Artificial Neural Networks
Report on Exercise Sessions and Projects

Laavanya Wudali Lakshmi Narsu (r0690809)
Master in Artificial Intelligence

June 2018

Contents

Chapter 1

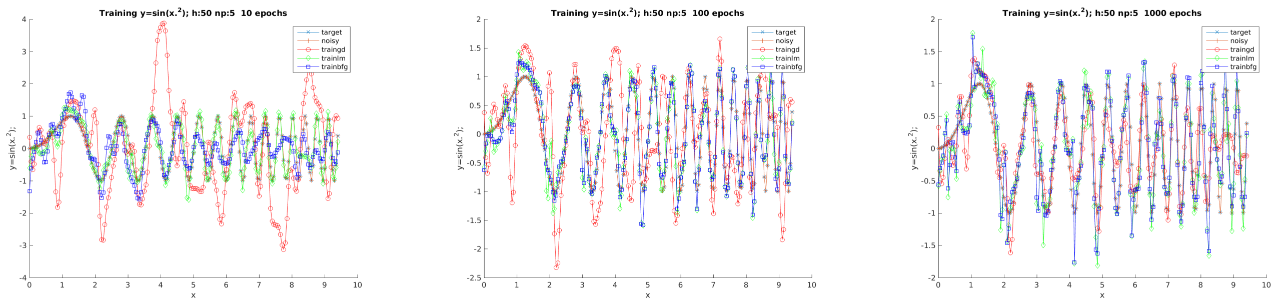
Exercise Session 1 - Supervised Learning and Generalization

In this exercise, we compare LM Algorithm, BFGS Algorithm and Gradient Descent Algorithm on their convergence rate and fit accuracy. We expect that LM Algorithm (second order) will converge fastest, followed by BFGS(quasi-second order), finally gradient descent(first order).

1.1 Approximating noise-free data

The nonlinear function used here is. $y = sin(x^2)$. It was observed that the LM algorithm is able to fit the training set well in just 10 epochs. It takes 100 epochs for BFGS to fit the training set. After 1000 epochs, gradient descent has not managed to fit the training set yet. This is in line with the theory, by which a second order method (LM) should be faster in convergence than a first order method(gradient descent). The same was observed even on the test set.

1.2 Generalization on Noisy Data



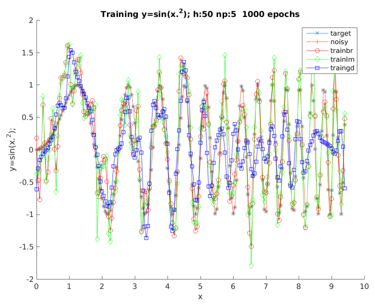
(a) 10 Epoch (b) 100 Epoch (c) 1000 Epoch

Figure 1.1: Comparison of algorithms on Noisy Function $y = sin(x^2)$

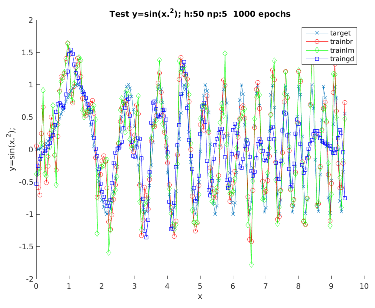
In this set of experiments, gaussian noise is added to the training set. It can be observed from Figs. 1.1 that the convergence behaviour of the three algorithms are similar. However it can be noticed that as the number of epochs increase, the error on the test set by the LM algorithm increases, because at this point the algorithm has started to overfit, also fitting the noise. This confirms the intuition behind early stopping.

1.3 Bayesian Neural Network Training

In this experiment, the bayesian neural network training algorithm is applied to the function $y = sin(x^2)$. In this set of experiments, gaussian noise of zero main and unit variance with amplitude equal to 5% of the RMSE power of the function is added to the training set. This training algorithm is compared to the steepest descent and the Levenberg Marquardt algroithms. Fig. 1.2 shows the results on the training and test sets. It can be observed that the convergence properties of LM and Bayesian are similar, this is because the Bayesian algorithm uses the LM as a base and applies regularization. From Fig. 1.2b it can be seen that at higher epochs, the regularization applied by the Bayesian approach improves generalization, indicating that it is an alternative to early stopping.



(a) 1000 Epoch Train

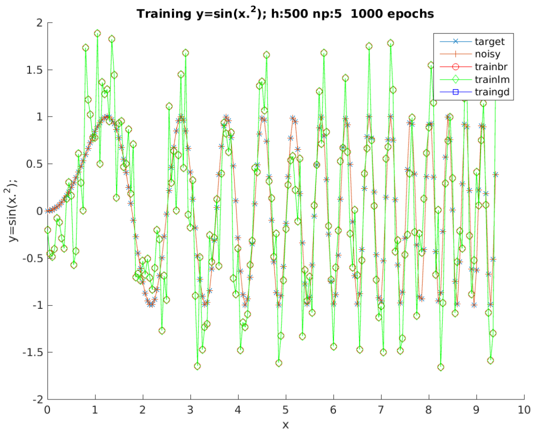


(b) 1000 Epoch Test

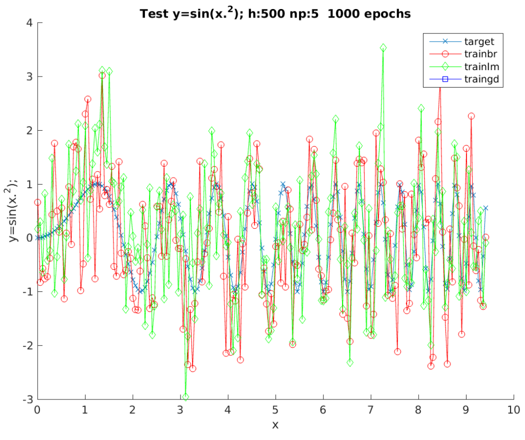
Figure 1.2: Comparison of algorithms on Noisy Function $y = sin(x^2)$

