AC50001 Introduction to Data Mining and Machine Learning   
**AC50001.2 Assignment: Classification and Clustering***Author: Vladislavs Ignatjevs (120015095)*

During last week I was working on Classification and Clustering assignment in attempt to create a system capable of dimension reduction using LDA, PCA and clustering using K-Means, and classification using SVM with a linear kernel, SVM with RBF kernel and Neural Network classifier. All these created components were tested/trained on limited MNIST handwritten digit database (only 3 numbers, 100 samples each).

**Q1. Principal Component Analysis (PCA)**

Applying PCA on data was the first classification problem I attempted to implement. PCA is a good way to compress data in regards to pattern recognition improvement. PCA transforms data to new, linearly uncorrelated coordinate system. It has its advantages and disadvantages. For example, if speaking about advantages, PCA is fast and easy to implement. However, if we overwrite data after applying PCA, it will not be possible to get the exact original data back, since PCA clears up the data that it considers unnecessary. As the result, we might lose significant information during the classification process. Apart from that, there are situations where we cannot apply PCA, since it works on linear spaces. For example, if the data is manifold structured, PCA will not perform an analysis in respect to expected results. The other drawback of PCA I found out comes from inability of Linear PCA to provide the information about class labels. LDA, on the other hand, solves this problem.

At the beginning I spent lots of time because I understood the task in a wrong way. Rather than applying PCA to all number data, I attempted to apply PCA on every number separately. This gave interesting results when plotted results. That was what I believe an attempt to reduce dimensions from 100 to 2 and create an interesting interpretation of most common features for numbers one, five and eight. Due to this being unsuitable for the task. I am not attaching the result of these application here, since it is not applicable for the task given, but I backed up the unused code into folder called “Unused”.

PCA can be summarized in the following steps:

1. Subtracting the mean for individual dimensions
2. Calculating covariance matrix
3. Calculating Eigen values and Eigen vectors
4. Transformation into new data

For the task allocated, I started with importing the data set and concatenating all the data sets into one larger matrix and transposing it:

%combine all data

all\_data = [digit\_one digit\_five digit\_eight]';

Then, I calculated the mean for this matrix and subtracted it from individual dimensions:

%substract the mean for individual dimenstions

all\_data = all\_data - repmat(meanAll, size(all\_data,1),1);

After doing that, it was possible to calculate covariance matrix (to see how much data vary from the mean in respect to each other). Then it was possible to calculate Eigen values and Eigen vectors from the covariance matrix calculate in the previous step. Then, by multiplying the matrix containing all data by specific Eigen vector’s values it made it possible to calculate PCA score:

score = all\_data\*v(:,1:2);

This made it ready for the data to get picked up for clustering.

**K-Means method for clustering.**

For data clustering I have used K-Means method because it was more familiar to me than the others. Using the following code I applied k-means to cluster the data into 3 clusters:

%do k-means

[idx, c] = kmeans(pca\_score,3);

%dim1

x1 = min(pca\_score(:,1)):0.01:max(pca\_score(:,1));

%dim2

x2 = min(pca\_score(:,2)):0.01:max(pca\_score(:,2));

%calculate mesh grid

[x1G,x2G] = meshgrid(x1,x2);

x\_grid = [x1G(:),x2G(:)]; % Defines a fine grid on the plot

% Assigns each node in the grid to the closest centroid

idx\_2\_region = kmeans(x\_grid,3,'MaxIter',1,'Start',c);

**Results:**

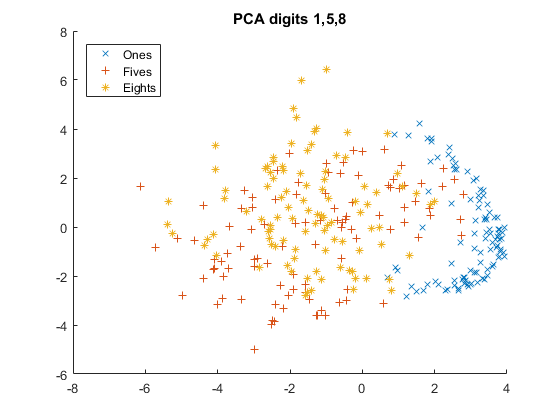


Figure 1: data after PCA projection

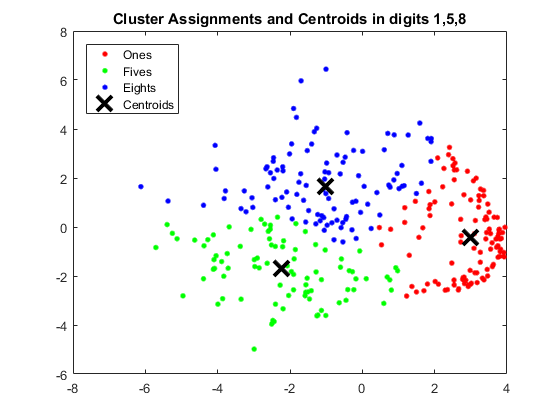


Figure 2: Data after PCA projection with Centroids

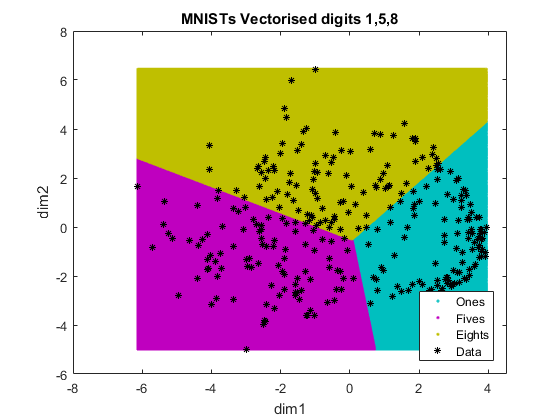


Figure 3: a more visualised plot of PCA projection on data, divided by regions for each of the classes

From the figures above we can see how PCA made it possible to classify all the data we had in the matrix. We can clearly see that due to classification some of the data got lost. For example, from the first figure, we can see that “Fives” data is spread all over the figure, but then classifier choses data “Eights” over “Fives”. Same for projection of “Ones”, if we compare figure one with figure 2 and figure 3, we can see that there is more “ones” data shown on figure 2 and figure 3 than in original projection on figure 1. I believe this happens in the result of implication of PCA discussed above. However, in respect of PCA, I believe that good results were achieved, especially if taking in mind, that it performed dimensionality reduction on data from 100 dimensions into 2 dimensions.

**Q2. Linear Discriminant Analysis (LDA)**

From what I learned, there are two types of Linear Discriminant Analysis applications. These are:

***Linear Discriminant Analysis on 2-class classification problem.***

The summary of steps for 2-class LDA:

1. Calculating means for each class
2. Calculating covariance matrices for each class
3. Calculating “within class scatter matrix” by summing up class covariance matrices
4. Computing Projection Vector by multiplying inverted “within class scatter matrix” on transposed difference of class means

***Linear Discriminant Analysis on multi-class classification problem.***

The summary of steps for n-class LDA:

1. Calculating means for each class and then calculate the mean of the class means
2. Calculating covariance matrices for each of the classes
3. Calculating “within class scatter matrix” by summing up all the covariance matrices
4. Calculating “between class scatter matrix”:
   1. for each of the classes:  
      multiplying number of size of the class by the difference of class means and mean of all class means, multiplied by the difference of class means and mean of all class means transposed.
   2. Summing up the results of above calculation from each class
5. Computing Eigen vectors (projection vectors) and Eigen values from inverted “within class scatter mask” multiplied by between class scatter mask
6. Calculating score by multiplying the matrix containing all data by specific Eigen vector’s values

If comparing LDA to PCA it is essential to mention that unlike PCA, LDA can pick up class labels, it can be used for jobs that PCA is not capable of doing. However, unlike expectations LDA is not guaranteed to perform better than PCA.

For the task allocated, I had to use 3-class LDA. I followed the steps I mentioned above to produce the 3-class LDA, but got stuck on the problem of calculating inverted “within-class scatter matrix”. Due to the data we have, the result of summing up class covariance matrices produced a matrix that had many zero values in it. It is obvious that it makes no sense to attempt to calculate inverse of 0 value. I solved the problem by adding bias to “within class scatter matrix” (I have picked some small number and to the “within-class scatter matrix“) in attempt to produce the matrix that would be suitable for inverting:

%addding small number to avoid inv on zero

dc=0.00001\*eye(size(sw));

sw\_new=sw+dc;

inverted\_SW=inv(sw\_new)\*sb;

It finally made it possible to calculate projection vectors and plot the results:

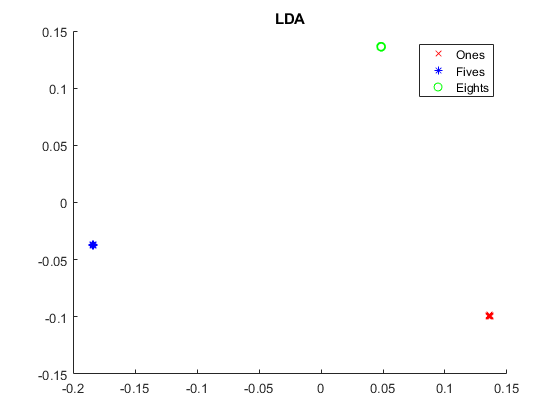
% computing the projection vectors:

[v1,d] = eig(inverted\_SW);

% calculating score

score = (all\_data\*v1(:,1:2));

**Results:**



From what we can see on the figure, LDA managed to classify classes and distinguishes the data. Compared to PCA, LDA treats data in slightly different way. By looking at PCA centroids, I believe that PCA performed better than LDA in this particular problem. Also, I believe that there is a bug somewhere in my LDA implementation, the results are not what I expected.

**Q3. Two-class problem classification**

For this question, I had to separate data for “Fives” from the rest of the data and treat it as two-class classification problem.

