Performance Comparison of Three Types of Autoencoder Neural Networks

Chun Chet Tan and C. Eswaran
Centre for Multimedia and Distributed Computing
Faculty of Information Technology
Multimedia University
63100 Cyberjaya, Selangor, Malaysia.
{cctan, eswaran}@mmu.edu.my

Abstract

This paper presents a comparison performance on three types of autoencoders, namely, the traditional autoencoder with Restricted Boltzmann Machine (RBM), the stacked autoencoder without RBM and the stacked autoencoder with RBM. The performances are compared based on the reconstruction error for face images and using the same values for the parameters such as the number of neurons in the hidden layers, the training method, and the learning rate. The results show that the RBM stacked autoencoder gives better performance in terms of the reconstruction error compared to the other two architectures.

1. Introduction

Principal component analysis (PCA) is a widely used method for reducing the dimensionality of data. This statistical method however can only determine linear interdependencies among the components of data vectors. In other words, PCA can only do a linear mapping from a high-dimensional space to a low-dimensional code space. However a neural network with at least one hidden layer can give a non-linear mapping from input layer to output layer. But the traditional neural networks are unable to reduce the dimensionality of data to the same extent as a PCA.

Autoencoder networks are feedforward neural networks which can have more than one hidden layer. These networks attempt to reconstruct the input data at the output layer [1]. The targets at the output layer are the same as the input data, thus the size of the output layer is also the same as the size of the input layer. Autoencoders are supervised neural networks which are trained using a gradient descent method, such as backpropagation. Since the size of the hidden layer in an autoencoder is smaller than the size of the input data, the dimensionality of input data is reduced to a smaller-dimensional code space at the hidden layer. The outputs from the hidden layer are then reconstructed into the original data at the output layer. Like PCA, the autoencoders can give mappings in both directions between the data and the code space.

By using more number of hidden layers in an autoencoder neural network, a high-dimensional input data can be reduced to much smaller code space. However, training a multilayer network with more than one hidden layer is tedious and further it will not also give good results [2]. This is due to the fact that the weights at deep hidden layers are hardly optimized. The time required for training such a network is also extremely high. In this paper, a stacked autoencoder neural network architecture which uses Restricted Boltzmann Machine (RBM) pre-training is proposed. Performance comparison is carried out on three types of autoencoder architectures, namely, the traditional autoencoder with RBM, the stacked autoencoder without RBM and the stacked autoencoder with RBM. The performances are compared based on the reconstruction error for face images.

The remainder of this paper is organized as follows: In Section 2, the architecture of autoencoder is described and in Section 3, the training methods are presented. Section 4 describes the experiments conducted and also the results obtained in the comparison study. Finally, the conclusions are given in Section 5.

2. Architecture

The architecture of a five-layer autoencoder neural network is depicted in Figure 1. An autoencoder is a feedforward neural network with one or more hidden layers that attempts to reconstruct its input activities at its output [1]. Hence the main difference of the autoencoder and the traditional neural network is the

978-0-7695-3136-6/08 $25.00 © 2008 IEEE
DOI 10.1109/AMS.2008.105

size of the output layer. In an autoencoder the size of the output layer is always the same as the size of the input layer. Besides that, the size of the deepest hidden layer in an autoencoder network is always smaller than the sizes of the input and output layers. As shown in Figure 1, the autoencoder network comprises two components namely "encoder" and "decoder". "Encoder" network transforms the high-dimensional input data into a low-dimensional code and the "decoder" network (which is similar to the "encoder" network) reconstructs the original high-dimensional data from the low-dimensional code.

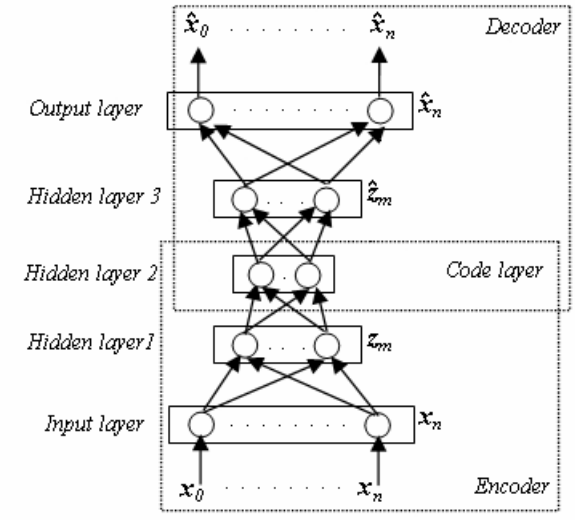


Figure 1. Multilayer autoencoder neural network architecture.

