encountered during training. In contrast to generalization, the back-propagation network does not extrapolate well. If it is inadequately or insufficiently trained on a particular class of input examples, subsequent identification of members of that class may be unreliable. The faces of the same person can have a large number of variations like presence or absence of glasses; facial expressions like smile, frown, etc; changes in pose in horizontal and vertical plane. This necessitates a large number of faces per person that capture almost all of the variations that a person's face can have. We work with two face databases namely ORL [6] and UMIST [7]. The ORL face database consists of 10 faces for each person, out of which, we use 8 faces per person in training the neural network and the rest 2 for testing the neural network. For the UMIST face database, 14 faces of a person are used in training the neural network and 5 faces are used in testing the neural network. This may be noted that the number of training examples should neither be so less that the MLP is not able to correctly generalize the classes, nor it should so large that the network memorizes the faces.

Dealing with the Local Minima

Sometimes during training it is observed, that after long training, the algorithms seem to stall. In other words, error remains high and the continuous training does not lead to its reduction. One of the explanations is that the optimization algorithm has found a local minimum, but not the global minimum. Once the network settles on a minimum, whether local or global, learning ceases. Since back-propagation uses a gradient-descent procedure, a back-propagation network follows the contour of an error surface with weight updates moving it in the direction of steepest descent.

As a general rule of thumb, the more hidden nodes we have in a network the less likely we are to encounter a local minimum during training. Although additional hidden nodes increase the complexity of the error surface, the extra dimensionality increases the number of possible escape routes.

In case the back-propagation seems to stall, some *help* is needed. The various suggestions to deal with the local minima and get out of it, may be listed as

1. We may make use of the adaptive learning rate.
2. Weights can be re-randomized and the process be repeated.
3. The number of hidden nodes can be changed.
4. Addition of the momentum term may help in taking large steps in the correct direction thus over-stepping some of the local minimums.

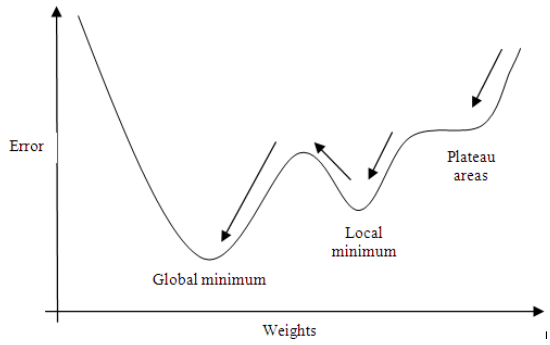

Fig 3. Variation of learning error with weight adjustment

Momentum

The original back-propagation algorithm is quite slow. By adding a term to the weight adjustment that is proportional to the amount of the previous weight change, the performance of the back-propagation algorithm can be improved. Such term is called momentum. The purpose of the momentum method is to accelerate the convergence of the back-propagation algorithm. The concept of momentum is that previous changes in the weights should influence the current direction of movement in weight space.

The benefits of using momentum in terms of optimization for the neural network learning are: it speeds up the back-propagation of errors, keeps the error minimization process going in the same

direction and also helps in jumping out of a local minimum and guides in finding the global minima.

The benefits of using momentum in terms of optimization for the neural network learning are listed as follows.

1. It speeds up the back-propagation of errors.
2. It keeps the error minimization process going in the same direction.
3. It helps in preventing the oscillations, which are very common in the traditional back-propagation approach.
4. The momentum term helps in jumping out of a local minimum and guides in finding the global minima.

Learning Rate

The back-propagation algorithm requires that the weight changes be proportional to the derivative of the error. The larger the learning rate the larger the weight changes on each epoch, and the quicker the network learns. However, the size of the learning rate can also influence whether the network achieves a stable solution. If the learning rate gets too large, then the weight changes no longer approximate a gradient descent procedure (true gradient descent requires infinitesimal steps). Oscillation of the weights is often the result. Ideally then, one should like to use the largest learning rate possible without triggering oscillation. This would offer the most rapid learning and the least amount of time spent waiting for the network to train. We may obtain the optimum value the learning rate by hit and trial. The learning rate may be kept the least in the beginning and then, it may be slowly increased in steps of 0.1 up to the optimum value, after which, its increase may lead to errors in correct recognition.