1.3.1 Overparametrized Networks

In this set of experiments, the same noisy function $y = sin(x^2)$ is used to train a larger network with 500 hidden layers. It can be noticed from Fig. 1.3a that the LM method has started fitting the noise,



(a) Training



(b) Test

Figure 1.3: Comparison of LM and Bayesian approach - 500 hidden neurons, 1000 epochs.

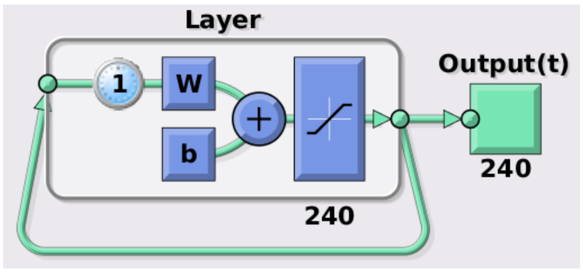
whereas the Bayesian approach fits the target well. In the case of the test set, the Bayesian approach performs better than LM, but still it can be seen that more regularization is needed.

Chapter 2

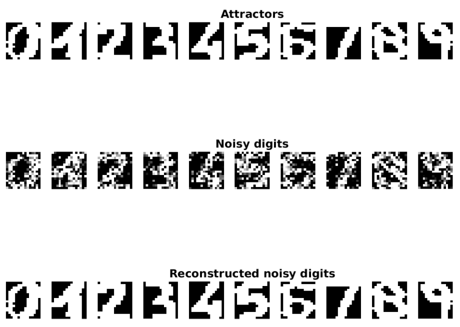
Exercise Session 2 - Recurrent Neural Networks

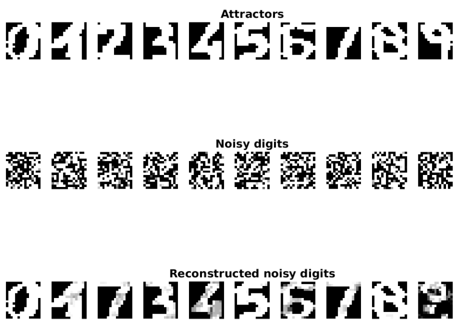
2.1 Hopfield Network on Handwritten Digits

The Hopfield network is a one-layer fully connected recurrent network with saturating linear activation functions. It can store binary patterns($\in [-1, 1]$) as attractor states) of its energy function $E = \frac{1}{2}\sum_{i,j} w_{ij}s_i s_j$. The weights are adapted using the Hebbian rule $w_{ij} = \sum_{i=1}^{p} s_i s_j$ to store a set of binary patterns. When the network is simulated with a certain input, it will converge to the nearest attractor state, this can be used to recover from noisy patterns. A simulation consists of an asynchronous operation over all the neurons of the network, where each neuron computes a weighted sum of its inputs and takes state 1 if the sum is non-negative, or -1 otherwise.



(a) Noisefactor 1





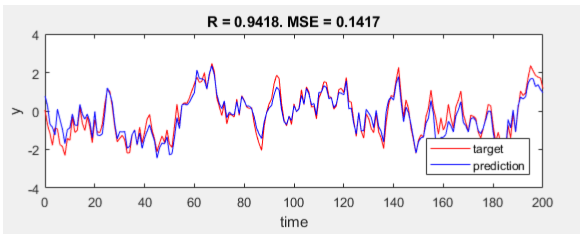
(b) Noisefactor 1

(c) Noisefactor 5

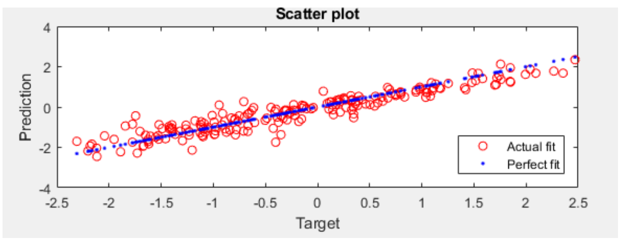
Figure 2.1: Hopfield Network on Handwritten Data

In this exercise the hopfield network is used to reconstruct noisy handwritten digits. In this case 10 different digits, each of size 15x16 is stored in a Hopfield network of 240 neurons(Fig. 2.1a). Noise is then added to the digits and this noisy input is fed in as the initial state and the network is simulated. The resulting attractor states on convergence should be the original digits. This is the case in the case of low noise levels (Fig. 2.1 b). At higher noise levels, it may not be possible to reconstruct the digits and a spurious attractor state may be reached as in the case of the digit 9(Fig. 2.1 c). Further, the wrong digit can also be recollected as in the case of a noisy 2 being reconstructed as a 7 (Fig. 2.1 c).

