CSCI 5521: Introduction to Machine Learning

Homework 4

Saksham Goel | Abhiraj Mohan
goelx029 | mohan056

1) **Write a matlab function that uses deltaNN.m to build and train a network with 1 hidden layer with 2 nodes (plus a bias node) that can recognize the XOR function. Use initial coefficient matrices and data matrices:**

   V0 = [.3 .3 .3; -.3 -.1 .1];
   W0 = [.20 .40 .10; -.16 -.44 -.09];
   Data = [1 -1 1 -1; 1 1 -1 -1];
   Labels = [-1 1 1 -1; 1 -1 -1 1];
   rate=.1;

   Notice, the data samples are given as column vectors: each column of Data is a data sample with two attributes. Its corresponding label is in Labels. This data consists of four data samples. The learning rate is set to 0.1. Notice there are two outputs which are supposed to be negatives of each other, hence the labels to be learned are set to that property. This usually speeds up training. Allow at most 100 epochs. Plot the error functional as a function.

   File: hw4_part1.m
   For our solution, we used the given initial parameters and ran the training code for 100 epochs doing cyclic training (Updating the weights based on each training example rather than summing everything up). To plot the error function, we calculated the total error per epoch by summing the error returned by the delta_nn function over each training sample per epoch. The plots are as follows:

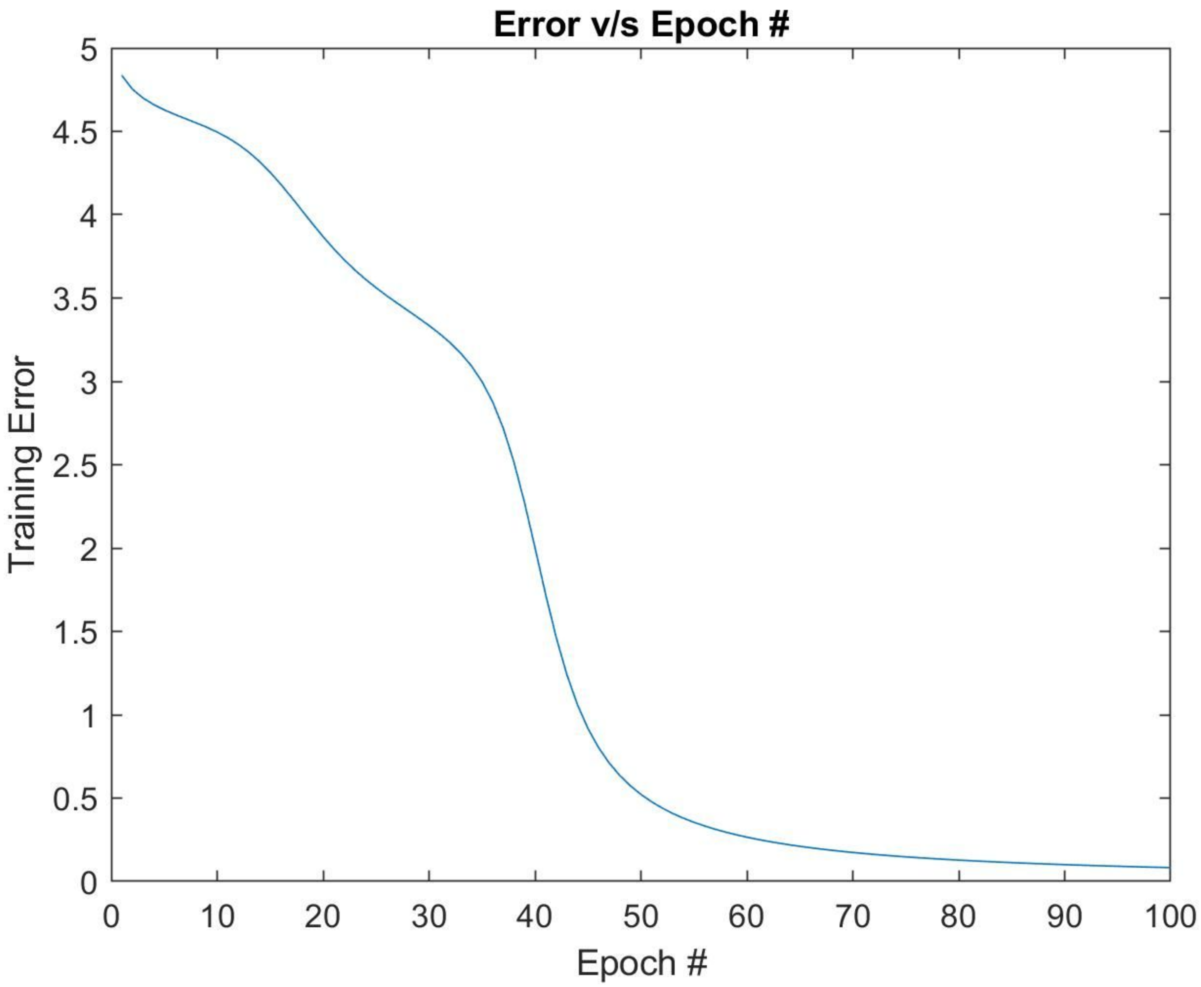


Figure 1: Training Error v/s Epoch using ‘plot’ command for the XOR function

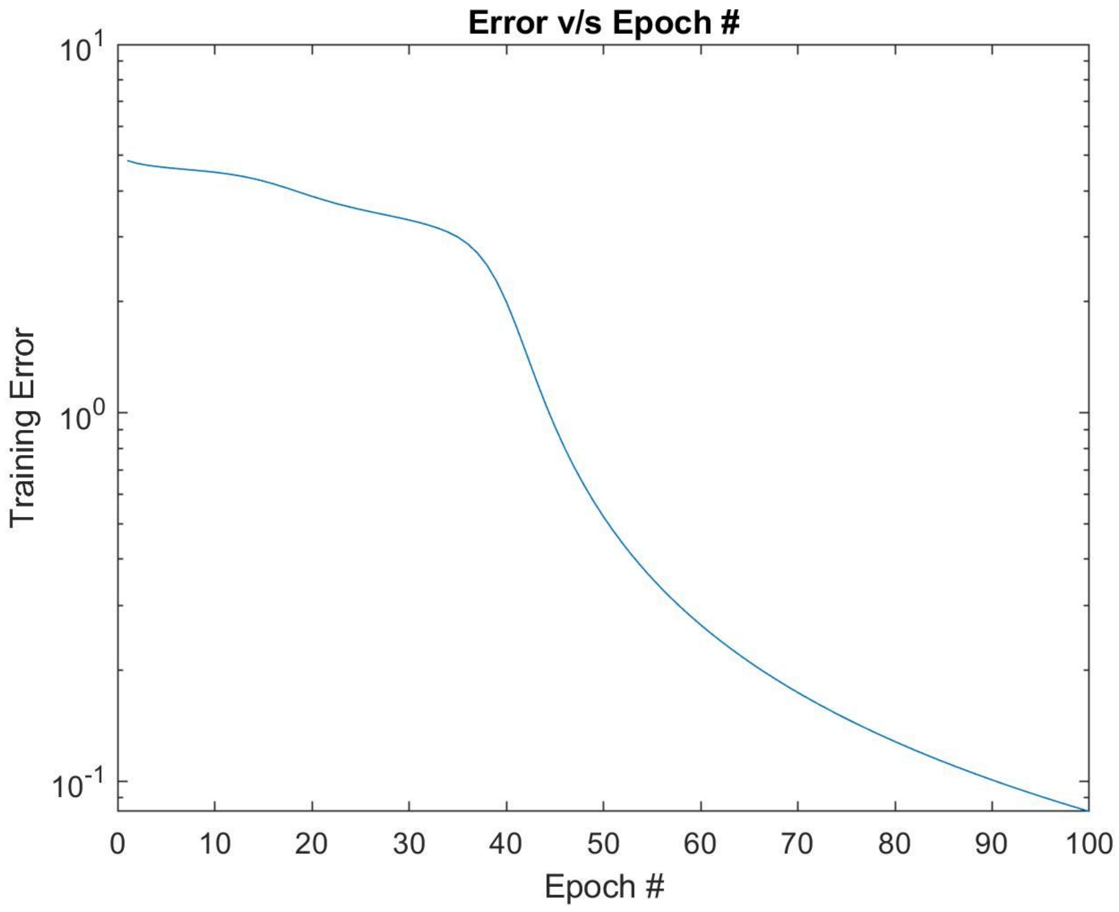


Figure 2: Training Error v/s Epoch using 'semilogy' command for the XOR function

2) **Repeat the same process on the abbreviated "digits" data in the file optdigits.txt. This file can be loaded with**

```
load optdigits.txt
all_labels=optdigits(:,65);
Data = [optdigits(all_labels==9 | all_labels==8,1:64)/256]';
Labels = [optdigits((all_labels==9 | all_labels==8),65)]';
Labels(Labels==9)=-1; Labels(Labels==8)= 1;
initVW; % only for the first run.
```

The data file here consists of 1797 low resolution digit images (8 × 8) with samples of all 10 digits. Each row of the input data consists of 64 pixel values followed by a single label. You must scale the pixel values to the range 0 to 1, and transpose them so each image is in a column. The code above accomplishes this preprocessing.
Train a neural network with one hidden layer with k hidden nodes using k = 18 and initial coefficient matrices in initVW.m. Allow at most 400 epochs.

File: hw4_part2.m
For our solution, we used the given initial parameters and ran the training code for 400 epochs doing cyclic training (Updating the weights based on each training example rather than summing everything up). To plot the error function, we calculated the total error per epoch by summing the error returned by the delta_nn function over each training sample per epoch. The plots are as follows:

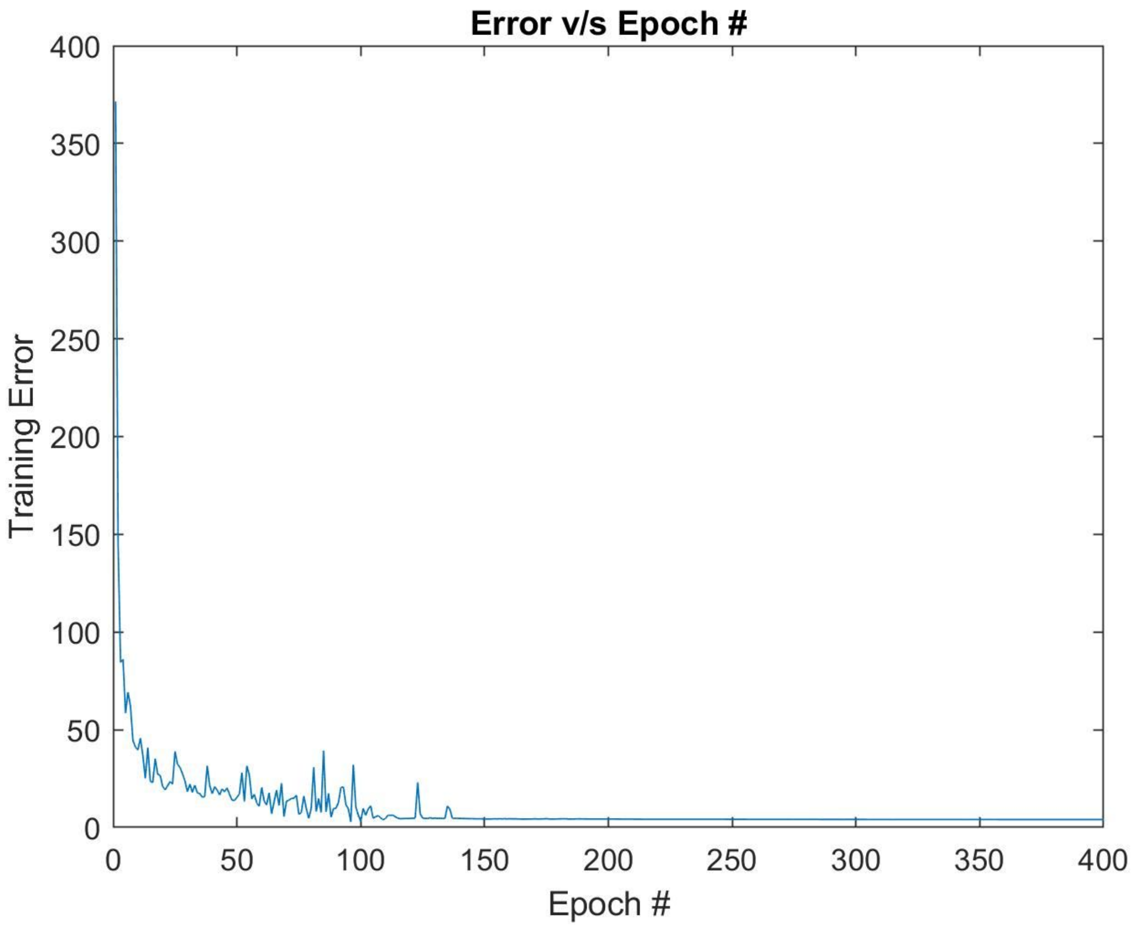


Figure 3: Training Error v/s Epoch using 'plot' command for the opt digits dataset

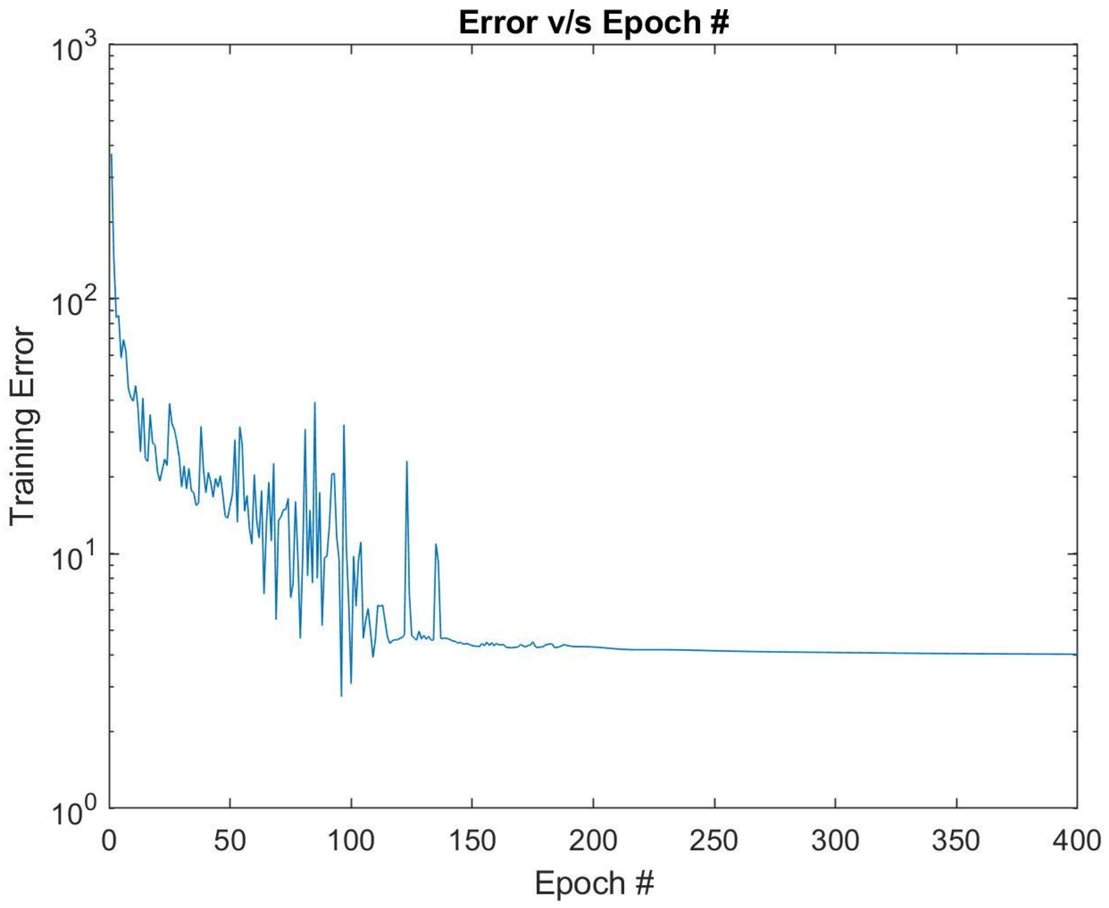


Figure 4: Training Error v/s Epoch using ‘semilogy’ command for the opt digits dataset

3) **Repeat the previous but use the same digits data set used in the previous homework. Use randomly selected initial coefficient matrices, and train on the data with flag values 1, 2, 3. Use the data corresponding to digits 8 & 9 only, and preprocess by PCA to reduce the dimension of the input data to 76, as in the previous homework.**

Use k = 18, and also try as small a k as possible that can yield an error rate under 10% on the data with flag value 4. Use randn to generate random initial coefficient values. For each k, initialize the random number generator with rng('default') for consistency. For each k, you may need to try several random starts to find a converging case, but you should not need to try more than 10 such starts. Then compute the error rate on the test set (flag value 5). In other words, use flag 1,2,3 data as training data and flag 4 data as validation data to tune the parameter k. Flag 5 data represents the future data, hence can be used only to measure the performance but not for tuning. Report the error rate on all three parts of the data: training, validation, test.