Updating the Weights

Back-propagation is essentially a learning algorithm that modifies the weights as is dictated by the error. The error is propagated backwards through the network and used to update the weights. After the weights are updated, the next instance is used to calculate the output, compute the errors, calculate the weights' updates, etc. There are two ways in which the weights can be updated. Updating the weights in a back-propagation network can be done either after the presentation of each pattern (say, pattern learning), or after all of the patterns in the training set have been presented (say, epoch learning). If the learning rate is small, there is little difference between the two procedures. However, substantial differences can be observed when the learning rate is large, as the derivation of the back-propagation algorithm assumes that the error derivatives are summed over all of the patterns. When the weights are updated immediately it leads to slow learning but the network learns in a better manner. On the other hand, when the weights are updated after each epoch it leads to faster convergence, speeding up of the learning process but may lead to improper training. In our case, the weights are updated after all the training examples have been presented.

Initialization with Free Parameter

The parameters that need to be set before the learning can start are: weights, biases and thresholds. A good choice for the initial values of the synaptic weights, biases and thresholds of the network can significantly accelerate learning.

A common practice is to set all the free parameters of the network to random numbers that are uniformly distributed inside a small range of values. If the weights are too large the sigmoid functions will start saturating from the very beginning of training and the system will become stuck in a kind of saddle point near the starting point itself. This phenomenon is known as "premature saturation". Premature saturation can be avoided by getting the initial values of the weights and threshold levels of the network, to be uniformly distributed inside a small range of values. The weights may be randomly set to some random values, when the process of training is started.

Pruning

When a MLP network is created the layers are fully connected with weights between all the nodes of the input, output and hidden layers. Some networks can learn with a much lower number of connections. The process of removing the weights that contain no useful information but just add to the network complexity is called pruning. When we examine the weights of the various connections, we may find many weights that are not changing at all or very leFGss. Such weights are redundant and do not help the network in learning. Pruning of the neural network serves two purposes. First, it speeds up the neural network learning by eliminating redundant weights and second, it reduces the network complexity. Pruning is performed when the values of the weights fall below some thresholds.

Stopping Criteria

Eventually training of the neural network has to terminate so that it can be queried to get the response for some input. The neural network will continue to learn until some stopping criteria is satisfied and the training can be stopped. There are a number of ways to suggest when the network training should stop. For example, the network training may be stopped if any of the following three criteria is satisfied.

1. The average/maximum/minimum training error falls below some user-defined level
2. The number of training cycles or epochs exceeds a certain value.
3. All the validation tests have been performed.

Performance Criteria

The performance criteria are parameters on the basis of which the performance of the network can be evaluated. Three of the performance criteria, which are suitable for the face recognition neural network, are- 1. the accuracy of the recognition results, 2. the number of cycles and 3, the absolute or relative errors in the training and testing phases.

Related Work

Attempts have been made in past to use the neural network for the problem of face recognition [3, 4, 8, 9]. For example, in [3], the first 50 principal components of the images are extracted and reduced to 5 dimensions using an auto associative neural network. The resulting representation is classified using a standard multi-layer perceptron. Good results are reported but the database is quite simple, the pictures are manually aligned and there is no lighting

variation, rotation, or tilting. There are 20 people in the database. Pan Z. et al., in their study [4], used the multilayer perceptron neural network. There is just one hidden layer with number of hidden units being in between 60 to 80. The input of neural network is a set of discrete cosine transform coefficients. They report that the achieved recognition rate is in between 94% to 97%. In [8], Henry Rowley et. al. used three types of hidden units, 4 looking at 10x10 pixel sub-regions, 16 looking at 5x5 pixel sub-regions and 6 looking at 20x5 pixel sub-regions. These sub-regions are chosen to represent facial features that are important to face detection. Overlapping detections are merged. To improve the performance of their system, multiple networks are applied. They are trained under different initial condition and have different self-selected negative examples. The outputs of these networks are arbitrated to produce the final decision. In [9], there is only one hidden layer in the network with 20 to 30 units in it. The number of units in the input layer is equal to the number of image pixels, 2576 (i.e. 46 × 56). Gray level of every pixel is linearly scaled from range (0, 255) to (-0.05, +0.05). The number of output units is equal to the number of classes, which is 40, the number of persons in the ORL database.

