Question 1:

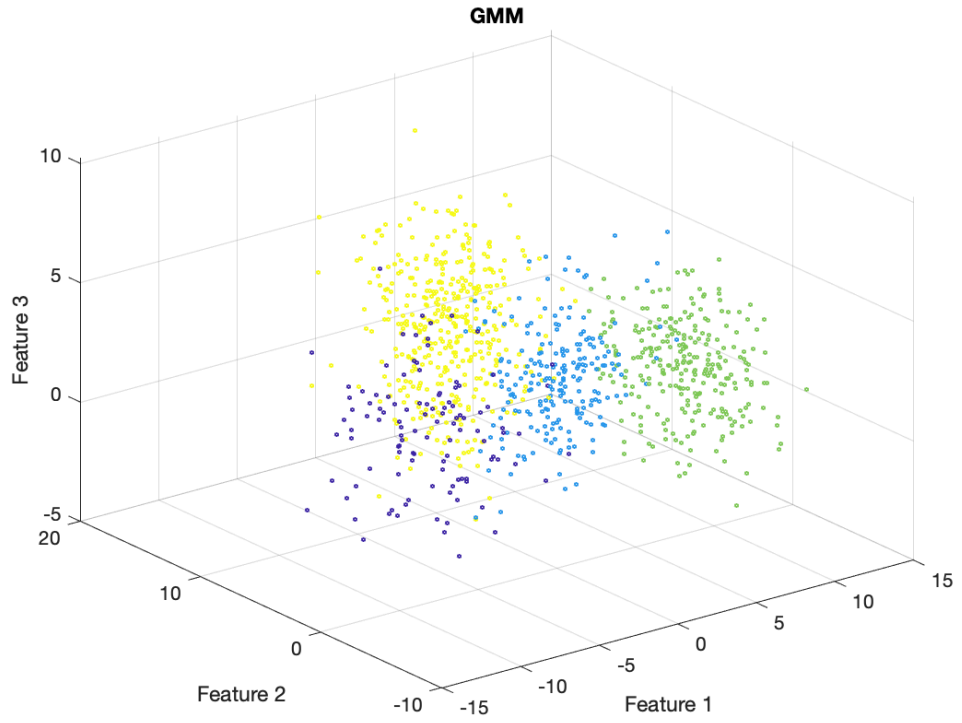


Figure 1: 1000 sample 4 component GMM

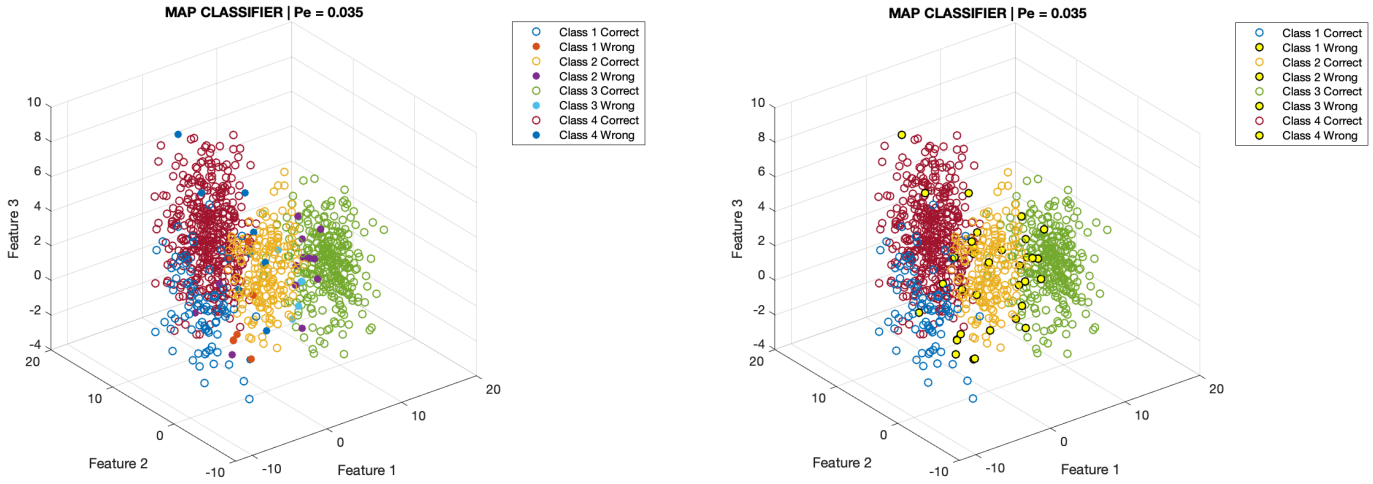


Figure 2: 1000 sample 4 Component GMM with MAP classification (different legends for easier visualization)

Value	Count	Percent
1	110	11.00%
2	213	21.30%
3	276	27.60%
4	401	40.10%

Figure 3: Priors (actual class distribution)