Epochs = 300	**Learning rate = 0.01**	**Learning rate = 0.005**
K = 3	Last Training Error: 437.265181 Least Validation Error: 0.115000 Test Error: 0.107500 Best Random Start: 3	Last Training Error: 451.372221 Least Validation Error: 0.110000 Test Error: 0.137500 Best Random Start: 10
K = 5 (Smallest K with Validation Error < 10%, Learning Rate: 0.005)	Last Training Error: 439.578730 Least Validation Error: 0.137500 Test Error: 0.172500 Best Random Start: 7	Last Training Error: 312.023043 Least Validation Error: **0.080000** Test Error: 0.102500 Best Random Start: 1
K = 9 (Smallest K with Validation Error < 10%, Learning Rate: 0.01)	Last Training Error: 314.442921 Least Validation Error: **0.097500** Test Error: 0.132500 Best Random Start: 2	Last Training Error: 355.942857 Least Validation Error: 0.087500 Test Error: 0.137500 Best Random Start: 5
K = 11	Last Training Error: 373.333211 Least Validation Error: 0.125000 Test Error: 0.125000 Best Random Start: 4	Last Training Error: 297.330541 Least Validation Error: 0.082500 Test Error: 0.110000 Best Random Start: 8
K = 15	Last Training Error: 314.320954 Least Validation Error: 0.092500 Test Error: 0.090000 Best Random Start: 7	Last Training Error: 296.767085 Least Validation Error: 0.075000 Test Error: 0.097500 Best Random Start: 4
K = 18 (Best Test Error, Learning Rate: 0.005, Epochs:	Last Training Error: 223.769259	Last Training Error: 225.988518

300)	Least Validation Error: 0.080000 Test Error: 0.085000 Best Random Start: 4	Least Validation Error: 0.082500 Test Error: **0.080000** Best Random Start: 4

Epochs = 100	**Learning rate = 0.01**	**Learning rate = 0.005**
K = 3	Last Training Error: 519.088932 Least Validation Error: 0.165000 Test Error: 0.145000 Best Random Start: 6	Last Training Error: 451.372222 Least Validation Error: 0.110000 Test Error: 0.137500 Best Random Start: 10
K = 5 (Smallest K with Validation Error < 10% with rate = 0.005)	Last Training Error: 439.600696 Least Validation Error: 0.137500 Test Error: 0.172500 Best Random Start: 7	Last Training Error: 313.611997 Least Validation Error: **0.082500** Test Error: 0.102500 Best Random Start: 1
K = 9 (Smallest K with Validation Error < 10% with rate = 0.01)	Last Training Error: 314.489887 Least Validation Error: **0.097500** Test Error: 0.132500 Best Random Start: 2	Last Training Error: 355.922693 Least Validation Error: 0.087500 Test Error: 0.137500 Best Random Start: 5
K = 11	Last Training Error: 373.807965 Least Validation Error: 0.125000 Test Error: 0.125000 Best Random Start: 4	Last Training Error: 297.341998 Validation Error: 0.082500 Test Error: 0.110000 Best Random Start: 8
K = 15	Last Training Error: 333.349252 Least Validation Error: 0.100000 Test Error: 0.120000 Best Random Start: 1	Last Training Error: 289.934580 Least Validation Error: 0.100000 Test Error: 0.105000 Best Random Start: 3
K = 18 (Best Test Error, Learning Rate: 0.005, Epochs: 300)	Last Training Error: 223.460026 Validation Error: 0.082500 Test Error: 0.090000 Best Random Start: 4	Last Training Error: 226.809506 Least Validation Error: 0.087500 **Test Error: 0.075000** Best Random Start: 4

It seems that for the above hyperparameter search, we were able to find very similar results when we changed the number of epochs from 100 to 300. We can attest this to the fact that most of the times with different random restarts, the network converges in these many epochs. Also one thing that we noticed was that for different learning rates like 0.01 and 0.005 the network at least seemed to learn from the training samples. When we tried training with larger values like 0.5 or 0.1, the network would not converge, and if it does the error is very high. Also it seems that increasing k beyond the smallest k value with validation error < 10% doesn't really offer much advantage over it because the network seems to sometimes overfit and lead to worse test and validation accuracies. However in terms of the lowest test error, it still belongs to k = 18, and we think this is because of the case, that we are using the same random initializations over 10 random initialization iterations, hence restricting the amount of parameters space that is being searched and optimized for fitting. With k = 18, more random values are being seen and also the best of all the 10 models are being selected which might lead to a better test set.

4) **Modify the deltaNN.m file as follows and apply the result to the XOR data and the PCA processed digits data.**

a) **The "sigmoid" function in both layers is the hyperbolic tangent tanh. Replace it with the ReLU function ReLU(ˆs) = (0 if ˆs ≤ 0 sˆ if ˆs > 0 or with a smoothed version ReLU(ˆs) = 1 10 log 1 + e 10ˆs Replace only the 'sigmoid' function in the hidden layer. The output layer should keep the 'tanh' function. You must replace the formula for the derivative in the hidden layer within the deltaNN.m file.**

For the XOR function I tried the following different hyperparameters:

Learning Rate	Number of Epochs	Training Error (last Epoch)	Accuracy
0.05	100	0.20672	1
0.05	200	0.041138	1
0.05	1600	0.002646	1
0.1	100	2.879	0.75
0.1	200	2.8744	0.75
0.1	1600	0.002042	1.00
0.2	100	3.059482	0.75
0.2	200	3.057907	0.75
0.2	1600	3.056758	0.75

All the plots with the training error plotted v/s epoch, refer to Appendix A. If we analyze the trends we can see that the learning rate of 0.1 and 0.2 are pretty big, although given enough epochs for the learning rate 0.1, the network converges. It could be concluded that the learning rate greater than 0.1 like 0.2 or something bigger would not be an ideal choice because it does not lead to convergence. When the learning rate is reduced to something like 0.05, the training error seems to decrease continuously and is converging faster than the learning rate 0.1. We can see that for learning rate 0.05 converges to a value of less than 10^{-2}, in around 500 epochs, while learning rate 0.1 converges to a value of less than 10^{-2}, in around 700 epochs. So it seems it is best to use the learning rate 0.05 with epochs around 800. One interesting thing that we see with epochs = 1600, is that for learning rate 0.1 the final training error rate is less than that of the learning rate 0.05 which is because of the sudden jump towards the local minima. Analyzing all the information we can say the lr = 0.05, epochs = 1600 is the best configuration.

For the PCA Digits Dataset, the information is as follows:
First I tried without normalizing the original dataset. When I did this, and just applied the Smooth Relu function over the PCA reduced dataset, I was not able to get a convergence over the dataset. I tried different values of parameters like:

- K: 18
  Learning Rate: 0.003
  Epochs: 300
  Notes: “Did not converge”
- K: 10
  Learning Rate: 0.001
  Epochs: 300
  Notes: “Did not converge”
- K: 20
  Learning Rate: 0.003
  Epochs: 300
  Notes: “Did not converge”
- K: 30
  Learning Rate: 0.003
  Epochs: 300
  Notes: “Did not converge”
- K: 30
  Learning Rate: 0.001
  Epochs: 300
  Notes: “Did not converge”
- K: 15
  Learning Rate: 0.01
  Epochs: 300
  Notes: “Did not converge”

After further inspection I found out that because I did not normalize the values it was leading to the problem of exploding gradients. Because of this the network did not perform good at all. So I normalized the dataset before applying the dataset and finally trained it. All the plots with the training error plotted v/s epoch, refer to Appendix B. The following are the analysis:

1. k: 20, Learning rate: 0.000100, Num Epochs: 300, Last Training Error: 251.075581
   Best Random Start: 9, Validation Error: 0.102500
   Test Error: 0.107500
2. k: 18, Learning rate: 0.003000, Num Epochs: 200, Last Training Error: 78.252879
   Best Random Start: 5, Validation Error: 0.037500
   Test Error: 0.047500
3. k: 15, Learning rate: 0.010000, Num Epochs: 200, Last Training Error: 60.002676
   Best Random Start: 6, Validation Error: 0.025000
   Test Error: 0.030000
4. k: 12, Learning rate: 0.010000, Num Epochs: 200, Last Training Error: 48.086744
   Best Random Start: 8, Validation Error: 0.025000
   Test Error: 0.042500
5. k: 15, Learning rate: 0.010000, Num Epochs: 300, Last Training Error: 60.001844
   Best Random Start: 6, Validation Error: 0.025000
   Test Error: 0.030000
6. k: 9, Learning rate: 0.010000, Num Epochs: 200, Last Training Error: 64.001800
   Best Random Start: 1, Validation Error: 0.035000
   Test Error: 0.040000

7. **k: 15, Learning rate: 0.050000, Num Epochs: 150, Last Training Error: 31.999970**
   Best Random Start: 8, Validation Error: 0.020000
   Test Error: 0.027500
8. k: 15, Learning rate: 0.100000, Num Epochs: 100, Last Training Error: 40.000016
   Best Random Start: 2, Validation Error: 0.025000
   Test Error: 0.032500

From the simple hyperparameter searching, I see that one of the best configurations was the one marked by bold in the above bullets. It seems that for larger k (> 15), the network overfits the data, while for smaller k (k < 15) the network seems to underfit the dataset. We tried running the = 15 case with large number of epochs, and found out that the network doesn't really learn anything more. It remains same. Also it seems that the learning rate is less than 0.01 is very less, however increasing the learning rate (btw 0.01 - 0.1) seems to reach a faster convergence and seems to have better training. However, we cannot really use a large learning rate lie 0.2 because it does not converge and leads to NaN value.

b) **Add a third layer to the network (keep the 'tanh' function). You must modify the deltaNN.m file to accommodate a third layer. It is easier to let z remain the output layer, y be the second hidden layer, turn x into the first hidden layer, and add a new input layer with w, connecting it with new coefficient matrix U. Apply to the XOR data using 2 nodes in the first hidden layer and 3 in the second (plus the biases). Hint: you must copy the derivative formulas for the old hidden layer and modify them to become the derivative formulas for the new hidden layer.**

For the XOR Function the different values are as follows:
k1: 2, k2: 3, Learning rate: 0.050000, Num Epochs: 100, Last Training Error: 0.243700, Accuracy Error: 0.000000
k1: 2, k2: 3, Learning rate: 0.050000, Num Epochs: 200, Last Training Error: 0.020692, Accuracy Error: 0.000000
k1: 2, k2: 3, Learning rate: 0.050000, Num Epochs: 1600, Last Training Error: 0.001493, Accuracy Error: 0.000000
k1: 2, k2: 3, Learning rate: 0.100000, Num Epochs: 100, Last Training Error: 0.019515, Accuracy Error: 0.000000
k1: 2, k2: 3, Learning rate: 0.100000, Num Epochs: 200, Last Training Error: 0.006771, Accuracy Error: 0.000000
k1: 2, k2: 3, Learning rate: 0.100000, Num Epochs: 1600, Last Training Error: 0.000752, Accuracy Error: 0.000000
k1: 2, k2: 3, Learning rate: 0.200000, Num Epochs: 100, Last Training Error: 0.005571, Accuracy Error: 0.000000
k1: 2, k2: 3, Learning rate: 0.200000, Num Epochs: 200, Last Training Error: 0.002548, Accuracy Error: 0.000000
k1: 2, k2: 3, Learning rate: 0.200000, Num Epochs: 1600, Last Training Error: 0.000266, Accuracy Error: 0.000000

It seems that for the XOR dataset, the data converges faster and to a much less training error value. Also it seems that for each training run the final training accuracy is 100% for all the training examples. It seems that for large learning rate the overall convergence is reached faster. Also for higher number of epochs, the training error just reduces even further. However it is worth noting that for the purpose of training a good enough model, we can just use lr = 0.2, and number of epochs = 100 as it has a very less training error overall and yet perfect accuracy.

Extra Credit 3 Layer NN on PCA Digits Dataset:

1. k1: 15, k2: 10, Learning rate: 0.050000, Num Epochs: 100, Last Training Error: 4.067764
   Best Random Start: 1, Validation Error: 0.037500
   Test Error: 0.065000
2. k1: 12, k2: 12, Learning rate: 0.050000, Num Epochs: 100, Last Training Error: 24.048267
   Best Random Start: 1, Validation Error: 0.017500
   Test Error: 0.040000
3. k1: 10, k2: 5, Learning rate: 0.050000, Num Epochs: 100, Last Training Error: 16.117536
   Best Random Start: 1, Validation Error: 0.040000
   Test Error: 0.037500
4. **k1: 5, k2: 5, Learning rate: 0.050000, Num Epochs: 100, Last Training Error: 6.673778**
   Best Random Start: 4, Validation Error: 0.025000
   Test Error: 0.017500
5. k1: 3, k2: 3, Learning rate: 0.100000, Num Epochs: 100, Last Training Error: 30.169063
   Best Random Start: 2, Validation Error: 0.020000
   Test Error: 0.027500
6. k1: 5, k2: 5, Learning rate: 0.200000, Num Epochs: 100, Last Training Error: 98.288972
   Best Random Start: 4, Validation Error: 0.025000
   Test Error: 0.040000
7. k1: 5, k2: 5, Learning rate: 0.100000, Num Epochs: 200, Last Training Error: 16.047847
   Best Random Start: 2, Validation Error: 0.025000
   Test Error: 0.022500
8. k1: 20, k2: 20, Learning rate: 0.100000, Num Epochs: 100, Last Training Error: 20.003248
   Best Random Start: 1, Validation Error: 0.022500
   Test Error: 0.030000

For the hyperparameter searching, It seems that one of the best performance was given by a network with two hidden layers of 5 nodes each with learning rate 0.05 and number of epochs 100. It seems that with layers having less number of nodes then the network underfits and the overall accuracy decreases. Also similarly with the network with much more number of nodes, tends to overfit the dataset and thus does not give almost that high accuracy. One other interesting thing that we notice is that the high learning rate tends to converge in more time, and similarly for low learning rate. However with the learning rate set to the values like 0.05 and 0.01, the convergence and the error seems to be the best.