EXPERIMENTS AND RESULTS

We experimented with the ORL[6] and UMIST[7] face databases using a neural network simulater called EasyNN-Plus[8]. The UMIST university face database consists of 564 images of 20 people, each covering a range of poses from profile to frontal views. Subjects cover a range of race/sex/appearance. Each subject exists in its own directory labeled as 1a, 1b, ... , 1t and the images are numbered consecutively as they were taken. The files are all in *Portable Gray Map* (PGM) format, 220 x 220 pixels in 256 shades of grey. As an example, the different images of face of a person in the directory 1a are shown in Fig.4.

Fig 4. Different images of the subject 1a in UMIST face database.

On the other hand, ORL database contains a set of faces taken between April 1992 and April 1994 at Olivetti Research Laboratory (ORL) in Cambridge, U.K. There are 10 different images of 40 distinct subjects. There are variations in facial expression (open/closed eyes, smiling/non-smiling), and facial details (glasses/no glasses). All the images are against a dark homogeneous background with the subjects in an up-right, frontal position, with tolerance for some tilting and rotation of up to about 20 degrees. There is some variation in scale of up to about 10%. The images are grayscale with a resolution of 92 x 112 pixels. As an example, 10 different images of face of a person are shown in Fig.5.

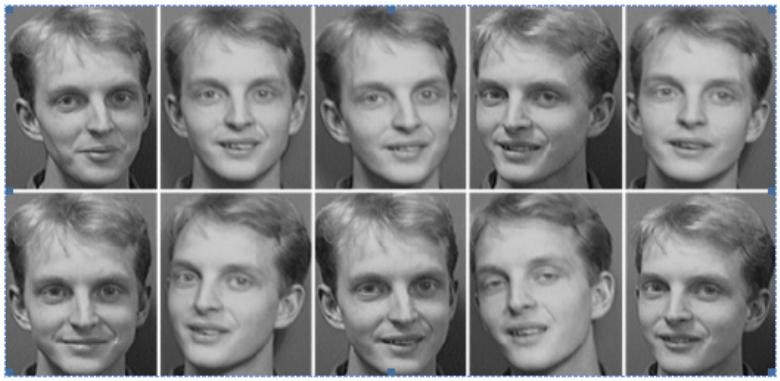

Fig 5. The set of 10 images for a subject in ORL database.

We report here, the recognition accuracies for the different faces of different persons. We obtain the optimal values of different design parameters such as number of training examples, number of hidden layers, number of nodes in hidden layers, learning rate, momentum, number of input nodes and number of output nodes.

Number of Training Examples

It is observed that as the number of training examples are increased the recognition accuracy increases. Fig. 6 shows the recognition accuracy for the first person in the ORL database as the number of faces is increased from 1 to 10.

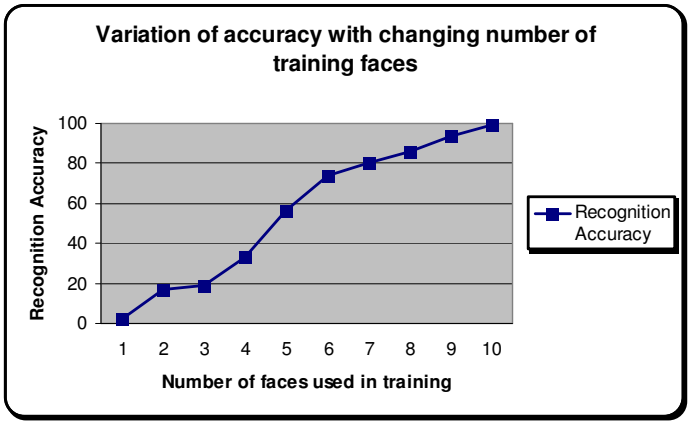

Fig 6. Variation of recognition accuracy as number of training faces is increased.