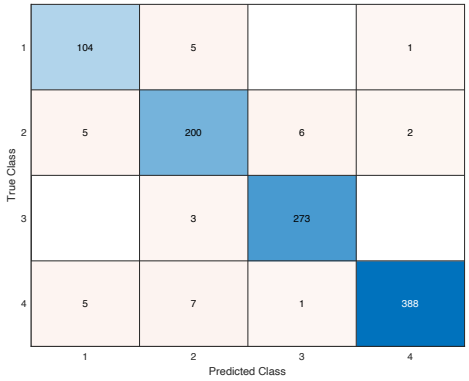


Figure 4: MAP Classification Confusion Matrix

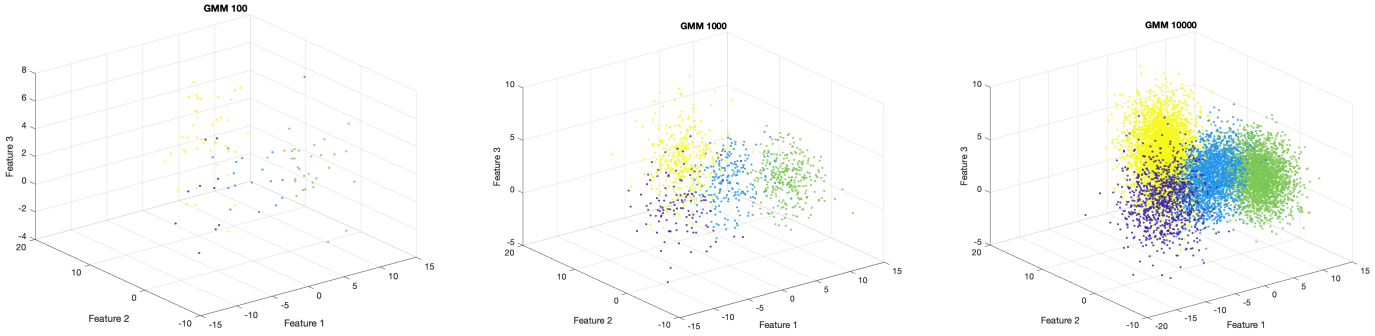


Figure 5: Three training datasets (100,1000,10000 samples)

Table 1: MATLAB OUTPUT

Set 1 - # Hidden Units 2 - CE 0.115392
Set 1 - # Hidden Units 5 - CE 0.084303
Set 1 - # Hidden Units 10 - CE 0.074914
Set 1 - # Hidden Units 15 - CE 0.072518
Set 1 - # Hidden Units 25 - CE 0.101746
Set 1 - # Hidden Units 50 - CE 0.085702
Set 2 - # Hidden Units 2 - CE 0.070045
Set 2 - # Hidden Units 5 - CE 0.040280
Set 2 - # Hidden Units 10 - CE 0.035730
Set 2 - # Hidden Units 15 - CE 0.039658
Set 2 - # Hidden Units 25 - CE 0.032660
Set 2 - # Hidden Units 50 - CE 0.130219
Set 3 - # Hidden Units 2 - CE 0.059482
Set 3 - # Hidden Units 5 - CE 0.027569
Set 3 - # Hidden Units 10 - CE 0.026504
Set 3 - # Hidden Units 15 - CE 0.026096
Set 3 - # Hidden Units 25 - CE 0.026603
Set 3 - # Hidden Units 50 - CE 0.027291
Best Dataset 3 - Best # Hidden Units 15
p_error =