Sigmoid activation functions are normally used in autoencoders for non-linear mapping. If linear activation functions are used, autoencoders will be performing similar to PCA. The networks can be trained by minimizing the mean square error between the original and the reconstructed data. The required gradient is easily obtained by using the chain rule to back propagate the error derivatives first through the decoder network and then through the encoder network [2].

Let the n-inputs and a bias received at a single node in a hidden-layer of autoencoder be (see Figure 2),

$$x_0, x_1, x_2, \ldots, x_n$$

where x_0 is a bias set to -1. The total weighted sum at a hidden node is,

$$net_j = \sum_{i=0}^{n} w_{ij} x_i \qquad (1)$$

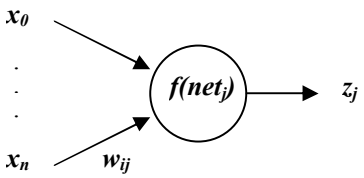
where w_{ij} is the weight of j-th hidden node associated with the input x_i. The output of a hidden node,

$$z_j = f(net_j) \qquad (2)$$

where

$$f(s) = \frac{1}{1+e^{-\beta s}} \qquad (3)$$

is a Sigmoid activation function.



j-th node

Figure 2. A single node in the hidden layer with its inputs, outputs and activation function.

The outputs from the last hidden layer are sent to the output layer as the inputs. The output of the l-th node in the output layer is obtained as

$$\hat{x}_l = f(net_l) \qquad (4)$$

where

$$net_l = \sum_{j=0}^{m} w_{jl} z_j \qquad (5)$$

3. Training Methods

3.1. Backpropagation

The outputs, $\hat{x}$, of the autoencoder are compared with the desired outputs, x, to derive the sum squared error. Connection weights of nodes are adjusted according to the weight changes. For the output layer, the change in the weight is given by [3]

$\Delta w_{jl} = \eta(x_l - \hat{x}_l)\lambda_0 f(net_l)(1 - f(net_l))z_j$ or

$$\Delta w_{jl} = \eta(x_l - \hat{x}_l) f'(net_l) z_j \qquad (6)$$

For hidden layer, the change of weight is given by [3]

$$\Delta w_{ij} = \eta\left(\sum_{l=0}^{n}(x_l - \hat{x}_l) f'(net_l) w_{jl}\right) f'(net_j) x_i \qquad (7)$$

where η is the learning rate.

3.2. Polak Ribiere conjugate gradient backpropagation

Polak Ribiere conjugate gradient (PR) backpropagation is used to train the autoencoders in our study. Conjugate backpropagation is one of the

variances of backpropagation. It speeds up the training process compared to the traditional backpropagation with momentum. Line search function of conjugate is used to locate the minimum point in the error function. The search direction is computed according to the formula:

$$p_k = -g_k + \beta_k p_{k-1} \quad (8)$$

where g_k is the new gradient and p_{k-1} is the previous search direction. The parameter β_k can be computed in different ways. For the Polak Ribiere variation of conjugate gradient, it is computed according to:

$$\beta_k = \frac{\Delta g_{k-1}^T g_k}{g_{k-1} g_{k-1}} \quad (9)$$

where it is a inner product of the previous change in the gradient with the current gradient divided by the norm squared of the previous gradient [4].

3.3. Stacked Autoencoder

Referring to Figure 1, it is difficult to optimize the weights in autoencoders that have multiple hidden layers (>=2). With large initial weights, autoencoders typically find poor local minima; with small initial weights, the gradients in the early layers are tiny, making the training impossible [2].

A method for training the multilayer autoencoders is explained in [5]. In this method, the training is implemented in phases. During the first phase, the autoencoder is assumed to have three layers, namely, input layer ***x***, output layer ***y*** and the hidden layer h_1 as shown in Figure 3(a). The weights W_1 and W'_1 are trained using backpropagation. During the second phase as depicted in Figure 3(b), a separate one-hidden-layer network consisting of input layer h_1, output layer h'_1 and hidden layer h_2 is stacked onto the existing autoencoder of Figure 3(a). The inputs and bias, z_0, to the input layer h_1 are $z_0, z_1, z_2, \ldots, z_m$ where $z_j = f(net_j); j \neq 0; m < n$.

