

ARTIFICIAL NEURAL NETWORKS AND DEEP LEARNING**REPORT**

<u>STUDENT NAME</u>	<u>Reema Qaiser Khan</u>
<u>STUDENT EMAIL</u>	<u>reemaqaiser.khan@student.kuleuven.be</u>
<u>STUDENT NUMBER</u>	<u>0775319</u>
<u>STUDENT OPTION</u>	<u>MASTER OF ARTIFICIAL INTELLIGENCE [ECS TRACK]</u>
<u>DATE</u>	<u>28th MAY 2020</u>

Exercise Sessions

- | | |
|--|---------------|
| 1. <i>Supervised learning and generalization</i> | <i>Pg. 1</i> |
| 2. <i>Recurrent neural networks</i> | <i>Pg. 7</i> |
| 3. <i>Deep feature learning</i> | <i>Pg. 18</i> |
| 4. <i>Generative models</i> | <i>Pg. 23</i> |

Artificial neural networks - Exercise session 1

Supervised learning and generalization

Reema Qaiser Khan- r0775319

1. Comparing Gradient Descent with Different Training Algorithms

1.1. Levenberg-Marquardt algorithm with Gradient Descent

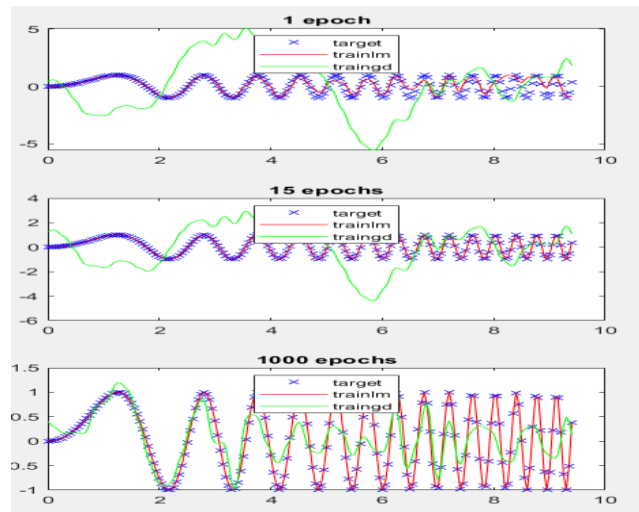


Figure 1: Plot of Levenberg-Marquardt algorithm with Gradient Descent

By analyzing the graph plots, we can clearly see that Levenberg-Marquardt has fitted very well w.r.t the function. However, Gradient Descent has performed poorly. At 1 epoch, LM has fitted well, GD has not fitted well. At 15 epochs, once again LM has performed well but GD is far from the function. At 1000 epochs, LM has fitted really well, GD has not fitted well.

1.2. Fletcher-Reeves conjugate gradient algorithm with Gradient Descent

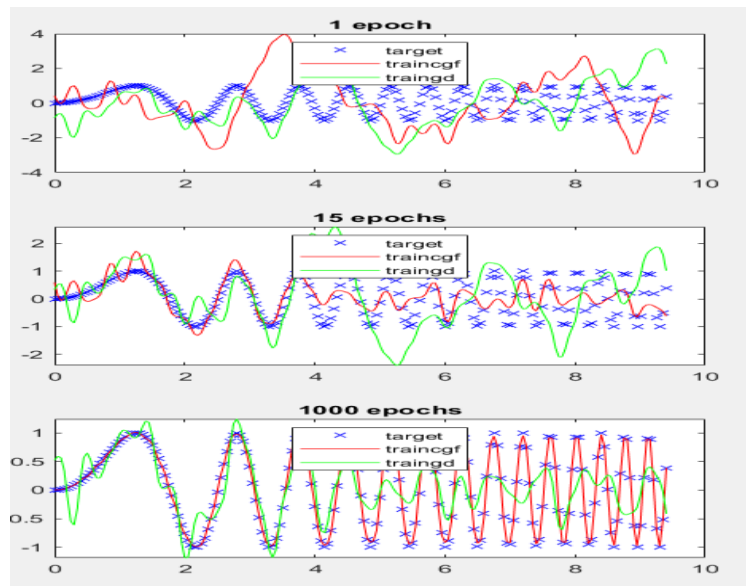


Figure 2: Plot of Fletcher-Reeves conjugate gradient algorithm with Gradient Descent

By analyzing the graph plots, FR has performed poorly at 1 epoch, GD is not that great either, however it is better than FR at 1 epoch. At 15 epochs, FR is close to fitting to the function, but it's still not good. GD has not fitted well at 15 epochs either. At 1000 epochs, FR has fitted well to the function, GD too has fitted well but FR has outperformed GD here.

1.1. BFGS quasi Newton algorithm (quasi Newton) with Gradient Descent

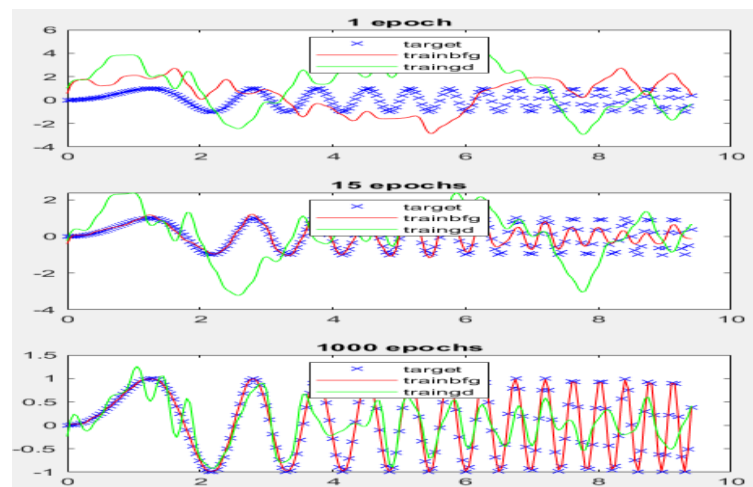


Figure 3: Plot of BFGS quasi Newton algorithm (quasi Newton) with Gradient Descent

By analyzing the graph plots, it is evident that BFGS quasi Newton has performed better than Gradient Descent. At 1000 epochs BFGS has fitted really well, whereas GD has not. BFGS when compared with LM, LM has performed better than BFGS.

2. Learning from noisy data: generalization

Noise has been added $y = y + 0.3 * \text{rand}(1, \text{length}(y))$;
It has an amplitude of 0.3.

2.1. Levenberg-Marquardt algorithm with Gradient Descent

LM performs really well when compared with GD. In all the 1, 15 and 1000 epochs, LM is very well fitted on the function in the presence of noise.

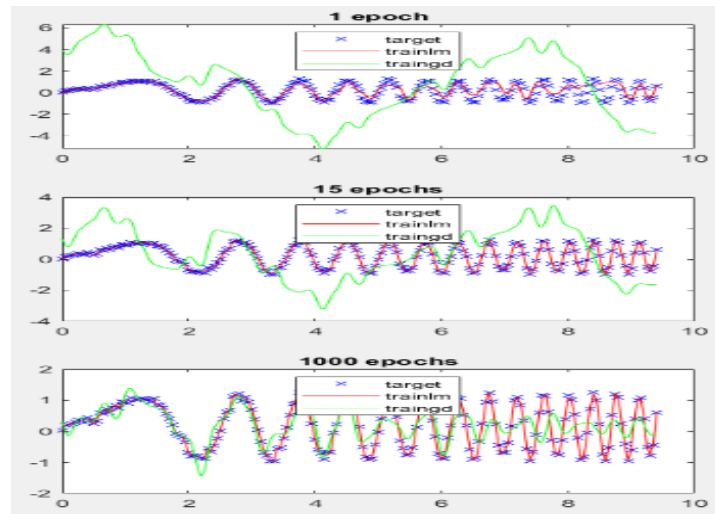


Figure 4: Plot of Levenberg-Marquardt algorithm with Gradient Descent

2.2. Fletcher-Reeves conjugate gradient algorithm with Gradient Descent

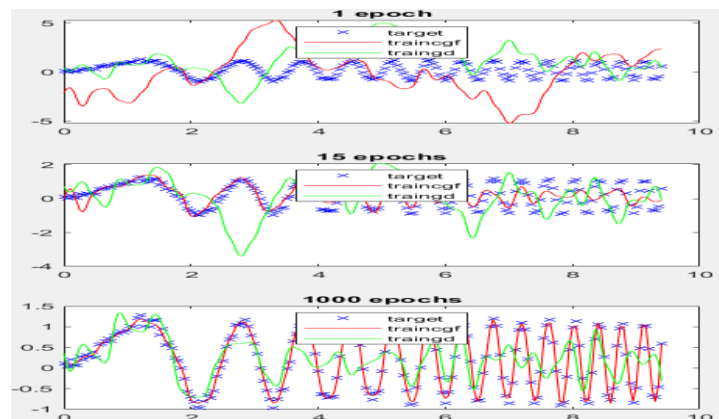


Figure 5: Plot of Fletcher-Reeves conjugate gradient algorithm with Gradient Descent

FR does not perform well at 1 epoch, however, through 15 till 1000 epochs it has improved and has eventually fitted well on the function. GD was better at 1 epoch than

FR, at 1000 epochs it is trying to fit on the function but still not that good when compared with FR.

2.3. BFGS quasi Newton algorithm (quasi Newton) with Gradient Descent

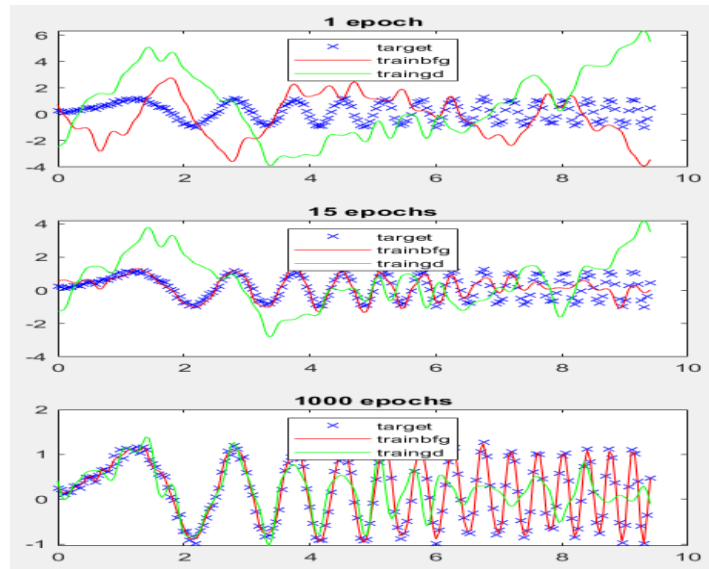


Figure 6: Plot of BFGS quasi Newton algorithm (quasi Newton) with Gradient Descent

BFGS in 1 epoch is far away from fitting on the function. It has improved through 15 epochs all the way till 1000 epochs. At 1000 epochs it has performed and fitted well on the function.