0.0330

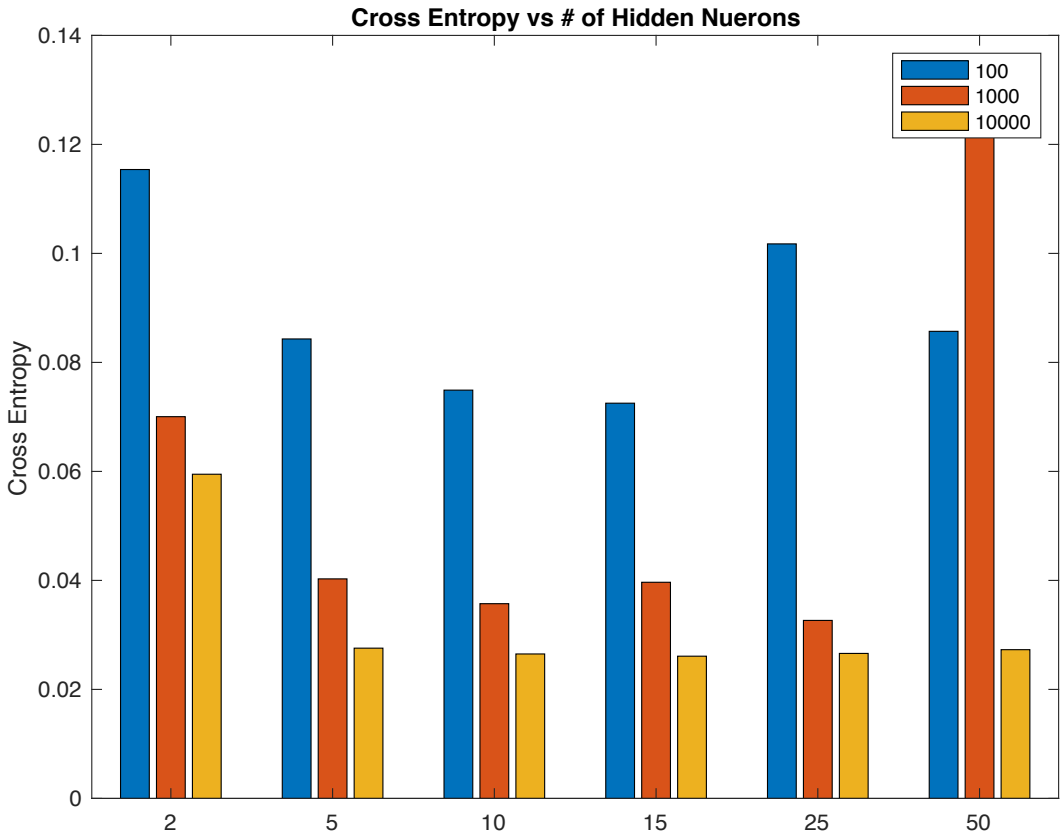


Figure 6: Cross Entropy vs # of Hidden Units

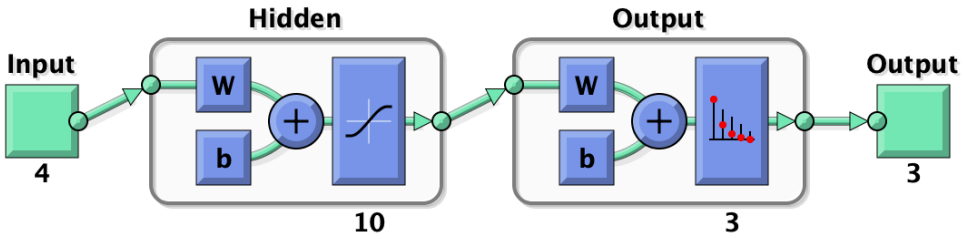


Figure 7: Neural Network Model (view function MATLAB)