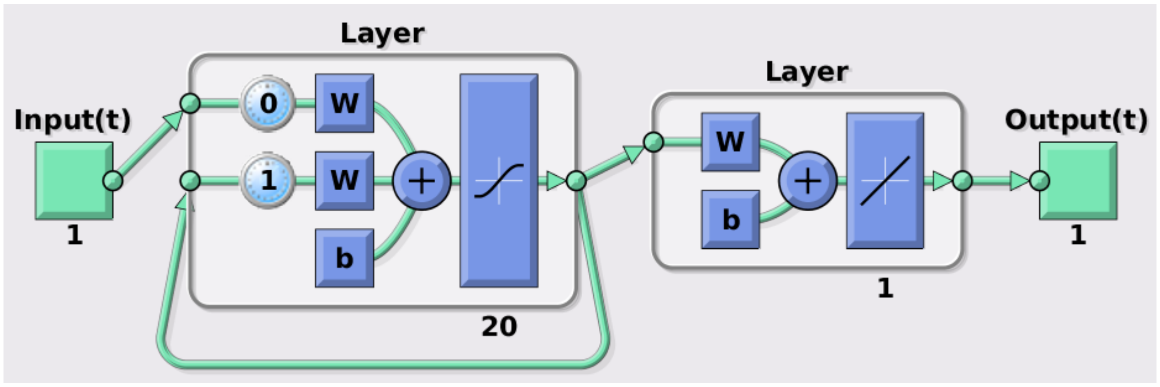
2.2 Elman Network and Hammerstein System Time Series



(a) Hits



(b) Correlation Plot



(c) Network Architecture

Figure 2.2: Elman Network on Hammerstein System Data

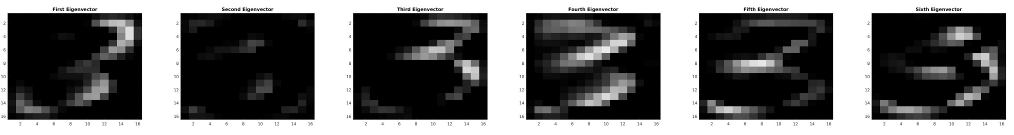
An elman network is a multilayer recurrent neural network where the hidden layer output is fed back as input (Fig. 2.2a). Thus the network can remember its last state and be used in sequence/time series modeling. In this exercise the network is trained on Hammerstein system time series data. The resulting output/predictions are shown in Fig. 2.2b. The predictions are close to the original values with a correlation coefficient of $R = 0.9418$ (Fig. 2.2c).

Chapter 3

Exercise Session 3 - Unsupervised Learning (PCA and SOM)

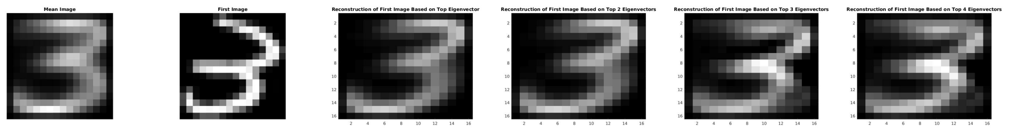
3.1 Principal Component Analysis on Handwritten Digits

PCA is performed on threes extracted US Post Handwritten Digits dataset. The Eigenvalue plot, variance fraction plots, the reconstruction error plots are plotted in Fig. 3.3a,b,c. This is in line with the theory that the magnitude of each eigenvalue gives the variance captured by the corresponding PC. At $n = 256$ we get perfect reconstruction as all the variance in the data is explained. Fig. 3.1 shows what each eigenvector captures. Fig. 3.2 shows the reconstructions with an increasing number of PC's.



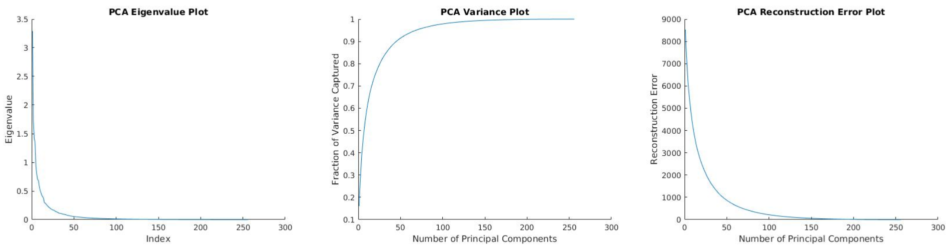
(a) 1st (b) 2nd (c) 3rd (d) 4th (e) 5th (f) 6th

Figure 3.1: Six largest eigenvalues (Intensities multiplied by 5)


Mean Image
First Image
Reconstruction of First Image Based on Top Eigenvector
Reconstruction of First Image Based on Top 2 Eigenvectors
Reconstruction of First Image Based on Top 3 Eigenvectors
Reconstruction of First Image Based on Top 4 Eigenvectors

(a) Mean (b) Original (c) 1 PC (d) 2 PC's (e) 3 PC's (f) 4 PC's

Figure 3.2: Reconstruction of the first 3 using principal components



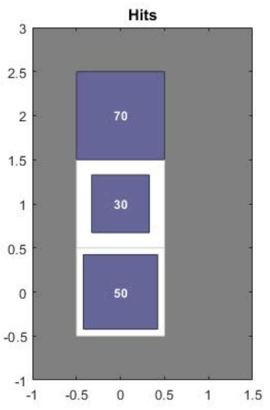
(a) Eigenvalues (b) V.F (c) R.E

Figure 3.3: Eigenvalues, Variance and Reconstruction Errors

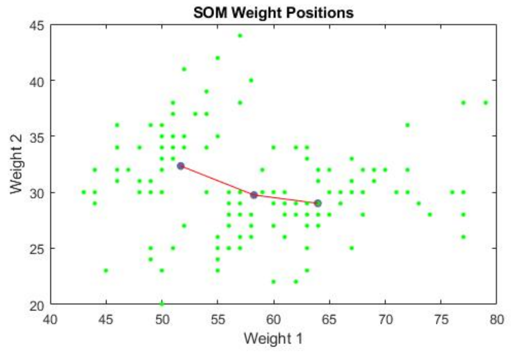
3.2 Self Organizing Feature Maps on Concentric Cylinders and Iris