It is important to discuss about how gradient descent performed w.r.t the three algorithms and when compared with the first section (no noise added); overall, it performed well and had better approximation.

3. Personal Regression

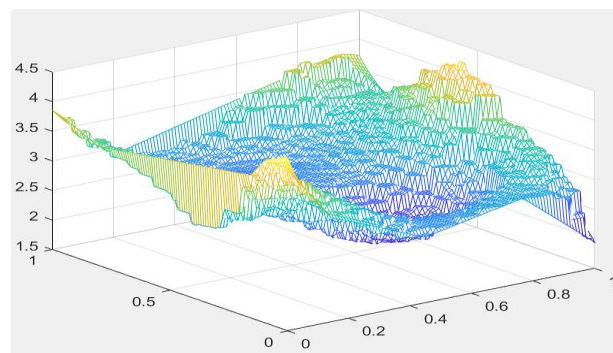


Figure 7: Plot of Personal Regression of Nonlinear function (train data)

For this task the following steps were performed:

- The function was created.
- Sampled it.
- Created the figure.
- Trained the network.
- Tested it.

From the above graph plot it can be analyzed that gradient descent has reached at it's minimum at 551 epochs.

It was also analyzed that in the Error histogram that there is an error of $-6.3e-07$ with a training dataset of almost 250. The mean squared performance is $1.08e-11$. The performance can be improved by using a different training algorithm, such as LM.

4. Comparing trainbr and Gradient Descent

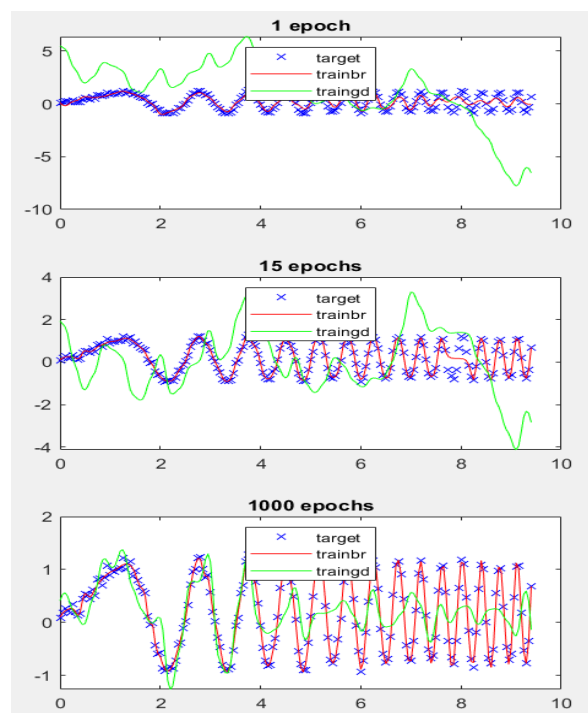


Figure 8: Plot of trainbr and Gradient Descent without noise

Neurons	Mean Squared Error	Epoch
50	0.2437	955
100	0.20607	985
10	0.46297	985

Table 1: trainbr and traingd Best Training Performance w.r.t Mean squared error and epochs without noise

Neurons	Mean Squared Error	Epoch
50	0.25743	985
100	0.20566	985
10	0.44972	985

Table 2: trainbr and traingd Best Training Performance w.r.t Mean squared error and epochs with noise

It was observed for both the cases that by increasing the number of neurons the error decreased. But by decreasing the number of neurons, the error increased.

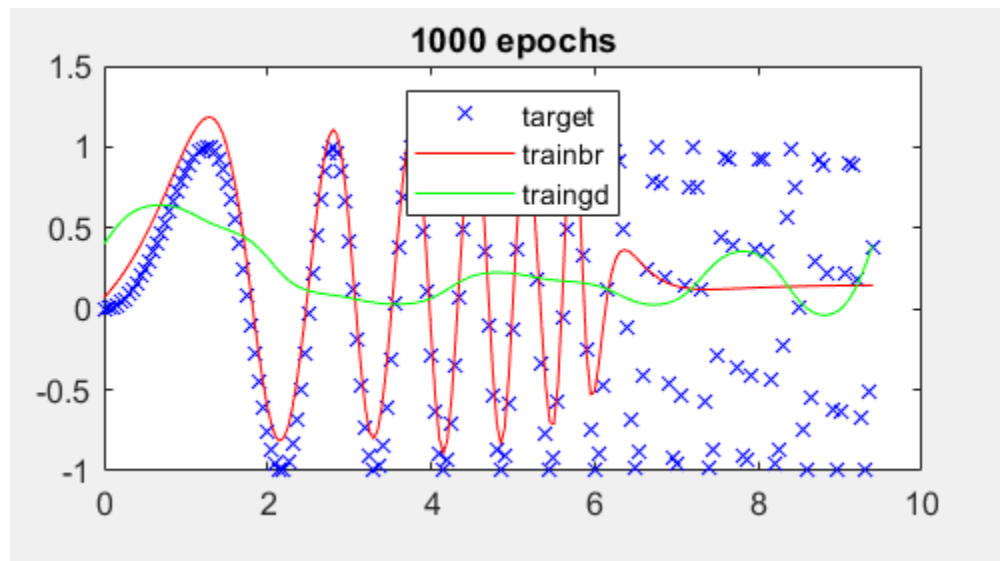


Figure 9: trainbr and traingd performance by decreasing the number of neurons (10).

Artificial neural networks - Exercise session 2

Recurrent neural networks

Reema Qaiser Khan-r0775319

1. Comparing various initial vectors and obtained attractors after a sufficient number of iterations.

1.1. Two target vectors $T = [+1 \ -1; -1 \ +1]'$ with 20 iterations

This Hopfield Network uses two neurons with 2 attractors $[+1 \ -1; -1 \ +1]$.

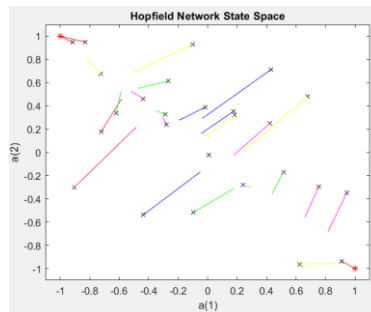


Figure 1: Two Target Vectors

From the graph plot it can be seen that all possible states are contained within the boundaries. The two stable points are plotted at the corners.

1.2. Three target vectors $T = [+1 \ +1 \ -1; -1 \ +1 \ -1]'$ with 20 iterations

This Hopfield Network uses three neurons with 3 attractors $[+1 \ +1 \ -1; -1 \ +1 \ -1]$.

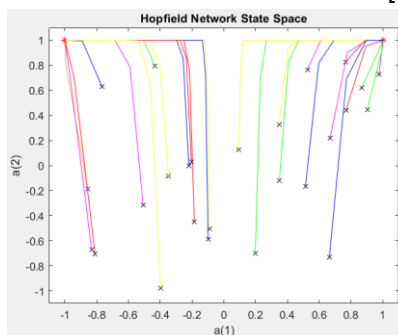


Figure 2: Three Target Vectors

From the graph plot it can be seen that there are two stable points that is shown at $[-1 \ +1]$ and $[+1 \ +1]$. All possible states are contained within the boundaries.

1.3. Four target vectors $T = [-1 \ +1 \ -1 \ +1; +1 \ -1 \ +1 \ +1]'$ with 20 iterations

This Hopfield Network uses four neurons with 4 attractors $[-1 \ +1 \ -1 \ +1; +1 \ -1 \ +1 \ +1]$.

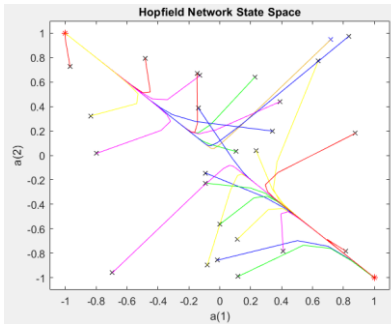


Figure 3: Four Target Vectors with 20 iterations

From the graph plot it can be seen that there are two stable points that are plotted at $[-1 \ 1]$ and $[1 \ -1]$. All possible states are contained within the boundaries.

1.4. Four target vectors $T = [-1 \ 1 \ -1 \ 1; +1 \ -1 \ 1 \ 1]'$ with 50 iterations

This Hopfield Network uses four neurons with 4 attractors $[-1 \ 1 \ -1 \ 1; +1 \ -1 \ 1 \ 1]$.

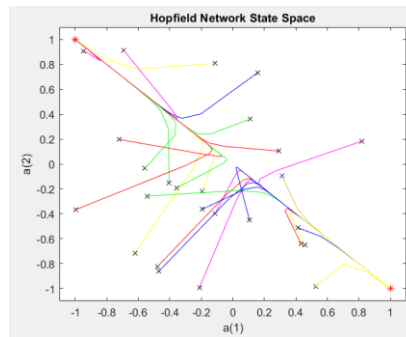


Figure 4: Four Target Vectors with 50 iterations

From the graph plot it can be observed that by increasing the number of iterations to 50, all the possible states are contained within the boundaries. There are two stable points plotted at $[-1 \ 1]$ and $[1 \ -1]$.

At 2 iterations, it can be seen that the points reach the attractors (at least one attractor).

2. Evaluating Script rep2 with high symmetry points.

2.1. Analysis with rands(2,1)

Running the original script, it can be observed that there are 4 attractors with 20 points.

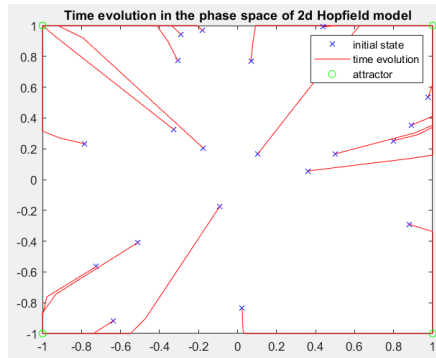


Figure 5: Analysis with rands(2,1)

2.2. Analysis with high symmetry points [0.5;0.5]

From the graph plot it can be observed that by using [0.5;0.5] as symmetry points, the 4 attractors are created at [1 1]. All attractors are at the same point [1 1] because of the initial state point [0.5;0.5] and all the possible states are contained within the boundaries.

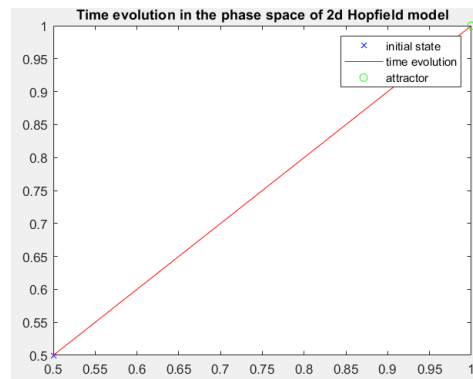


Figure 6: High symmetry points [0.5;0.5]

2.3. Analysis with high symmetry points [1;1]

From the graph plot it can be observed that by using [1;1] as symmetry points, the 4 attractors are created at [1 1]. All attractors are at the same point [1 1] because of the initial state point [1;1] and all the possible states are contained within the boundaries.