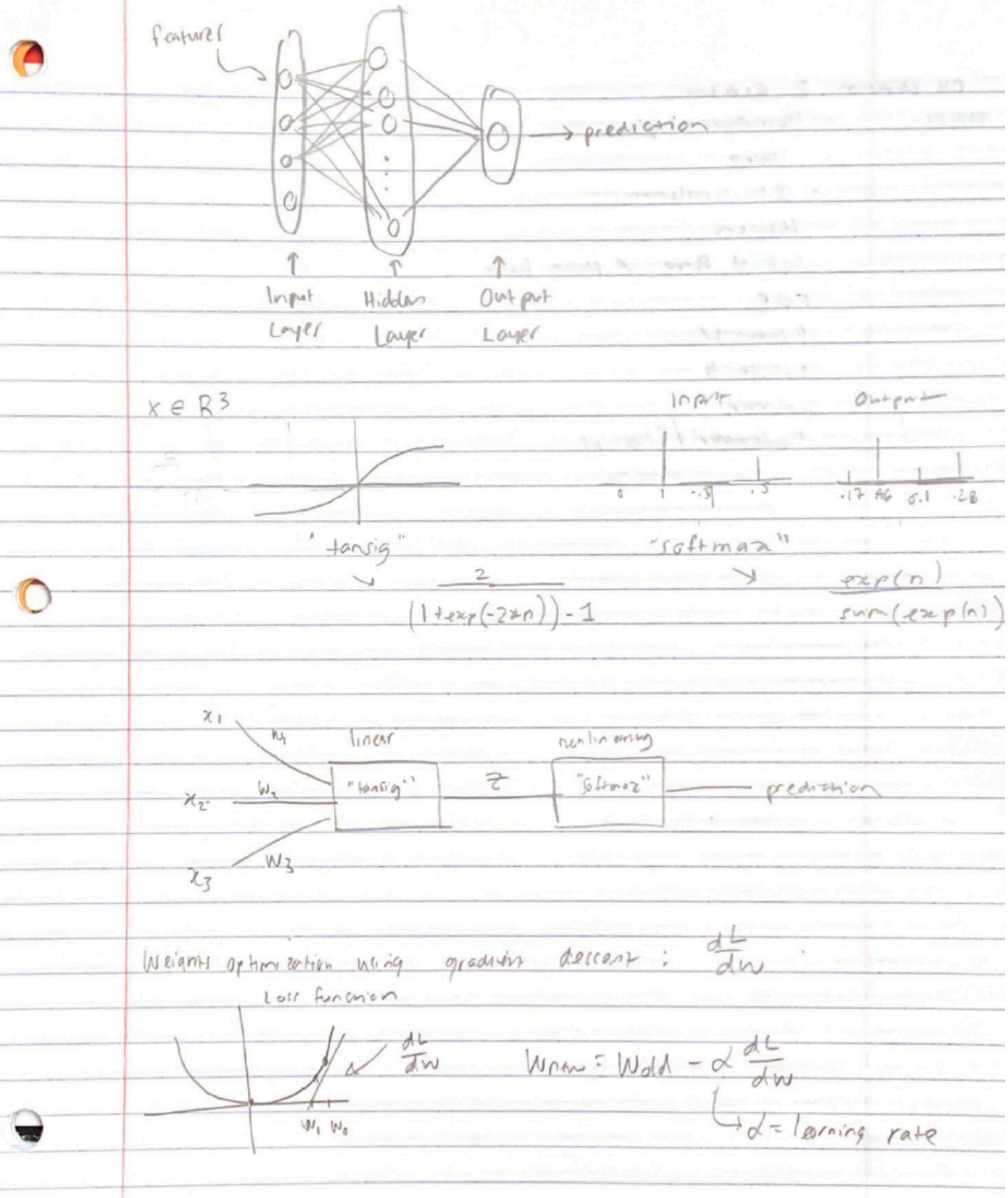


Figure 8: Neural Network Internals and Optimization Expression

Report: To begin the experiment MATLAB's gmmdistribution function is used to generate a 3D 4 component GMM with the following properties.

```
% Generate samples from a 4-component GMM
alpha_true = [0.1,0.2,0.3,.4]; %priors
mu_true = [-8 0 8 0;0 0 0 9; 2 3 2 3]';
Sigma_true(:,:,1) = [7 1 1;1 8 1; 1 1 4];
Sigma_true(:,:,2) = [7 1 1;1 2 1; 1 1 3];
Sigma_true(:,:,3) = [4 1 1;1 8 1; 1 1 2];
Sigma_true(:,:,4) = [4 1 1;1 8 1; 1 1 5];
```

The resulting 3D GMM is shown in figure 1. Next off, the MAP classifier is applied to the GMM and is derived in the code as follows by picking the max class prior probabilities.

```
% decision rule is decide class j if p(c_j | x) > p(c_k | x) and  p(c_l | x) for all j
pc1_x = mvnpdf(Y, mu_true(1,:), Sigma_true(:,:,1)) * alpha_true(1);
pc2_x = mvnpdf(Y, mu_true(2,:), Sigma_true(:,:,2)) * alpha_true(2);
pc3_x = mvnpdf(Y, mu_true(3,:), Sigma_true(:,:,3)) * alpha_true(3);
pc4_x = mvnpdf(Y, mu_true(4,:), Sigma_true(:,:,3)) * alpha_true(4);
[~,predictions] = max([pc1_x,pc2_x,pc3_x,pc4_x],[],2);
```

The results of applying the MAP classifier are shown in figure 2, where the wrongly classified samples are denoted by a yellow mark. In order to calculate the minimum probability of error, the number of misclassified samples is divided by the total number of samples and is determined to be .035.

Next off, three separate training datasets are generated with 100, 1000, 10000 samples along with their true labels and is visualized in figure 5. For each dataset, a multilayer perceptron is trained using the maximum likelihood principle. To setup the neural net, the tool of choice is patternnet from MATLAB's machine learning toolbox (the following is a description from documentation)

patternnet R2019b

Pattern recognition network

Syntax

```
patternnet(hiddenSizes,trainFcn,performFcn)
```

Description

Pattern recognition networks are feedforward networks that can be trained to classify inputs according to target classes. The target data for pattern recognition networks should consist of vectors of all zero values except for a 1 in element `i`, where `i` is the class they are to represent.

`patternnet(hiddenSizes,trainFcn,performFcn)` takes these arguments,

`hiddenSizes`	Row vector of one or more hidden layer sizes (default = 10)
`trainFcn`	Training function (default = `'trainscg'`)
`performFcn`	Performance function (default = `'crossentropy'`)

and returns a pattern recognition neural network.

The defaults of patternnet are a cross entropy performance measurement and a multilayer neural network with input layer being a sigmoid, "tansig" and output layer being a softmax function, or in other words, the ouput layer nonlinearity is a normalized exponential function. The following function is used to setup this model.

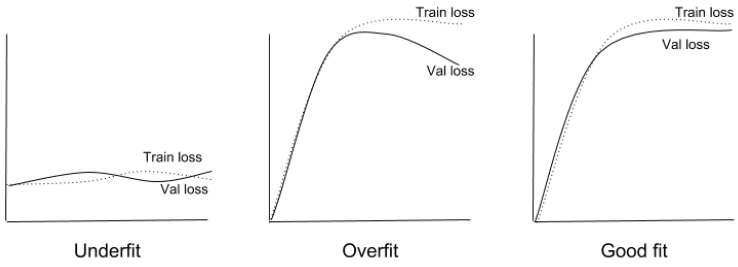
```
function net = setupNN(activation_function,neurons)
    net = patternnet(neurons);
    %net.layers{1}.transferFcn = activation_function;
    %net.layers{2}.transferFcn = "softmax";
    net.divideParam.trainRatio=0.80;
    net.divideParam.valRatio=0.20;
    net.divideParam.testRatio=0;
    net.trainParam.showWindow=0;
    net.trainParam.epochs=2500;
end
```

Figure 7 shows the Neural Network Model and Figure 8 shows the mathematical internals of a neural network and how gradient descent is used to find the local minimum and optimize the parameters of the weights to determine the maximum likelihood of fitting the model.