EXTRA CREDIT

For extra credit, I tried to train a Deep Convolutional Neural Network over the Digits Dataset. In file: extra_credit.m. I do not really have much experience with training Deep Convolutional Neural Networks, however I tried training it for different CNN networks as follows:

- 20 Convolutional Filters (5 * 5)
  1 Batch Normalization Layer
  ReLU Non-Linearity
  Fully Connection Layer
  Softmax

  Training Accuracy: 78%
  Validation Accuracy: 57%
  Testing Accuracy: 52%

- 10 Convolutional Filters (5 * 5)
  1 Batch Normalization Layer
  ReLU Non-Linearity
  Fully Connection Layer
  Softmax

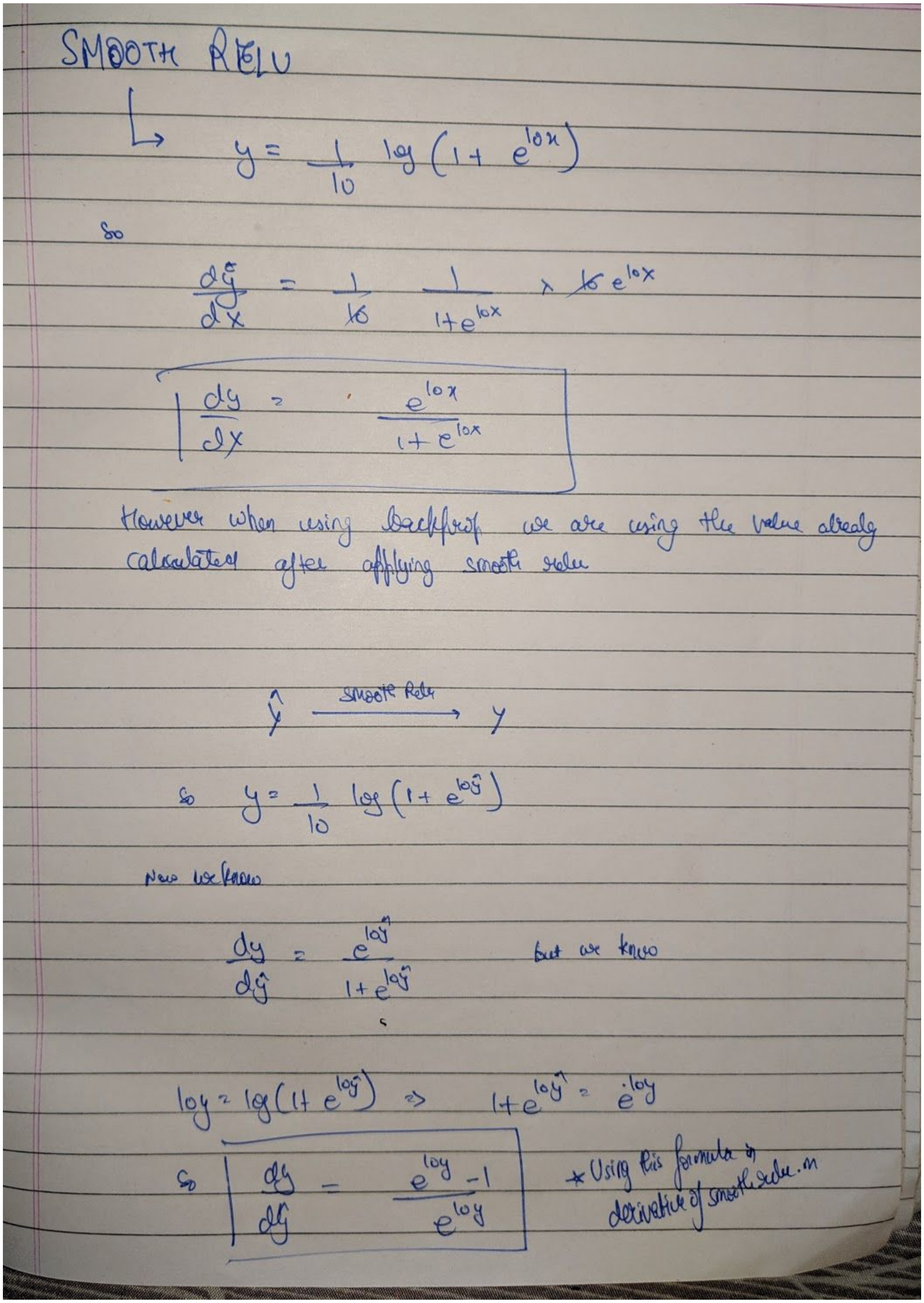
  Training Accuracy: 66%
  Validation Accuracy: 54%
  Testing Accuracy: 51%

- 20 Convolutional Filters (5 * 5)
  1 Batch Normalization Layer
  ReLU Non-Linearity
  10 Convolutional Filters (5 * 5)
  1 Batch Normalization Layer
  ReLU Non-Linearity
  Fully Connection Layer
  Softmax

  Training Accuracy: 65%
  Validation Accuracy: 55%
  Testing Accuracy: 51.08%

I was not able to get a good enough validation accuracy. I think this is because the network is overfitting the dataset, because of the number of examples v/s the number of parameters to train.

Smooth Relu Mathematics



SMOOTH RELU

↳ $y = \frac{1}{10}\log\left(1+e^{10x}\right)$

So

$$\frac{dy}{dx} = \frac{1}{\cancel{10}}\,\frac{1}{1+e^{10x}} \times \cancel{10}\,e^{10x}$$

$$\boxed{\frac{dy}{dx} = \frac{e^{10x}}{1+e^{10x}}}$$

However when using backprop we are using the value already calculated after applying smooth relu

$$\hat{y} \xrightarrow{\text{Smooth Relu}} y$$

So $y = \frac{1}{10}\log\left(1+e^{10\hat{y}}\right)$

Now we know

$$\frac{dy}{d\hat{y}} = \frac{e^{10\hat{y}}}{1+e^{10\hat{y}}}$$ but we know

$$10y = \log\left(1+e^{10\hat{y}}\right) \Rightarrow 1+e^{10\hat{y}} = e^{10y}$$

So $$\boxed{\frac{dy}{d\hat{y}} = \frac{e^{10y}-1}{e^{10y}}}$$

* Using this formula in derivative of smooth relu.

<u>APPENDIX A</u>: Plots for XOR Data using RELU Network

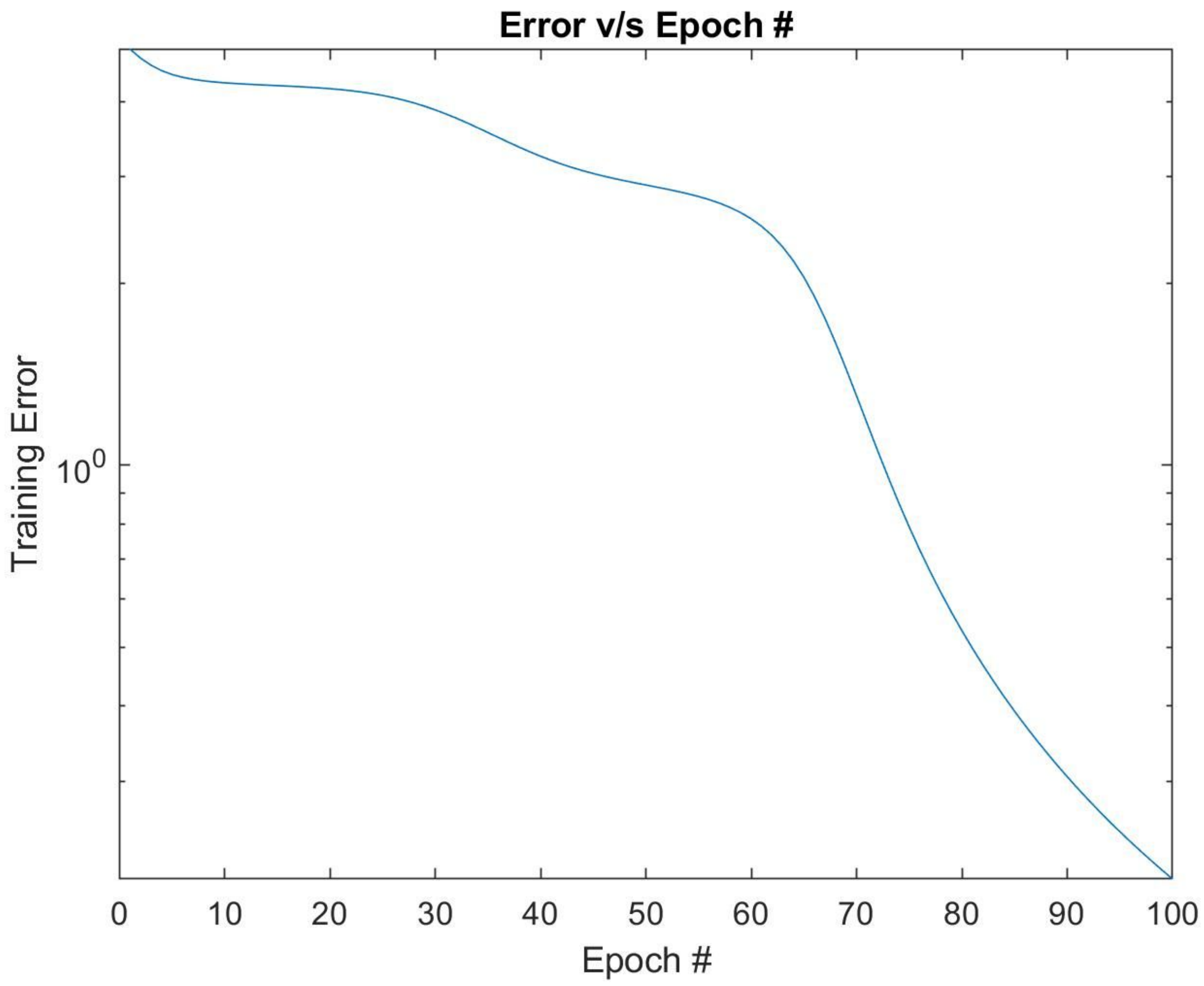


Figure A.1: Training Error v/s Epoch using 'semilogy' command for the XOR dataset with Epochs = 100, Learning Rate = 0.05

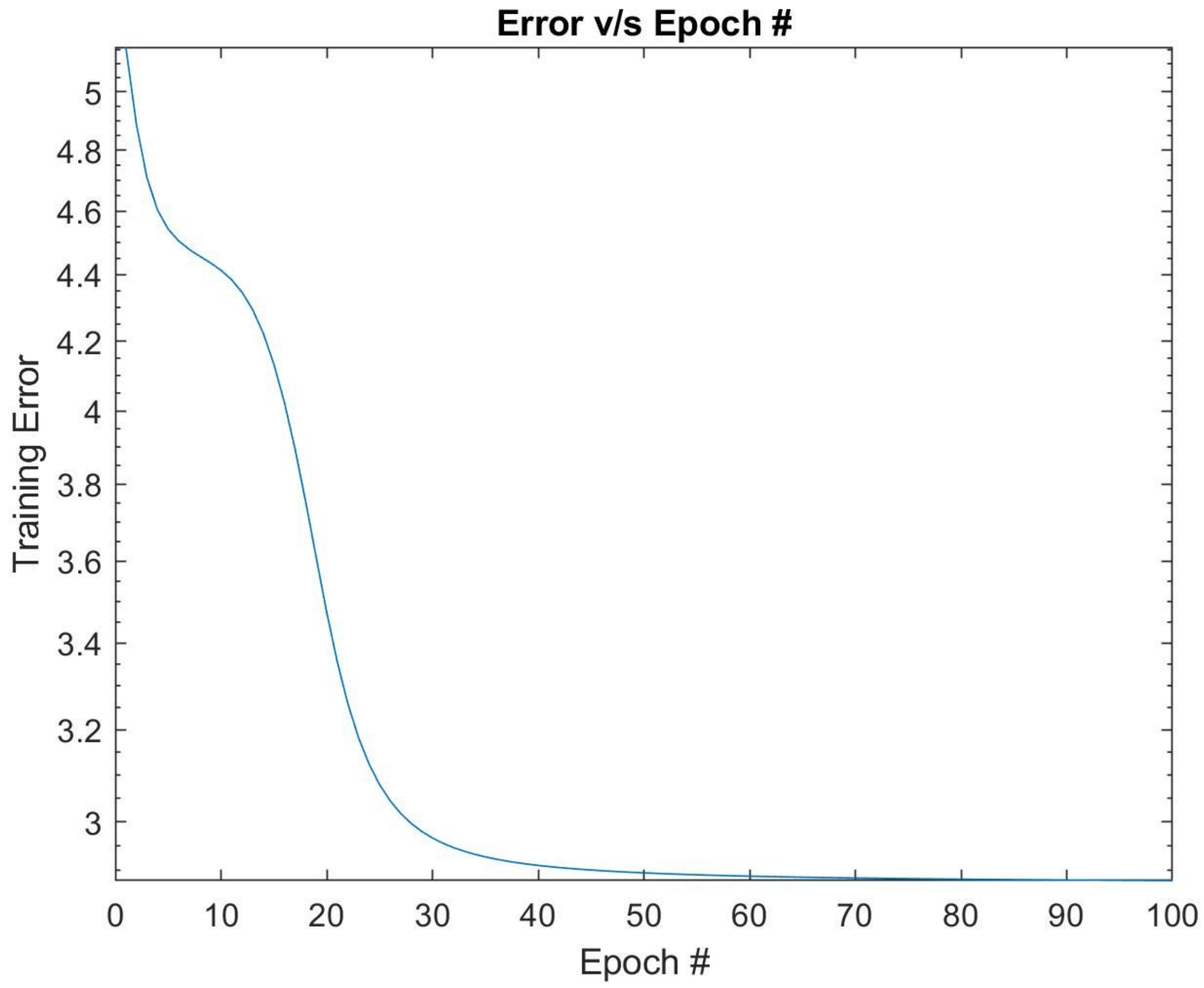


Figure A.2: Training Error v/s Epoch using 'semilogy' command for the XOR dataset with Epochs = 100, Learning Rate = 0.1

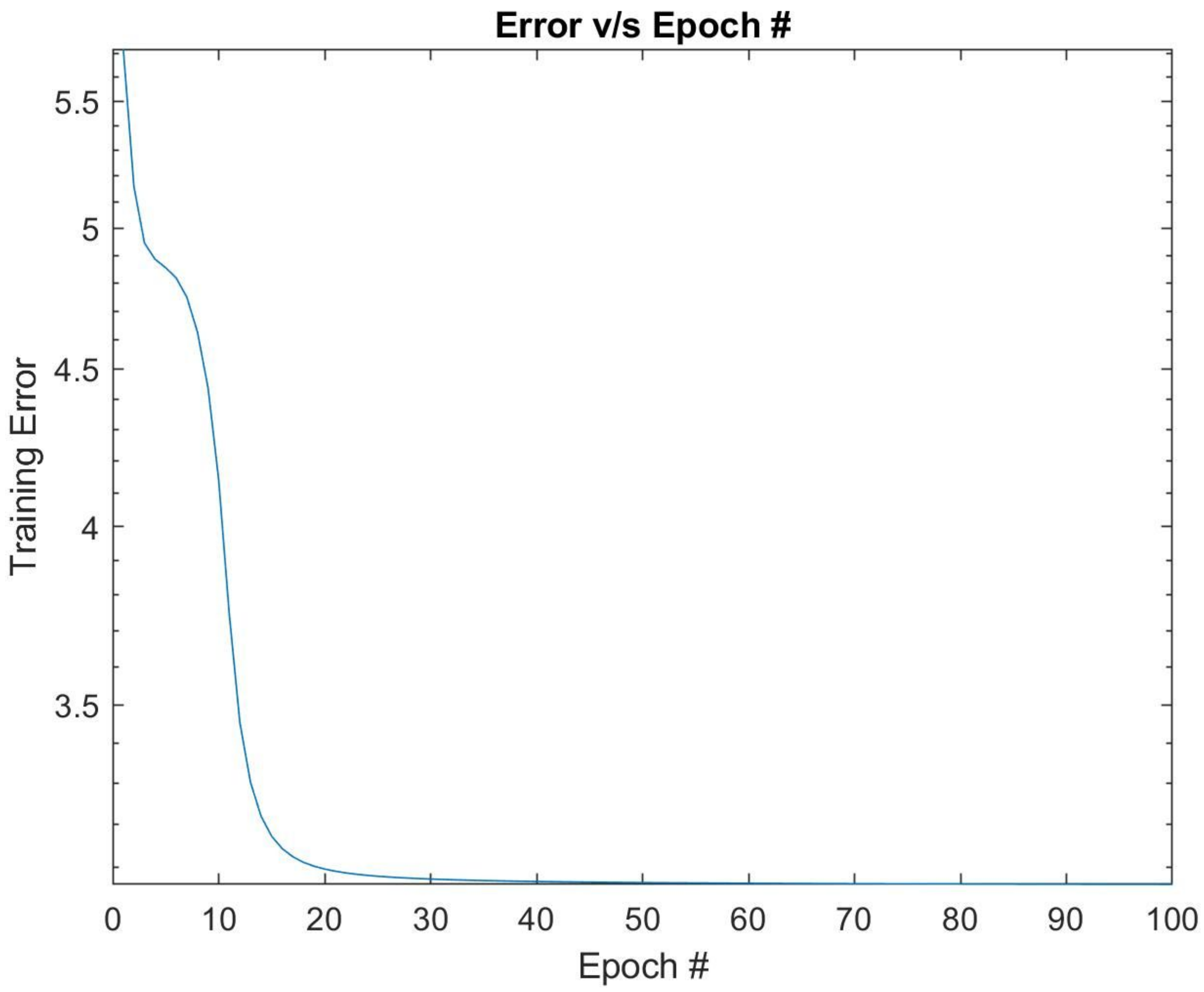


Figure A.3: Training Error v/s Epoch using 'semilogy' command for the XOR dataset with Epochs = 100, Learning Rate = 0.2