Number of Hidden Layers

If only one hidden layer is taken, then, the number of training cycles that are required, might be very less, but, the accuracy may not be acceptable. The ORL faces have great variation in terms of presence of glasses, moustaches, beard, female faces, etc. So, one hidden layer is not found to be resulting in correct accuracy. So, we include two hidden layers in the multi-layer perceptron and discover that the accuracy is increased. In the case of UMIST faces, there is not much variation in the faces and the faces differ only slightly from each other in terms of the pose. So only a single hidden layer is found to be sufficient for these faces and it gives excellent results.

Number of Nodes in Each Hidden Layer

The number of nodes in the hidden layer(s) should be such that it is neither too large to result in unnecessary network complexity nor too small to result in slow and inaccurate learning. The criteria to judge the optimum number of nodes is number of training cycles and the recognition accuracy.

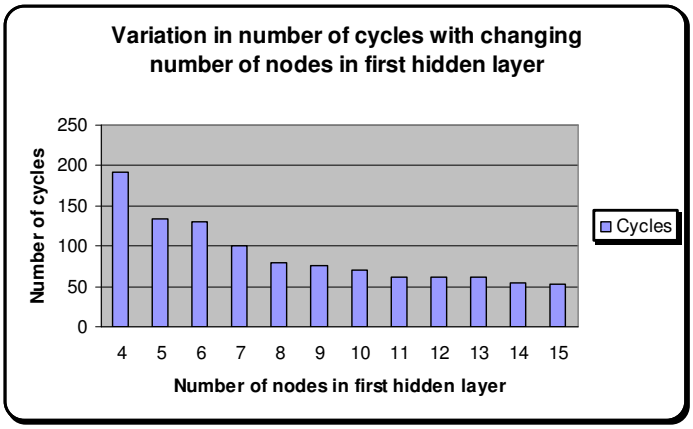

Fig 7. Variation in training cycles as number of nodes in first hidden layer is increased.

Fig. 7 shows that as the number of nodes is increased, lesser number of training cycles is required. But, in order to reduce the network complexity, the optimum number of nodes is taken as 11 and the recognition accuracy for this number of nodes is found to be around 99%.

For the ORL faces, it is observed that a MLP having 16 nodes in the first hidden layer gives good accuracy. As shown in Fig.8, number of training cycles is least, when the second hidden layer has 8 hidden nodes. So, for ORL faces, the neural network are optimized if, we include 16 and 8 nodes in first and second hidden layer, respectively.

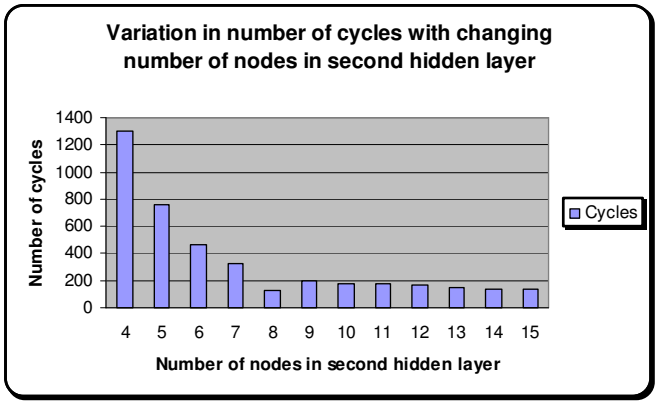

Fig 8. Variation in training cycles as number of nodes in second hidden layer is increased.