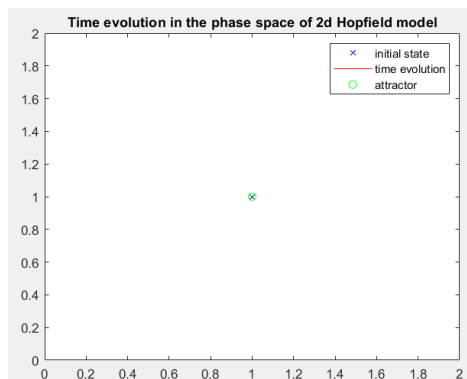


Figure 7: High symmetry points [1;1]

It is therefore observed that the number of attractors getting plotted are more than what is given, i.e. given 3, plotted 4.

3. Evaluating Script rep3 with high symmetry points.

3.1. Analysis with rands(3,1)

Running the original script, it can be observed that there are 3 attractors with 10 points.

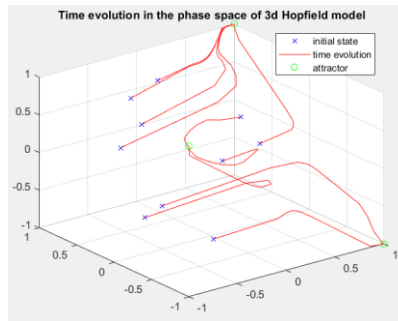


Figure 8: Analysis with rands(3,1)

3.2. Analysis with high symmetry points [0.5;0.5;0.5]

From the graph plot it can be observed that by using [0.5;0.5;0.5] as symmetry points, the 3 attractors are created at [1 1 1]. All attractors are at the same point [1 1 1] because of the initial state point [0.5;0.5;0.5] and all the possible states are contained within the boundaries.

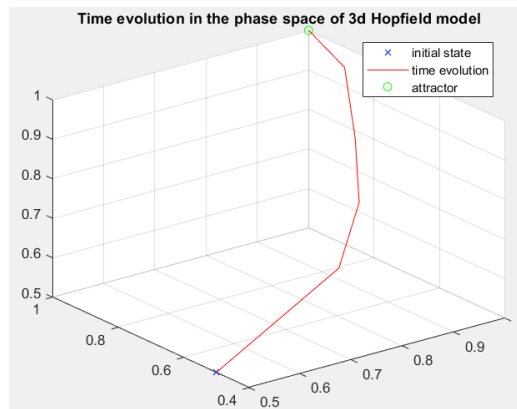


Figure 9: High symmetry points [0.5;0.5;0.5]

3.3. Analysis with high symmetry points [1;1;1]

From the graph plot it can be observed that by using [1;1;1] as symmetry points, the 3 attractors are created at [1 1 1]. All attractors are at the same point [1 1 1] because of the initial state point [1;1;1] and all the possible states are contained within the boundaries.

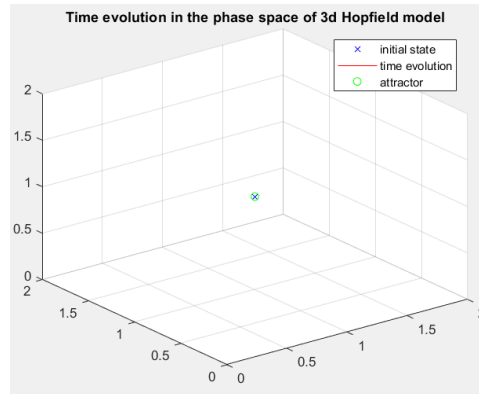


Figure 10: High symmetry points [1;1;1]

4. Hopfield network which has as attractors the handwritten digits.

4.1. Evaluating hopdigit_v2(2,100) with noise 2 and 100 iterations

By using a lower value of noise, it can be observed that the attractors are reconstructed pretty well and are correctly identified.

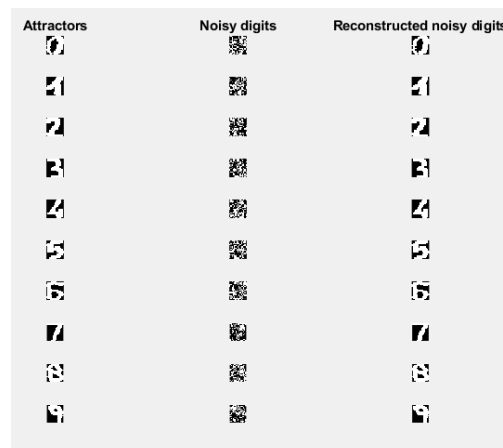


Figure 11: hopdigit_v2(2,100) with noise 2 and 100 iterations

4.2. Evaluating hopdigit_v2(5,100) with noise 5 and 100 iterations

The reconstructed noisy digits are not reproduced correctly.

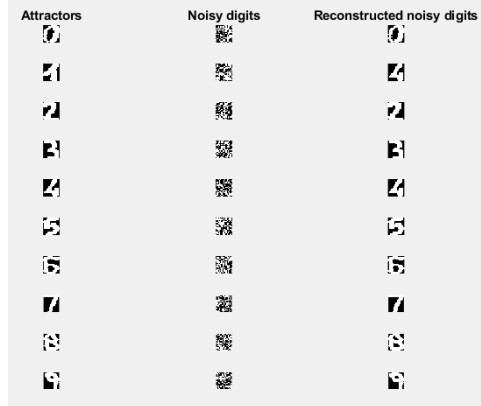


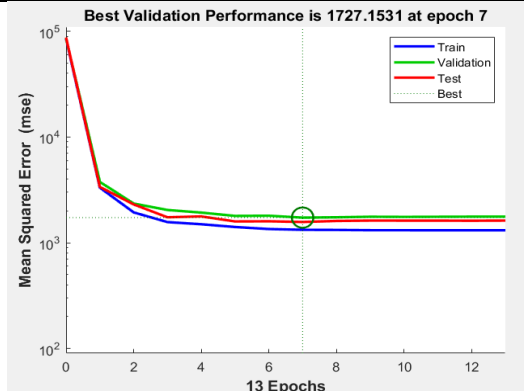
Figure 12: hopdigit_v2(5,100) with noise 5 and 100 iterations

Even by decreasing the number of iterations, it was observed that at hopdigit_v2(5,50) the reconstructed images were not correct, however, when hopdigit_v2(2,50) was applied, the images were correctly reproduced.

5. MLP with one hidden layer

The MLP with one hidden layer is created as follows:

- Load laser.dat file to a trainset variable
- Load laserpred.dat file prediction variable
- Assign a lag value p
- Create a network
- Assign number of neurons
- Number of iterations (Epochs)
- Train the network using Levenberg-Marquardt algorithm
- Predict the data

LAG P Value	Number of Neurons	Epochs	Best Validation Performance	Data of two iterations
1	100	100	 <p>Best Validation Performance is 1727.1531 at epoch 7</p>	data = 37.3808 data = 60.4229

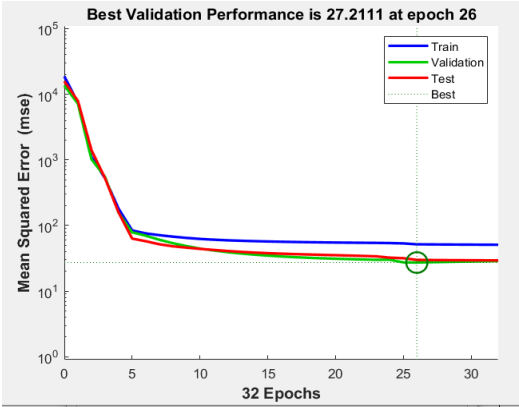
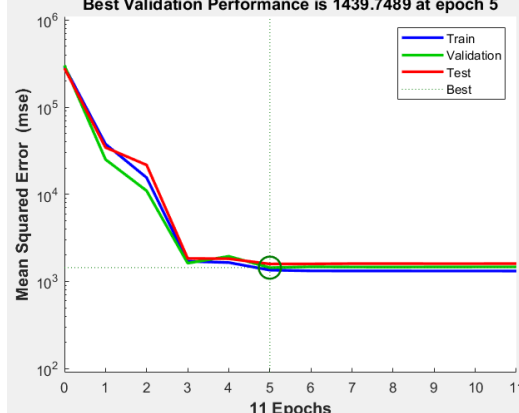
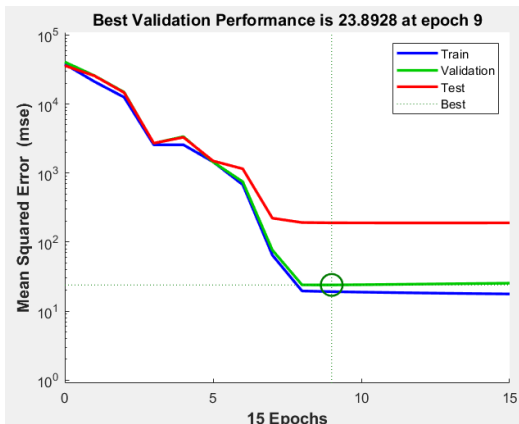
5	100	100	 <p>Best Validation Performance is 27.2111 at epoch 26</p>	<p>data =</p> <p>20.0000</p> <p>12.0000</p> <p>13.0000</p> <p>23.0000</p> <p>70.7827</p> <p>data =</p> <p>12.0000</p> <p>13.0000</p> <p>23.0000</p> <p>70.7827</p> <p>168.7215</p>
1	50	100	 <p>Best Validation Performance is 1439.7489 at epoch 5</p>	<p>data =</p> <p>34.7174</p> <p>data =</p> <p>52.7983</p>
5	50	100	 <p>Best Validation Performance is 23.8928 at epoch 9</p>	<p>data =</p> <p>20.0000</p> <p>12.0000</p> <p>13.0000</p> <p>23.0000</p> <p>69.0605</p> <p>data =</p> <p>12.0000</p> <p>13.0000</p> <p>23.0000</p> <p>69.0605</p> <p>173.6927</p>

Table 1: Investigation of the model with different lag values and neurons

It can be concluded by Table 1 that by increasing the lag value and keeping the number of neurons same, there is a decrease in the mean square error, which indicates of how close a fitted line is to data points. The lesser the error the better the performance.