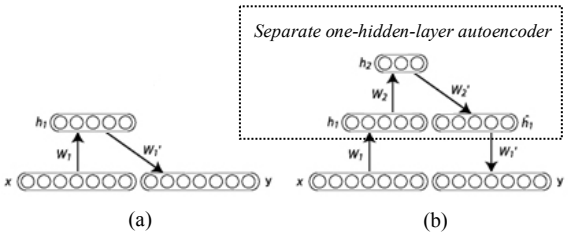


Figure 3. Stacked Autoencoder.

Since the size and the value of input and output layers of an autoencoder are the same, output layer h'_1 is the same as input layer h_1. Hidden layer h_2 is a new hidden layer added onto the autoencoder. The weights of W_2 and W'_2 can be trained by using backpropagation. After the training of the separate one-hidden-layer network is completed, all the weights of the autoencoder are trained together to converge to a global minimum. This kind of autoencoders is called stacked autoassociators [6]. Figure 3 illustrates the iterative training procedure. With this method, a deep autoencoder with more than one single hidden layer can be trained to converge.

3.4. Restricted Boltzmann Machine (RBM)

An alternative method for training the multilayer autoencoders was proposed by Hinton et al in [2]. All the weights of the multilayer autoencoder can be trained in a single phase provided the weights are pre-trained with RBM. Since the weights are close to good solution after RBM pre-training, backpropagation works well for fine-tuning the weights in the multilayer autoencoders.

The RBM can be modeled as a two-layer network in which the energy function of pixels connected to feature detectors is given by

$$E(v,h) = -\sum_{i \in pixels} v_i b_i - \sum_{j \in features} b_j h_j - \sum_{i,j} v_i h_j w_{ij} \quad (10)$$

where v_i and h_j are the binary states of pixel ***i*** and feature ***j***, b_i and b_j are the biases respectively and w_{ij} is the weight between them. Given a training image, h_j of each feature detector ***j*** is set to 1 with probability $p(b_j + \sum_i v_i w_{ij})$ where

$$prob(x_i = 1) = 1/[1 + \exp(-x_i)] \quad (11)$$

is a logistic function [2]. The network eventually will reach a Boltzmann distribution in which the probability of the equilibrium state, ***v***, is determined solely by the "energy" of that state vector relative to the energy of all possible binary state vectors [2] as shown below

$$P(v) = \exp(-E(v,h)) / \sum \exp(-E(v',h')) \quad (12)$$

The change in weight can be derived as

$$\Delta w_{ij} = \varepsilon \langle \partial \log P(v) / \partial w_{ij} \rangle$$
$$= \varepsilon (\langle v_i h_j \rangle_{data} - \langle v_i h_j \rangle_{\text{model}}) \quad (13)$$

where ε is the learning rate, $\langle . \rangle_{data}$ is an expected value in the data distribution and $\langle . \rangle_{model}$ is an expected value of Boltzmann sampling state vectors from its equilibrium distribution at temperature 1 [7][8]. The temperature will be increased as the learning converges. The same learning rule is applied to the biases, b_i and b_j.

After pre-training with RBM, backpropagation can be used for fine-tuning the weights in multilayer autoencoders provided the initial weights and biases are close to optimum solution.

3.5. Stacked Autoencoder with RBM pre-training

This architecture uses RBM pre-training for each hidden layer. Starting from a traditional one-hidden layer autoencoder, the weights are initialized and pre-trained with RBM. Then the weights are fine-tuned with PR backpropagation. The outputs from the hidden layer are used as the inputs for the next autoencoder in the stack. The same training process is carried out for the new separate one-hidden layer autoencoder to be stacked onto the existing trained autoencoder. The resulting autoencoder has a smaller number of hidden layer neurons, which leads to dimensionality reduction. After stacking the new autoencoder, PR backpropagation is applied to fine tune the overall weights. This process is repeated for stacking more hidden layers and for constructing a 'slimmer-waist' autoencoder.

4. Experiments and Results

In this section, we describe the experiments conducted to compare three models. The test images used are ORL face dataset [9]. This dataset contains 400 images of size 112 X 92 for forty different people with 10 images for each person. The images are rescaled to size 37 X 30 using nearest neighbor interpolation. The intensities of the images are scaled to make the mean and the pixel variance values to be 0 and 1 respectively. The dataset is then divided into 200 training images, which contain 5 images of each person, and 200 testing images, the remaining 5 images of each person.