Next off in the main for loop of the program, the indices of the data are divided into 10 groups 10-fold cross validation where the cross-entropy loss is determined from the fitting of various neural net models. These models are multi-layer MLP functions with a hidden hyperbolic tangent transfer function layer and an output normalized exponential function layer. From there, the number of neurons is varied from 2,5,10,15,25,50 neurons. Keeping track of the cross-entropy scores, the results are reported in Table 1 and visualized in Figure 6 where the best model has a number of neurons somewhere in the middle of the set being tested. Cross entropy loss is a good candidate for similarly measuring probability of error since it measures the performance of a classification model whose output is a probability between 0 and 1. The best dataset is set 3 (10000 samples) and 15 neurons and a cross entropy score of 0.026096. The results are somewhat expectable from the properties of neural networks. Neural networks usually have nonconvex functions and there are a lot of local minimum which results in the model getting "stuck" as a less than optimal set of weights without realization. Thus, there is the need to fit different models and number of hidden perceptions. Finally, the trained neural network classifiers are applied to the test dataset previously generated. As shown in table 1, the probability of error is.0330 and is calculated using the following equation where "classes" is the predicted labels and "compIdx" is the true label.

```
%% P(error)
p_error = sum(classes'~=compIdx)/N
```

The neural network probability of error is nearly identical the MAP classifier theoretical minimum probability of error and actually performs slightly better (.0330 as opposed to .035). The results clearly show that a larger training size helps the model to learn better and this prevents overfitting to a specific dataset which helps generalization to new or unseen datasets. This balance is shown in the following charts of learning curves taken online (https://mc.ai/introduction-to-regularization-to-reduce-overfitting-of-deep-learning-neural-networks/)



This shows the importance of testing different models with different datasets and dataset sizes. Neural networks are a powerful tool and it is beneficial to understand how they work interanally.

QUESTION 2:

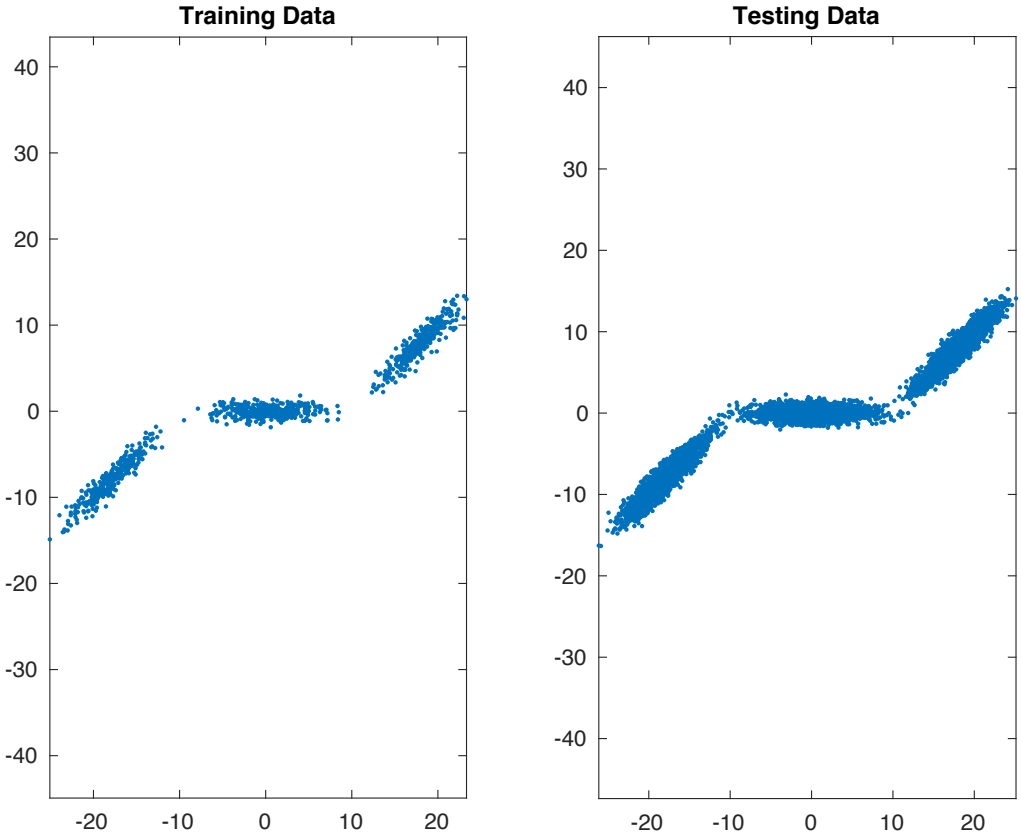


Figure 1: Generated Data

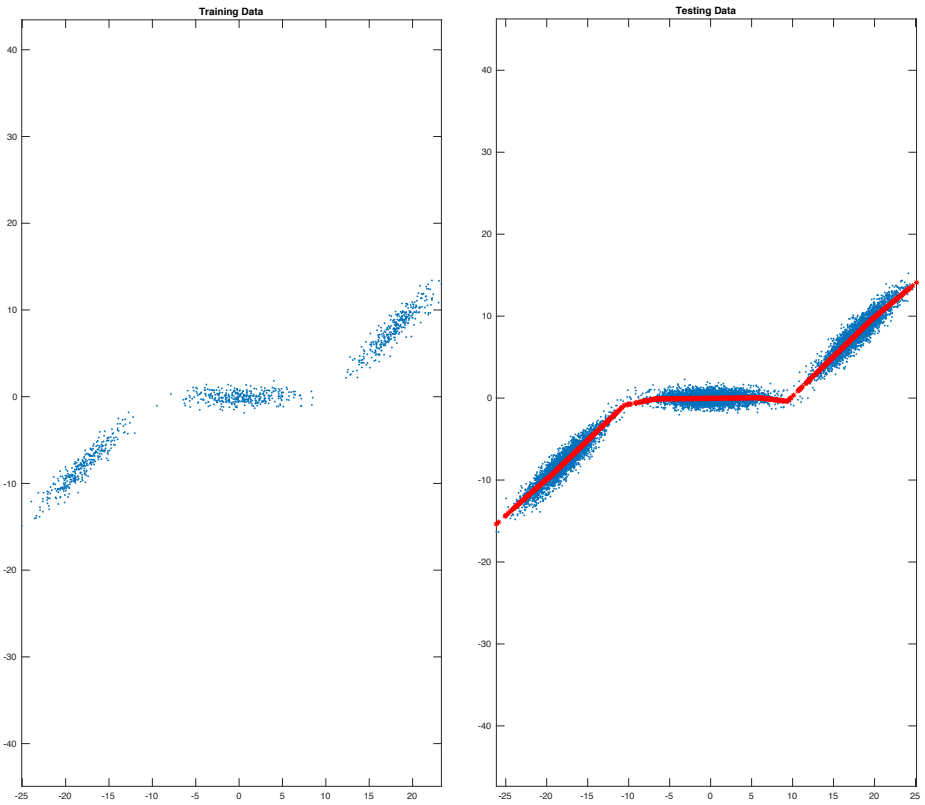


Figure 2: Best Model Fitting

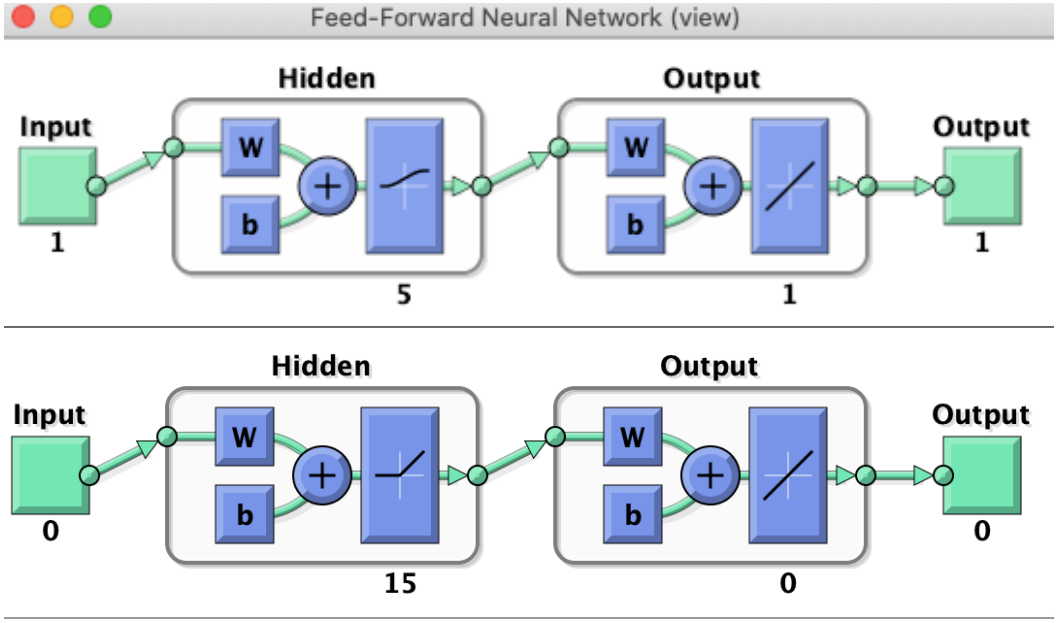


Figure 3: Neural Network

Table 1: MATLAB OUTPUT:

Transfer Fun logsig - # Hidden Units 2 - MSE 0.653572
Transfer Fun logsig - # Hidden Units 5 - MSE 0.609288
Transfer Fun logsig - # Hidden Units 7 - MSE 0.625990
Transfer Fun logsig - # Hidden Units 10 - MSE 0.646949
Transfer Fun logsig - # Hidden Units 15 - MSE 0.662029
Transfer Fun logsig - # Hidden Units 20 - MSE 0.768704
Transfer Fun poslin - # Hidden Units 2 - MSE 9.974078
Transfer Fun poslin - # Hidden Units 5 - MSE 1.490921
Transfer Fun poslin - # Hidden Units 7 - MSE 1.083377
Transfer Fun poslin - # Hidden Units 10 - MSE 0.882189
Transfer Fun poslin - # Hidden Units 15 - MSE 0.607321
Transfer Fun poslin - # Hidden Units 20 - MSE 0.605699
Best TSFN Function poslin - Best # Hidden Units 20