6. Long short-term memory network-LSTM

6.1 First Part Train the LSTM

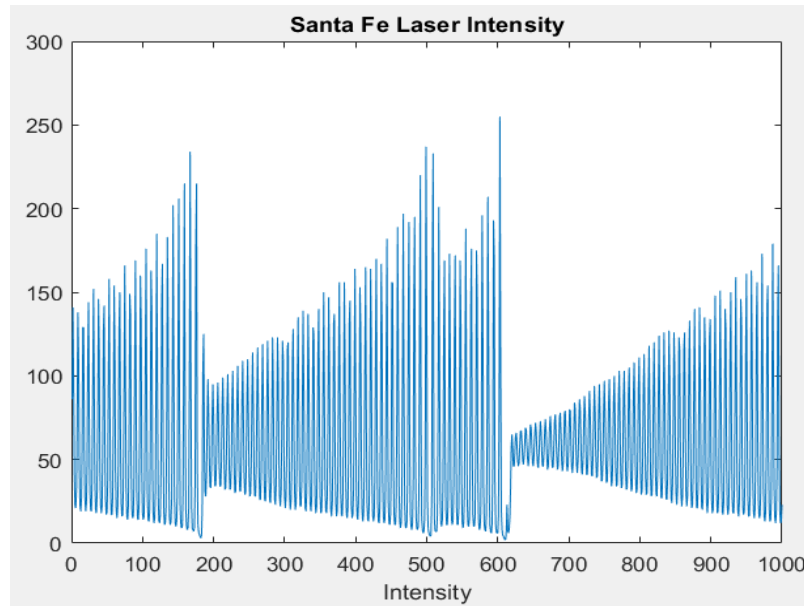
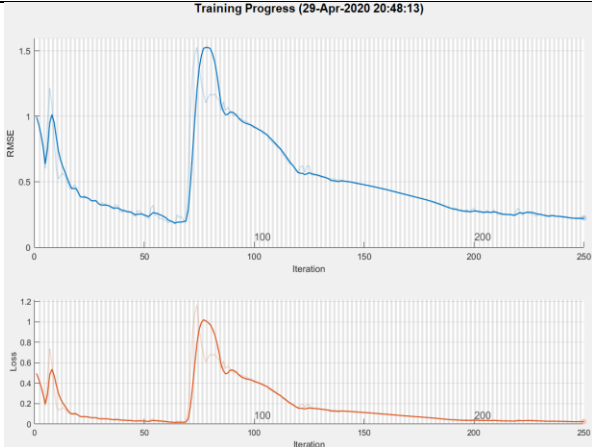
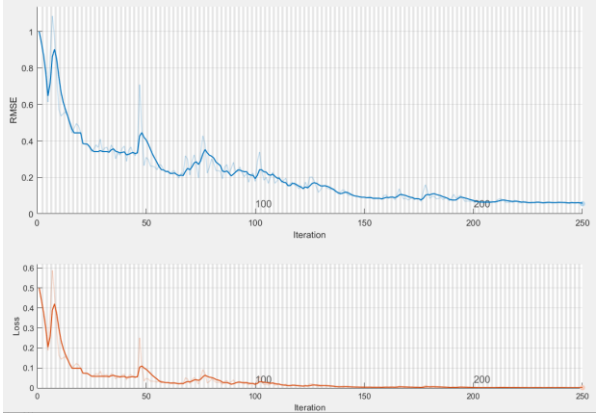
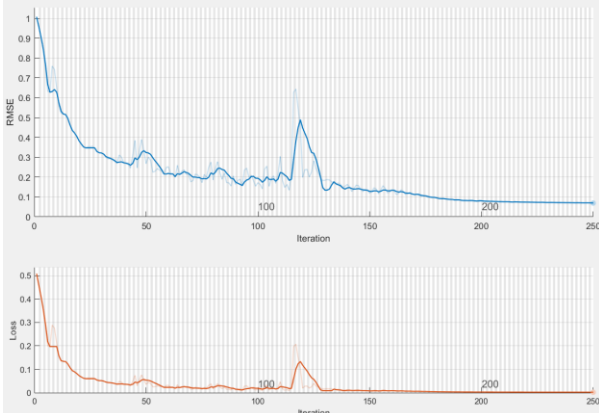
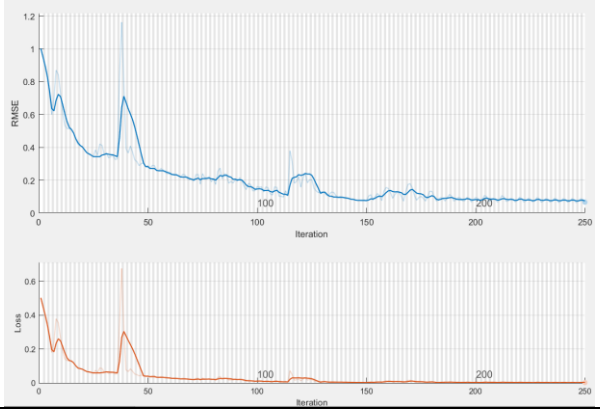


Figure 13: Santa Fe Laser Intensity graph

Lag	Hidden Units	Epochs	Training Progress	Elapsed Time
1	200	250	 <p>The training progress graph shows two metrics over 250 iterations. The top plot shows RMSE (Root Mean Square Error) on the y-axis (0 to 1.5) against Iteration on the x-axis (0 to 250). The RMSE starts at approximately 1.2, drops to a minimum of about 0.2 at iteration 75, then spikes to a peak of about 1.5 at iteration 80, before gradually decreasing to around 0.2 by iteration 250. The bottom plot shows Loss on the y-axis (0 to 1.2) against Iteration on the x-axis (0 to 250). The Loss starts at approximately 0.8, drops to a minimum of about 0.1 at iteration 75, then spikes to a peak of about 1.1 at iteration 80, before gradually decreasing to around 0.1 by iteration 250.</p>	1 Min 54 sec

5	200	250	<p>Training Progress (29-Apr-2020 20:53:45)</p> 	1 Min 54 sec
1	150	250	<p>Training Progress (29-Apr-2020 21:09:00)</p> 	1 Min 29 sec
5	150	250	<p>Training Progress (29-Apr-2020 21:13:47)</p> 	1 Min 28 sec

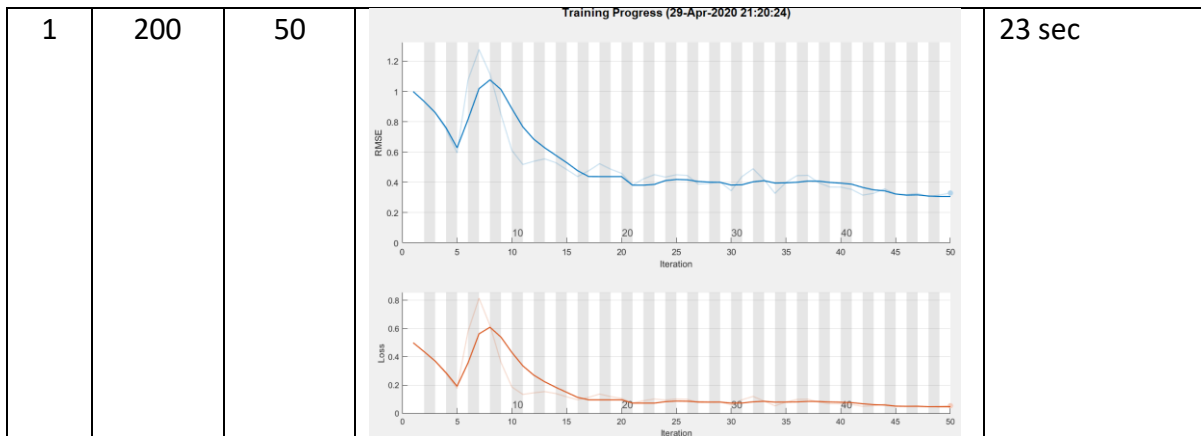


Table 1: Parameters (Lag, Hidden Units, Epochs) Tuned

It can be analyzed from Table 1, for the first two examples where the lag value 1 was increased by 5, the values of Hidden Units and Epochs were kept same, the output plot showed that the RMSE and Loss error decreased smoothly and gradually. This indicates that Lag 5 seems to be better than Lag 1. The elapsed time was same.

For the next example, Lag value was 1, Hidden Units value was reduced to 150, Epochs was same. For the RMSE and Loss, the values decreases gradually but there is a slight spike in the middle. It's not a problem because eventually the error and loss is reducing and that's what we want. The elapsed time reduced.

For the next example, the lag value was increased to 5 keeping Hidden Units 150 and Epochs 250. The RMSE and Loss too eventually decreases. When compared to Lag 1, there was only a 1 sec difference in the elapsed time.

For the last example, the lag value is kept 1, Hidden Units 200, and Epochs 50. When compared to the example where the Epochs is 200. This seems to decrease smoothly. Elapsed was 23 sec.

6.2 Second Part Predict and Update

For the second part the following parameters have been chosen:

- Train Initial Lag= 1
- Test Lag= 1
- Hidden Units= 200
- Epochs= 250

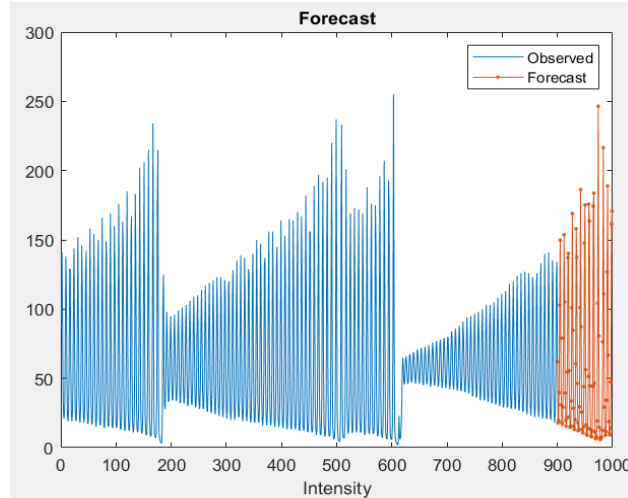


Figure 14: Santa Fe Laser Intensity graph with Forecast

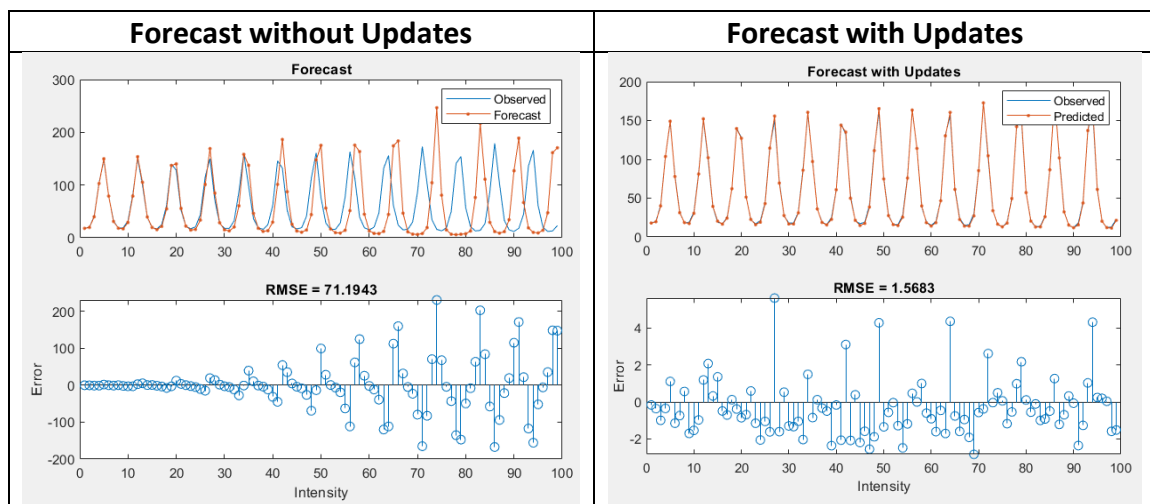


Table 2: RMSE and Forecast with/without updates

From Table 2 we can analyze that there is a significant decrease in the RMSE when the Forecast is updated. The observed and predicted data fit well.