The three architectures are compared using a layer size: (37 X 30)-500-300-100-30-100-300-500-1110. For training the networks, PR backpropagation method is used. The total number of training epochs employed is 230. For the two architectures which use RBM, the networks are pre-trained with RBM for 200 epochs for each hidden layer. For stacked autoencoder without RBM, the network is initialized with small random weights. In the case of stacked autoencoders, starting from a conventional one-hidden-layer network as shown in Figure 3(a), the networks are trained for 50 epochs. In the next step, the outputs from the hidden neurons are assumed to be the inputs for the new separate one-hidden-layer autoencoder stacked on the top of the trained autoencoder, as shown in Figure 3(b). The new autoencoder is then trained by using the same method of PR backpropagation for 10 epochs. In the next phase, the whole network with three-hidden layers is again trained for 50 epochs. This process is iterated to stack more hidden layers onto the autoencoders. Every new separate one-hidden-layer autoencoder is undergoing training for 10 epochs. For the whole network after each stacking, 50 epochs of training is carried out. Thus a total number of 230 epochs are used in the training process.

The plot of MSE VS epochs shown in Figure 4 reveals that the RBM pre-trains the autoencoder effectively as seen by the convergence of MSE during the training phase.

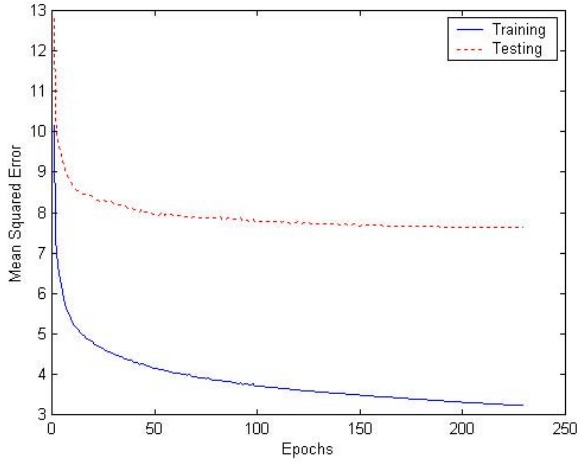


Figure 4. MSE of traditional autoencoder with RBM.

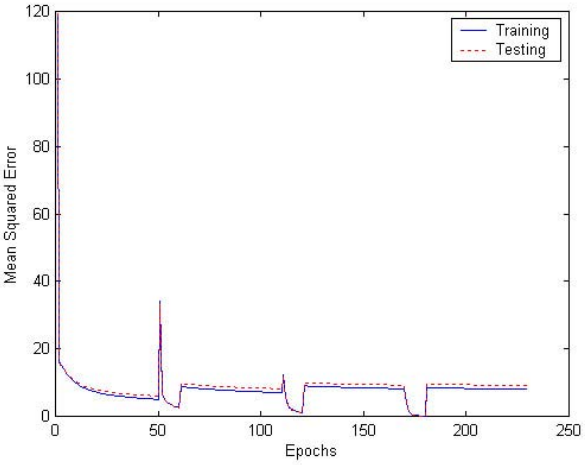


Figure 5a. MSE of stacked autoencoder without RBM.

Figure 5a shows the test results obtained for stacked autoencoder without RBM. From Figure 5a, it is clear that the MSE for both training and testing phases are extremely high at the beginning and lead to slow

convergence at the end. Figure 5b shows the zoom-in view of the results shown in Figure 5a. From this figure, we observe that this model has very low difference between the training and testing MSEs.

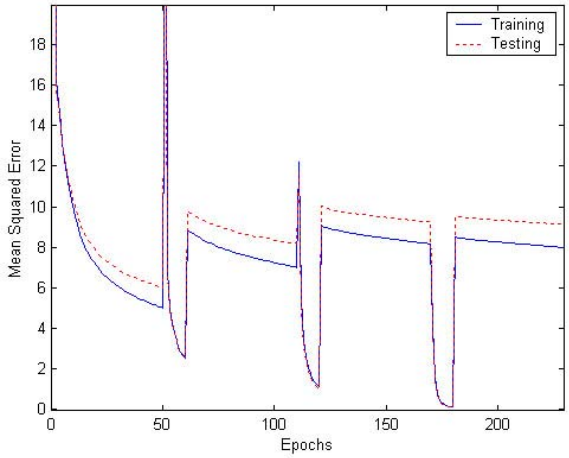


Figure 5b. MSE of stacked autoencoder without RBM (zoom-in view).