Learning Rate and Momentum

The back-propagation algorithm that is used to train the MLP has two very important parameters namely, learning rate and momentum. We vary the learning rate and momentum by carrying out several simulations and discover that these parameters do not play a significant role in the accuracy of the recognition system. But, they have a great bearing on the number of cycles that the system takes to converge and reach the level, where the error is acceptable. If the values of the learning rate and momentum are very large then the neural network shows oscillations and does not converge even after 4200 cycles for a target error of 1%. If on the other hand, the values of learning rate and momentum are very less then the neural network takes a lot of time to converge to the desired error level. When both the learning rate and momentum are kept at 0.1 then the learning takes more than 5000 cycles. Fig. 9 shows the variation of cycles with changing momentum and learning rate. The number of cycles first decreases with increase in these parameters but after the values exceed a certain threshold the number of cycles takes an upward swing.

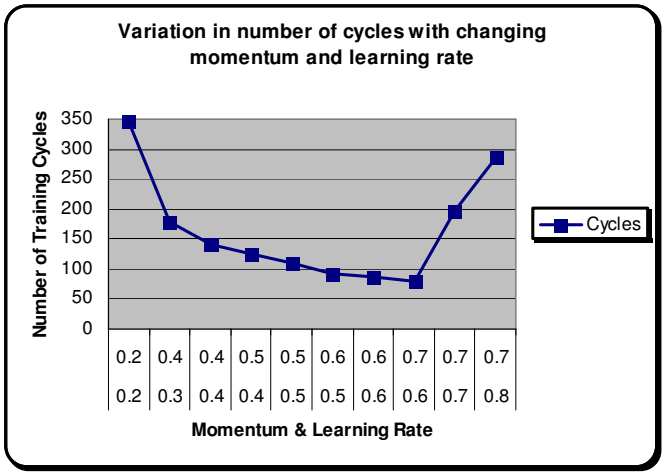

Fig 9. Effect of learning rate and momentum on the number of cycles.

So, the optimal values of learning rate and momentum are selected as follows.

1. For the ORL database, learning rate is taken to be 0.4 and momentum is also taken to be 0.4.
2. For the UMIST database, learning rate is taken to be 0.3 and momentum is taken to be 0.4.

Number of Training Cycles

As it is clear from Fig. 9, the number of training cycle's decreases as the learning rate and momentum are increased. But this happens only up to some threshold values of learning rate and momentum after which, the number of cycles required increases because of the oscillations in the learning process. Fig. 10 and fig. 11 show the effect of increase in the number of nodes in the first hidden layer, on the number of training cycles. It is found that number of training cycles generally decreases with increase in the number of nodes. It is also clear that when the target error is halved from 1% to 0.5%, the number of training cycles roughly doubles. We find that, when the number of hidden layers is increased, the number of cycles needed to converge also increases.

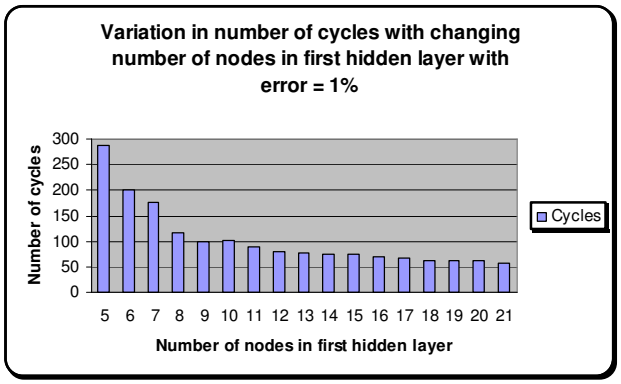

Fig 10. Variation in training cycles as number of nodes in first hidden layer is increased, with target error set to 1%.

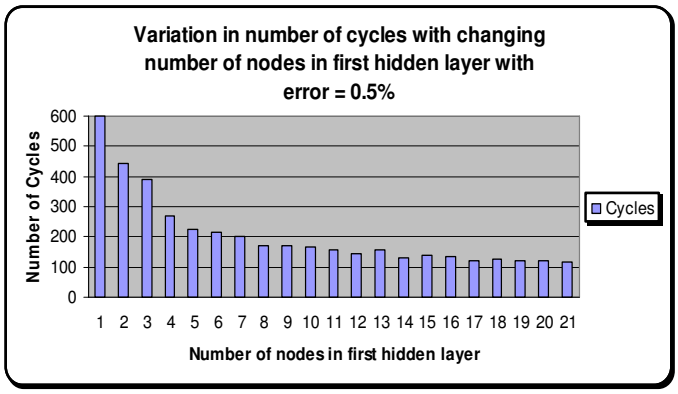

Fig 11. Variation in training cycles as number of nodes in first hidden layer is increased, with target error set to 0.5%.