When comparing the RNN with the LSTM, so LSTM gives Better Results. But in LSTM we have to deal with the complexity and operation cost.

Artificial neural networks - Exercise session 3

Deep feature learning

Reema Qaiser Khan-r0775319

1. Principal Component Analysis on Handwritten Digits

Performed Principal Component Analysis on the handwritten digit three using the database threes.mat.

In figure 1, the 7th three image of the dataset has been displayed.

In figure 2, the mean has been plotted.

In figure 3, the diagonal matrix of Eigen values has been plotted.

In figure 4, 4 reconstructed images have been displayed from the compressed dataset that were projected on 1,2,3, and 4 principal components.

In figure 5, the reconstruction error has been plotted, it can be observed that with each principal component the error is gradually decreasing. The reconstruction error for $q=256$ should be zero. However, in my case its 0.0826.

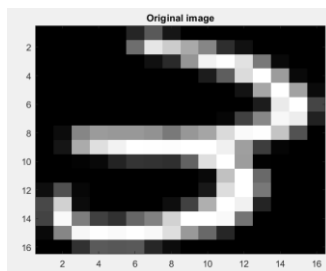


Figure 1: Original Image

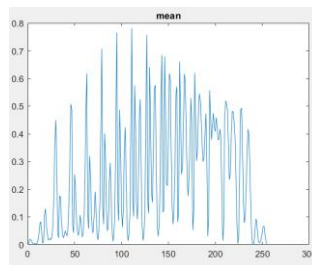


Figure 2: Mean

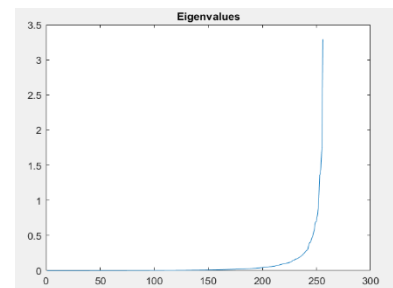


Figure 3: Eigen Values

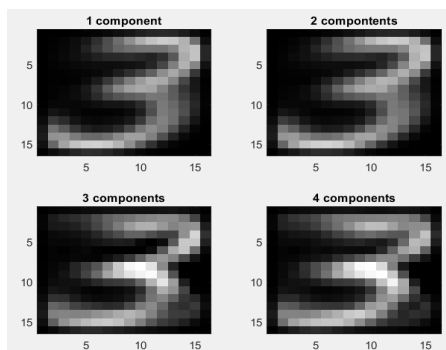


Figure 4: Reconstructed Images

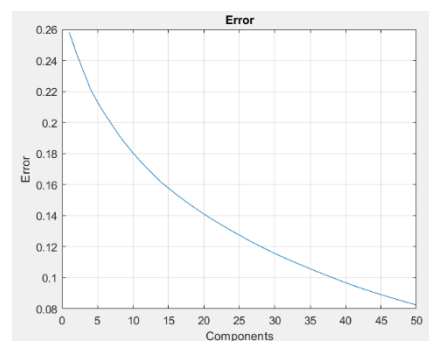


Figure 5: Error

2. Digit Classification with Stacked Autoencoders

A Stacked AutoEncoder was executed using a dataset of digittrain_dataset.mat.

It consists of 5000 training examples.

Stacked autoencoder is divided into different parts:

Training First Autoencoder, Training Second AutoEncoder, Training SoftMax Layer, Forming a stacked neural network, and finally fine tuning the neural network.

It showed us how a deep neural network is trained to classify digits. For each AutoEncoder, we had to set the size of the hidden layer, which is supposed to be smaller than the input size(dimensions).

It can be observed that in AutoEncoder1: Input size=784 and Hidden Layer size 100, in AutoEncoder2: Input size=100 and Hidden Layer size=50, and in Softmax Layer: Input size=50 and Hidden Layer=10. To check the result we uploaded digittest_dataset.mat. Before fine tuning the accuracy of the Confusion matrix is 81.5%, after fine tuning the confusion matrix accuracy is improved to 99.7%.

This was achieved with epochs as follows: AutoEncoder1=400, AutoEncoder2=100, and Softmax Layer=400.

Figure 6, shows the images of the digit.

Figure 7, shows the result after the first AutoEncoder executes.

Figure 8 shows a stacked network view.

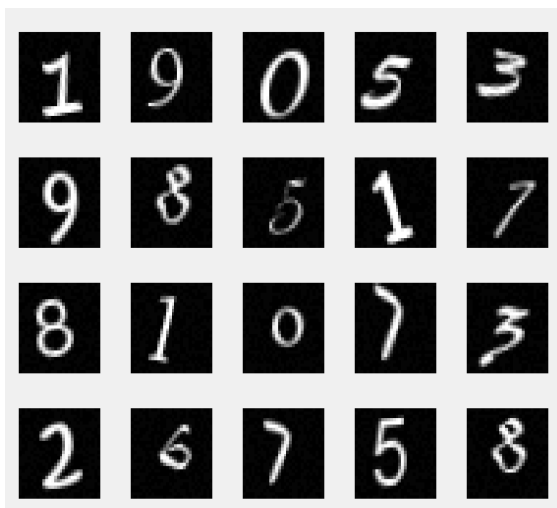


Figure 6: Images of the Digits

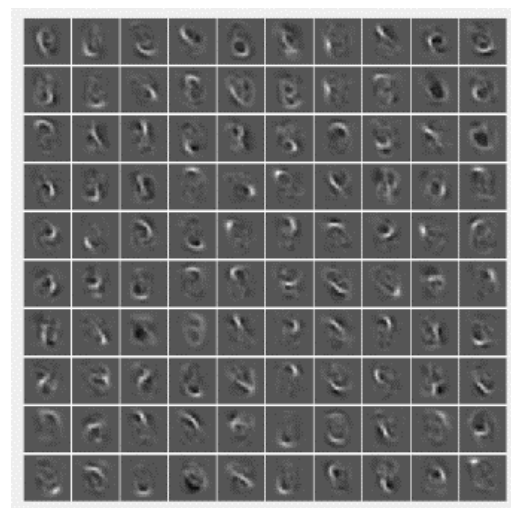


Figure 7: Result after the first AutoEncoder

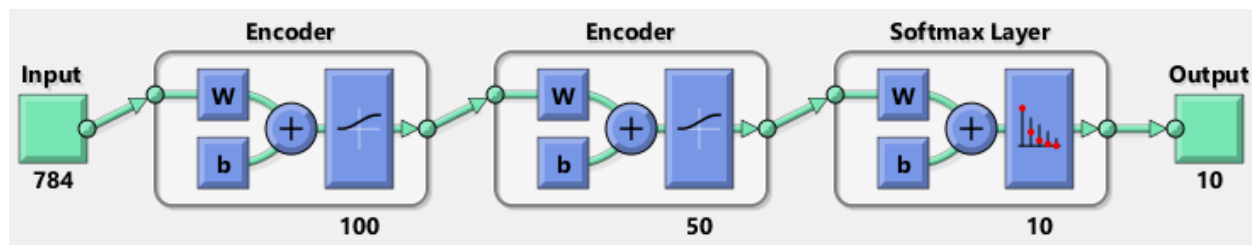


Figure 8: Stacked Network

Next, we execute the DigitClassification.m. The epochs and input size for AutoEncoder1, AutoEncoder2, and Softmax is same as the previous example. The accuracy calculated is 99.6800. When compared to normal neural network with 1 hidden layer, the accuracy calculated is 95.800. For the normal neural network with 2 hidden layers the accuracy is 96.5800.

In this default setting it can be observed that the AutoEncoder performed well as compared to the normal neural network. By changing the epochs to [300,90,300] and the patternnet for 1st hidden layer to 90 and for the 2 hidden layers to [90 40], it was still observed that AutoEncoder performed well having an accuracy of 99.6800. Whereas normal neural network had an accuracy of 97.2600 for the 1st hidden layer and 96.1800 for 2 hidden layers. Figure 9 shows Pattern Recognition Neural Network View.

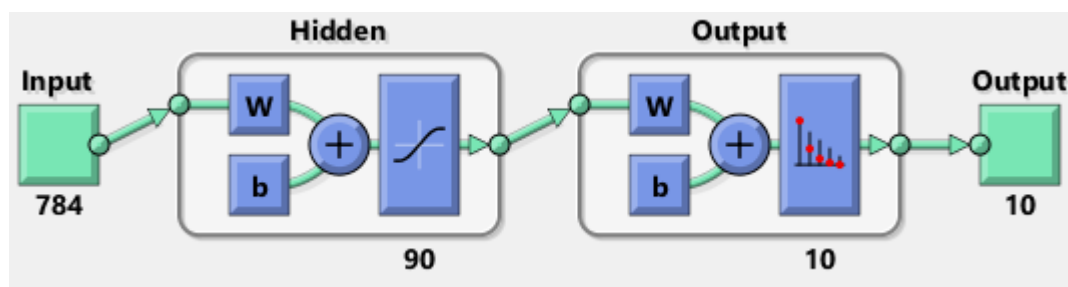


Figure 9: Pattern Recognition Neural Network View

3. Convolutional Neural Networks

3.1. Meaning of weights:

In convolutional Neural Network we use Filters, Filters can be of many types: horizontal edge detection, vertical edge detection and so on. Therefore, the values that are used in these filters are weights. During back propagation these weights get updated.

3.2. Dimension of the input at the start of layer 6:

The CNN has the following layers:

- Input Layer
- 1st Convolutional Layer
- ReLu Layer
- Pooling Layer
- Fully Connected Layer.

In the Input Layer the colored RGB image has the dimensions 227x227x3. This is passed onto the convolutional layer.

Here the dimensions are 227x227x96. 96 here is representing the 96 filters used. The filters have the dimensions 11x11x3.

For calculating the new dimensions that will be passed to the Relu and Cross-Channel normalization layer, we use the following formula:

$$n_H = [(n + 2P - F) / S] + 1$$

Here,

n is size 227

F is filter size

P is padding which is zero

S is stride which is 4

Formula for calculating new dimensions

The new dimensions after 1st convolutional layer is 55x55x96. In the Relu Layer and the Cross-Channel normalization Layer dimensions do not change. For the max pooling layer, we use 2x2 filter with stride 2.