3.2.1 SOM based clustering on Iris dataset

The Iris dataset has three data classes of 50 points each. In this experiment, a SOM with 3 neurons is fit to this dataset to see if a good clustering can be obtained using SOM.The obtained ARI is 0.66 < 1, implying it is not a perfect clustering. Fig. 3.4 shows the neurons in weight space (1st two principal components). It can be seen that one cluster separates well but the other two do not. The neuron hits show that one neuron has exactly 50 hits, which corresponds to one cluster. The other two neurons show 70 and 30 hits.



(a) Hits



(b) Weight Space

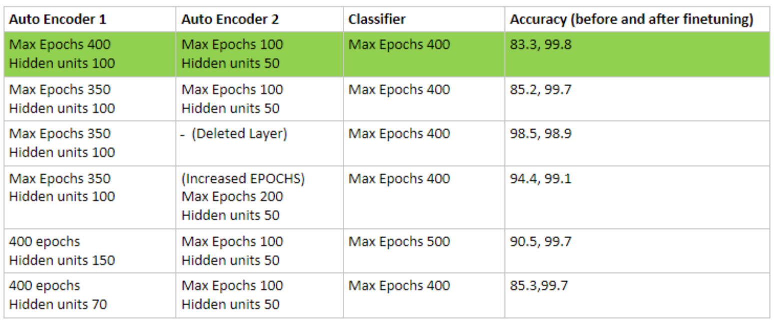
Figure 3.4: SOM on Iris

Chapter 4

Exercise Session 4 - Deep learning: Stacked Autoencoders and Convolutional Neural Networks

4.1 Digit Classification with Stacked Autoencoders

In this experiment, we classify Digits using a Stacked Autoencoder and investigate the effect of different parameters on the classification accuracy. Fig. 4.1 shows the different changes made to the hyperparameters and the resulting accuracies. 99.8 was the highest achievable accuracy with the default parameters. It was observed that even one layer of an autoencoder gives a better performance than a conventional neural network.The pre-training phase initializes the weights of each layer in an efficient manner which helps the Fine-tuning phase to start much closer to a good minima than random weight initialization. The Fine-tuning stage treats all layers of the stacked autoencoder as a single model, and improves on all the weights simultaneously which increases the accuracy of the model.



Auto Encoder 1	Auto Encoder 2	Classifier	Accuracy (before and after finetuning)
Max Epochs 400 Hidden units 100	Max Epochs 100 Hidden units 50	Max Epochs 400	83.3, 99.8
Max Epochs 350 Hidden units 100	Max Epochs 100 Hidden units 50	Max Epochs 400	85.2, 99.7
Max Epochs 350 Hidden units 100	- (Deleted Layer)	Max Epochs 400	98.5, 98.9
Max Epochs 350 Hidden units 100	(Increased EPOCHS) Max Epochs 200 Hidden units 50	Max Epochs 400	94.4, 99.1
400 epochs Hidden units 150	Max Epochs 100 Hidden units 50	Max Epochs 500	90.5, 99.7
400 epochs Hidden units 70	Max Epochs 100 Hidden units 50	Max Epochs 400	85.3,99.7

Figure 4.1: Experiments on Stacked AutoEncoder

4.2 Convolutional Neural Networks

In this exercise, a pre-trained CNN (AlexNet) is used to classify images as either an airplane, a ferry or a laptop. The input layer accepts images of 227px * 227px. The images must contain 3 color channels. The second layer is a convolutional layer with weights of size 11*11*3*96. This means there are 96 convolutional masks of size 11*11*3(where 3 corresponds to the three color channels). Each of these masks will be convolved with the image using a stride ([4 4] in this case). Every step in the convolution results in a single value that is assigned to the pixel in the output of the layer. Since there is no stride value for depth, these are presumably compressed into one dimension. The second layer outputs 96 2D images(one for each mask) with sizes of approximately 227/4 * 227/4 ≈ 55*55.

Layers 3 (ReLu) and 4 (normalisation) do not change these dimensions.

Layer 5 is a max pooling layer of size 3*3 using stride [2 2]. This means the 96 images are halved in each dimension once more to 27*27.

The last fully connected layer consists of 1000 neurons since this network was pre-trained to classify images in 1000 different classes. However, in our case, only three are really used. Regardless, this is a huge reduction considering that the problem started from images with dimensionality 227*227*3 = 154587 pixels using 8 bits per pixel.