**Support Vector Machine (SVM)**

**SVM with linear kernel**

The SVM classifier with linear network was developed for use with libsvm library (version 3.2.2) that was provided. The following steps were replicated in order to develop SVM with linear kernel:

1. Combining all data samples into one
2. Scaling the data in each column into the range [0,1]
3. Computing data labels
4. Preparing vectors for “fives” against the rest (‘1’ meaning “fives”, ‘0’ the rest of the data)
5. Performing data partition
6. Getting the index for training and test samples
7. Creating training label ground truth and creating training data
8. Creating testing label ground truth and creating test data
9. Training SVM using linear kernel
10. Classifying

%combine all data

all\_data = [digit\_one digit\_five digit\_eight];

all\_data\_label = [];

%scale the data in each column of all data into the range of [0, 1]

for i=1:max(size(all\_data,2))

t = all\_data(:,i);

t = (t-min(t(:)))./(max(t(:))-min(t(:)));

all\_data(:,i) = t;

end

whos;

%get labels (digit one: 1-100, digit five 101-200, digit eight 201-300)

for k=1:size(all\_data,2)

if k<= 100

all\_data\_label = [all\_data\_label;1];

end

if k >100 && k <= 200

all\_data\_label = [all\_data\_label;5];

end

if k > 200

all\_data\_label = [all\_data\_label;8];

end

end

% preparing the vectors for 'fives' against the rest

a = zeros(size(all\_data\_label));

for i=1:max(size(a))

a(i) = isequal(all\_data\_label(i),5);

% in the vector a, 1 means 'fives'; and ‘0’ means rest

end

cvo = cvpartition(a,'k',2);

% get the index of training samples

trIdx = cvo.training(1);

% get the index of the test samples

teIdx = cvo.test(1);

% creating the training label ground truth

training\_label\_vector = a(trIdx);

%creating the training data

training\_instance\_matrix = all\_data(trIdx,:);

% creating the testing label ground truth

test\_label\_vector = a(teIdx);

% creating the test data

test\_instance\_matrix = all\_data(teIdx,:);

%training SVM using a linear kernel

model = svmtrain(training\_label\_vector, training\_instance\_matrix, '-t 0');

%classification on the test data:

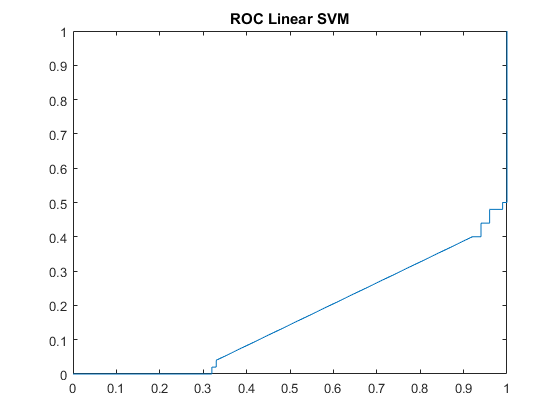
[predict\_label, accuracy, dec\_values] = svmpredict(test\_label\_vector,test\_instance\_matrix, model);

**Results:**

I have ran the script around 20 times in attempt to find the best results. I am providing the result set with highest accuracy percentage that was achieved after the number of script executions. For the results I consider the best, optimization finished at 140th iteration providing the following:

|  |
| --- |
| **nu** = 0.504592 |
| **obj** = -65.854545, **rho** = -0.999837 |
| **nSV** = 88, **nBSV** = 69 |
| **Total nSV** = 88 |
| **Accuracy** **= 82.6667% (124/150)** (classification) |

I have also plotted the Receiver operating characteristic (ROC) curve for this result



From what I can see from results, the classification job was very successful. I expected SVM with Linear kernel to provide lower accuracy. What is interesting that although the classification accuracy values was high, it was quite unstable and vary. Through the number of times I ran classification job it varied from 73% to 82.7%.

**SVM with RBF (radial basis function) kernel**

The steps to replicate SVM classifier with RBF kernel are exactly the same as for SVM with linear kernel, the only difference appear in the training technique:

%training SVM using a linear kernel

modelDef = svmtrain(training\_label\_vector, training\_instance\_matrix, '-t 1 -g 0.07');

Due variable G being available this time, I decided to change it number of times and produce number of model. I have picked different values for “g” (0.07,0.01,0.20,0.50,0.91) to see how it changes the result. The best results I have achieved after running the classification job for around 20 times:

|  |  |
| --- | --- |
| G = 0.07 (default value) *finished at iter 131* | G = 0.01 (smaller value) *finished at iter 70* |
| **nu** = 0.640605 | **nu** = 0.666667 |
| **obj** = -94.462968, **rho** = -0.999798 | **obj** = -99.967466, **rho** = -0.999836 |
| **nSV** = 109, **nBSV** = 93 | **nSV** =111, **nBSV** = 97 |
| **Total nSV** = 109 | **Total nSV** = 111 |
| **Accuracy** **= 66% (99/150)** | **Accuracy** **= 66.6667% (100/150)** |

|  |  |
| --- | --- |
| G = 0.20 (higher value) *finished at iter 152* | G = 0.50 (even higher value) *finished at iter 132* |
| **nu** = 0.599665 | **nu** = 0.526810 |
| **obj** = -85.704811, **rho** = -0.999760 | **obj** = -75.200986, **rho** = -0.999928 |
| **nSV** =108, **nBSV** = 86 | **nSV** =96, **nBSV** = 77 |
| **Total nSV** = 108 | **Total nSV** = 96 |
| **Accuracy** **= 67.3333% (101/150)** | **Accuracy** **= 72% (108/150)** |

|  |
| --- |
| G = 0.91 (highest value) *finished at iter 126* |
| **nu** = 0.480768 |
| **obj** = -67.443118, **rho** = -0.999913 |
| **nSV** =89, **nBSV** = 70 |
| **Total nSV** = 89 |
| **Accuracy** **= 76.6667% (115/150)** |

What I figured out through the amount of times I ran the classification job was rather interesting for me. By setting up “g” value to or 0.01, 0.07, 0.20 the accuracy was lover then the rest, but more stable. During around 20 times of running classification job it the accuracy suffered from small changes (0.5%-2%). However, by increasing “g” to 0.9 I have managed to get the highest accuracy, but the accuracy became very unstable. During around 20 times of running classification job the accuracy was suffering from large changes (5%-10%).

**Comparing SVM with RBF to SVM with Linear:**

From what it is possible to see on the results of SVM with linear kernel and SVM with RBF kernel, the SVM with linear kernel provides higher accuracy (10% more accurate than best result of SVM with RBF kernel). At the same time, at the higher result it is much more stable. I believe it is possible to imrove the result of SVM with RBF kernel by finding the best size combination of training/testing set size.

**ROC curves for the results of SVM with RVF kernel:**

|  |  |
| --- | --- |
| C:\Users\TEMP.COMPUTING\Desktop\machine learning assignment\MachineLearning17\Graphs\SVM\RBF\g01.png  Figure 4: ROC plot SVM with RBF kernel (g=0.01) | C:\Users\TEMP.COMPUTING\Desktop\machine learning assignment\MachineLearning17\Graphs\SVM\RBF\g02.png  Figure 5: ROC plot SVM with RBF kernel (g=0.20) |
| C:\Users\TEMP.COMPUTING\Desktop\machine learning assignment\MachineLearning17\Graphs\SVM\RBF\g05.png  Figure 6: ROC plot SVM with RBF kernel (g=0.50) | C:\Users\TEMP.COMPUTING\Desktop\machine learning assignment\MachineLearning17\Graphs\SVM\RBF\g07.png  Figure 7: ROC plot SVM with RBF kernel (g=0.07) |
| C:\Users\TEMP.COMPUTING\Desktop\machine learning assignment\MachineLearning17\Graphs\SVM\RBF\g091.png  Figure 8: ROC plot SVM with RBF kernel (g=0.91) |

**Neural Network classifier**

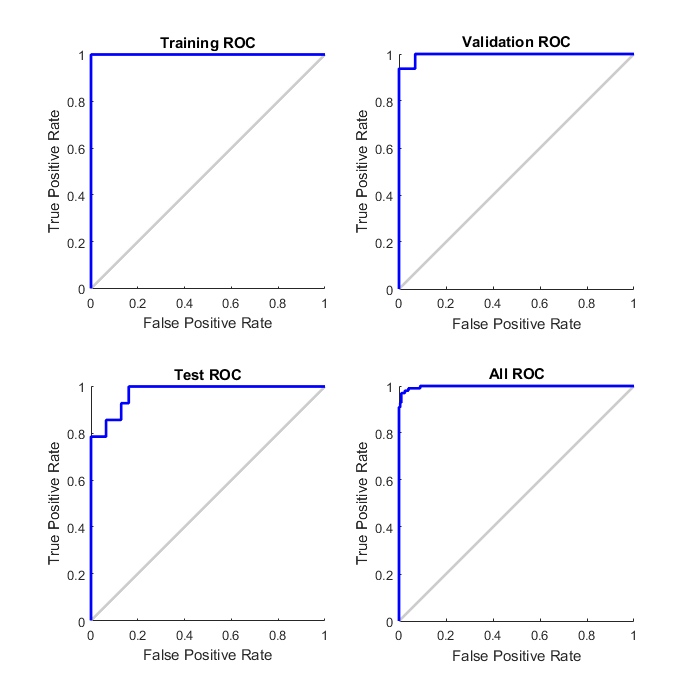
For this task I have used Matlab’s build in “Neural Net Pattern Recognition” tool (nprtool). I have created a neural network with the following samples: 70% training, 15% validation, 15% testing. I have used 1 hidden layer. I have never applied 5-fold cross validation setting to the data due to lack of time. The correct process would be to split the data into 5 chunks and perform training 5 times in attempt to get better results.