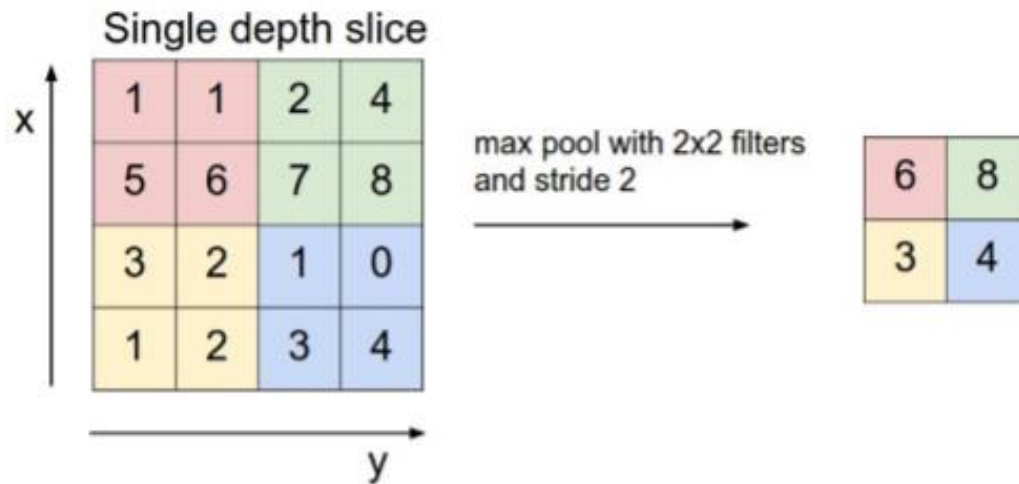


Figure 10: Example of Max Pool and stride

This returns 27x27x96. The original image has been down sampled to 27x27x96.

3.3. Network Final Dimension:

There are 1000 neurons in the last layer according to the 'ClassificationOutputLayer' properties. However, the network was trained to classify 1000 unique classes and in our case we are using only 3 [airplanes ferry laptop]. We should therefore be using 3 neurons based on our initial input image [227x227x3].

Artificial neural networks - Exercise session 4

Generative models

Reema Kaiser Khan-r0775319

1. Restricted Boltzmann Machine

1.1. Effect of Different Parameters (# epochs and # components)

Numbers of Components is the Number of binary hidden units.

Number of Epochs is the Number of iterations/sweeps over the training dataset to perform during training.

By increasing the Number of components, the training time has increased, which means that the result will be more accurate. Therefore, the reconstruction error is less.

By decreasing the Number of components, the training time has decreased, which means that the result will be less accurate. Therefore, the reconstruction error is more.

Number of Error has less effect on Error as compared to Number of Components.

Pseudo_Likelihood is a reconstruction loss because it is decreasing.

```
[BernoulliRBM] Iteration 1, pseudo-likelihood = -112.04, time = 16.00s
[BernoulliRBM] Iteration 2, pseudo-likelihood = -98.47, time = 17.64s
[BernoulliRBM] Iteration 3, pseudo-likelihood = -91.87, time = 17.67s
[BernoulliRBM] Iteration 4, pseudo-likelihood = -89.06, time = 17.67s
[BernoulliRBM] Iteration 5, pseudo-likelihood = -85.56, time = 17.61s
[BernoulliRBM] Iteration 6, pseudo-likelihood = -84.18, time = 17.64s
[BernoulliRBM] Iteration 7, pseudo-likelihood = -81.96, time = 17.86s
[BernoulliRBM] Iteration 8, pseudo-likelihood = -81.36, time = 17.93s
[BernoulliRBM] Iteration 9, pseudo-likelihood = -81.01, time = 17.86s
[BernoulliRBM] Iteration 10, pseudo-likelihood = -80.16, time = 17.95s
[BernoulliRBM] Iteration 11, pseudo-likelihood = -78.98, time = 18.04s
[BernoulliRBM] Iteration 12, pseudo-likelihood = -78.10, time = 17.93s
BernoulliRBM(batch_size=10, learning_rate=0.01, n_components=100, n_iter=12,
random_state=0, verbose=True)
```

Figure 1: Performance of RBM when Number of Components=100 and Number of Iterations= 12

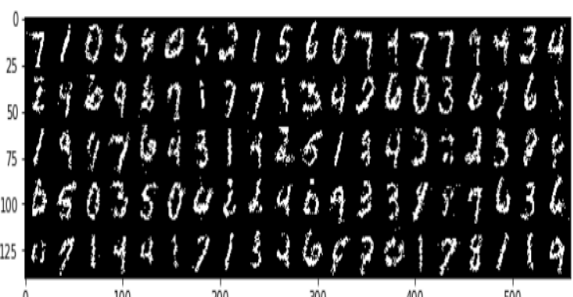
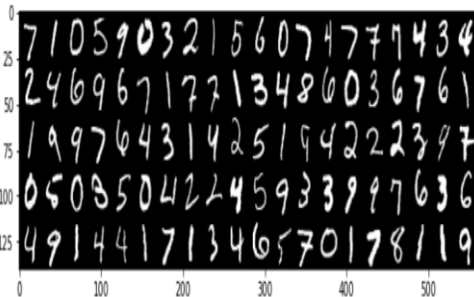
NUMBER OF COMPONENTS= 100, NUMBER OF ITERATIONS= 12	
RECONSTRUCTED IMAGE	TEST IMAGE
	

Table 1: Comparison of Reconstructed Image and Test Image


```
[BernoulliRBM] Iteration 1, pseudo-likelihood = -198.55, time = 4.50s
[BernoulliRBM] Iteration 2, pseudo-likelihood = -190.71, time = 5.09s
[BernoulliRBM] Iteration 3, pseudo-likelihood = -183.02, time = 5.15s
[BernoulliRBM] Iteration 4, pseudo-likelihood = -172.03, time = 5.08s
[BernoulliRBM] Iteration 5, pseudo-likelihood = -166.73, time = 5.13s
BernoulliRBM(batch_size=10, learning_rate=0.01, n_components=20, n_iter=5,
              random_state=0, verbose=True)
```

Figure 2: Performance of RBM when Number of Components=20 and
Number of Iterations= 5

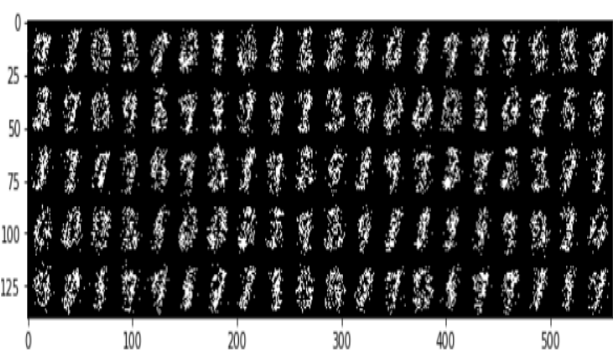
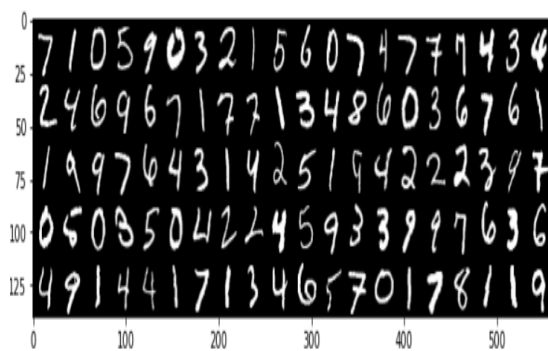
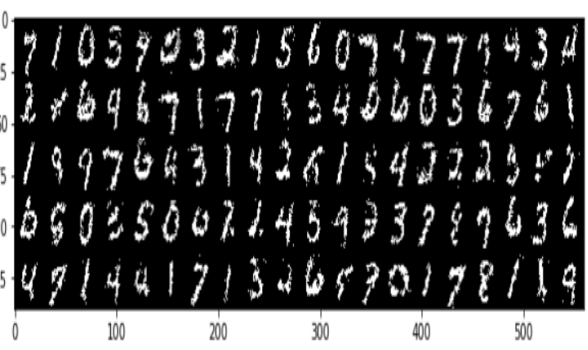
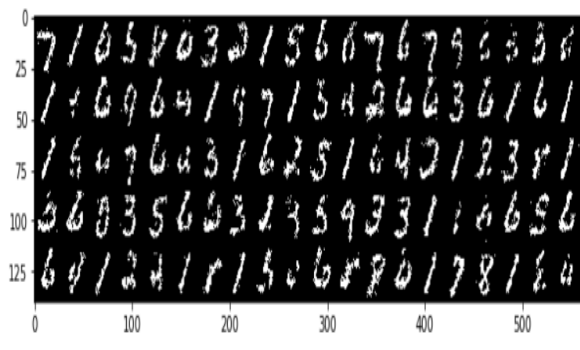
NUMBER OF COMPONENTS= 20, NUMBER OF ITERATIONS= 5	
RECONSTRUCTED IMAGE	TEST IMAGE
	

Table 2: Comparison of Reconstructed Image and Test Image

By comparing Table 2 and 4, it can clearly be seen that Table 2 is more accurate (less reconstruction error), whereas Table 4 is less accurate (more reconstruction error).

1.2. Gibbs Sampling

Reconstruction quality decreases as the Gibbs steps increases. The accuracy of the images is also affected.

NUMBER OF COMPONENTS= 100, NUMBER OF ITERATIONS= 12	
Gibbs_steps= 10	Gibbs_steps= 50
	
Gibbs_steps= 100	

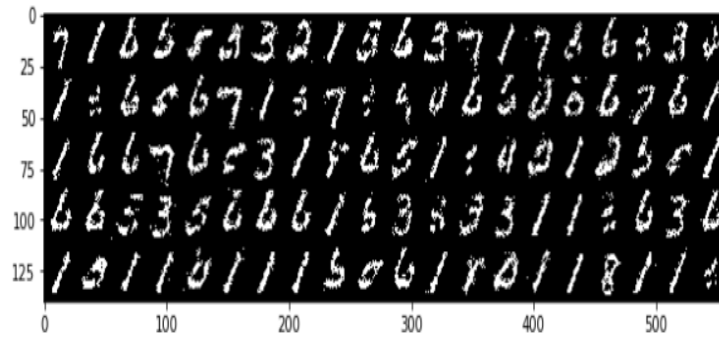


Table 3: Reconstructed Images with Gibbs steps 10, 50, and 100

1.3. Reconstruction of unseen images

The role of the number of hidden units is that it helps in capturing more features in the training set. If the number of hidden units is more, then in this case it will capture more features and the reconstruction of unseen images will be more accurate.

The learning rate is a tuning parameter that moves towards a local minimum by determining the step size at each iteration. A very high learning rate may skip the minima but a very low learning rate will take too long to converge or in another case may get stuck in a local minimum which is undesired.

By changing the number of iterations, error saturates (system has reached equilibrium).

By changing the hyperparameters it does affect the performance in the reconstruction of unseen images.

Observe Table 4, 5, and 6. The number of components and number of iterations were not changed, however just by changing the learning rate by different values has affected the reconstructed image.