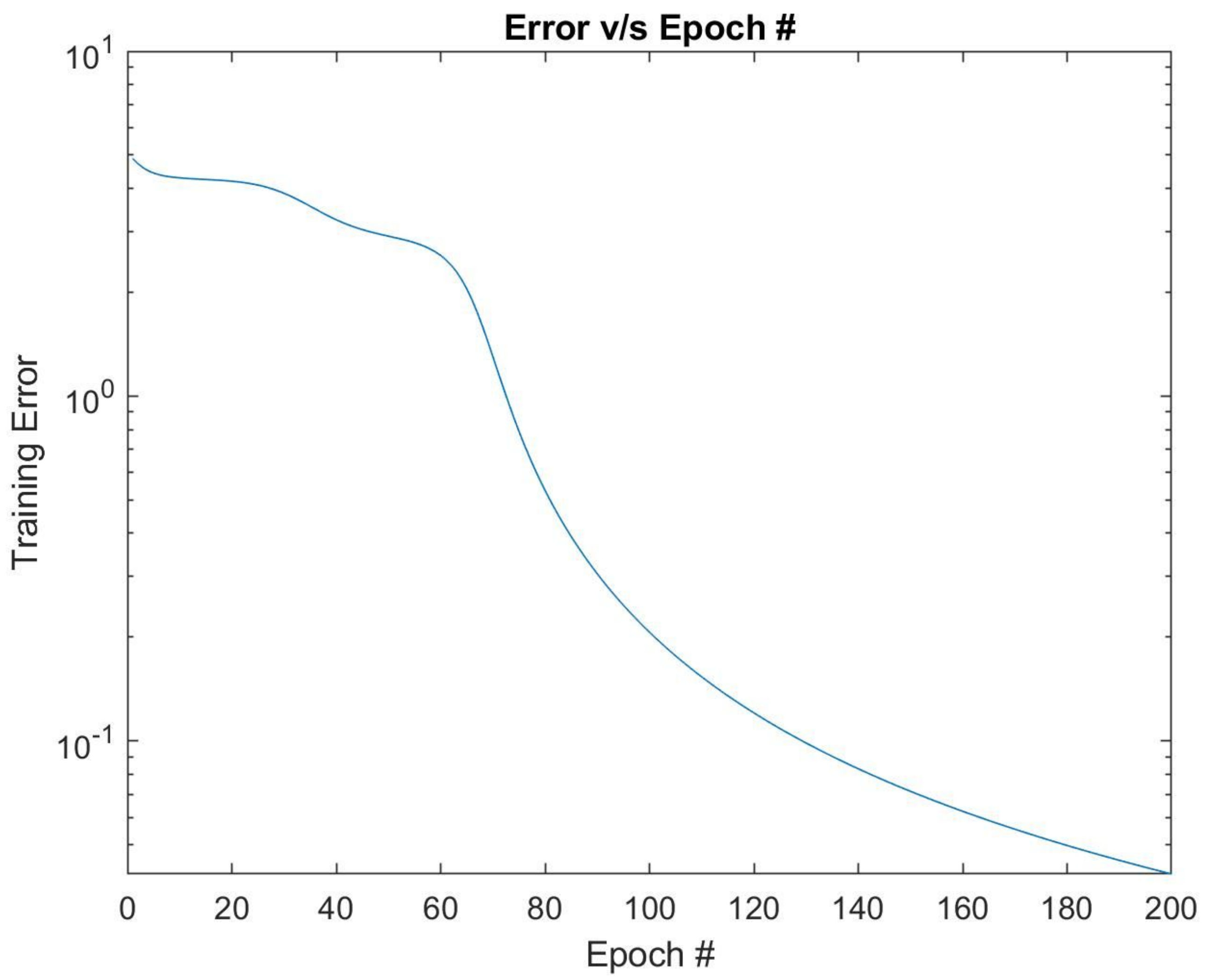


Figure A.4: Training Error v/s Epoch using ‘semilogy’ command for the XOR dataset with Epochs = 200, Learning Rate = 0.05

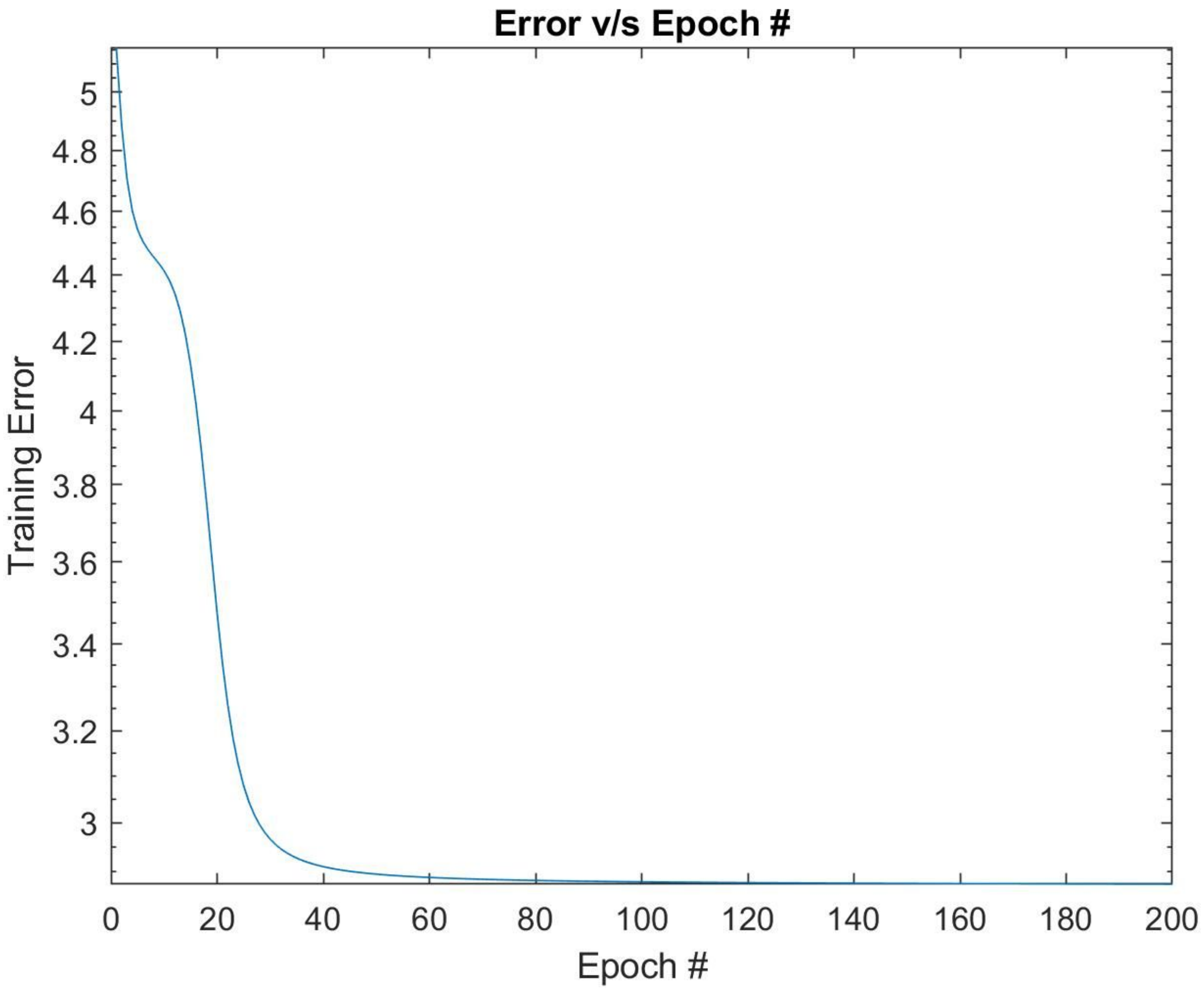


Figure A.5: Training Error v/s Epoch using 'semilogy' command for the XOR dataset with Epochs = 200, Learning Rate = 0.1

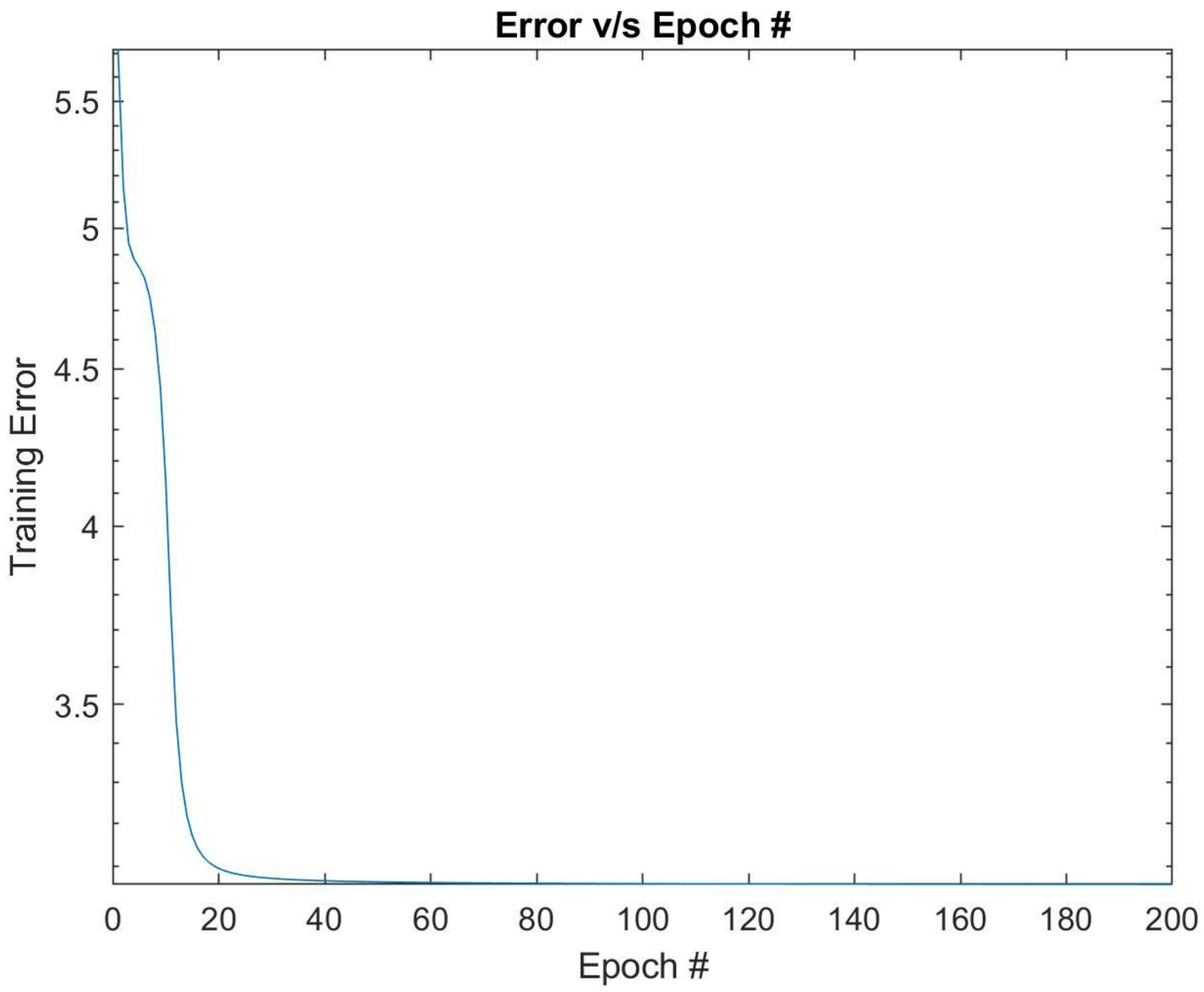


Figure A.6: Training Error v/s Epoch using ‘semilogy’ command for the XOR dataset with Epochs = 100, Learning Rate = 0.2

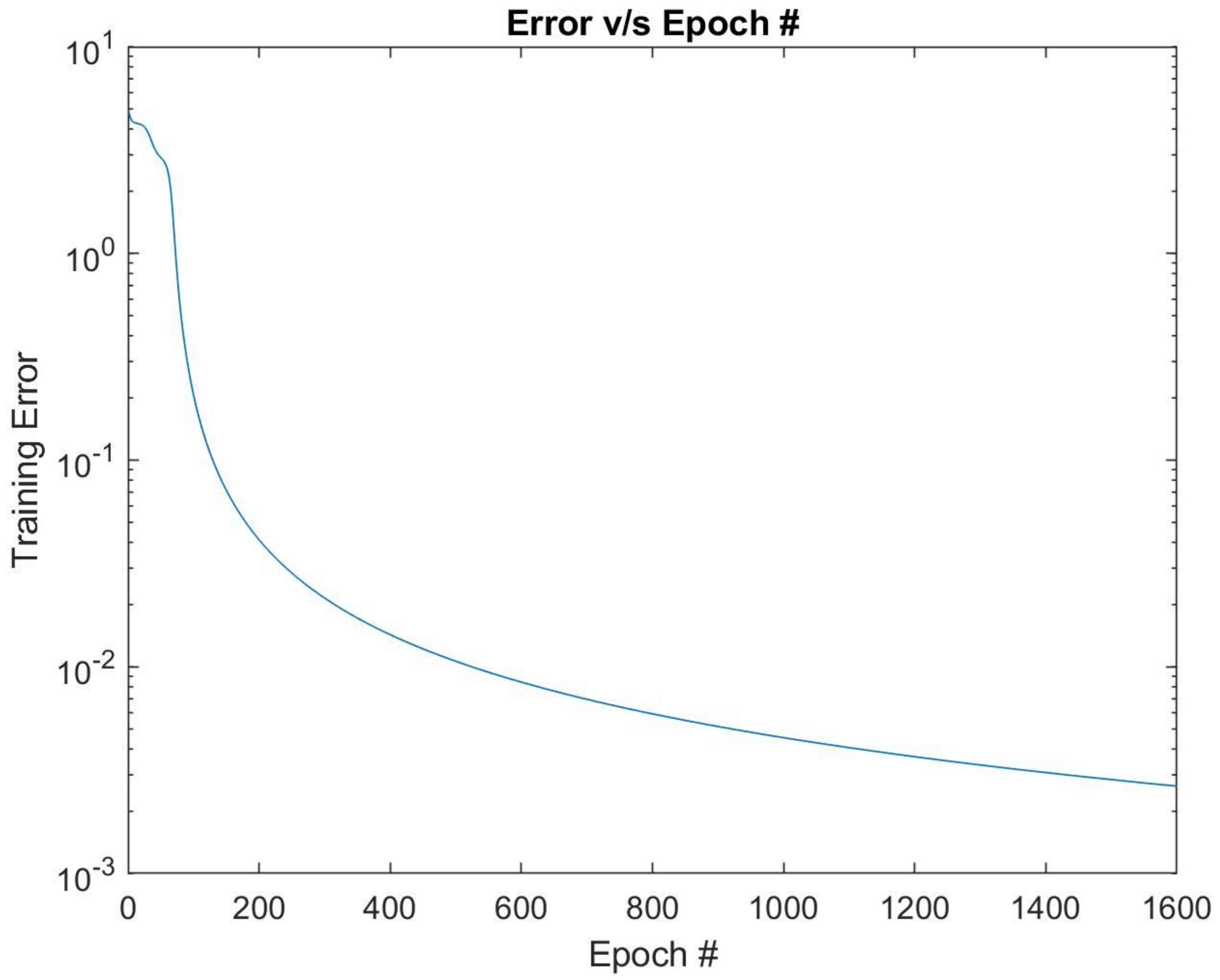


Figure A.7: Training Error v/s Epoch using 'semilogy' command for the XOR dataset with Epochs = 1600, Learning Rate = 0.05