**Results:**

In attempt to find the best results, 5 same neural network classifiers were created and trained. The best performance I could achieve was 81%.

|  |  |
| --- | --- |
| C:\Users\TEMP.COMPUTING\AppData\Local\Microsoft\Windows\INetCache\Content.Word\diagram.png  Figure 9: Neural Network diagram | C:\Users\TEMP.COMPUTING\AppData\Local\Microsoft\Windows\INetCache\Content.Word\results.png  Figure 10: Neural network training results |
| C:\Users\TEMP.COMPUTING\AppData\Local\Microsoft\Windows\INetCache\Content.Word\training_state.png  Figure 11: Neural network training state | C:\Users\TEMP.COMPUTING\AppData\Local\Microsoft\Windows\INetCache\Content.Word\error_histogram.png  Figure 12: Neural network error histogram |
| C:\Users\TEMP.COMPUTING\AppData\Local\Microsoft\Windows\INetCache\Content.Word\performance.png  Figure 13: Neural network performance | C:\Users\TEMP.COMPUTING\Desktop\machine learning assignment\MachineLearning17\Graphs\NeuralNet\confusion.png  Figure 14: Neural network confusion matrix |

**ROC curves for Neural Network:**



**Comparing results in Q3**

By looking at performance results I can conclude that SVM with Linear kernel provides the highest accuracy result (82.6667%). The neural network achieved 81.1515% which is slightly smaller than SVM with Linear kernel, but much better than SVM with RBF Kernel. The parameter we would be interested in for comparing classifiers is ROC curve plotted from results of performing classifier testing. The ROC curve for Neural Network looks better than ROC curves for SVM, but after having discussion with other student we made a conclusion that although it looks the best, it is an overkill due to the way neural network performs testing. By looking at the other ROC curves, I believe that it is bug somewhere in ROC curve plotting. I believe that SVM with Linear kernel should provide the best ROC curve by looking at the results.

**Instructions on running the code:**

In the ZIP file there are 5 folder corresponding to classifiers.  
**For LDA:**  
Run lda.m from the LDA folder  
**For PCA:**  
Run PCAforAll.m from PCA folder  
**For SVM with Linear Kernel:**  
Run svm\_linear.m from SVM folder  
**For SVM with RBF Kernel:**  
Run svm\_rbf.m from SVM folder

**For Neural Network:**

Run train\_advanced.m to replicate the neural network I created.

**Code:**

**LDA:**

%load data

load('AC50001\_assignment2\_data.mat');

%combine all data

all\_data = [digit\_one digit\_five digit\_eight]' ;

all\_data\_label = [];

meanAll = mean(all\_data);

all\_data = all\_data - repmat(meanAll, size(all\_data,1),1);

%get labels

for k=1:size(all\_data,1)

if k<= 100

all\_data\_label = [all\_data\_label;'1'];

end

if k >100 && k <= 200

all\_data\_label = [all\_data\_label;'5'];

end

if k > 200

all\_data\_label = [all\_data\_label;'8'];

end

end

classOnes = all\_data(all\_data\_label=='1',:);

classFives = all\_data(all\_data\_label=='5',:);

classEights = all\_data(all\_data\_label=='8',:);

%means

muOnes = mean(classOnes,1);

muFives = mean(classFives,1);

muEights = mean(classEights,1);

%class covariance matrices (from transposed class matrices)

covarOnes = cov(classOnes);

covarFives = cov(classFives);

covarEights = cov(classEights);

%within class scatter matrix

sw = covarOnes + covarFives + covarEights;

%mean of the class means

meanClassMeans = (muOnes + muFives + muEights)./3;

%each of classes has 100 samples, no need to

%recalculate using size(class,2)

%between class scatter matrix

sbOnes = 100 .\* (muOnes-meanClassMeans)'\*(muOnes-meanClassMeans);

sbFives = 100 .\* (muFives-meanClassMeans)'\*(muFives-meanClassMeans);

sbEights = 100 .\* (muEights-meanClassMeans)'\*(muEights-meanClassMeans);

sb = sbOnes+ sbFives+sbEights;

%LDA projection vector

%addding small number to avoid inv on zero

dc=0.00001\*eye(size(sw));

sw\_new=sw+dc;

inverted\_SW=inv(sw\_new)\*sb;

% computing the projection vectors:

[v1,d] = eig(inverted\_SW);

% calculating score

score = (all\_data\*v1(:,1:2));

%Or the following code to plot data after LDA projection:

% the class ones:

figure

scatter(score(1:100,1), score(1:100,2), 'r', 'x'); %% look at only one direction will be fine

hold on;

scatter(score(101:200,1), score(101:200,2),'b', '\*');

hold on;

scatter(score(201:300,1), score(201:300,2), 'g', 'o');

title 'LDA';

legend('Ones','Fives','Eights','Data','Location','NorthEast');

hold off;

**PCA:**

*PcaforAll.m*

%clear workspace

clear;

%load data

load('AC50001\_assignment2\_data.mat');

%for reuse

orig\_digit\_one = digit\_one;

orig\_digit\_five = digit\_five;

orig\_digit\_eight = digit\_eight;

%combine all data

all\_data = [digit\_one digit\_five digit\_eight]';

[score] = calculatePCAforAll(all\_data);

%cluster those data points in the 2-D space into 3 clusters using k-means

%idx - cluster indices

%c - centroids locations

%k-means(data, number\_of\_clusters)

%prepare data for scatter plot

[idx,c,x1,x2,x\_grid,idx\_2\_region] = PCAKmeansFiguresALL(score);

*prepareDataForScatter3Clust.m*

function [idx,c,x1,x2,x\_grid,idx\_2\_region] = prepareDataForScatter3Clust(pca\_score)

%do k-means

[idx, c] = kmeans(pca\_score,3);

%dim1

x1 = min(pca\_score(:,1)):0.01:max(pca\_score(:,1));

%dim2

x2 = min(pca\_score(:,2)):0.01:max(pca\_score(:,2));

%calculate mesh grid

[x1G,x2G] = meshgrid(x1,x2);

x\_grid = [x1G(:),x2G(:)]; % Defines a fine grid on the plot

% Assigns each node in the grid to the closest centroid

idx\_2\_region = kmeans(x\_grid,3,'MaxIter',1,'Start',c);

end

*PCAKmeansFiguresALL.m*

function [idx,c,x1,x2,x\_grid,idx\_2\_region] = PCAKmeansFiguresALL(PCA\_score)

[idx,c,x1,x2,x\_grid,idx\_2\_region] = prepareDataForScatter3Clust(PCA\_score);

figure;

gscatter(x\_grid(:,1),x\_grid(:,2),idx\_2\_region,...

[0,0.75,0.75;0.75,0,0.75;0.75,0.75,0],'..');

hold on;

plot(PCA\_score(:,1),PCA\_score(:,2),'k\*','MarkerSize',5);

title 'MNISTs Vectorised digits 1,5,8';

xlabel 'dim1';

ylabel 'dim2';

legend('Ones','Fives','Eights','Data','Location','SouthEast');

hold off;

opts = statset('Display','final');

[idx, c] = kmeans(PCA\_score,3,'Distance','cityblock',...

'Replicates',5,'Options',opts);

%figure 2

figure;

plot(PCA\_score(idx==1,1),PCA\_score(idx==1,2),'r.','MarkerSize',12)

hold on

plot(PCA\_score(idx==2,1),PCA\_score(idx==2,2),'g.','MarkerSize',12)

hold on

plot(PCA\_score(idx==3,1),PCA\_score(idx==3,2),'b.','MarkerSize',12)

plot(c(:,1),c(:,2),'kx',...

'MarkerSize',15,'LineWidth',3)

legend('Ones','Fives','Eights','Centroids',...

'Location','NW')

title 'Cluster Assignments and Centroids in digits 1,5,8'

hold off

%plot data after PCA projection:

figure;

scatter(PCA\_score(1:100,1), PCA\_score(1:100,2),'Marker','x'); %% look at only one direction will be fine

hold on;

scatter(PCA\_score(101:200,1), PCA\_score(101:200,2),'Marker','+');

hold on;

scatter(PCA\_score(201:300,1), PCA\_score(201:300,2), 'Marker','\*');

title 'PCA digits 1,5,8'

legend('Ones','Fives','Eights',...

'Location','NW')

hold off;

end

**SVM with Linear Kernel:**

%load data

load('AC50001\_assignment2\_data.mat');

%combine all data

all\_data = [digit\_one digit\_five digit\_eight] ;

all\_data\_label = [];

%scale the data in each column of all data into the range of [0, 1]

for i=1:max(size(all\_data,2))

t = all\_data(:,i);

t = (t-min(t(:)))./(max(t(:))-min(t(:)));

all\_data(:,i) = t;

end

whos;

%get labels (digit one: 1-100, digit five 101-200, digit eight 201-300)

for k=1:size(all\_data,2)

if k<= 100

all\_data\_label = [all\_data\_label;1];

end

if k >100 && k <= 200

all\_data\_label = [all\_data\_label;5];

end

if k > 200

all\_data\_label = [all\_data\_label;8];

end

end

%fives against the rest

% preparing the vectors for 'fives' against the rest

a = zeros(size(all\_data\_label));

for i=1:max(size(a))

a(i) = isequal(all\_data\_label(i),5);

% in the vector a, 1 means 'fives'; and ‘0’ means rest

end

cvo = cvpartition(a,'k',2);

% get the index of training samples

trIdx = cvo.training(1);

% get the index of the test samples

teIdx = cvo.test(1);

% creating the training label ground truth

training\_label\_vector = a(trIdx);

%creating the training data

training\_instance\_matrix = all\_data(trIdx,:);

% creating the testing label ground truth

test\_label\_vector = a(teIdx);

% creating the test data

test\_instance\_matrix = all\_data(teIdx,:);

%training SVM using a linear kernel

model = svmtrain(training\_label\_vector, training\_instance\_matrix, '-t 0');

%classification on the test data:

[predict\_label, accuracy, dec\_values] = svmpredict(test\_label\_vector,test\_instance\_matrix, model);

[X,Y] = perfcurve(test\_label\_vector,dec\_values,1);

figure;

plot(X,Y);

hold on;

title('ROC Linear SVM')

hold off;

**SVM with RBF Kernel:**

%load data

load('AC50001\_assignment2\_data.mat');

%combine all data

all\_data = [digit\_one digit\_five digit\_eight] ;

all\_data\_label = [];