Number of Components:100 Number of iterations:12 Learning Rate:0.2

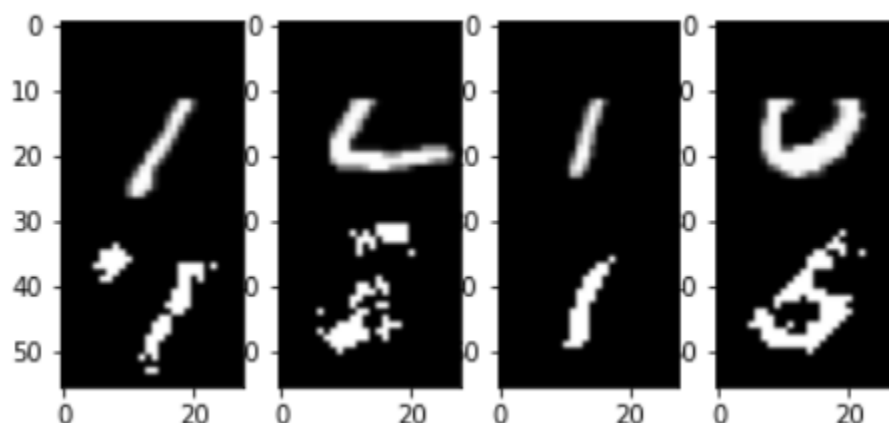


Table 4: Reconstructed Image with Number of Components:100 Number of iterations:12 Learning Rate:0.2

Number of Components:100 Number of iterations:12 Learning Rate:0.01

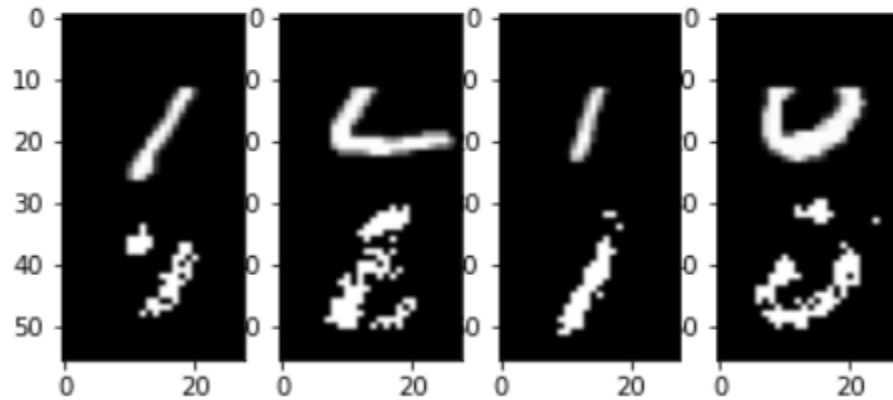


Table 5: Reconstructed Image with Number of Components:100 Number of iterations:12 Learning Rate:0.01

Number of Components:100 Number of iterations:12 Learning Rate: 3

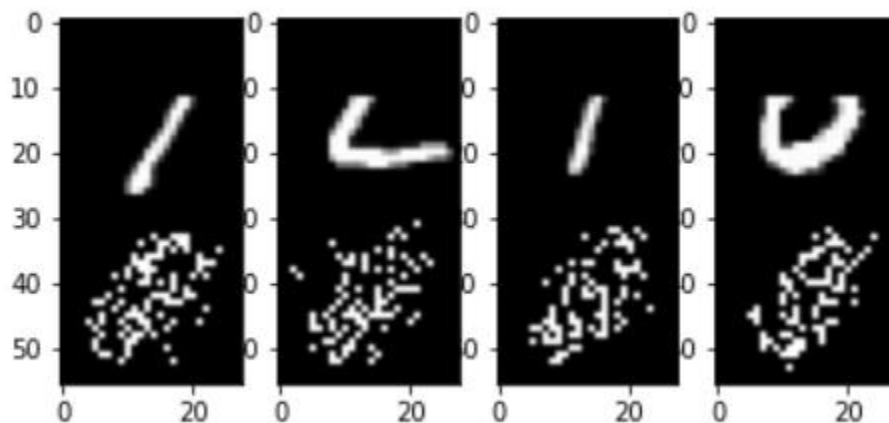


Table 6: Reconstructed Image with Number of Components:100 Number of iterations:12 Learning Rate:3

By removing more rows it is unable to reconstruct images.

Rows removed 0 to 15

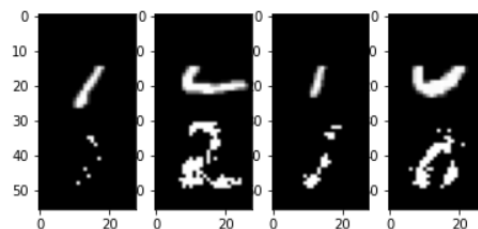


Table 7: Reconstructed images with missing rows

If rows are removed from the middle, as long as the difference between the start row and the end row isn't much, the system is able to reconstruct images accurately.

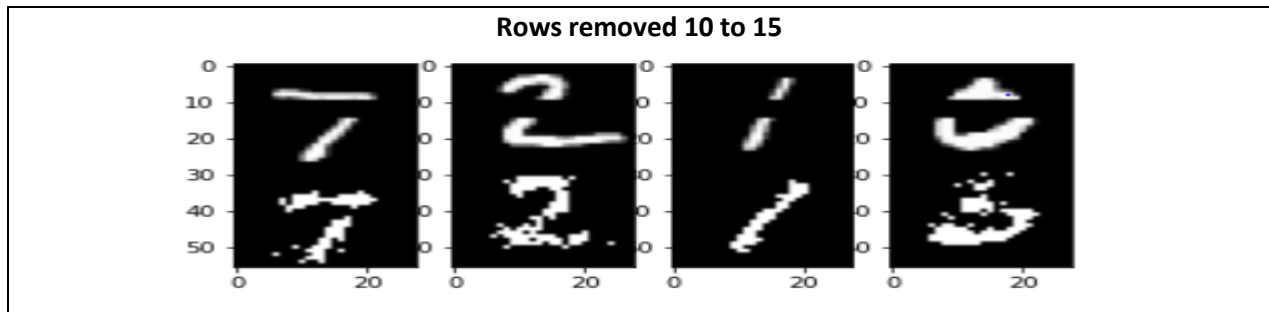


Table 8: Reconstructed Images with middle rows missing

2. Deep Boltzmann Machines

2.1 Difference between RBM and DBM (Interconnection Weights)

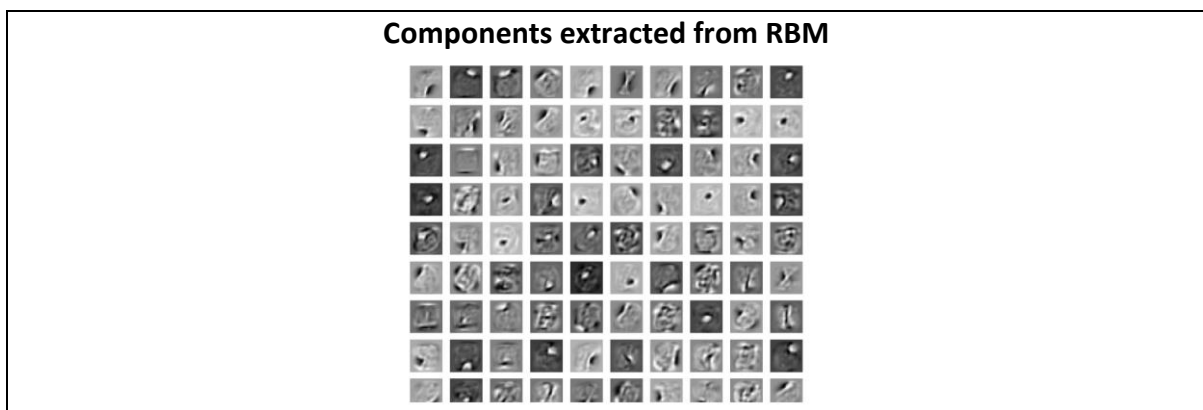


Table 9: Components extracted from RBM

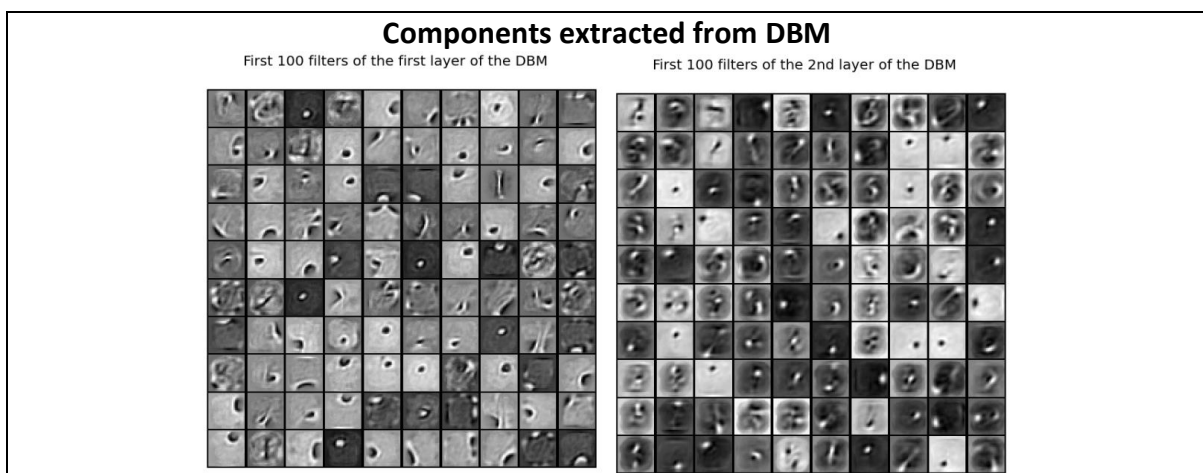


Table 10: Components extracted from DBM

RBM is a 2-layer neural network. The 1st layer is the visible, or input, layer. 2nd layer is the hidden layer.

Deep Boltzmann machines can be understood as a series of restricted Boltzmann machines stacked on top of each other. No connectivity within layers or between non-neighboring layers.

By looking at Table 9 and 10, it is obvious that DBM is a higher level of RBM therefore the weights in Table 10 are more prominent meaning that they are able to extract better features.

The difference between the filters of the first and second layer of the DBM is this that the first layer extracts the initial features (Low level features) and the second layer extracts the abstract features (Higher level Features).