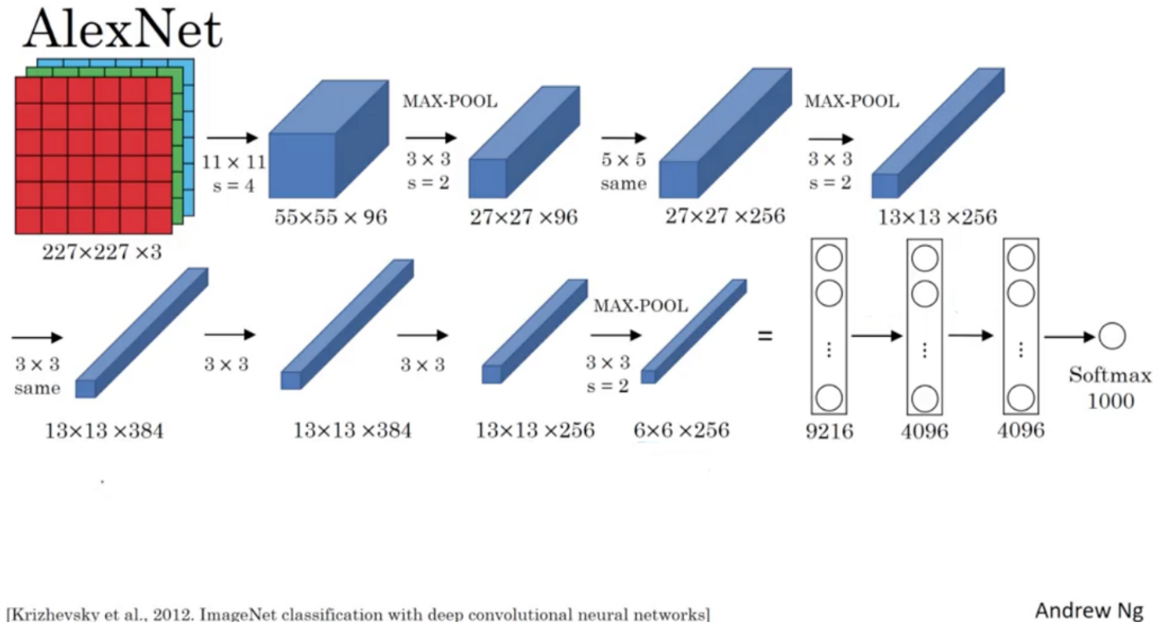


Figure 4.2: Convolutional Neural Network - AlexNet

4.3 Answers to Exercise 2.2

1.The weights of the Convolutional Layer represent the filters that are tuned to extract the right features from the input in order to help with the classification.

2.The dimensions after the first convolutional layer (96 filters with dim 11*11*3 and stride 4) are 55*55*96. Further, the dimensions again change after the 5th layer (Max Pooling 3*3 filter with stride 2) to 27*27*96.

3.The final dimension of the problem used in the Classification task is 1000. This indicates the number of classes the input can be classified to. However, in this case we use only three of these classes.

Chapter 5

Project Works

5.1 Problem 1: Nonlinear Regression and Classification with MLPs

5.1.1 Nonlinear Regression

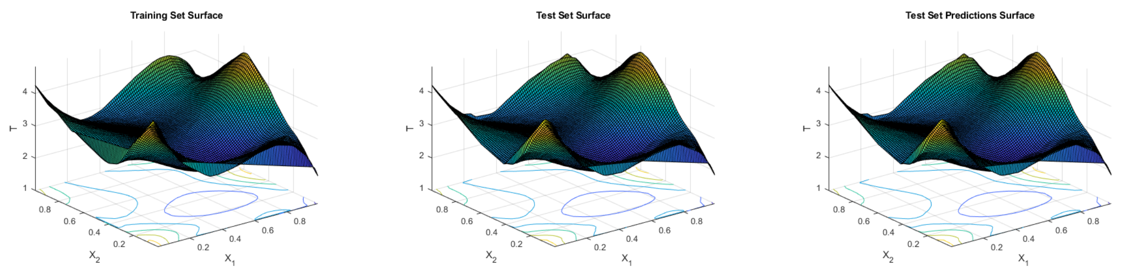
In this project, the task is to design and tune a feedforward neural network for regression in order to estimate an unknown nonlinear function of two variables.

Data Definition

The nonlinear function $T(X_1, X_2)$ for this task is constructed as a linear combination of 5 unknown nonlinear functions. There are 13600 points sampled from these unknown functions $T1$, $T2$, $T3$, $T4$ and $T5$. The weights for the combination are determined by the top 5 digits of my student number - $9, 9, 8, 6, 0$ (from $r0690809$ in descending order). It is normalized by the sum of the 5 values (32). The function is -

$$T = \frac{1}{32}(9T1 + 9T2 + 8T3 + 6T4). \tag{5.1}$$

Thus a dataset of matrices X and y of size 13600×2 and 13600×1 respectively are obtained, these are used for the rest of this task. From this dataset, 3 sets of size 1000 points are sampled randomly with replacement (independent). These are used as the training set, validation set and test set. As they are independent samples, the assignment of these sets as training, validation or test can be arbitrary. The surface of the training, validation and test sets are plotted in Fig. 5.1.



(a) Training (b) Test (c) Predictions on Test Surface

Figure 5.1: Surfaces

Network Architectures

Design Choices:

- **Number of Hidden Layers and Units:** The number of hidden layers is set to 1 owing the simplicity of training. Three different values 10, 50 and 100 are tested and the best determined using the validation set.
- **Hidden Layer and Output Activation Function:** The choices available here are *tansig*, *logsig* and *purelin*. *purelin*, a linear function cannot be used as the multilayer neural network universal approximation theorem will not hold. The hyperbolic tangent sigmoid is preferred here as it has been shown to lead to faster convergence when compared to the logistic sigmoid. Here *purelin* is used as the output activation function as the task is a regression task.
- **Input and Output Preprocessing:** Here the input is scaled such that the min and max values are -1 and 1 respectively. It can be observed from the data(Fig. 5.1) that the scales of the two independent variables are similar, thus standardizing is not necessary. The min max scaling is a simpler approach to use and is preferred here.
- **Training Algorithm:** The Levenberg-Marquardt (trainlm) algorithm is used. The dataset being of small size(1000) means that it is not necessary to opt for conjugate gradient methods. quasi newton methods again are not necessary as the Hessian can be computed, stored in memory and inverted. In such conditions, trainlm, which uses the inverse of the Hessian and is known to converge faster than steepest descent (traingd) is apt for this problem.