%scale the data in each column of all data into the range of [0, 1]

for i=1:max(size(all\_data,2))

t = all\_data(:,i);

t = (t-min(t(:)))./(max(t(:))-min(t(:)));

all\_data(:,i) = t;

end

whos;

%get labels (digit one: 1-100, digit five 101-200, digit eight 201-300)

for k=1:size(all\_data,2)

if k<= 100

all\_data\_label = [all\_data\_label;1];

end

if k >100 && k <= 200

all\_data\_label = [all\_data\_label;5];

end

if k > 200

all\_data\_label = [all\_data\_label;8];

end

end

%fives against the rest

% preparing the vectors for 'fives' against the rest

a = zeros(size(all\_data\_label));

for i=1:max(size(a))

a(i) = isequal(all\_data\_label(i),5);

% in the vector a, 1 means 'fives'; and ‘0’ means rest

end

cvo = cvpartition(a,'k',2);

% get the index of training samples

trIdx = cvo.training(1);

% get the index of the test samples

teIdx = cvo.test(1);

% creating the training label ground truth

training\_label\_vector = a(trIdx);

%creating the training data

training\_instance\_matrix = all\_data(trIdx,:);

% creating the testing label ground truth

test\_label\_vector = a(teIdx);

% creating the test data

test\_instance\_matrix = all\_data(teIdx,:);

%training SVM using a linear kernel

modelDef = svmtrain(training\_label\_vector, training\_instance\_matrix, '-t 1 -g 0.07');

modelSMallerG = svmtrain(training\_label\_vector, training\_instance\_matrix, '-t 1 -g 0.01');

modelHigherG = svmtrain(training\_label\_vector, training\_instance\_matrix, '-t 1 -g 0.20');

modelEvenHigherG = svmtrain(training\_label\_vector, training\_instance\_matrix, '-t 1 -g 0.50');

modelHighestG = svmtrain(training\_label\_vector, training\_instance\_matrix, '-t 1 -g 0.91');

%classification on the test data:

%[predict\_label, accuracy, dec\_values] = svmpredict(test\_label\_vector,test\_instance\_matrix, model);

%default g

defaultGclassifResults = []; %0.07

smallerGclassifResults = []; %0.01

higherGclassifResults = []; %0.20

evenHigherGclassifResults = []; %0.50

highestGclassifResults = []; %0.91

%default g

disp('default g = 0.07');

[predict\_label, accuracy, dec\_values] = svmpredict(test\_label\_vector,test\_instance\_matrix, modelDef);

[X,Y] = perfcurve(test\_label\_vector,dec\_values,1);

figure;

plot(X,Y);

hold on;

title('ROC RBF SVM (g=0.07)')

hold off;

%smaller g

disp('smaller g = 0.01');

[predict\_label, accuracy, dec\_values] = svmpredict(test\_label\_vector,test\_instance\_matrix, modelSMallerG);

[X,Y] = perfcurve(test\_label\_vector,dec\_values,1);

figure;

plot(X,Y);

hold on;

title('ROC RBF SVM (g=0.01)')

hold off;

%higher g

disp('higher g = 0.20');

[predict\_label, accuracy, dec\_values] = svmpredict(test\_label\_vector,test\_instance\_matrix, modelHigherG);

[X,Y] = perfcurve(test\_label\_vector,dec\_values,1);

figure;

plot(X,Y);

hold on;

title('ROC RBF SVM (g=0.20)')

hold off;

%even more higher g

disp('even higher g = 0.50');

[predict\_label, accuracy, dec\_values] = svmpredict(test\_label\_vector,test\_instance\_matrix, modelEvenHigherG);

[X,Y] = perfcurve(test\_label\_vector,dec\_values,1);

figure;

plot(X,Y);

hold on;

title('ROC RBF SVM (g=0.50)')

hold off;

%highest g

disp('highest g = 0.91');

[predict\_label, accuracy, dec\_values] = svmpredict(test\_label\_vector,test\_instance\_matrix, modelHighestG);

[X,Y] = perfcurve(test\_label\_vector,dec\_values,1);

figure;

plot(X,Y);

hold on;

title('ROC RBF SVM (g=0.91)')

hold off;

%default G - classification accuracy lowest but remains stable over different trainings

%hightest G - classsification accuracy highest, but accurracy is not stable

%(+-2-5% every time the training is performed

**For Neural Network:**

*Train\_avanced.m*

% Solve a Pattern Recognition Problem with a Neural Network

% Script generated by Neural Pattern Recognition app

% Created 20-Mar-2017 16:59:18

%

% This script assumes these variables are defined:

%

% all\_data - input data.

% all\_data\_label - target data.

%load data

load('AC50001\_assignment2\_data.mat');

% neural network classifier with one hidden layer to classify the

%dataset in a 5-fold cross validation setting.

%combine all data

all\_data = [digit\_one digit\_five digit\_eight] ;

all\_data\_label = [];

%scale the data in each column of all data into the range of [0, 1]

for i=1:max(size(all\_data,2))

t = all\_data(:,i);

t = (t-min(t(:)))./(max(t(:))-min(t(:)));

all\_data(:,i) = t;

end

whos;

%get labels (digit one: 1-100, digit five 101-200, digit eight 201-300)

for k=1:size(all\_data,2)

if k<= 100

all\_data\_label = [all\_data\_label;1];

end

if k >100 && k <= 200

all\_data\_label = [all\_data\_label;5];

end

if k > 200

all\_data\_label = [all\_data\_label;8];

end

end

%fives against the rest

% preparing the vectors for 'fives' against the rest

a = zeros(size(all\_data\_label));

for i=1:max(size(a))

a(i) = isequal(all\_data\_label(i),5);

% in the vector a, 1 means 'fives'; and ‘0’ means rest

end

all\_data\_label = a';

x = all\_data;

t = all\_data\_label;

% Choose a Training Function

% For a list of all training functions type: help nntrain

% 'trainlm' is usually fastest.

% 'trainbr' takes longer but may be better for challenging problems.

% 'trainscg' uses less memory. Suitable in low memory situations.

trainFcn = 'trainscg'; % Scaled conjugate gradient backpropagation.

% Create a Pattern Recognition Network

hiddenLayerSize = 1;

net = patternnet(hiddenLayerSize);

% Choose Input and Output Pre/Post-Processing Functions

% For a list of all processing functions type: help nnprocess

net.input.processFcns = {'removeconstantrows','mapminmax'};

net.output.processFcns = {'removeconstantrows','mapminmax'};

% Setup Division of Data for Training, Validation, Testing

% For a list of all data division functions type: help nndivide

net.divideFcn = 'dividerand'; % Divide data randomly

net.divideMode = 'sample'; % Divide up every sample

net.divideParam.trainRatio = 70/100;

net.divideParam.valRatio = 15/100;

net.divideParam.testRatio = 15/100;

% Choose a Performance Function

% For a list of all performance functions type: help nnperformance

net.performFcn = 'crossentropy'; % Cross-Entropy

% Choose Plot Functions

% For a list of all plot functions type: help nnplot

net.plotFcns = {'plotperform','plottrainstate','ploterrhist', ...

'plotconfusion', 'plotroc'};

% Train the Network

[net,tr] = train(net,x,t);

% Test the Network

y = net(x);

e = gsubtract(t,y);

performance = perform(net,t,y)

tind = vec2ind(t);

yind = vec2ind(y);

percentErrors = sum(tind ~= yind)/numel(tind);

% Recalculate Training, Validation and Test Performance

trainTargets = t .\* tr.trainMask{1};

valTargets = t .\* tr.valMask{1};

testTargets = t .\* tr.testMask{1};

trainPerformance = perform(net,trainTargets,y)

valPerformance = perform(net,valTargets,y)

testPerformance = perform(net,testTargets,y)

% View the Network

view(net)

% Plots

% Uncomment these lines to enable various plots.

%figure, plotperform(tr)

%figure, plottrainstate(tr)

%figure, ploterrhist(e)

%figure, plotconfusion(t,y)

%figure, plotroc(t,y)

% Deployment

% Change the (false) values to (true) to enable the following code blocks.

% See the help for each generation function for more information.

if (false)

% Generate MATLAB function for neural network for application

% deployment in MATLAB scripts or with MATLAB Compiler and Builder

% tools, or simply to examine the calculations your trained neural

% network performs.

genFunction(net,'myNeuralNetworkFunction');

y = myNeuralNetworkFunction(x);

end

if (false)

% Generate a matrix-only MATLAB function for neural network code

% generation with MATLAB Coder tools.

genFunction(net,'myNeuralNetworkFunction','MatrixOnly','yes');

y = myNeuralNetworkFunction(x);

end

if (false)

% Generate a Simulink diagram for simulation or deployment with.

% Simulink Coder tools.

gensim(net);

end

*neural\_net\_function.m*

function [y1] = neural\_net\_function(x1)

%MYNEURALNETWORKFUNCTION neural network simulation function.

%

% Generated by Neural Network Toolbox function genFunction, 20-Mar-2017 16:57:16.

%

% [y1] = myNeuralNetworkFunction(x1) takes these arguments:

% x = 784xQ matrix, input #1

% and returns:

% y = 1xQ matrix, output #1

% where Q is the number of samples.