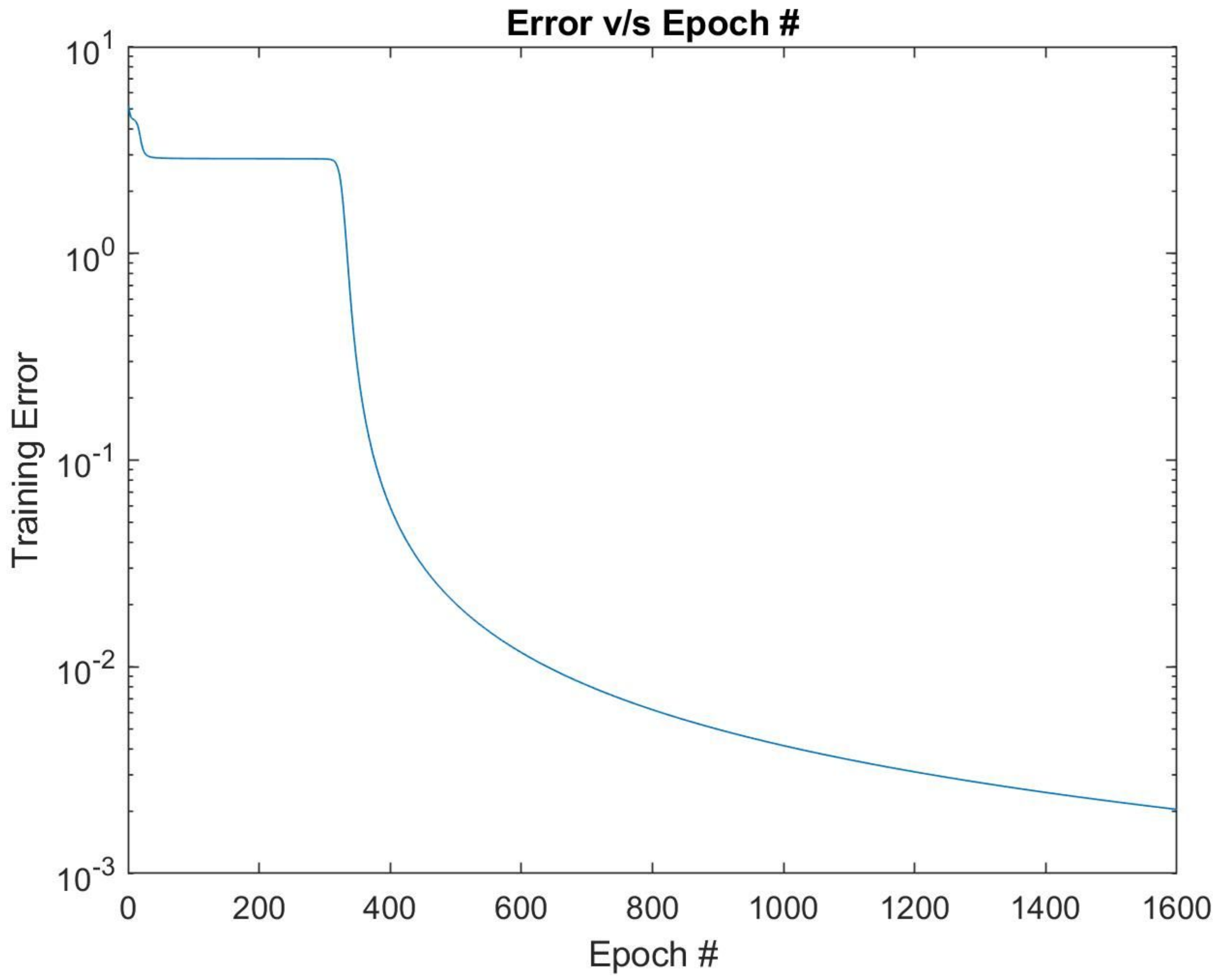


Figure A.8: Training Error v/s Epoch using 'semilogy' command for the XOR dataset with Epochs = 1600, Learning Rate = 0.1

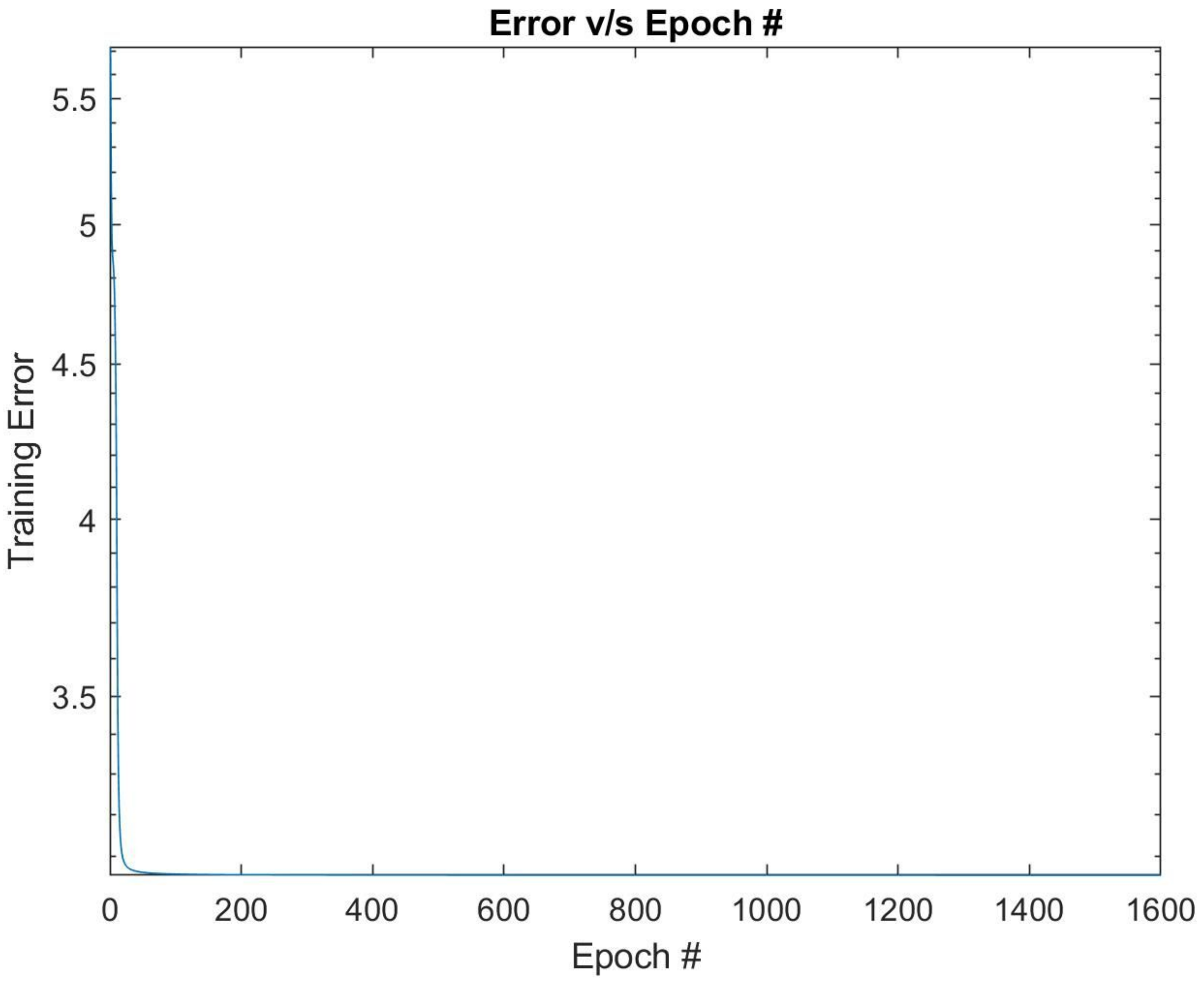


Figure A.9: Training Error v/s Epoch using 'semilogy' command for the XOR dataset with Epochs = 1600, Learning Rate = 0.2

<u>APPENDIX B</u>: Plots for PCA Digits Dataset for RELU Neural Network

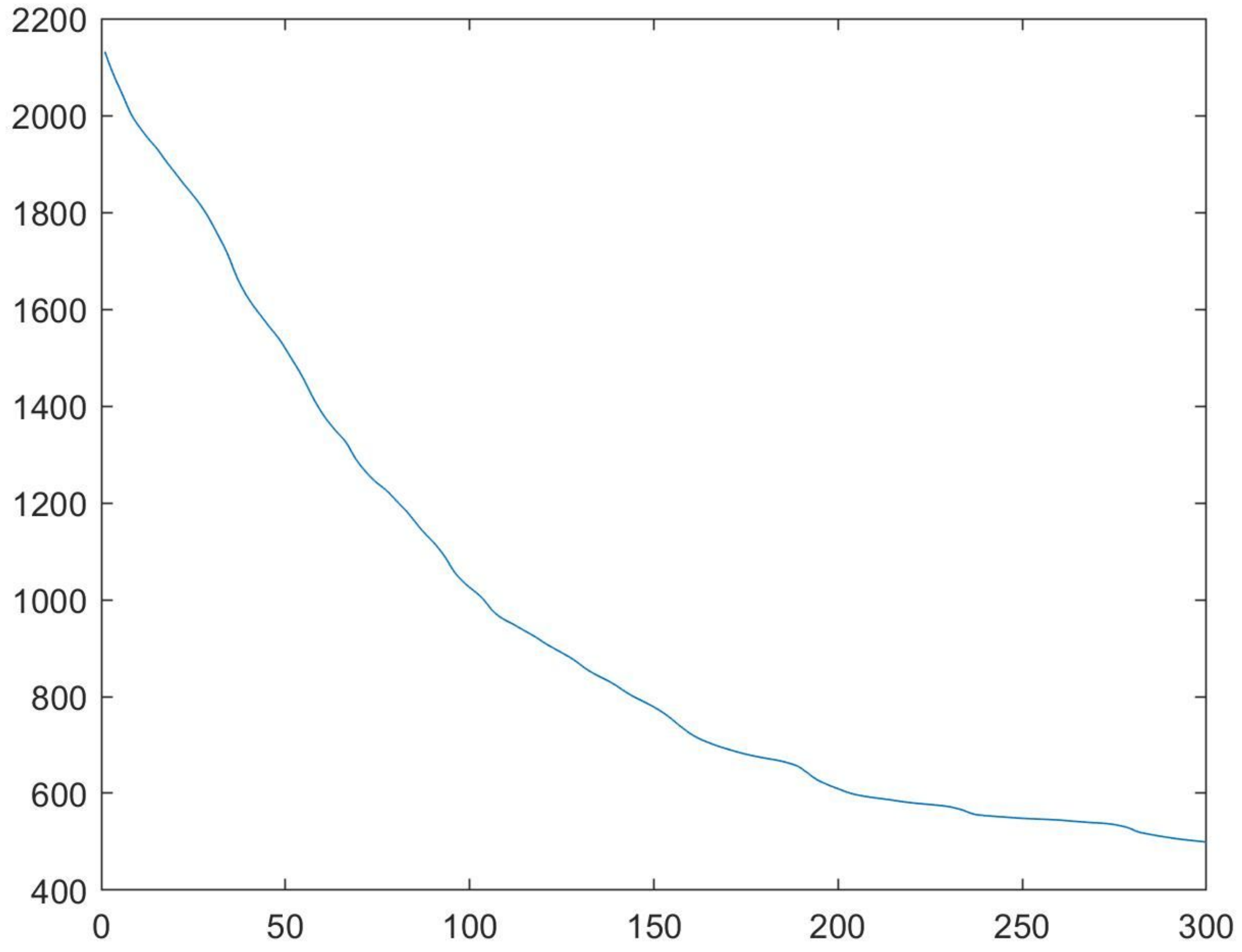


Figure B.1: Training Error v/s Epoch using 'semilogy' command for the Reduced PCA dataset for observation 1

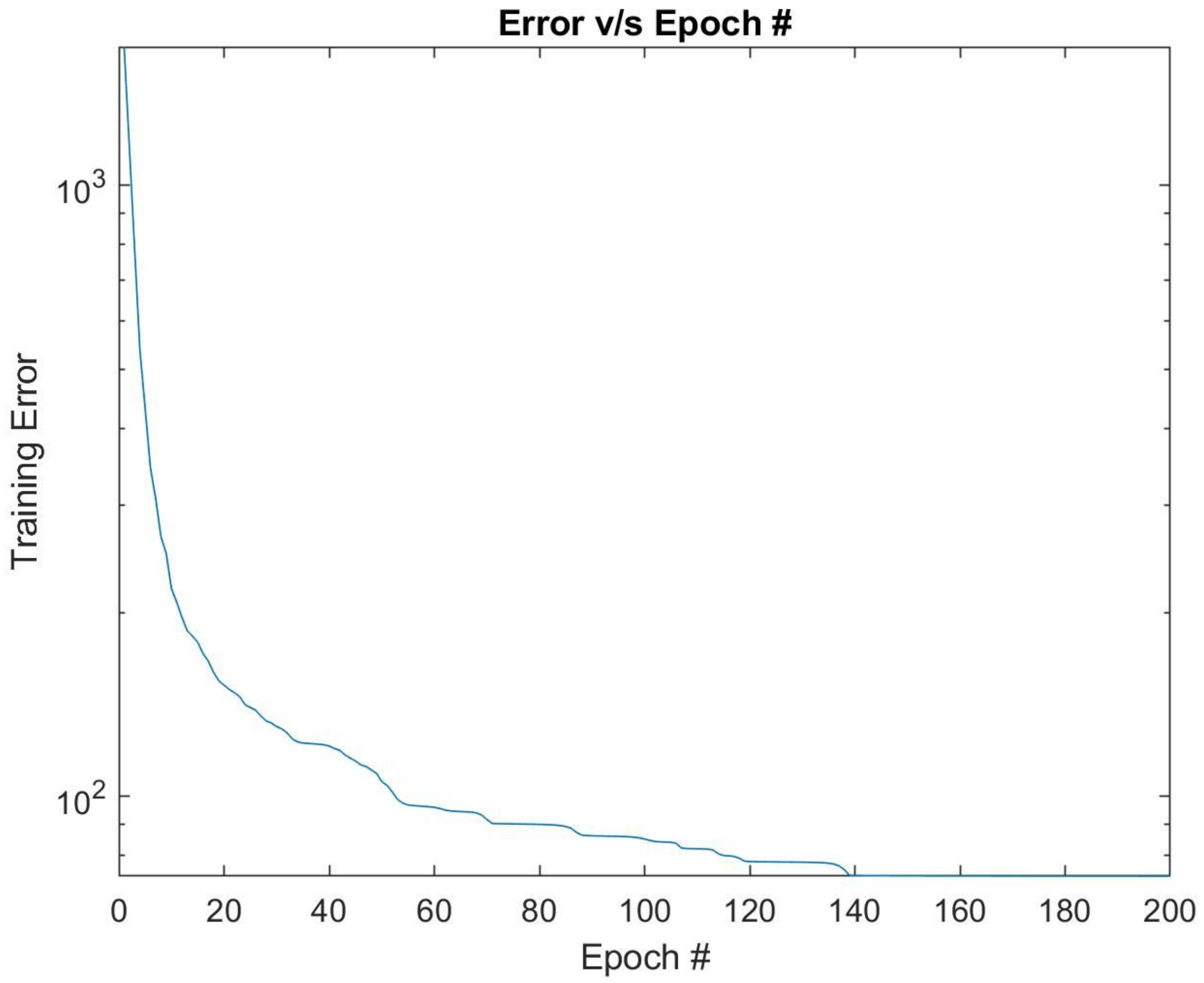


Figure B.2: Training Error v/s Epoch using 'semilogy' command for the Reduced PCA dataset for observation 2