Results

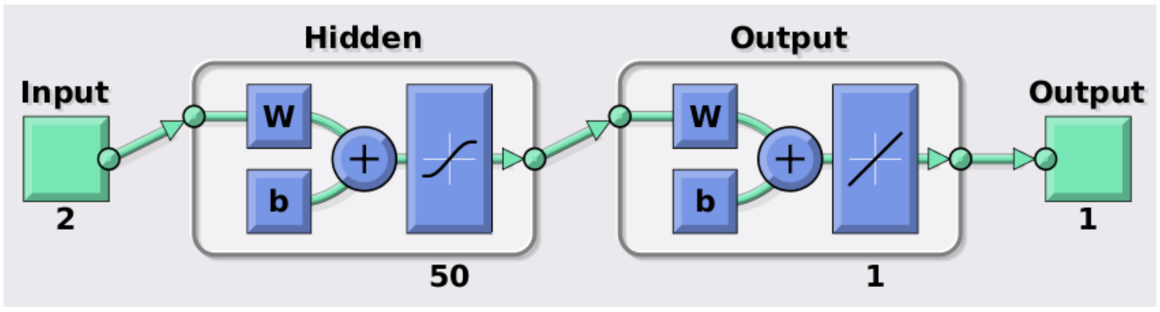
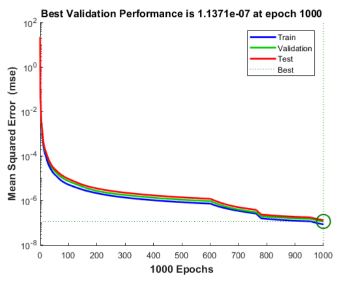
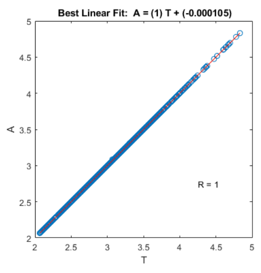


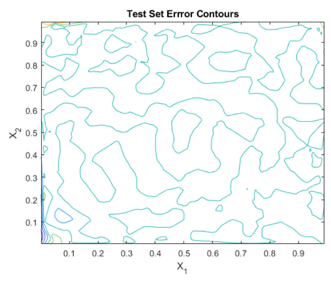
Figure 5.2: Final Network Architecture



(a) Performance plot



(b) Test Set Prediction Correlations



(c) Test Error Contours

Figure 5.3: Results

The final architecture is shown in Fig. 5.2. From the Performance Plot, it is visible that the MSE of both Test and Validation data are almost 0. We can also see that the predictions surface closely approximates the test set surface. (Fig. 5.1b,c). The correlations between the test set ground truth and the test predictions are found to be perfect $R = 1$ (Fig. 5.3(b)). The test error contours are plotted in Fig. 5.3c, and most of the values are near 0. As the results are nearly perfect, no improvements may be needed.

5.1.2 Classification

In this task a feedforward network is used to perform binary classification on the wine dataset. The class combinations are according to the student number((r069080)9). In my case, the positive class was 5 and the negative class was 6 , 7 and 8.

Task 1: Classification

In the first task a feedforward neural net has to be designed for this classification task.

Network Architectures

Design Choices:

- **Number of Hidden Layers and Units:** The number of hidden layers is set to 1 owing the simplicity of training. Three different values 10, 50 and 100 are tested and the best determined using the validation set.
- **Hidden Layer and Output Activation Function:** The hyperbolic tangent sigmoid is used here as the hidden layer actiation function. Here the logsig is used as the output activation function as the task is a binary classification task.
- **Input and Output Preprocessing:** Here the input is scaled such that the min and max values are -1 and 1 respectively.
- **Training Algorithm:** The Levenberg-Marquardt (trainlm) algorithm is used.

Results:

The final architecture is shown in Fig. 5.5a. The hyperparameters, Correct Classification Rate (CCR) on the test set are tabulated in Table. 5.1. The CCR on the test set is found to be 0.7174.

Task 2: PCA Dimensionality Reduction

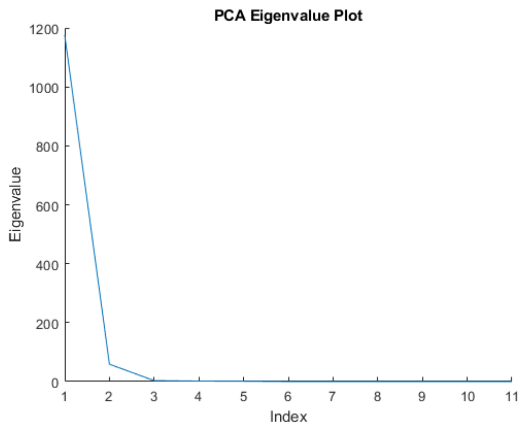
In this section Principal Component Analysis (PCA) is applied in order to reduce dimensionality. Looking at the plots in Fig. 5.4, particularly 5.4b, it can be seen that the first two components already account for $> 95\%$ of the variance.