%#ok<\*RPMT0>

% ===== NEURAL NETWORK CONSTANTS =====

% Input 1

x1\_step1.keep = [100 101 102 105 106 107 108 119 120 121 122 123 124 125 128 129 130 131 132 133 134 135 136 146 147 148 149 150 151 152 153 154 155 156 157 158 159 160 161 162 163 164 165 173 174 175 176 177 178 179 180 181 182 183 184 185 186 187 188 189 190 191 192 193 194 201 202 203 204 205 206 207 208 209 210 211 212 213 214 215 216 217 218 219 220 221 222 229 230 231 232 233 234 235 236 237 238 239 240 241 242 243 244 245 246 247 248 249 250 256 257 258 259 260 261 262 263 264 265 266 267 268 269 270 271 272 273 274 275 276 277 278 284 285 286 287 288 289 290 291 292 293 294 295 296 297 298 299 300 301 302 303 304 305 306 307 312 313 314 315 316 317 318 319 320 321 322 323 324 325 326 327 328 329 330 331 332 333 334 335 340 341 342 343 344 345 346 347 348 349 350 351 352 353 354 355 356 357 358 359 360 361 362 363 368 369 370 371 372 373 374 375 376 377 378 379 380 381 382 383 384 385 386 387 388 389 390 391 396 397 398 399 400 401 402 403 404 405 406 407 408 409 410 411 412 413 414 415 416 417 418 424 425 426 427 428 429 430 431 432 433 434 435 436 437 438 439 440 441 442 443 444 445 446 452 453 454 455 456 457 458 459 460 461 462 463 464 465 466 467 468 469 470 471 472 473 474 480 481 482 483 484 485 486 487 488 489 490 491 492 493 494 495 496 497 498 499 500 501 502 503 508 509 510 511 512 513 514 515 516 517 518 519 520 521 522 523 524 525 526 527 528 529 530 531 536 537 538 539 540 541 542 543 544 545 546 547 548 549 550 551 552 553 554 555 556 557 558 559 564 565 566 567 568 569 570 571 572 573 574 575 576 577 578 579 580 581 582 583 584 585 586 587 592 593 594 595 596 597 598 599 600 601 602 603 604 605 606 607 608 609 610 611 612 613 620 621 622 623 624 625 626 627 628 629 630 631 632 633 634 635 636 637 638 639 640 648 649 650 651 652 653 654 655 656 657 658 659 660 661 662 663 664 665 666 676 677 678 679 680 681 682 683 684 685 687 688 689 690 691 692 693 705 706 707 708 709 710 711 712 734 735 736 737 738 739 740 766];

x1\_step2.xoffset = [0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0];

x1\_step2.gain = [3.77777777777778;2.28699551569507;4.67889908256881;51;39.2307692307692;39.2307692307692;39.2307692307692;30;127.5;7.96875;2;2;2;7.96875;2.58883248730964;2.00787401574803;2.83333333333333;2;2.01581027667984;2.42857142857143;2.02380952380952;2.02380952380952;2.02380952380952;56.6666666666667;2.35023041474654;2.08163265306122;2;2;2;2;2;2.00787401574803;2.36111111111111;3.26923076923077;2;2.40566037735849;2;2;2;2;2;2.00790513833992;508;72.5714285714286;3.56643356643357;2.01581027667984;2;2;2;2;2;2;2;2.01581027667984;2.00787401574803;2;2;2;2;2;2;2;2.00790513833992;2.17948717948718;5.77272727272727;12.4390243902439;2.09876543209877;2.00787401574803;2;2;2.01581027667984;2.00787401574803;2;2;2;2;2;2;2.00787401574803;2;2;2;2;2;2;2;2.00790513833992;46.3636363636364;2.77173913043478;2;2;2;2;2.00787401574803;2;2;2;2;2;2.00787401574803;2.00787401574803;2;2;2;2;2;2;2.00790513833992;2.00790513833992;42.5;2.40566037735849;2.00787401574803;2;2;2;2;2;2;2;2;2;2;2.00787401574803;2;2;2;2;2;2;2;2.00790513833992;2.00790513833992;3;2;2;2.00787401574803;2;2;2;2;2;2;2;2;2;2;2;2;2;2;2;2;2;2;2.00787401574803;15.9375;2.00787401574803;2.01581027667984;2;2;2;2;2;2;2;2;2;2;2;2;2;2;2;2;2;2;2;2;2.00787401574803;3.35526315789474;2;2;2;2;2;2;2;2;2;2;2;2;2;2;2;2;2;2;2;2;2;2;2.01581027667984;3.984375;2.40566037735849;2;2;2;2;2;2;2;2;2;2;2;2;2;2;2;2;2;2;2;2;2;2.01581027667984;8.09523809523809;3.89312977099237;2;2;2;2;2;2;2;2;2;2;2;2;2;2;2;2;2;2;2;2;2;2.01581027667984;2.64248704663212;2;2;2;2;2;2;2;2;2;2;2;2;2;2;2;2;2;2;2;2;2.00787401574803;2.00787401574803;2;2;2;2;2;2;2;2;2;2;2;2;2;2;2;2;2;2;2;2;2;2.00787401574803;2.01581027667984;2.01581027667984;2;2;2;2;2;2;2;2;2;2;2;2;2;2;2;2;2;2;2;2;2.00787401574803;5.3125;13.025641025641;2;2;2;2;2;2;2;2;2;2;2;2;2;2;2;2;2;2;2;2;2;2.00787401574803;2.00790513833992;3.93798449612403;2;2;2;2;2;2;2;2;2;2;2;2;2;2;2;2;2;2;2;2;2.00787401574803;2.00787401574803;2.04016064257028;3.93798449612403;2.36111111111111;2;2;2;2;2;2;2;2;2.01581027667984;2;2;2;2;2;2;2;2;2;2.00787401574803;2.00787401574803;2.40566037735849;14.5142857142857;26.7368421052632;3.29032258064516;2;2;2;2;2;2;2;2;2.03187250996016;2.01581027667984;2;2;2;2;2;2;2;2;2.00787401574803;2.84916201117318;23.1818181818182;3.29032258064516;2;2;2;2;2;2;2;2.00787401574803;2.41706161137441;2.41706161137441;2.01581027667984;2;2;2;2;2;2;2.25663716814159;3.984375;42.5;6.2962962962963;2.00787401574803;2;2;2;2;2;2;2.18025751072961;4.76635514018692;2.56281407035176;2.09016393442623;2.10743801652893;2.125;2;2;2;2.04;5.20408163265306;21.25;2;2;2.00787401574803;2;2;2.00787401574803;2.00787401574803;2.2972972972973;13.0769230769231;6.14457831325301;2.86516853932584;2.01581027667984;2.2972972972973;2;2.37209302325581;5.73033707865169;4.11290322580645;2.23684210526316;2.00787401574803;2.01581027667984;2;2;2.00787401574803;2.50246305418719;9.62264150943396;3.14814814814815;3.56643356643357;2.67015706806283;2.67015706806283;3.04191616766467;3.29032258064516;4.21487603305785];

x1\_step2.ymin = -1;

% Layer 1

b1 = 0.03012836904768269;