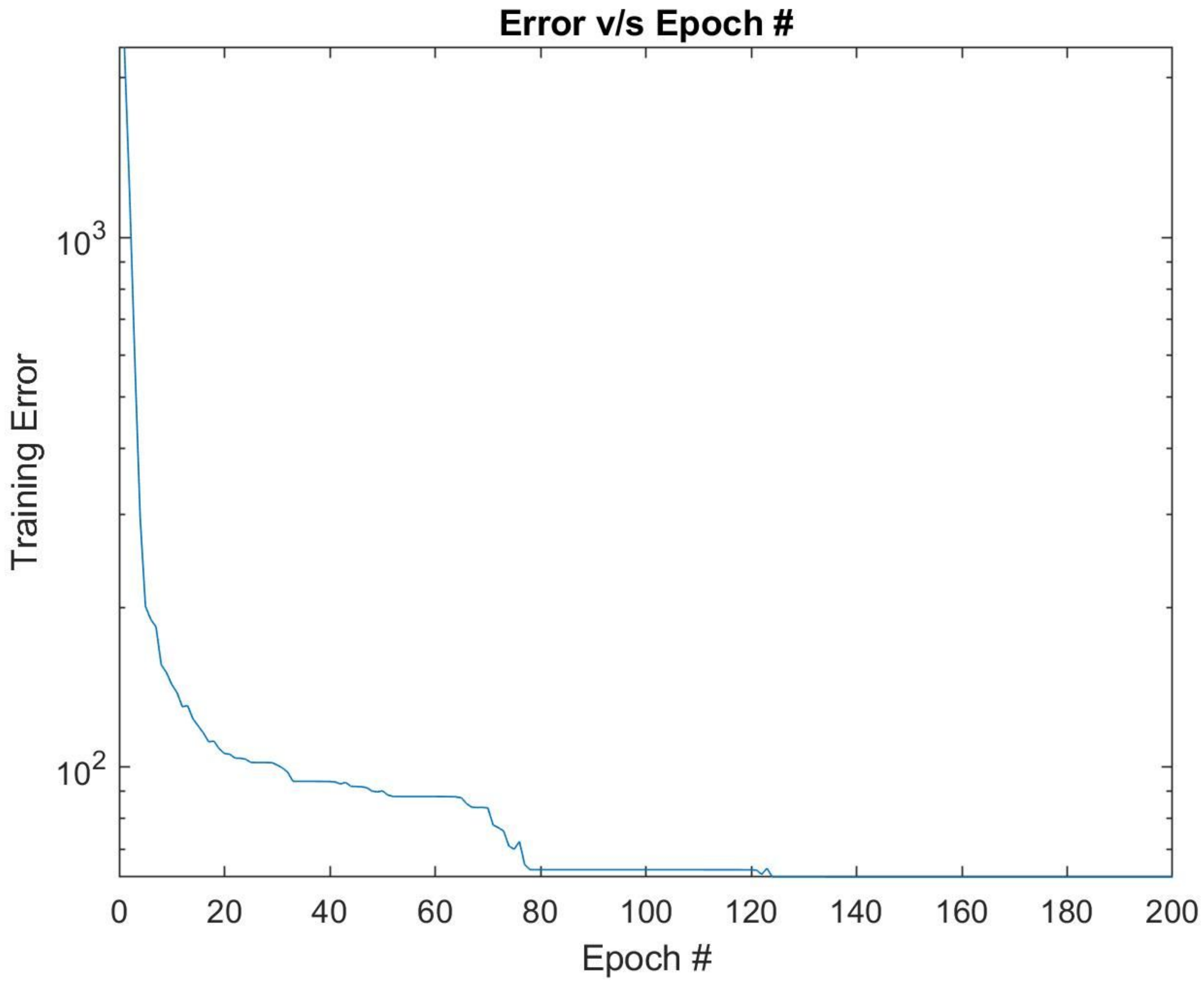


Figure B.3: Training Error v/s Epoch using ‘semilogy’ command for the Reduced PCA dataset for observation 3

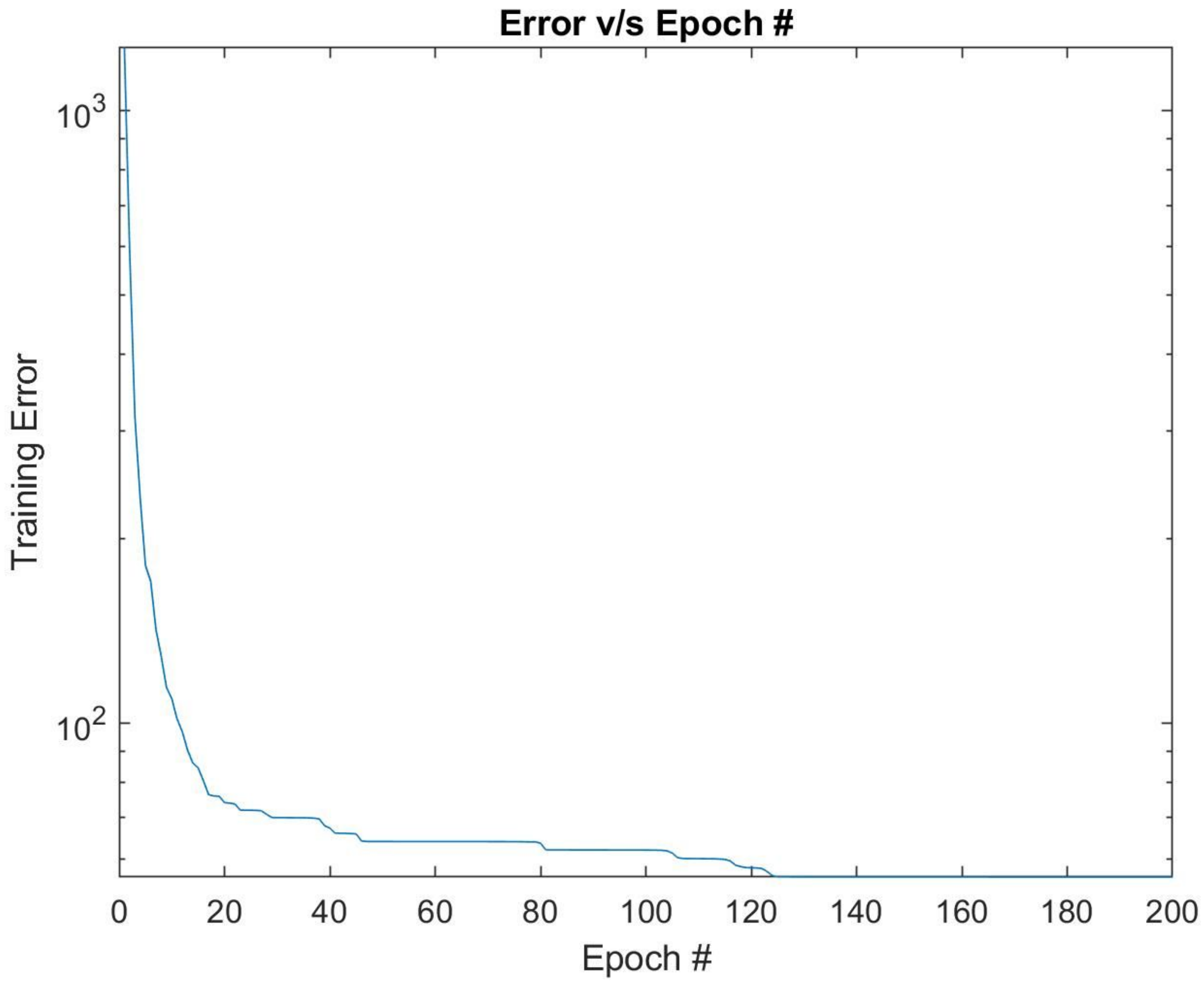


Figure B.4: Training Error v/s Epoch using ‘semilogy’ command for the Reduced PCA dataset for observation 4

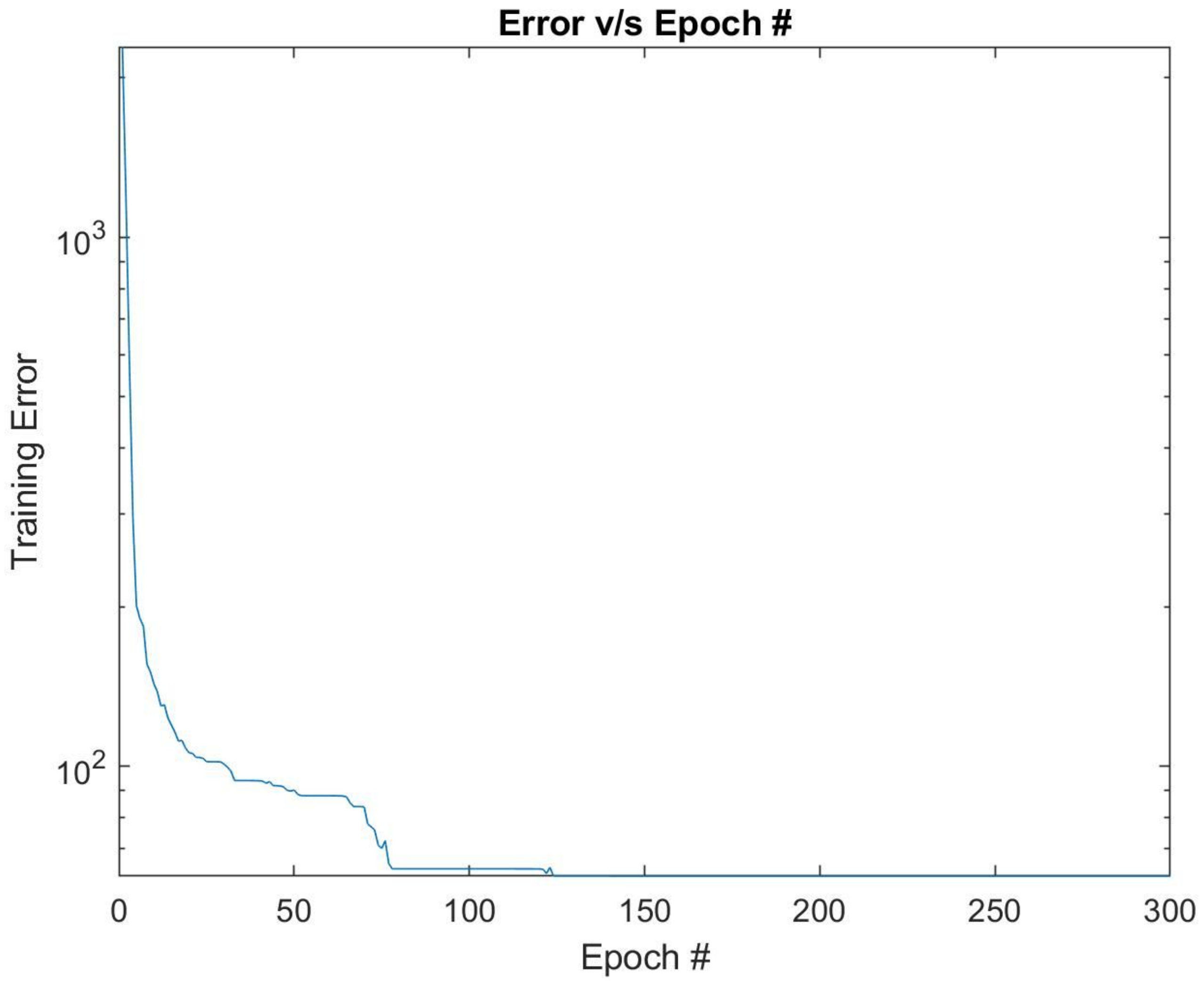


Figure B.5: Training Error v/s Epoch using 'semilogy' command for the Reduced PCA dataset for observation 5

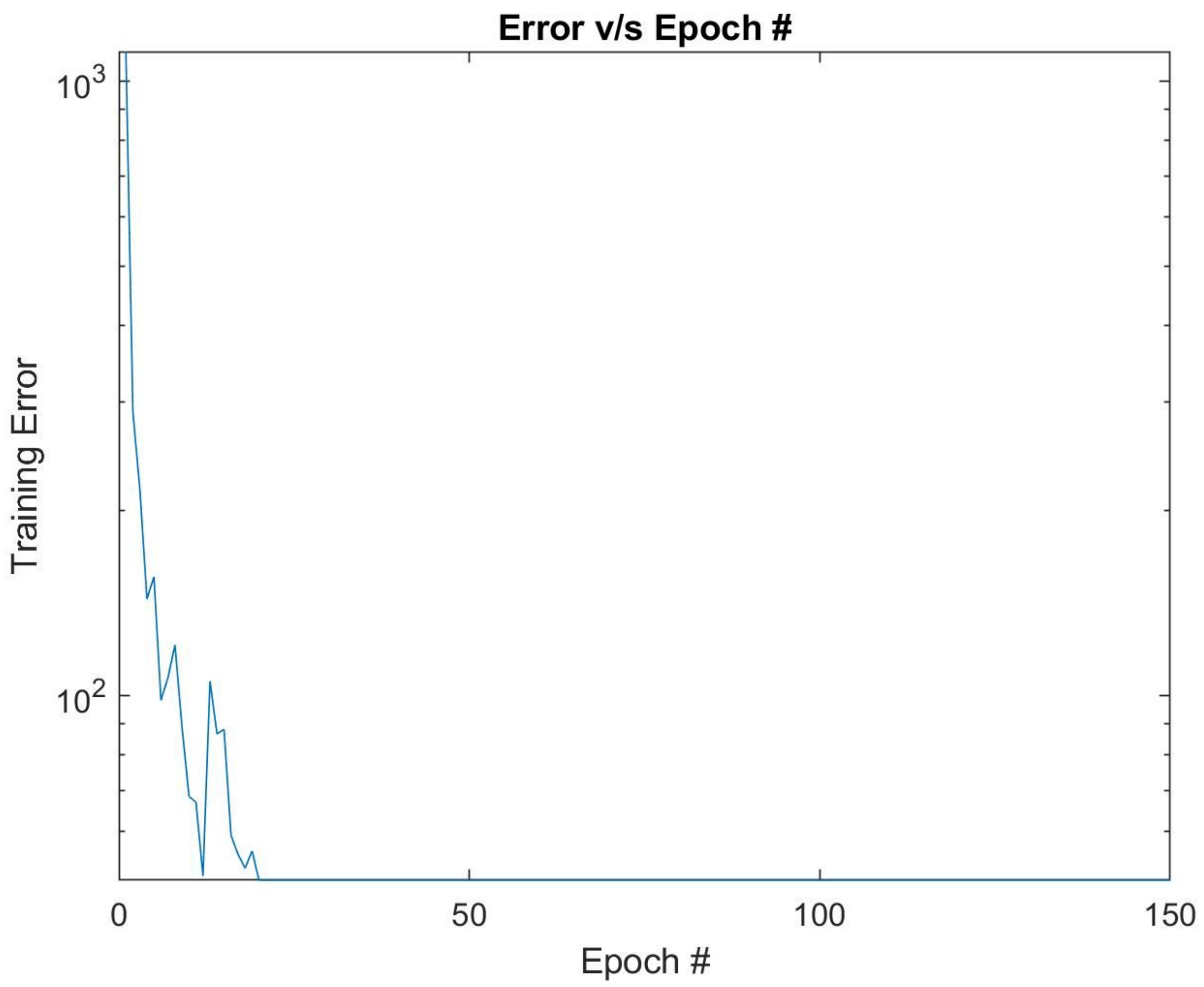


Figure B.7: Training Error v/s Epoch using 'semilogy' command for the Reduced PCA dataset for observation 7

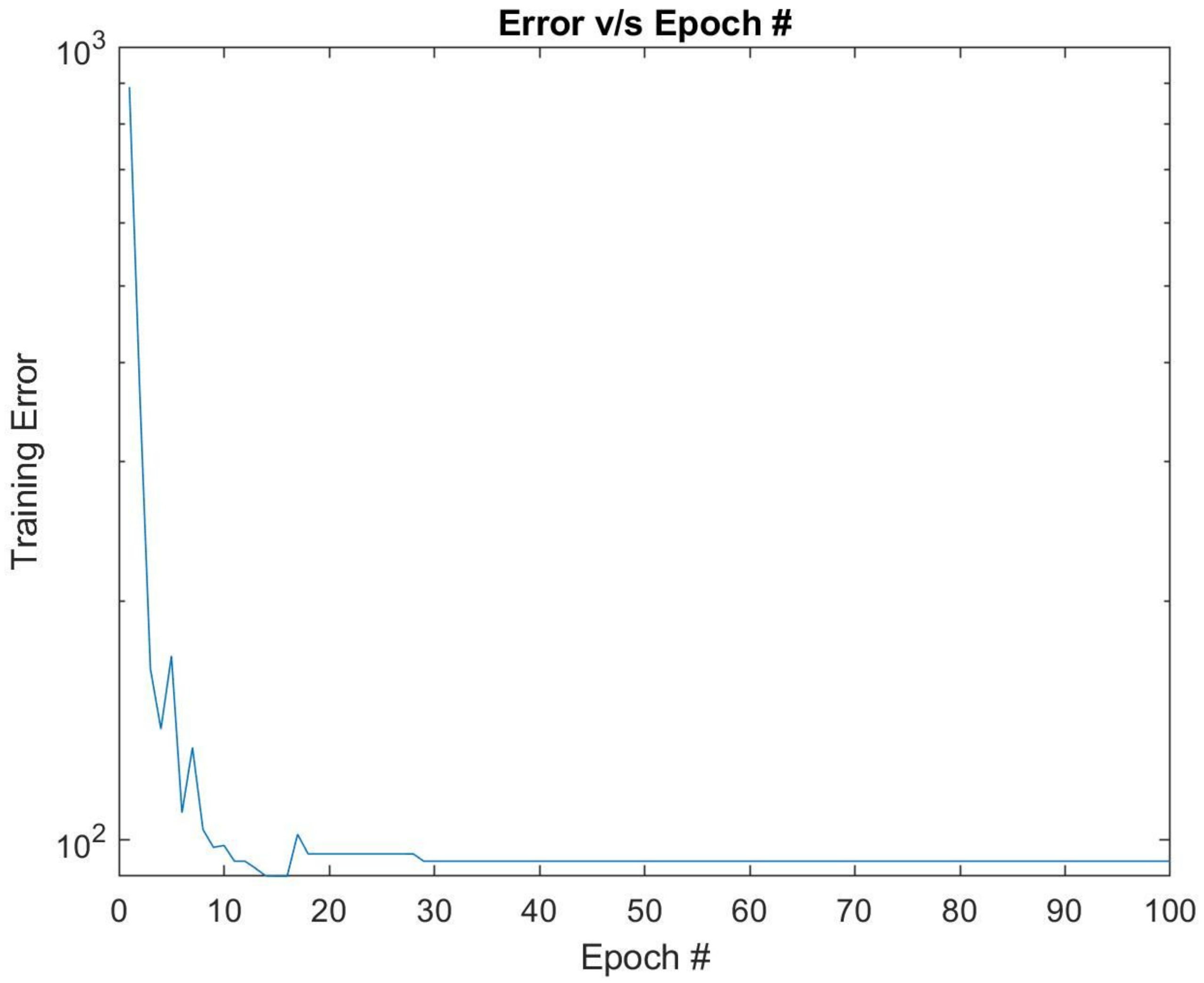


Figure B.8: Training Error v/s Epoch using ‘semilogy’ command for the Reduced PCA dataset for observation 8