MSE_FINAL =

0.6039

RESULTS:

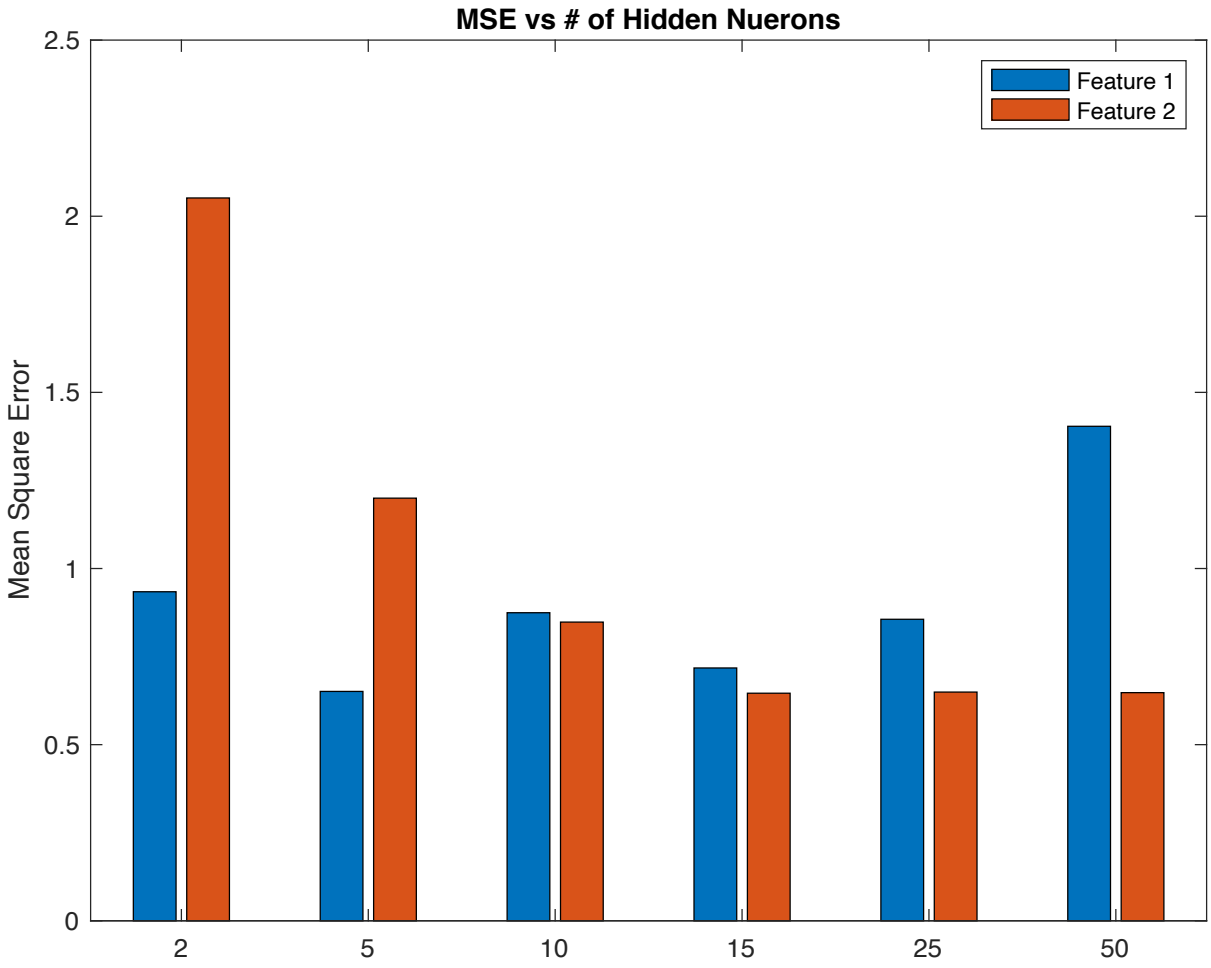


Figure 4: MSE vs # of Hidden Units

Report: To begin the experiment, 1000 samples of two-dimensional data is generated as a training set and 10000 samples are generated as a testing dataset. The resulting data is plotted and shown in Figure 1. Next off, a function is created to define the parameters of the neural network layers. The MATLAB feedforwardnet toolbox is used and below is a description of the function.

Syntax

```
feedforwardnet(hiddenSizes,trainFcn)
```

Description

Feedforward networks consist of a series of layers. The first layer has a connection from the network input. Each subsequent layer has a connection from the previous layer. The final layer produces the network's output.

Feedforward networks can be used for any kind of input to output mapping. A feedforward network with one hidden layer and enough neurons in the hidden layers, can fit any finite input-output mapping problem.