Figure 6 shows the performance of the stacked autoencoder with RBM pre-training. It is clear from this figure that the RBM has proven to be a good pre-training method as it leads to a faster convergence compared to the stacked autoencoder without RBM. For the same number of training epochs, it is found that the autoencoder with RBM has the minimum testing MSE among all the architectures. One important phenomenon observed from these experiments is that the RBM pre-training tends to make the network more focused on the training dataset, resulting in a low generalization. Due to this, we notice from Figs.4 and 6 that the difference between the training and testing MSEs is higher in these two architectures compared to the one without RBM.

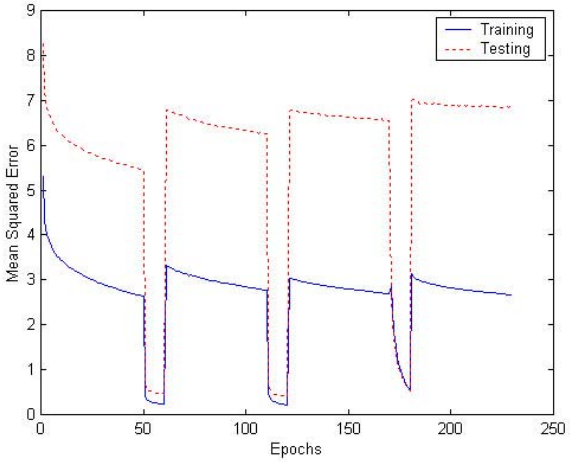


Figure 6. MSE of stacked autoencoder with RBM.

We also note from Figs.5 and 6 that the MSEs become worse when a new hidden layer is stacked on to the existing autoencoder. This means that the dimensionality reduction of autoencoders occurs at the expense of reconstruction efficiency as shown by the increase in the value of MSEs with each stacking. Table 1 shows the testing MSEs obtained for the three architectures. From the results of Table 1, we can conclude that the stacked autoencoder with RBM outperforms the other two architectures with respect to the reconstruction error.

Table 1. MSE of testing phase after 230 epochs.

	Testing MSE
Traditional Autoencoder with RBM	7.6178
Stacked Autoencoder without RBM	9.1092
Stacked Autoencoder with RBM	6.8483

5. Conclusions

We have compared the performances of three types of autoencoders. The experiments conducted for evaluating the performances have been described in detail. From the results it is found that stacked autoencoder with RBM has the lowest reconstruction error compared to the other two architectures.

6. References

[1] Geoffrey E. Hinton, Peter Dayan, and Michael Revow, "Modeling the Manifolds of Images of Handwritten Digits", *IEEE TRANSACTIONS ON NEURAL NETWORKS, VOL. 8, NO. 1,* 1997, pp. 1045–9227.

[2] G. E. Hinton and R. R. Salakhutdinov, "Reducing the Dimensionality of Data with Neural Networks", *Science Vol 313,* 2006, pp. 504–507.

[3] Willi-Hans Steeb, The Nonlinear Workbook 2nd Edition, World Scientific, 2002, pp. 417–418.

[4] Neural Network Toolbox. (No date). Polak-Ribiere Update (traincgp). [Online]. The MathWorks. < http://www.mathworks.com/access/helpdesk/help/toolbox/nnet/index.html?/access/helpdesk/help/toolbox/nnet/backpr14.html>

[5] Bengio, Y., Lamblin, P., Popovici, D., & Larochelle, H. "Greedy layer-wise training of deep networks", *Advances in Neural Information Processing Systems 19, MIT Press*, 2007.

[6] Hugo Larochelle, Dumitru Erhan, Aaron Courville, James Bergstra and Yoshua Bengio. "An Empirical Evaluation of Deep Architectures on Problems with Many Factors of Variation", to appear in *Proceedings of the 24 th*

International Conference on Machine Learning (ICLM), 2007.

[7] Ruslan Salakhutdinov, Andriy Mnih, Geoffrey Hinton, "Restricted Boltzmann Machines for Collaborative Filtering", *Proceedings of the 24th International Conference on Machine Learning*, 2007.

[8] Yee Whye Tee and Geoffrey E. Hinton, "Rate-coded Restricted Boltzmann Machines for Face Recognition", *Advances in Neural Information Processing Systems 13, MIT Press*, 2001.

[9] The Database of Faces. (2002). [Online]. The ORL Database of Faces. AT&T Laboratories Cambridge. <http://www.cl.cam.ac.uk/research/dtg/attarchive/facedatabase.html>