<u>APPENDIX C</u>: Training Error Plots for PCA Digits Dataset over the 3 Layer Neural Network

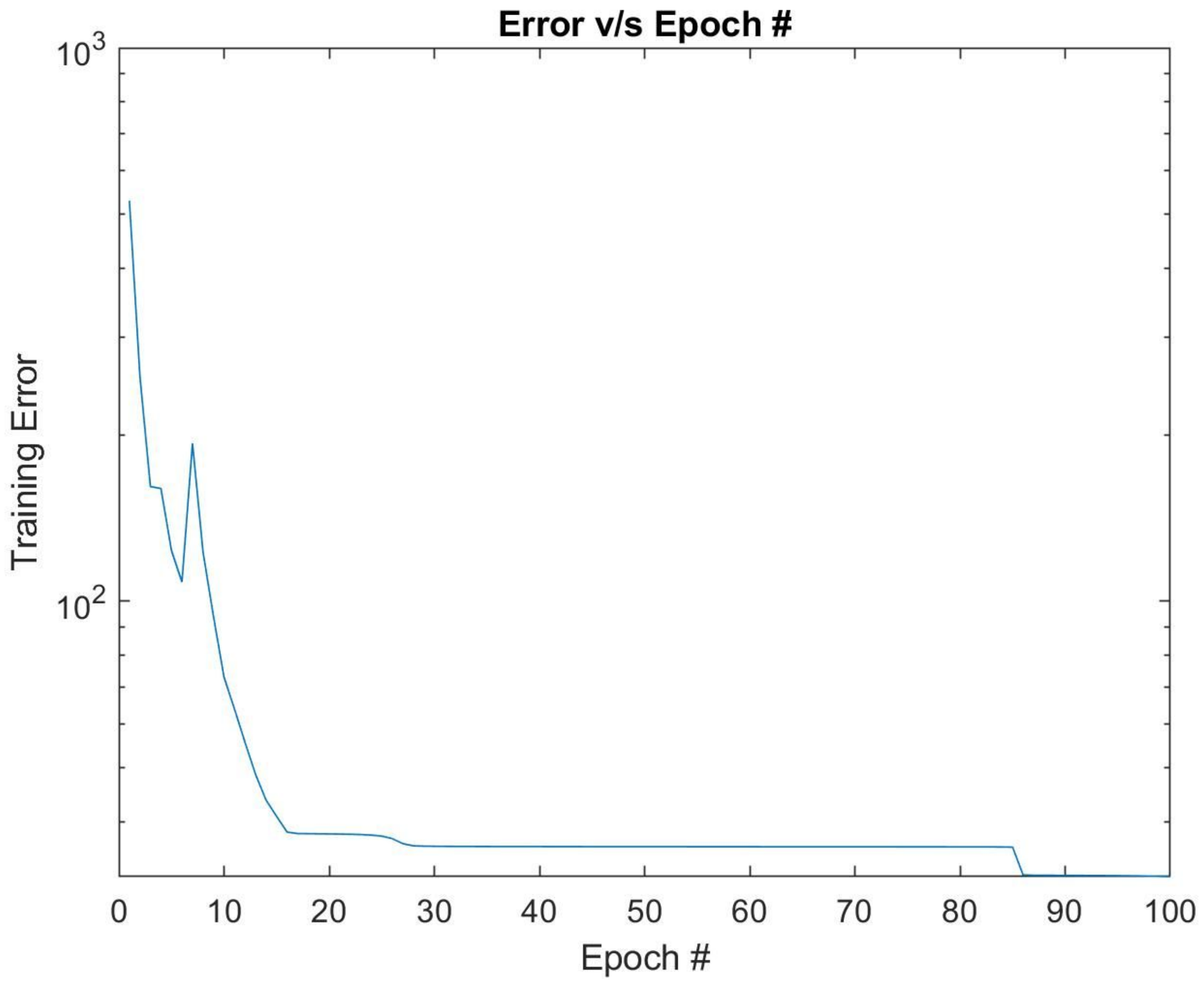


Figure C.1: Training Error v/s Epoch using 'semilogy' command for the Reduced PCA dataset for observation 1

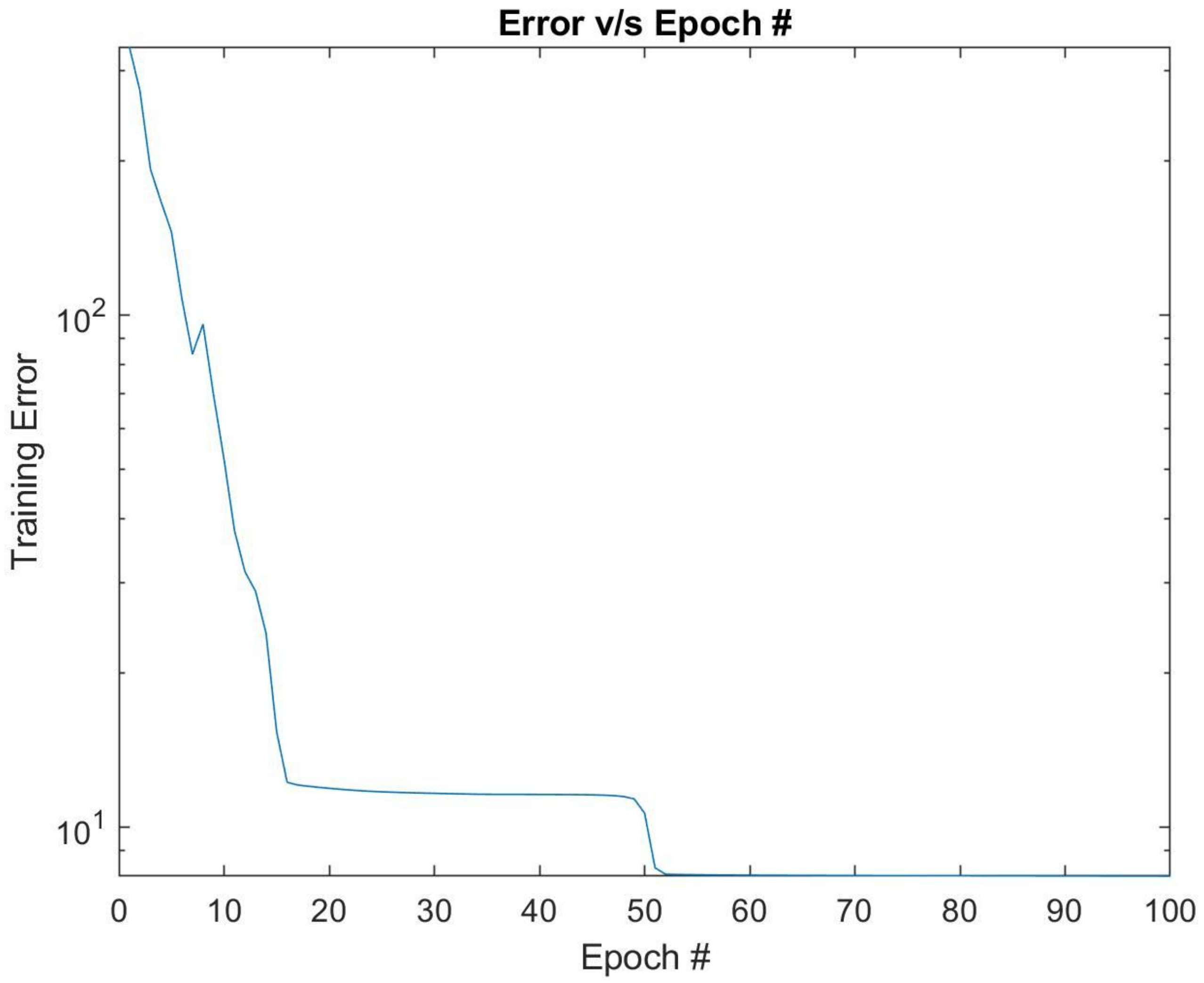


Figure C.2: Training Error v/s Epoch using ‘semilogy’ command for the Reduced PCA dataset for observation 2

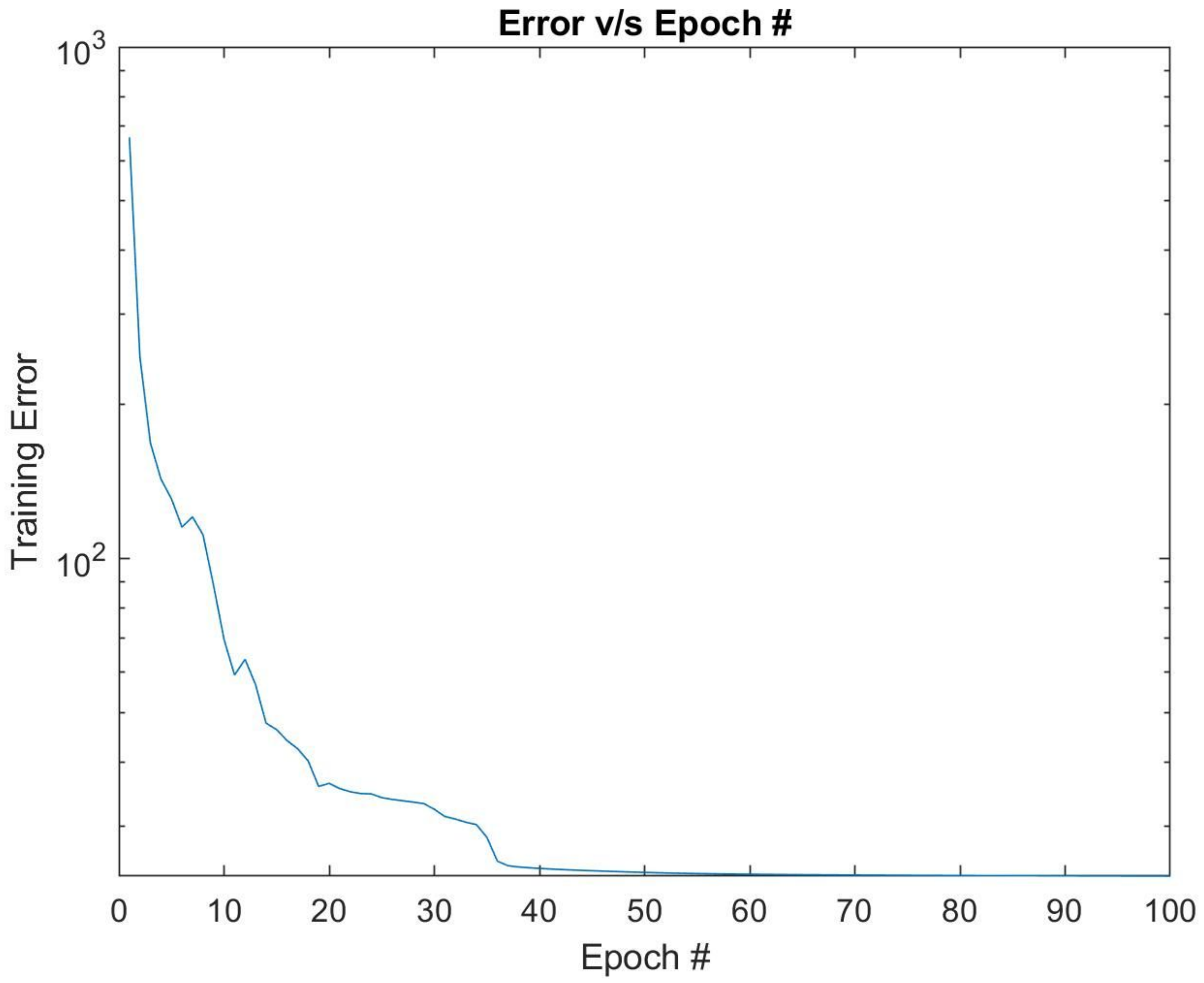


Figure C.3: Training Error v/s Epoch using 'semilogy' command for the Reduced PCA dataset for observation 3

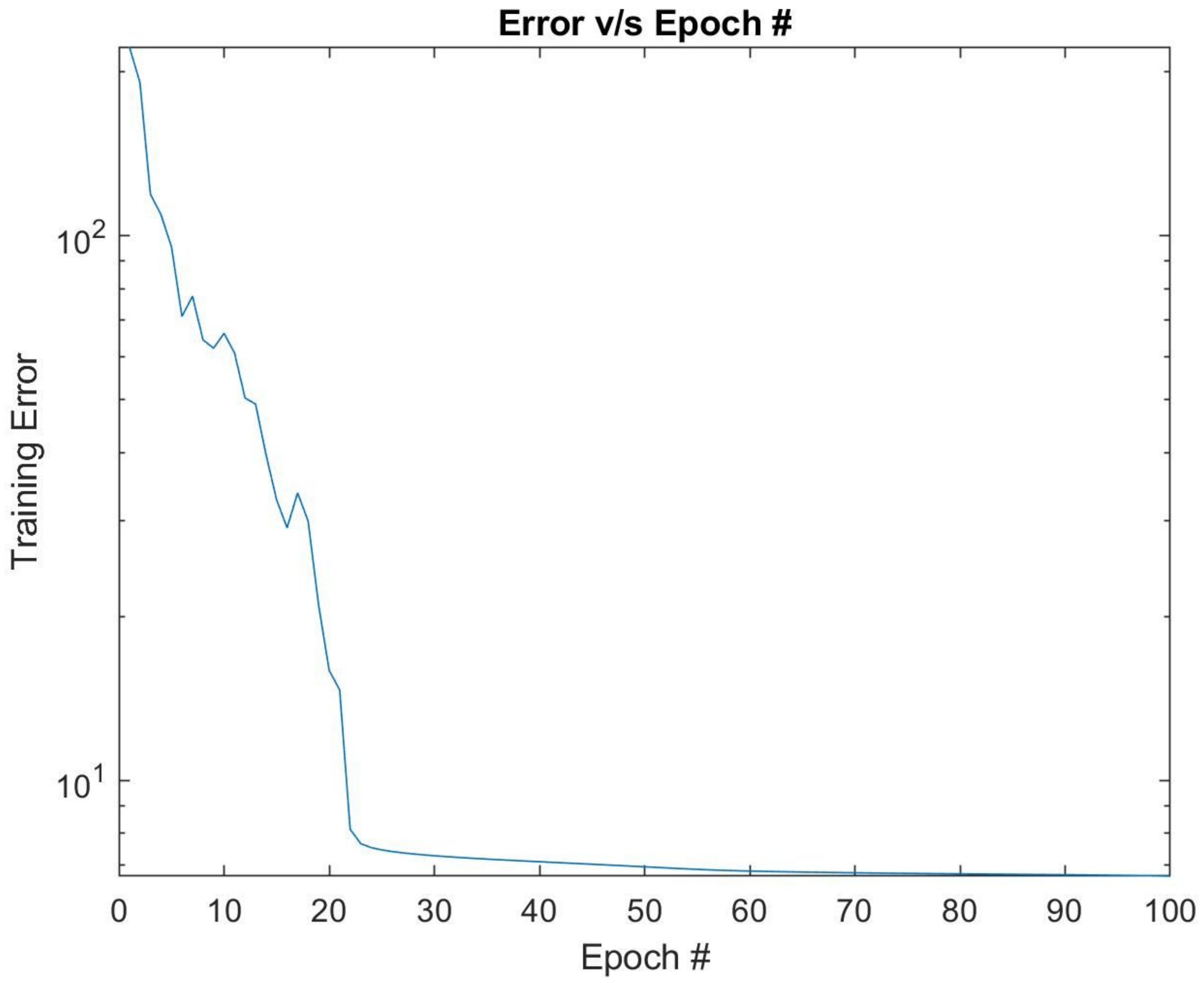


Figure C.4: Training Error v/s Epoch using 'semilogy' command for the Reduced PCA dataset for observation 4