IW1\_1 = [0.019137608992262349 -0.01443692162317467 0.11292956733270548 0.10016583777742544 0.10457493134512869 -0.002886636468552022 0.046869672987817079 0.057044492483111803 0.066880889511887864 0.035271970930163071 -0.068779739406530024 0.032034786596480243 -0.099159997566007005 -0.023691220471225703 0.051999178584983083 0.11216084157111901 0.086886122811670921 -0.025316049323215673 0.041013825435205611 0.0097317875161897043 0.13554442758792787 0.1406786010367409 0.1081069418814023 -0.022382818382365232 -0.0070973310089470689 -0.040628126115876337 0.067377997618538743 0.05893987533567082 0.042299600551607489 -0.045585783542576502 -0.020003955851602299 -0.11361190411120618 0.064538906938074708 -0.022709940631922335 -0.050381193037511492 0.096404345894019541 0.053690883349711227 0.15471070768905792 -0.017748129038161819 0.083828674489239974 0.074720924291607704 -0.05423526723051899 -0.067389706900232008 -0.080800814086384454 0.059297021107150821 -0.035863843624441158 0.12924020265631078 0.069538688294707079 -0.080108941979769907 -0.12829809895134736 -0.076509027893223974 0.0083470547034485039 -0.018034063134559221 0.03480811113169157 -0.043860249704767124 -0.014083701418296565 0.08043022478491979 0.13505013760281345 0.1210033005891222 -0.09736741159911648 0.01543553985881667 0.037638914654107347 0.021229764170801867 -0.098999474746768734 0.055904978677485007 -0.055964412698635196 -0.049961230533333498 -0.032703692001177342 -0.047697558428312747 -0.047199776418645194 -0.081180229613683222 -0.079240267568912143 -0.055057396776735011 0.0093689773963254678 0.083424888969525643 0.16269771705079566 0.021872942589691559 -0.015853705001523364 0.13025583614193717 0.086048579817314327 0.049214540882870778 -0.0030635042399198759 -0.12486661676318596 -0.11508645855777778 -0.034568598591478997 -0.10514021950576664 -0.0037058159264315403 -0.10558203974204253 -0.070845607201410379 -0.038510425765433731 0.012992111284421059 0.13156782642966316 -0.022085014260530597 0.085245806600887805 -0.0023893822017744047 0.12647758445025609 0.05683468232364107 0.019570222775362453 -0.079210600055848845 -0.243373480500231 -0.00556565171550066 -0.029541454652755234 0.14822569607212532 0.25474622660060825 0.038146694780632186 -0.053626357552939104 0.020026253817079266 0.081393872281689247 0.092192994347217172 -0.043271210177060392 -0.027861369553280658 0.12167871762523957 0.028360094708177064 0.005167737917683355 0.051198430466877484 -0.038497663786283824 -0.073730200376061844 -0.037123661768447704 0.061689398036433821 0.099207629165820641 0.16911956594864377 0.047909588009408741 -0.091960608471746391 -0.086067871310703328 -0.10393011727103248 0.066912654902669205 0.15757933174399769 0.0078395439527928282 0.020628588160043652 0.027030552383477378 -0.072267079841082674 0.05119969272850574 0.078633799265100626 -0.0048053319215409793 0.0030759585450046176 0.046469017837837009 0.1384069084787353 0.067134010275923847 -0.0022800413916728604 0.078215080151588873 0.14288013014746312 0.1251175686955055 0.21097376023406311 0.059078720046979796 0.086337743913057777 -0.075836773835695884 -0.21252089288319806 -0.20272868436302571 -0.049442912110847573 0.10753320476009492 -0.050756299767459483 0.13458737511438754 -0.021304210853496302 0.036977966881670987 -0.054612065726213285 -0.081793338010407501 0.042605608954331069 -0.12533765401347113 0.041955630032319349 0.047912271995610704 -0.090720733551086244 -0.0163909492270078 0.020037703410500686 -0.0013720116079216074 0.09031532958123073 0.23512411321329052 0.14055430996192908 0.0095250062345079278 0.053762585939886386 -0.16937643209329756 -0.073903880962149268 -0.13653701048351541 0.073332193370085383 -0.007149996597745777 0.042178776234380912 -0.10894656602136016 0.030140576516168128 0.10514956174773762 0.051888158327118995 0.091609217386870445 0.027613880123312864 -0.12958450578317574 -0.11760623314185185 -0.068098142192578925 -0.045913819433528527 0.1533820539328076 -0.0070577824565935116 0.072855107425284116 0.10979895148266611 0.096510129234315858 0.090135947853823922 -0.035072351361668495 -0.16193200963371068 -0.0016571439773712866 -0.022219498522770453 -0.054319936407915485 0.070984686035643313 0.0012125968911195417 0.060642082567054625 0.022441141433528045 -0.098069276009096207 -0.023338379022553014 -0.031768860063349502 0.064531087915133289 0.005625261015226115 -0.16351766717295579 -0.042887255381118705 -0.04429525981888105 -0.043013085788544236 0.13726655650065356 0.080151249494376672 0.052585501770074108 -0.0074070492817731965 -0.046656752263361696 -0.21319524209525673 -0.21378317024303434 -0.21301058143134943 -0.26969935548865664 -0.16934298857636984 -0.093655099415031931 -0.10033552189883119 0.086930473022725482 0.14388340218688192 -0.040882338814389518 -0.097155026938643016 -0.13075209726335227 -0.070762713430699287 -0.13018523309727603 -0.06712417982861163 -0.20470947448675686 -0.13159054791189545 -0.062111875399899749 0.041225975075257146 -0.044907366922435486 -0.015750423602667152 0.042784789896563459 0.011250338264366652 -0.072439261006356503 -0.28341689376311857 -0.14831606597963048 -0.20734397264650983 -0.21228416172683334 -0.23415650991627243 -0.15920771640373249 -0.010850168183131934 -0.041109604497813396 -0.056403571670406362 0.064494990614318848 -0.17269163012311178 0.062669377720203701 0.036780823325167404 -0.16138739228793739 -0.13306413275522927 -0.02409718381070456 0.044865135785268691 -0.029269973882624013 -0.060074083469468839 0.053318915458253624 -0.093109281697148064 -0.16953470305665649 -0.13639954537473889 -0.19579960804288937 -0.20086016136220758 -0.20270087775301801 -0.065890963498783345 0.045673370066556931 0.048394050147828598 0.02363098818902036 -0.037136403653754403 0.10390015454594297 -0.035800829301284573 -0.036676568546593583 -0.12831217295910113 -0.0065753639398887596 0.01043177412883434 -0.15516249670880747 -0.057732133418380911 -0.075448853361747503 -0.134011187526523 -0.039024856991218354 0.062721694423836516 -0.10046863542789426 -0.083284278005411702 -0.19628339655206281 -0.21858354699091248 -0.09537585331253895 -0.016993480614357546 0.11317188977731443 0.080037204928226982 0.0072093468889599241 0.10655192218747493 -0.015731041338075034 0.046587074289662307 -0.044187829022985278 -0.12371748790905651 -0.060689973155748797 0.024857325683054599 0.0064944310771937333 0.018826467437886771 0.020370974441894181 -0.043548385740742289 -0.032895399506157386 0.037355886709539959 0.099601347102908716 -0.069428233723849778 -0.061257829730421855 -0.026882493479804852 -0.13542505867593077 -0.0075313795426411676 0.09118734783900391 0.0089679279092816904 0.036662760272442994 0.0079585100849090783 0.16685224633067275 0.089477584173144192 0.11234000002869482 -0.04677195469710322 -0.116790160782589 -0.028301958393442395 -0.074356003323890529 0.07049584063324249 0.060027576563154009 0.037876311330217238 0.018956357687236147 -0.011731244878329066 -0.041179803163148156 0.02156664986285841 -0.069153715188981765 -0.065242873381797087 -0.13892143034777332 -0.069817418969463765 0.01175492233539217 0.045739368911700794 0.16690511682680881 0.14903615280443983 0.10427730540527737 -0.019772767955206043 0.0287707139012532 0.24084035496620668 -0.021040627831129158 -0.0022917959901033657 0.051818165153726378 -0.12290920525108255 -0.099347415939016859 -0.065837378608566907 0.042511112537370728 -0.027495190456223395 0.070255445626334401 -0.13723063329063998 -0.11351641926300032 -0.03416498901120435 0.033742058672215559 -0.042446024047579842 -0.21969484693194929 -0.14060595217435515 -0.19857819406493865 0.046860394595495791 0.2142788906933065 0.085332483010114243 0.084833321963148328 0.016797248154706471 -0.099874630477845269 -0.028230292840624047 -0.038645629949668435 -0.080763659245955383 0.031942063687202996 0.055263639222826719 0.047978181583412285 -0.028775501020410214 -0.10945483053739057 0.037780428974281223 -0.037192068022394727 0.14016352802268173 0.051312926772595496 0.062028159620769421 0.0053361561191142444 -0.16365563555055773 -0.22077695219486315 -0.052655126613466489 0.060213482695041183 0.09024484850410798 0.1245611631826745 0.085196751041303059 0.10499313751071311 0.12709757561113863 -0.029120020250265805 0.040174361267952097 -0.15276565930745523 -0.069716261596956056 -0.041007934673758142 -0.1499844068871744 0.020763908764994312 -0.017375135190505809 -0.020048094286663924 0.051570537751539525 0.047958798449073137 0.1158284996437707 0.0387401116907145 0.10111624156304946 0.16799304664137529 -0.087392050517947115 -0.20316589774460189 -0.1407555517313307 -0.010161077979125818 0.1406813140625997 0.07085078367822098 -0.0034175274334081324 0.059359451034197722 0.21112780034660378 0.25204804606165199 0.080805823264027674 -0.080176506505924228 -0.1226415015583435 0.025574472069962977 -0.03159445715662576 0.0772570430492618 -0.14150576521804734 -0.055143810006738861 -0.063158129707129834 0.14759803420544046 0.13232776825703671 0.14852181888131583 0.027524426871754874 0.018301820563720143 -0.0033979252337834678 -0.09808128882352346 -0.048383078613559005 -0.066418670065912816 0.061189029808700578 0.086230064150190966 0.063127766548632017 0.021141100254962283 0.023256847745865825 0.073602157948292002 -0.0022102681553640311 -0.13575998604740142 -0.12439400423503662 0.02349334235450036 -0.083575279665006094 -0.018808991744987014 0.15486467473365856 0.17530622962234177 0.13972547524733928 0.098782893283373072 0.040733845842209869 -0.074265380048521457 0.0035941654693459751 0.07652362081332327 -0.097731727566846074 -0.00077900935229284875 -0.01358715578846011 -0.016930200245843331 -0.047130637964232946 0.037815673406940947 0.1182005950307783 -0.09446843867937281 -0.014056660961557922 -0.11616625208668291 0.13221537022883437 0.11355470494029632 0.060583761680133373 0.12926381938441833 0.022466343245415021 0.036334011884996881 -0.067438061981709693 -0.1263613147414602 -0.0096099365694523684 0.063154391942270349 -0.13054777573002008 -0.081566647835072878 0.0048748995617351448 -0.10141952903286967 0.05669245618921312 0.031008631262091066 0.0019858474488492514 -0.087756506679707316 -0.021110097096229779 -0.014629812961575469 0.14027528491894872 0.048476440533617458 0.072497414787200579 0.010724290076564726 -0.060322182345216607 0.047513495420633906 -0.055199819210461257 -0.093193514671119998 -0.059739506790463789 0.016116284510142111];

% Layer 2

b2 = -0.30002921168309854;

LW2\_1 = 4.8083981591072806;

% ===== SIMULATION ========

% Dimensions

Q = size(x1,2); % samples

% Input 1

xp1 = removeconstantrows\_apply(x1,x1\_step1);

xp1 = mapminmax\_apply(xp1,x1\_step2);

% Layer 1

a1 = tansig\_apply(repmat(b1,1,Q) + IW1\_1\*xp1);

% Layer 2

a2 = logsig\_apply(repmat(b2,1,Q) + LW2\_1\*a1);

% Output 1

y1 = a2;

end

% ===== MODULE FUNCTIONS ========

% Map Minimum and Maximum Input Processing Function

function y = mapminmax\_apply(x,settings)

y = bsxfun(@minus,x,settings.xoffset);

y = bsxfun(@times,y,settings.gain);

y = bsxfun(@plus,y,settings.ymin);

end

% Remove Constants Input Processing Function

function y = removeconstantrows\_apply(x,settings)

y = x(settings.keep,:);

end

% Sigmoid Positive Transfer Function

function a = logsig\_apply(n,~)

a = 1 ./ (1 + exp(-n));

end

% Sigmoid Symmetric Transfer Function

function a = tansig\_apply(n,~)

a = 2 ./ (1 + exp(-2\*n)) - 1;

end

*neural\_net\_function\_cells.m*

function [Y,Xf,Af] = neural\_net\_function\_cells(X,~,~)

%MYNEURALNETWORKFUNCTION neural network simulation function.

%

% Generated by Neural Network Toolbox function genFunction, 20-Mar-2017 16:56:01.

%

% [Y] = myNeuralNetworkFunction(X,~,~) takes these arguments:

%

% X = 1xTS cell, 1 inputs over TS timesteps

% Each X{1,ts} = 784xQ matrix, input #1 at timestep ts.

%

% and returns:

% Y = 1xTS cell of 1 outputs over TS timesteps.

% Each Y{1,ts} = 1xQ matrix, output #1 at timestep ts.

%

% where Q is number of samples (or series) and TS is the number of timesteps.