We list the values of different design parameters for the optimal multi-layer perceptron for ORL and UMIST databases in the table 1 and 2 respectively. From the Table 1 and 2, it is clear that the different parameters get different values for the two different databases, for the same target error. While for ORL database, we require two hidden layers, just one hidden is layer required for face recognition in UMIST database.

Table 1. The optimized neural network for ORL faces

Name of the variable	Value
Number of nodes in input layer	50
Number of nodes in output layer	25
Number of hidden layers	2
Number of nodes in first hidden layer	16
Number of nodes in second hidden layer	8
Learning rate	0.4
Momentum	0.4
Target error	0.5 %

Table 2. The optimized neural network for UMIST faces

Name of the variable	Value
Number of nodes in input layer	40
Number of nodes in output layer	15
Number of hidden layers	1
Number of nodes in first hidden layer	11
Learning rate	0.3
Momentum	0.4
Target error	0.5 %

CONCLUSION

In this paper, we discussed the design of an optimal multi-layer perceptron for eigenfaces based face recognition. We discussed the various design issues and obtained the optimal values of different design parameters. The design of the neural network was discussed in the context of the face recognition on two face database namely ORL and UMIST. Important findings of the study may be summarized as follows.

1. If the number of faces that are used in training of the neural network is increased, the accuracy of recognition generally improves because the system is able to draw better generalizations with the increase in the number of training faces.
2. Learning rate and momentum have a significant effect on the number of training cycles that the neural network takes to converge but does not have much impact on the accuracy of the face recognition system.
3. The number of eigenfaces that are used to represent the actual faces have a significant bearing on the accuracy of the face recognition system. Increasing the number of eigenfaces leads to more inputs to the system thus leading to better learning and accuracy.
4. The number of training cycles that the system needs for convergence also depends on the number of hidden layers and the number of nodes in the hidden layer(s).

REFERENCES

[1] M. Turk and A. Pentland,1991. “Eigenfaces for recognition,” *Journal of Cognitive Neuroscience*, vol. 3, no. 1, pp. 71–86.

[2] F. Rosenblatt, 1958. “The Perceptron: A Probabilistic Model for Information Storage and Organization in the Brain”, *Psychological Review*, vol. 65, issue 6, pp. 386-408.

[3] D. DeMers and G.W. Cottrell, 1993. “Non-linear dimensionality reduction”, In S.J. Hanson, J.D. Cowan, and C. Lee Giles, editors, *Advances in Neural Information Processing Systems,* vol. *5*, Morgan Kaufmann Publishers, San Mateo, CA, pp. 580-587.

[4] Z. Pan, A. G. Rust and H. Bolouri, 2000. “Image Redundancy Reduction for Neural Network Classification using Discrete Cosine Transforms”, *Proceedings of the International Joint Conference on Neural Networks*, vol. 3, pp 149-154.

[5] M. L. Minsky and S. S. Papert,1969. “Perceptrons: An Introduction to Computational Geometry”, MIT Press, Cambridge.

[6] ORL face database, http://www.uk.research.att.com/facedatabase.html

[7] UMIST face database, http://images.ee.umist.ac.uk/danny/database.html

[8] H. A Rowely, 1987. “A Trainable View-based object for face detection”, IEEE ASSP Magzine.

[9] D. Bryliuk and V. Starovoitov. 2002. “Access control by face recognition using neural networks and negative examples”. *Proceedings of the 2nd International Conference on Artificial Intelligence,* Crimea, Ukraine, pp. 428-436, Sep.

[10] EasyNN-Plus, http://www.easynn.com/