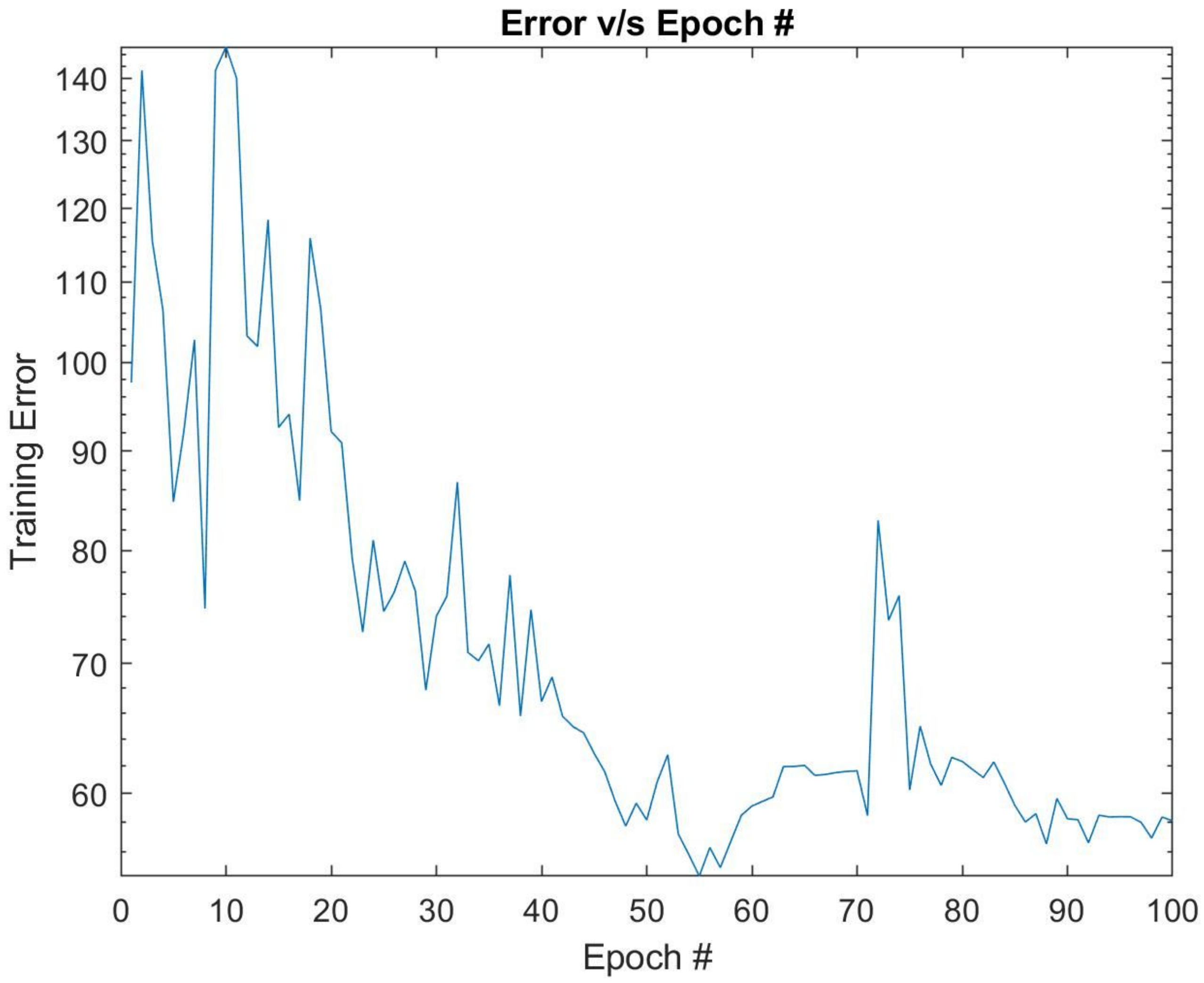


Figure C.5: Training Error v/s Epoch using 'semilogy' command for the Reduced PCA dataset for observation 5

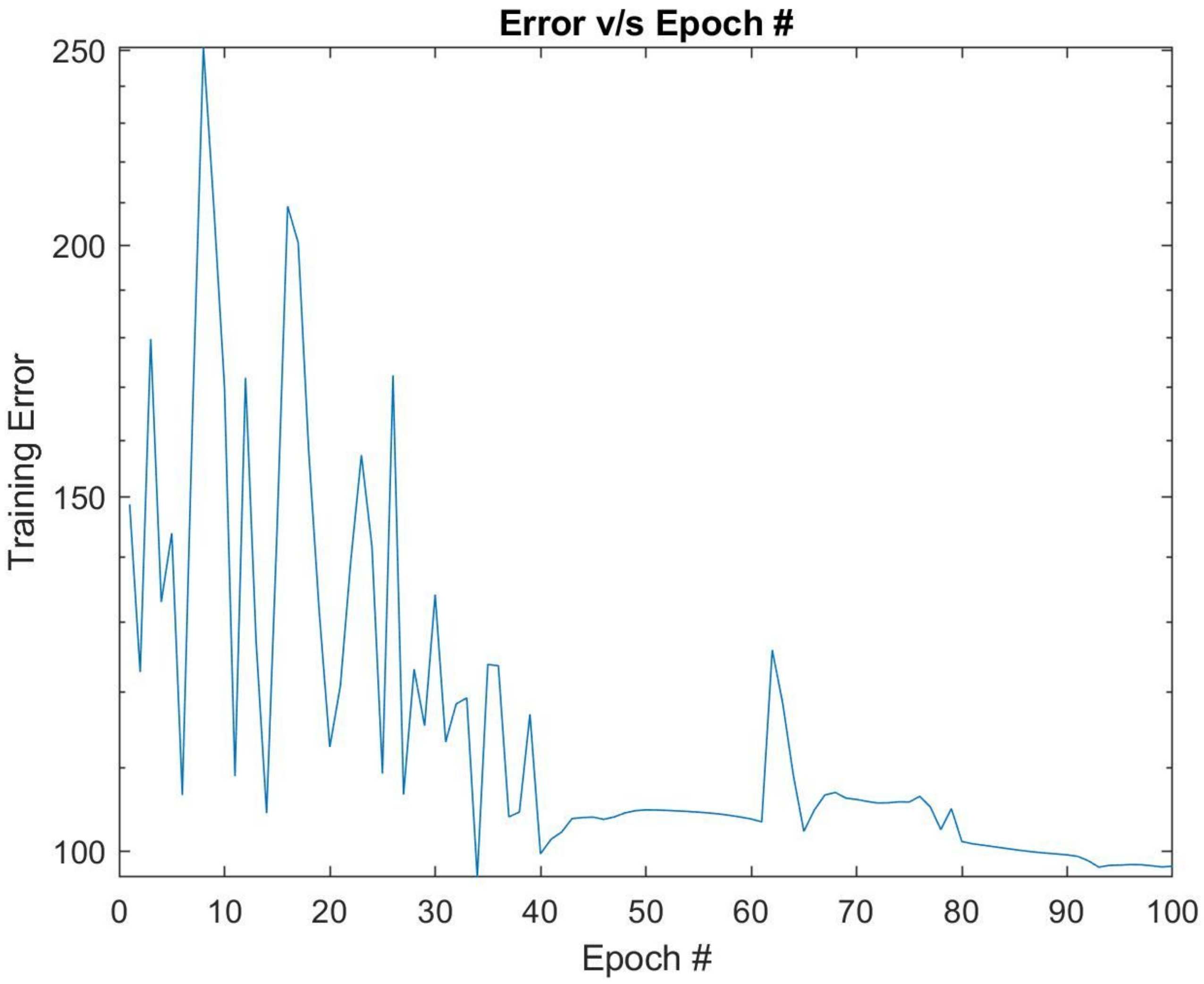


Figure C.6: Training Error v/s Epoch using 'semilogy' command for the Reduced PCA dataset for observation 6

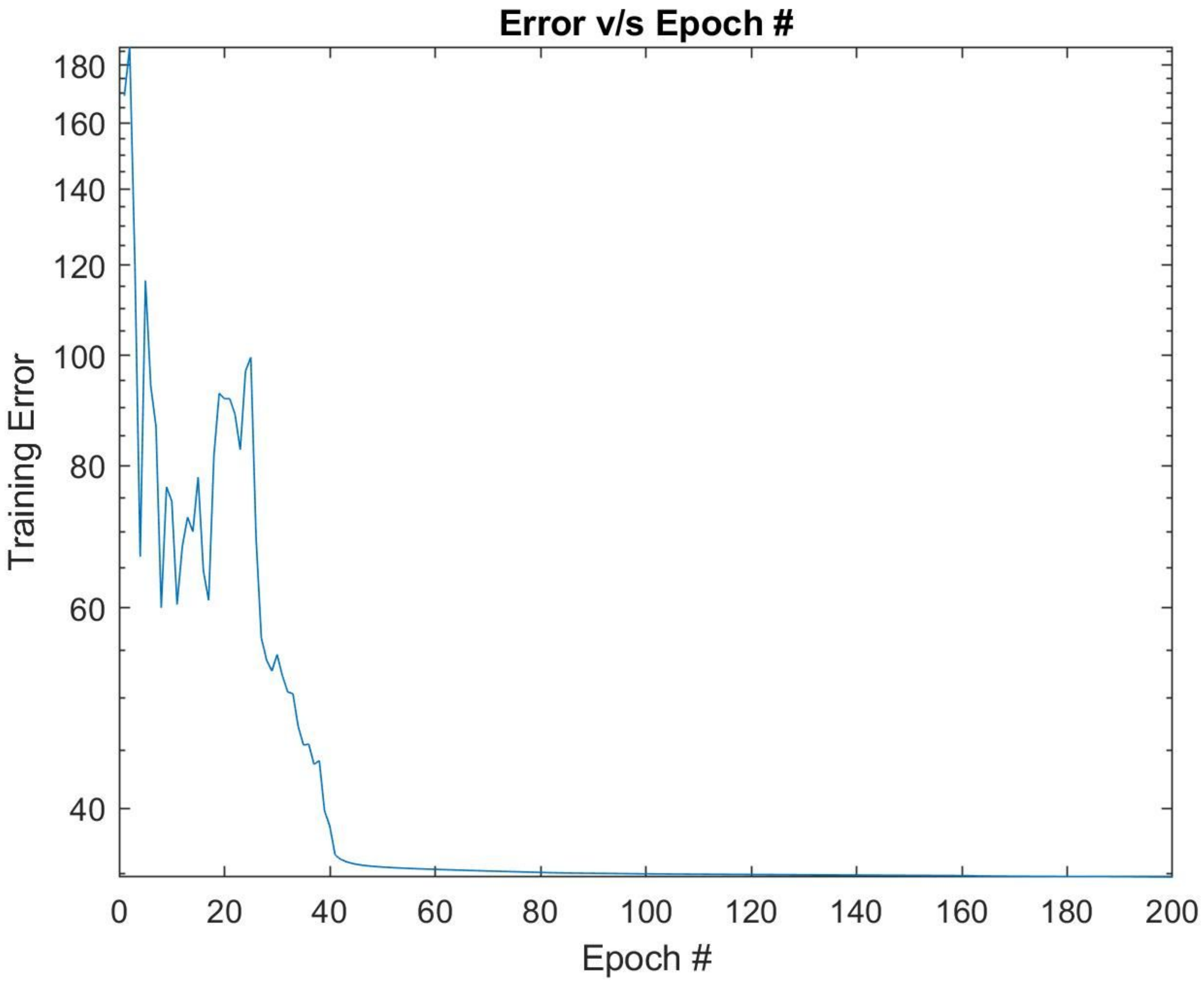


Figure C.7: Training Error v/s Epoch using 'semilogy' command for the Reduced PCA dataset for observation 7

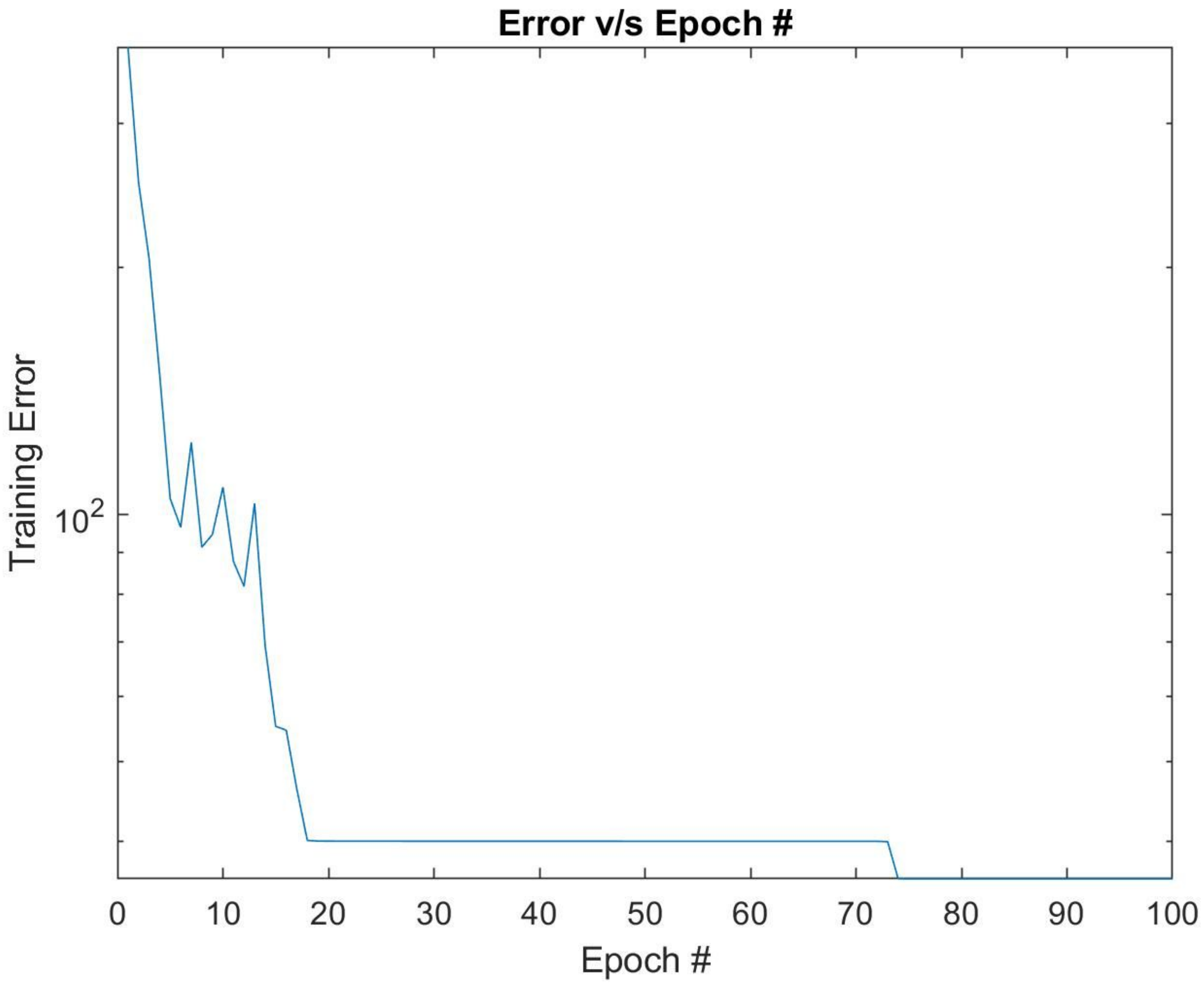


Figure C.8: Training Error v/s Epoch using 'semilogy' command for the Reduced PCA dataset for observation 8