%#ok<\*RPMT0>

% ===== NEURAL NETWORK CONSTANTS =====

% Input 1

x1\_step1.keep = [100 101 102 105 106 107 108 119 120 121 122 123 124 125 128 129 130 131 132 133 134 135 136 146 147 148 149 150 151 152 153 154 155 156 157 158 159 160 161 162 163 164 165 173 174 175 176 177 178 179 180 181 182 183 184 185 186 187 188 189 190 191 192 193 194 201 202 203 204 205 206 207 208 209 210 211 212 213 214 215 216 217 218 219 220 221 222 229 230 231 232 233 234 235 236 237 238 239 240 241 242 243 244 245 246 247 248 249 250 256 257 258 259 260 261 262 263 264 265 266 267 268 269 270 271 272 273 274 275 276 277 278 284 285 286 287 288 289 290 291 292 293 294 295 296 297 298 299 300 301 302 303 304 305 306 307 312 313 314 315 316 317 318 319 320 321 322 323 324 325 326 327 328 329 330 331 332 333 334 335 340 341 342 343 344 345 346 347 348 349 350 351 352 353 354 355 356 357 358 359 360 361 362 363 368 369 370 371 372 373 374 375 376 377 378 379 380 381 382 383 384 385 386 387 388 389 390 391 396 397 398 399 400 401 402 403 404 405 406 407 408 409 410 411 412 413 414 415 416 417 418 424 425 426 427 428 429 430 431 432 433 434 435 436 437 438 439 440 441 442 443 444 445 446 452 453 454 455 456 457 458 459 460 461 462 463 464 465 466 467 468 469 470 471 472 473 474 480 481 482 483 484 485 486 487 488 489 490 491 492 493 494 495 496 497 498 499 500 501 502 503 508 509 510 511 512 513 514 515 516 517 518 519 520 521 522 523 524 525 526 527 528 529 530 531 536 537 538 539 540 541 542 543 544 545 546 547 548 549 550 551 552 553 554 555 556 557 558 559 564 565 566 567 568 569 570 571 572 573 574 575 576 577 578 579 580 581 582 583 584 585 586 587 592 593 594 595 596 597 598 599 600 601 602 603 604 605 606 607 608 609 610 611 612 613 620 621 622 623 624 625 626 627 628 629 630 631 632 633 634 635 636 637 638 639 640 648 649 650 651 652 653 654 655 656 657 658 659 660 661 662 663 664 665 666 676 677 678 679 680 681 682 683 684 685 687 688 689 690 691 692 693 705 706 707 708 709 710 711 712 734 735 736 737 738 739 740 766];

x1\_step2.xoffset = [0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0];

x1\_step2.gain = [3.77777777777778;2.28699551569507;4.67889908256881;51;39.2307692307692;39.2307692307692;39.2307692307692;30;127.5;7.96875;2;2;2;7.96875;2.58883248730964;2.00787401574803;2.83333333333333;2;2.01581027667984;2.42857142857143;2.02380952380952;2.02380952380952;2.02380952380952;56.6666666666667;2.35023041474654;2.08163265306122;2;2;2;2;2;2.00787401574803;2.36111111111111;3.26923076923077;2;2.40566037735849;2;2;2;2;2;2.00790513833992;508;72.5714285714286;3.56643356643357;2.01581027667984;2;2;2;2;2;2;2;2.01581027667984;2.00787401574803;2;2;2;2;2;2;2;2.00790513833992;2.17948717948718;5.77272727272727;12.4390243902439;2.09876543209877;2.00787401574803;2;2;2.01581027667984;2.00787401574803;2;2;2;2;2;2;2.00787401574803;2;2;2;2;2;2;2;2.00790513833992;46.3636363636364;2.77173913043478;2;2;2;2;2.00787401574803;2;2;2;2;2;2.00787401574803;2.00787401574803;2;2;2;2;2;2;2.00790513833992;2.00790513833992;42.5;2.40566037735849;2.00787401574803;2;2;2;2;2;2;2;2;2;2;2.00787401574803;2;2;2;2;2;2;2;2.00790513833992;2.00790513833992;3;2;2;2.00787401574803;2;2;2;2;2;2;2;2;2;2;2;2;2;2;2;2;2;2;2.00787401574803;15.9375;2.00787401574803;2.01581027667984;2;2;2;2;2;2;2;2;2;2;2;2;2;2;2;2;2;2;2;2;2.00787401574803;3.35526315789474;2;2;2;2;2;2;2;2;2;2;2;2;2;2;2;2;2;2;2;2;2;2;2.01581027667984;3.984375;2.40566037735849;2;2;2;2;2;2;2;2;2;2;2;2;2;2;2;2;2;2;2;2;2;2.01581027667984;8.09523809523809;3.89312977099237;2;2;2;2;2;2;2;2;2;2;2;2;2;2;2;2;2;2;2;2;2;2.01581027667984;2.64248704663212;2;2;2;2;2;2;2;2;2;2;2;2;2;2;2;2;2;2;2;2;2.00787401574803;2.00787401574803;2;2;2;2;2;2;2;2;2;2;2;2;2;2;2;2;2;2;2;2;2;2.00787401574803;2.01581027667984;2.01581027667984;2;2;2;2;2;2;2;2;2;2;2;2;2;2;2;2;2;2;2;2;2.00787401574803;5.3125;13.025641025641;2;2;2;2;2;2;2;2;2;2;2;2;2;2;2;2;2;2;2;2;2;2.00787401574803;2.00790513833992;3.93798449612403;2;2;2;2;2;2;2;2;2;2;2;2;2;2;2;2;2;2;2;2;2.00787401574803;2.00787401574803;2.04016064257028;3.93798449612403;2.36111111111111;2;2;2;2;2;2;2;2;2.01581027667984;2;2;2;2;2;2;2;2;2;2.00787401574803;2.00787401574803;2.40566037735849;14.5142857142857;26.7368421052632;3.29032258064516;2;2;2;2;2;2;2;2;2.03187250996016;2.01581027667984;2;2;2;2;2;2;2;2;2.00787401574803;2.84916201117318;23.1818181818182;3.29032258064516;2;2;2;2;2;2;2;2.00787401574803;2.41706161137441;2.41706161137441;2.01581027667984;2;2;2;2;2;2;2.25663716814159;3.984375;42.5;6.2962962962963;2.00787401574803;2;2;2;2;2;2;2.18025751072961;4.76635514018692;2.56281407035176;2.09016393442623;2.10743801652893;2.125;2;2;2;2.04;5.20408163265306;21.25;2;2;2.00787401574803;2;2;2.00787401574803;2.00787401574803;2.2972972972973;13.0769230769231;6.14457831325301;2.86516853932584;2.01581027667984;2.2972972972973;2;2.37209302325581;5.73033707865169;4.11290322580645;2.23684210526316;2.00787401574803;2.01581027667984;2;2;2.00787401574803;2.50246305418719;9.62264150943396;3.14814814814815;3.56643356643357;2.67015706806283;2.67015706806283;3.04191616766467;3.29032258064516;4.21487603305785];

x1\_step2.ymin = -1;

% Layer 1

b1 = 0.03012836904768269;