Task 3: Combining PCA and the neural network

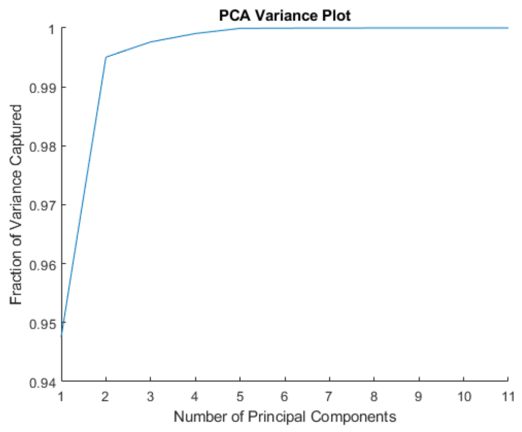
In this experiment, the features output from PCA are used as input to the neural network. The architecture is the same as in the first task. This time, apart from the number of hidden units, the number of principal components can be determined using the validation set.

Results:

The results of the PCA + Neural Net is tabulated in table 5.1 along with the tuned hyperparameters. It is found that the accuracy in the test set improves slightly from 0.717 to 0.726. Interestingly the number of components was found to be 7, which is higher than 2, which would have been chosen based in case the variance captured was used to select the number of components.

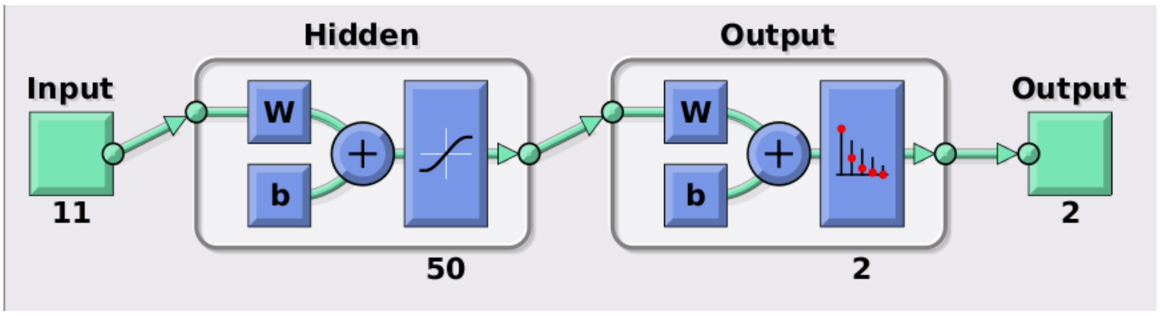


(a) Eigenvalues



(b) Variance Explained

Figure 5.4: Dimensionality Reduction using PCA



(a) Without PCA

Figure 5.5: Final Network Architecture for Classification

Case	**Hyperparameters**	CCR_{val}	CC_{test}
Classification	hidden units = 50	0.7351	0.7174
PCA+Classification	hidden units = 50, principal components = 7	0.7454	0.7266

Table 5.1: Classification Results

5.2 Problem 2: character recognition with Hopfield networks

This project studies the retrieval capabilities of Hopfield networks in the context of character recognition. It consists of three parts (tasks) that aim at retrieving stored characters given distorted characters as input.

5.2.1 Data

For the three tasks, a set of 36 characters including all uppercase characters and the lowercase characters l, a, v, n, y (from Laavanya) are used. Each of these characters are of size 7×5. This is shown in Fig. . These characters are reshaped to a 35 length vector which are used as patterns that can be stored in a Hopfield net.

5.2.2 Task 1 - 5-character experiment

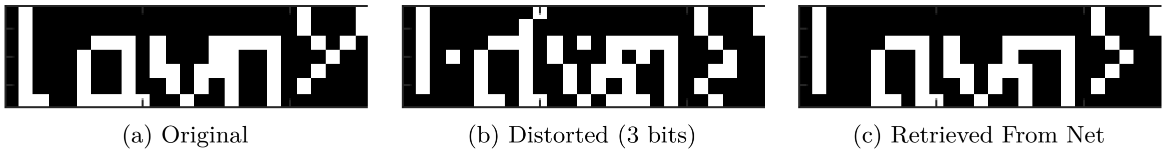
In this task the Hopfield network of $7 \times 5 = 35$ neurons is used to store the 5 patterns. Then distorted (3 bits at random) versions of the first 5 lowercase characters l, a, v, n, y are fed into the network in order to find the correct attractor state pattern, The results of this are shown in Fig. 5.6

Recall

It can be seen from the figure that only character v is retrieved without any error and all other characters have one or two pixels missing.

Spurious Patterns

The character l is retrieved with two pixels missing, a is retrieved with one pixel missing, n is retrieved with one pixel missing and y is also retrieved with one pixel missing. All these are spurious patterns.



(a) Original (b) Distorted (3 bits) (c) Retrieved From Net

Figure 5.6: Experiment using the first 5 characters on a 35 node Hopfield Network. Recollection error = 5 pixels, Total Distortion = 15 pixels

5.2.3 Task 2 - Storage capacity experiment

In this task, the relation between the number of errors and the number of patterns stored in a 35 node Hopfield net are studied. The number of patterns stored are increased from 1 to 36 with a step of 1. For each step the distorted versions of the pattern stored in the network in that step are fed into the network and the recollected patterns are evaluated w.r.t the original patterns in order to evaluate the error. This is shown in Fig. 5.7. It can be seen from the figure that the error increases as the number of patterns stored increase, which is expected.