2.2 Difference between RBM and DBM (Quality)

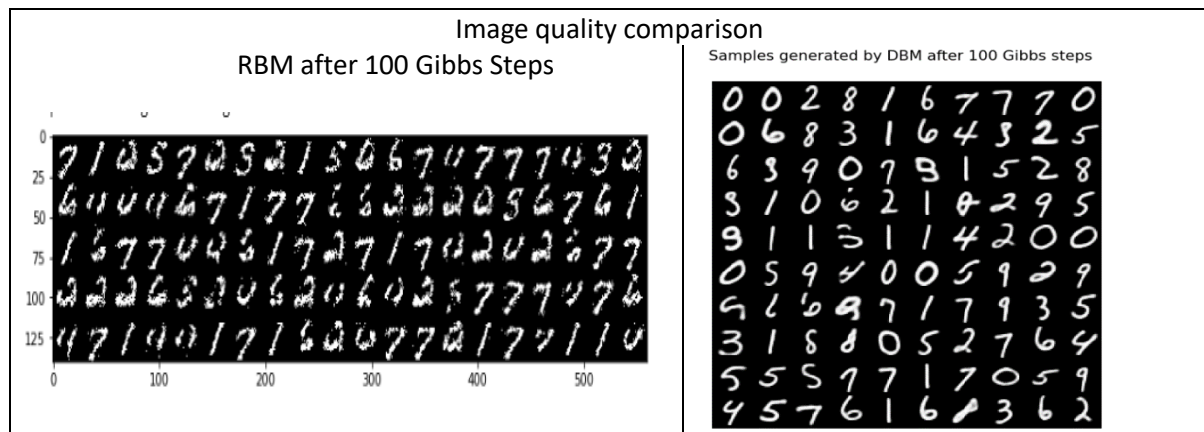


Table 11: Image quality comparison after 100 Gibbs Step

DBM has a better result as compared to RBM. This is mainly because of how DBM works. DBM has a different connectivity from that of an RBM. The DBM groups the hidden nodes into several hidden layers h_1, \dots, h_N with the constraint that the visible layer is connected to only the first hidden layer, and then each hidden layer is connected only to the layer below and above it, with no lateral. Whereas RBMs have very simple inference rules, inference in a DBM is approximate and depends on multiple passes through the hidden layers. In order to infer the state of a hidden layer, you need to know the state of the layer above and below it, so you have to do some sort of iterative sampling to find the hidden states (e.g. Gibbs sampling in this case).

3. Generative Adversarial Networks

3.1. Generator vs Discriminator Analysis

Generative model G to be trained on training data X sampled from some true distribution D is the one which, given some standard random distribution Z produces a distribution D' which is close to D according to some closeness metric. Mathematically, $z \sim Z$ maps to a sample $G(Z) \sim D'$. So, we are making the generator smart enough to fool the discriminator and produce output as 1.

Discriminative model D is the one that discriminates between two different classes of data. For example, object is a face or not, if yes then it will produce output as 1, if not then the model will produce output as 0.

By analyzing the DCGAN.ipynb, there are two things that we have to look at:

- 1- Discriminative Loss: How well the discriminator can distinguish between images that are real and images that are fake.

2- Generator Loss: How well the generator is able to trick the discriminator

For both the discriminator and generator, the model converges to a stable equilibrium eventually. Here the stability is alright.

Therefore, by looking at Figure, we can analyze that the artificial instances from $G(Z)$ are becoming stronger with each back propagation, and the generator is capable of easily tricking the Discriminator. When monitoring the loss of generator and discriminator, the values are varying for each batch. However, looking at the above definition, it is observed that generator's loss value is higher in many batches as compared to the discriminator's loss value which may quantify that the generator is able to trick the discriminator.



Table 12: Different Batches with Discriminator and Generator Loss and accuracy.

4. Optimal Transport

4.1. OT.ipynb

By looking at Table 13, it is observed that the color histograms are transported by extracting the Blue and Red channel pixels of each image. After that the EMD (Earth Mover's Distance) Transport is applied on the initial images giving us Image 1/2 Adapt. Sinkhorn Transport is also applied on the initial images giving us Image 1/2 Adapt (reg).

This technique is different from non-optimal color swapping because in that case all the 3 RGB channels are used. Number of steps are applied to the source and target image: computing color statistics, applying mean, clipping the pixel intensities, merging, returning transferred image.

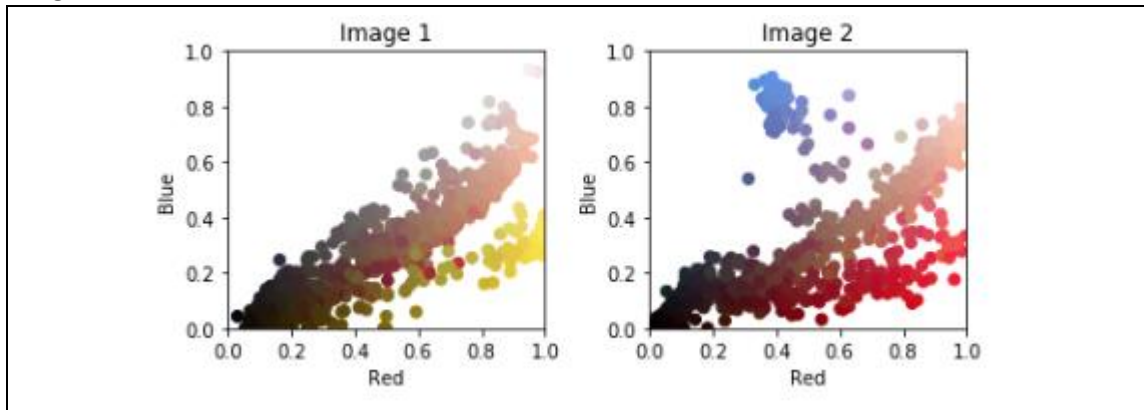


Table 13: Scatter Plot of Colors

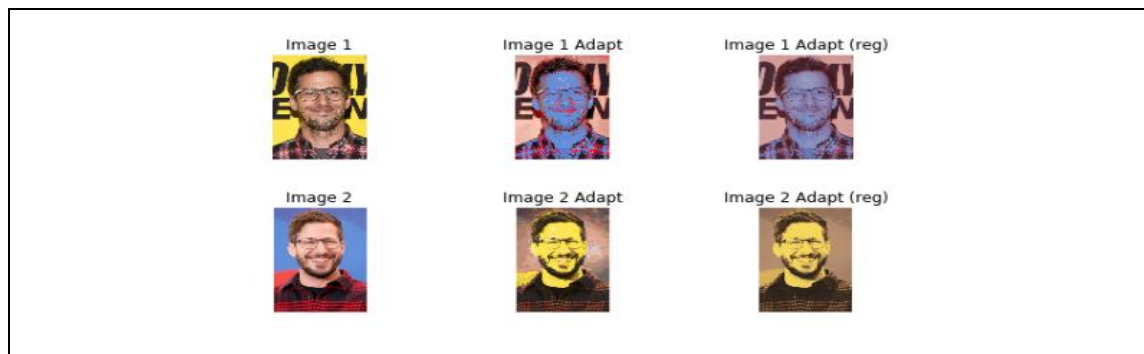


Table 14: Initial Images, EMD (Earth Mover's Distance) Transport Images (Adapt), and SinkhornTransport Images(Adapt Reg)

4.2. WGAN.ipynb

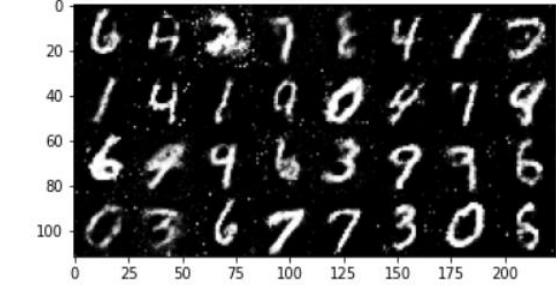
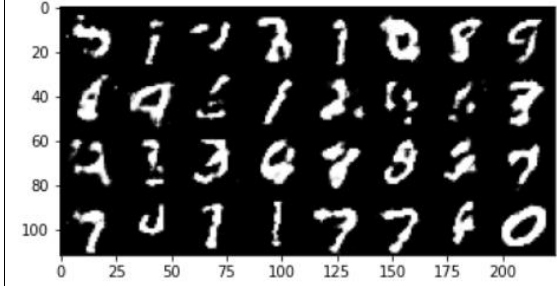
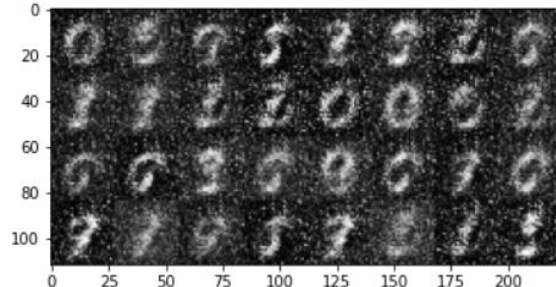
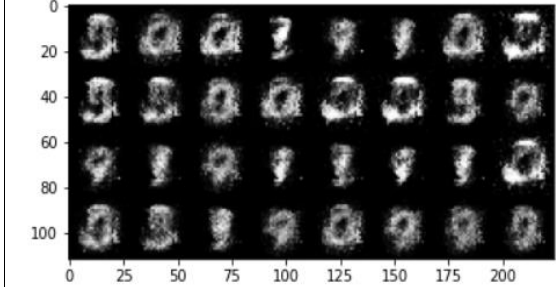
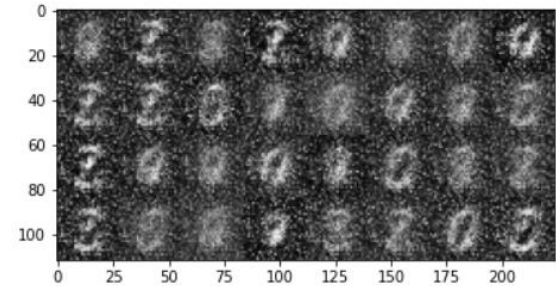
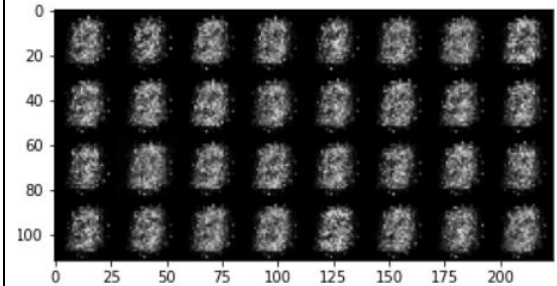
GAN	WGAN
BATCH 20000	
	
BATCH 1500	
	
BATCH 1000	
	

Table 15: Comparison of GAN and WGAN at different Batches

By analyzing Table 15, It can be observed that WGAN performs better than GAN. It takes longer time but the output, stability and generated samples are better as compared to GAN.

Optimal Transport: is a concept that compares two probability distributions. Through all possible ways, transport, morph, and reshape first probability distribution into the second probability distribution. The idea is to associate a global cost to every such transport, using the local considerations of moving 1 pile of sand to another. Optimal Transport is used in Image Sciences for color and texture processing. In computer vision for shape manipulation. In machine learning for classification.

Wasserstein distance is used to solve the Optimal transport problem by finding the optimal transport plan with minimum cost and optimal way of allocating resources. In WGAN, the distance should be minimum between the distributions of data in the training dataset and generated examples. This is the main objective of when generator is trained.