IW1\_1 = [0.019137608992262349 -0.01443692162317467 0.11292956733270548 0.10016583777742544 0.10457493134512869 -0.002886636468552022 0.046869672987817079 0.057044492483111803 0.066880889511887864 0.035271970930163071 -0.068779739406530024 0.032034786596480243 -0.099159997566007005 -0.023691220471225703 0.051999178584983083 0.11216084157111901 0.086886122811670921 -0.025316049323215673 0.041013825435205611 0.0097317875161897043 0.13554442758792787 0.1406786010367409 0.1081069418814023 -0.022382818382365232 -0.0070973310089470689 -0.040628126115876337 0.067377997618538743 0.05893987533567082 0.042299600551607489 -0.045585783542576502 -0.020003955851602299 -0.11361190411120618 0.064538906938074708 -0.022709940631922335 -0.050381193037511492 0.096404345894019541 0.053690883349711227 0.15471070768905792 -0.017748129038161819 0.083828674489239974 0.074720924291607704 -0.05423526723051899 -0.067389706900232008 -0.080800814086384454 0.059297021107150821 -0.035863843624441158 0.12924020265631078 0.069538688294707079 -0.080108941979769907 -0.12829809895134736 -0.076509027893223974 0.0083470547034485039 -0.018034063134559221 0.03480811113169157 -0.043860249704767124 -0.014083701418296565 0.08043022478491979 0.13505013760281345 0.1210033005891222 -0.09736741159911648 0.01543553985881667 0.037638914654107347 0.021229764170801867 -0.098999474746768734 0.055904978677485007 -0.055964412698635196 -0.049961230533333498 -0.032703692001177342 -0.047697558428312747 -0.047199776418645194 -0.081180229613683222 -0.079240267568912143 -0.055057396776735011 0.0093689773963254678 0.083424888969525643 0.16269771705079566 0.021872942589691559 -0.015853705001523364 0.13025583614193717 0.086048579817314327 0.049214540882870778 -0.0030635042399198759 -0.12486661676318596 -0.11508645855777778 -0.034568598591478997 -0.10514021950576664 -0.0037058159264315403 -0.10558203974204253 -0.070845607201410379 -0.038510425765433731 0.012992111284421059 0.13156782642966316 -0.022085014260530597 0.085245806600887805 -0.0023893822017744047 0.12647758445025609 0.05683468232364107 0.019570222775362453 -0.079210600055848845 -0.243373480500231 -0.00556565171550066 -0.029541454652755234 0.14822569607212532 0.25474622660060825 0.038146694780632186 -0.053626357552939104 0.020026253817079266 0.081393872281689247 0.092192994347217172 -0.043271210177060392 -0.027861369553280658 0.12167871762523957 0.028360094708177064 0.005167737917683355 0.051198430466877484 -0.038497663786283824 -0.073730200376061844 -0.037123661768447704 0.061689398036433821 0.099207629165820641 0.16911956594864377 0.047909588009408741 -0.091960608471746391 -0.086067871310703328 -0.10393011727103248 0.066912654902669205 0.15757933174399769 0.0078395439527928282 0.020628588160043652 0.027030552383477378 -0.072267079841082674 0.05119969272850574 0.078633799265100626 -0.0048053319215409793 0.0030759585450046176 0.046469017837837009 0.1384069084787353 0.067134010275923847 -0.0022800413916728604 0.078215080151588873 0.14288013014746312 0.1251175686955055 0.21097376023406311 0.059078720046979796 0.086337743913057777 -0.075836773835695884 -0.21252089288319806 -0.20272868436302571 -0.049442912110847573 0.10753320476009492 -0.050756299767459483 0.13458737511438754 -0.021304210853496302 0.036977966881670987 -0.054612065726213285 -0.081793338010407501 0.042605608954331069 -0.12533765401347113 0.041955630032319349 0.047912271995610704 -0.090720733551086244 -0.0163909492270078 0.020037703410500686 -0.0013720116079216074 0.09031532958123073 0.23512411321329052 0.14055430996192908 0.0095250062345079278 0.053762585939886386 -0.16937643209329756 -0.073903880962149268 -0.13653701048351541 0.073332193370085383 -0.007149996597745777 0.042178776234380912 -0.10894656602136016 0.030140576516168128 0.10514956174773762 0.051888158327118995 0.091609217386870445 0.027613880123312864 -0.12958450578317574 -0.11760623314185185 -0.068098142192578925 -0.045913819433528527 0.1533820539328076 -0.0070577824565935116 0.072855107425284116 0.10979895148266611 0.096510129234315858 0.090135947853823922 -0.035072351361668495 -0.16193200963371068 -0.0016571439773712866 -0.022219498522770453 -0.054319936407915485 0.070984686035643313 0.0012125968911195417 0.060642082567054625 0.022441141433528045 -0.098069276009096207 -0.023338379022553014 -0.031768860063349502 0.064531087915133289 0.005625261015226115 -0.16351766717295579 -0.042887255381118705 -0.04429525981888105 -0.043013085788544236 0.13726655650065356 0.080151249494376672 0.052585501770074108 -0.0074070492817731965 -0.046656752263361696 -0.21319524209525673 -0.21378317024303434 -0.21301058143134943 -0.26969935548865664 -0.16934298857636984 -0.093655099415031931 -0.10033552189883119 0.086930473022725482 0.14388340218688192 -0.040882338814389518 -0.097155026938643016 -0.13075209726335227 -0.070762713430699287 -0.13018523309727603 -0.06712417982861163 -0.20470947448675686 -0.13159054791189545 -0.062111875399899749 0.041225975075257146 -0.044907366922435486 -0.015750423602667152 0.042784789896563459 0.011250338264366652 -0.072439261006356503 -0.28341689376311857 -0.14831606597963048 -0.20734397264650983 -0.21228416172683334 -0.23415650991627243 -0.15920771640373249 -0.010850168183131934 -0.041109604497813396 -0.056403571670406362 0.064494990614318848 -0.17269163012311178 0.062669377720203701 0.036780823325167404 -0.16138739228793739 -0.13306413275522927 -0.02409718381070456 0.044865135785268691 -0.029269973882624013 -0.060074083469468839 0.053318915458253624 -0.093109281697148064 -0.16953470305665649 -0.13639954537473889 -0.19579960804288937 -0.20086016136220758 -0.20270087775301801 -0.065890963498783345 0.045673370066556931 0.048394050147828598 0.02363098818902036 -0.037136403653754403 0.10390015454594297 -0.035800829301284573 -0.036676568546593583 -0.12831217295910113 -0.0065753639398887596 0.01043177412883434 -0.15516249670880747 -0.057732133418380911 -0.075448853361747503 -0.134011187526523 -0.039024856991218354 0.062721694423836516 -0.10046863542789426 -0.083284278005411702 -0.19628339655206281 -0.21858354699091248 -0.09537585331253895 -0.016993480614357546 0.11317188977731443 0.080037204928226982 0.0072093468889599241 0.10655192218747493 -0.015731041338075034 0.046587074289662307 -0.044187829022985278 -0.12371748790905651 -0.060689973155748797 0.024857325683054599 0.0064944310771937333 0.018826467437886771 0.020370974441894181 -0.043548385740742289 -0.032895399506157386 0.037355886709539959 0.099601347102908716 -0.069428233723849778 -0.061257829730421855 -0.026882493479804852 -0.13542505867593077 -0.0075313795426411676 0.09118734783900391 0.0089679279092816904 0.036662760272442994 0.0079585100849090783 0.16685224633067275 0.089477584173144192 0.11234000002869482 -0.04677195469710322 -0.116790160782589 -0.028301958393442395 -0.074356003323890529 0.07049584063324249 0.060027576563154009 0.037876311330217238 0.018956357687236147 -0.011731244878329066 -0.041179803163148156 0.02156664986285841 -0.069153715188981765 -0.065242873381797087 -0.13892143034777332 -0.069817418969463765 0.01175492233539217 0.045739368911700794 0.16690511682680881 0.14903615280443983 0.10427730540527737 -0.019772767955206043 0.0287707139012532 0.24084035496620668 -0.021040627831129158 -0.0022917959901033657 0.051818165153726378 -0.12290920525108255 -0.099347415939016859 -0.065837378608566907 0.042511112537370728 -0.027495190456223395 0.070255445626334401 -0.13723063329063998 -0.11351641926300032 -0.03416498901120435 0.033742058672215559 -0.042446024047579842 -0.21969484693194929 -0.14060595217435515 -0.19857819406493865 0.046860394595495791 0.2142788906933065 0.085332483010114243 0.084833321963148328 0.016797248154706471 -0.099874630477845269 -0.028230292840624047 -0.038645629949668435 -0.080763659245955383 0.031942063687202996 0.055263639222826719 0.047978181583412285 -0.028775501020410214 -0.10945483053739057 0.037780428974281223 -0.037192068022394727 0.14016352802268173 0.051312926772595496 0.062028159620769421 0.0053361561191142444 -0.16365563555055773 -0.22077695219486315 -0.052655126613466489 0.060213482695041183 0.09024484850410798 0.1245611631826745 0.085196751041303059 0.10499313751071311 0.12709757561113863 -0.029120020250265805 0.040174361267952097 -0.15276565930745523 -0.069716261596956056 -0.041007934673758142 -0.1499844068871744 0.020763908764994312 -0.017375135190505809 -0.020048094286663924 0.051570537751539525 0.047958798449073137 0.1158284996437707 0.0387401116907145 0.10111624156304946 0.16799304664137529 -0.087392050517947115 -0.20316589774460189 -0.1407555517313307 -0.010161077979125818 0.1406813140625997 0.07085078367822098 -0.0034175274334081324 0.059359451034197722 0.21112780034660378 0.25204804606165199 0.080805823264027674 -0.080176506505924228 -0.1226415015583435 0.025574472069962977 -0.03159445715662576 0.0772570430492618 -0.14150576521804734 -0.055143810006738861 -0.063158129707129834 0.14759803420544046 0.13232776825703671 0.14852181888131583 0.027524426871754874 0.018301820563720143 -0.0033979252337834678 -0.09808128882352346 -0.048383078613559005 -0.066418670065912816 0.061189029808700578 0.086230064150190966 0.063127766548632017 0.021141100254962283 0.023256847745865825 0.073602157948292002 -0.0022102681553640311 -0.13575998604740142 -0.12439400423503662 0.02349334235450036 -0.083575279665006094 -0.018808991744987014 0.15486467473365856 0.17530622962234177 0.13972547524733928 0.098782893283373072 0.040733845842209869 -0.074265380048521457 0.0035941654693459751 0.07652362081332327 -0.097731727566846074 -0.00077900935229284875 -0.01358715578846011 -0.016930200245843331 -0.047130637964232946 0.037815673406940947 0.1182005950307783 -0.09446843867937281 -0.014056660961557922 -0.11616625208668291 0.13221537022883437 0.11355470494029632 0.060583761680133373 0.12926381938441833 0.022466343245415021 0.036334011884996881 -0.067438061981709693 -0.1263613147414602 -0.0096099365694523684 0.063154391942270349 -0.13054777573002008 -0.081566647835072878 0.0048748995617351448 -0.10141952903286967 0.05669245618921312 0.031008631262091066 0.0019858474488492514 -0.087756506679707316 -0.021110097096229779 -0.014629812961575469 0.14027528491894872 0.048476440533617458 0.072497414787200579 0.010724290076564726 -0.060322182345216607 0.047513495420633906 -0.055199819210461257 -0.093193514671119998 -0.059739506790463789 0.016116284510142111];

% Layer 2

b2 = -0.30002921168309854;

LW2\_1 = 4.8083981591072806;

% ===== SIMULATION ========

% Format Input Arguments

isCellX = iscell(X);

if ~isCellX, X = {X}; end;

% Dimensions

TS = size(X,2); % timesteps

if ~isempty(X)

Q = size(X{1},2); % samples/series

else

Q = 0;

end

% Allocate Outputs

Y = cell(1,TS);

% Time loop

for ts=1:TS

% Input 1

temp = removeconstantrows\_apply(X{1,ts},x1\_step1);

Xp1 = mapminmax\_apply(temp,x1\_step2);

% Layer 1

a1 = tansig\_apply(repmat(b1,1,Q) + IW1\_1\*Xp1);

% Layer 2

a2 = logsig\_apply(repmat(b2,1,Q) + LW2\_1\*a1);

% Output 1

Y{1,ts} = a2;

end

% Final Delay States

Xf = cell(1,0);

Af = cell(2,0);

% Format Output Arguments

if ~isCellX, Y = cell2mat(Y); end

end

% ===== MODULE FUNCTIONS ========

% Map Minimum and Maximum Input Processing Function

function y = mapminmax\_apply(x,settings)

y = bsxfun(@minus,x,settings.xoffset);

y = bsxfun(@times,y,settings.gain);

y = bsxfun(@plus,y,settings.ymin);

end

% Remove Constants Input Processing Function

function y = removeconstantrows\_apply(x,settings)

y = x(settings.keep,:);

end

% Sigmoid Positive Transfer Function

function a = logsig\_apply(n,~)

a = 1 ./ (1 + exp(-n));

end

% Sigmoid Symmetric Transfer Function

function a = tansig\_apply(n,~)

a = 2 ./ (1 + exp(-2\*n)) - 1;

end