Critical Loading Capacity

The network is able to recall up to 2 patterns correctly (Fig.5.7).

Theoretical Capacity

The estimated critical loading capacity is in line with the theoretical limit (green line in Fig, 5.7) $\frac{N}{4logN} = \frac{35}{4log35} = 2.46$, where $N = 35$ is the number of neurons.

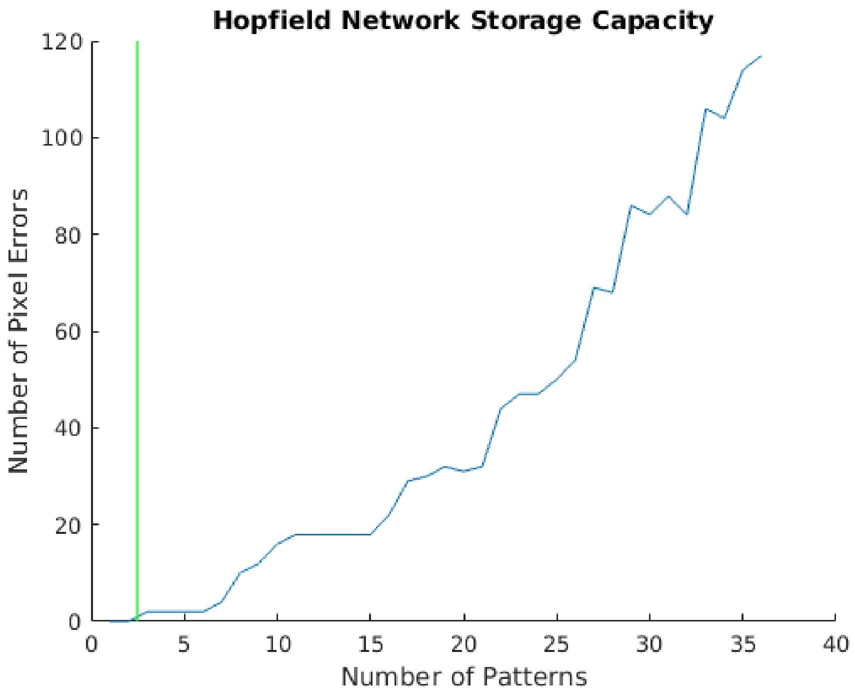


Figure 5.7: Actual storage capacity of Hopfield network. The green line shows the theoretical value ($\approxeq 2$)

5.2.4 Task 3 - 25-character experiment on scaled up network

In this task, the aim is to construct a network that can correctly recall 25 patterns.

Network Design

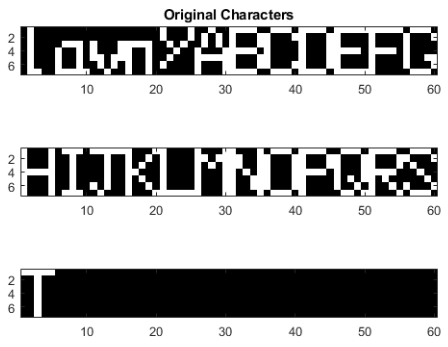
The way this can be done is by increasing the neurons N in the network. In order to get a storage capacity of 25, $\frac{N}{4logN} = 25$, which is achieved by using 875 (multiple of 35, $35 * 25$) neurons, which results in storage capacity being $\frac{N}{4logN} = \frac{875}{4log875} \approxeq 32$. The images thus have to be scaled up by a factor of $\sqrt{25} = 5$ in each dimension.
Thus this task the Hopfield network of $35 \times 25 = 875$ neurons is used to store the 25 patterns. Then distorted (again 3 bits at random) versions of the first 25 characters are fed into the network in order to find the correct attractor state pattern, The results of this are shown in Fig. 5.8.

Image Scaling

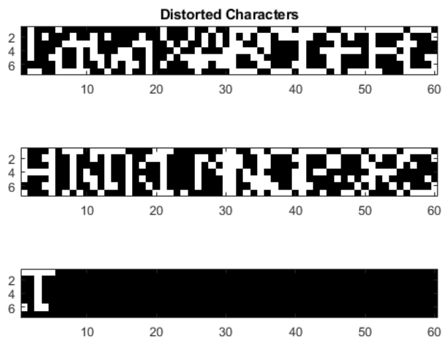
In order to use the new network, the images have to be resized. This is done by bicubic interpolation followed by rounding to ensure that the values remain between -1 and 1. The distorted images are also resized instead of distorting the resized images by swapping 3 bits. The outputs of the net when the scaled distorted patterns are fed in are rescaled down by 25 similarly in order to match the size of the original patterns.

Recall

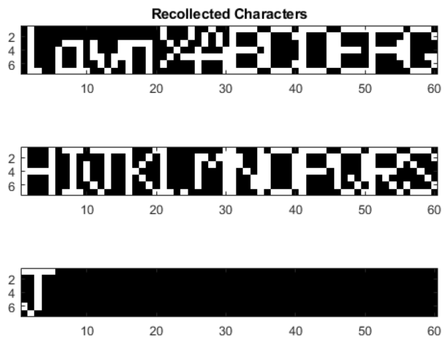
It can be seen from the figure there are 10 errors. These errors are in the characters A, F, J, L, M and T.



(a) Original



(b) Distorted (3 bits)



(c) Retrieved From Net

Figure 5.8: Experiment using the first 25 characters on the scaled up 875 node hopfield network. Recollection error = 10 pixels, Total Distortion = 75 pixels