I created a function to define the neural net as follows:

```
function net = define_NN(activation_function,neurons)
    net = feedforwardnet(neurons);
    net.layers{1}.transferFcn = activation_function;
    net.divideParam.trainRatio=1.0;
    net.divideParam.valRatio=0;
    net.divideParam.testRatio=0;
    net.trainParam.showWindow=0;
    net.trainParam.epochs=2500;
end
```

The activation functions passed are “logsig” and “poslin” which are MATLAB ready logistic and Relu functions. The MATLAB toolbox does not have a softplus activation function so poslin, the positive linear transfer function, is used for reproducible results. The output layer is linear (no nonlinearity). This structure is shown in figure 3 Similar to question 1, the internals of a neural network can be explained as shown in figure 8. In the main for loop of the program, the indices of the data are divided into 10 groups 10-fold cross validation where the average mean square error is determined from the fitting of various neural net models. These models are single layer MLP functions with either logistic or Relu transfer function. From there, the number of neurons is varied from 2,5,10,15,25,50 neurons. Keeping track of the MSE scores, the results are reported in Table 1 and visualized in Figure 4 where the best model has a number of neurons somewhere in the middle of the set being tested. The results are somewhat expectable from the properties of neural networks. Neural networks usually have nonconvex functions and there are a lot of local minimum which results in the model getting “stuck” as a less than optimal set of weights without realization. Thus, there is the need to fit different models and number of hidden perceptions. The results showed the model that minimizes the mean square error is “poslin” activation function with 20 hidden units. The number of hidden units to test with were chosen on an arbitrary linear scale. The MSE on the training set was 0.605699 as shown in Table 1. This value is calculated using the perform function as follows from the toolbox:

```
MSE_FINAL = perform(net,y_pred,Xtest(2,:))
```

This trained model is then applied to the test dataset where the predicted x2 values are overlaid in the original plot and is shown in figure 2. Similarly, the MSE on the final test dataset is MSE = .6039 which is similar to the training MSE.

Appendix Code:

```
%% Generate Data Question 1

clear all, close all, clc

plotData = 1;
n = 2; Ntrain = 1000; Ntest = 10000;
alpha = [0.33,0.34,0.33]; % must add to 1.0
meanVectors = [-18 0 18;-8 0 8];
covEvalues = [3.2^2 0;0 0.6^2];
covEvectors(:,:,1) = [1 -1;1 1]/sqrt(2);
covEvectors(:,:,2) = [1 0;0 1];
covEvectors(:,:,3) = [1 -1;1 1]/sqrt(2);

t = rand(1,Ntrain);
ind1 = find(0 <= t & t <= alpha(1));
ind2 = find(alpha(1) < t & t <= alpha(1)+alpha(2));
ind3 = find(alpha(1)+alpha(2) <= t & t <= 1);
Xtrain = zeros(n,Ntrain);
Xtrain(:,ind1) =
covEvectors(:,:,1)*covEvalues^(1/2)*randn(n,length(ind1))+meanVectors(:,1);
Xtrain(:,ind2) =
covEvectors(:,:,2)*covEvalues^(1/2)*randn(n,length(ind2))+meanVectors(:,2);
Xtrain(:,ind3) =
covEvectors(:,:,3)*covEvalues^(1/2)*randn(n,length(ind3))+meanVectors(:,3);

t = rand(1,Ntest);
ind1 = find(0 <= t & t <= alpha(1));
ind2 = find(alpha(1) < t & t <= alpha(1)+alpha(2));
ind3 = find(alpha(1)+alpha(2) <= t & t <= 1);
Xtest = zeros(n,Ntrain);
Xtest(:,ind1) =
covEvectors(:,:,1)*covEvalues^(1/2)*randn(n,length(ind1))+meanVectors(:,1);
Xtest(:,ind2) =
covEvectors(:,:,2)*covEvalues^(1/2)*randn(n,length(ind2))+meanVectors(:,2);
Xtest(:,ind3) =
covEvectors(:,:,3)*covEvalues^(1/2)*randn(n,length(ind3))+meanVectors(:,3);

if plotData == 1
    figure(1), subplot(1,2,1),
    plot(Xtrain(1,:),Xtrain(2,:),'.')
    title('Training Data'), axis equal,
    subplot(1,2,2),
    plot(Xtest(1,:),Xtest(2,:),'.')
    title('Testing Data'), axis equal,
end


%%
data = Xtrain(1,:);
label = Xtrain(2,:);
indices = crossvalind('Kfold',Xtrain(2,:),10);
%%
tsfn_fun = ["logsig","poslin"];
%hidden_units = [2 5 7 10 25 50 100];
hidden_units = [2 5 10 15 25 50];
for i = 1:length(tsfn_fun)
    tsfn_ = tsfn_fun(i);
    for j = 1:length(hidden_units)
```

```
        hidden = hidden_units(j);
        net = define_NN(tsfn_,hidden);
        mean_cv = 0;
        for k = 1:10
            test_data = (indices == k);
            train_data = ~test_data;
            trained_net = train(net,data(train_data),label(train_data));
            y_pred = trained_net(data(test_data));
            MSE = perform(trained_net,y_pred,label(test_data));
            cv(k) = MSE;
            if k == 10
                mean_cv = mean(cv);
            end
        end
        metrics(i,j) = mean_cv;
        fprintf("Transfer Fun %s - # Hidden Units %d - MSE %f\n",tsfn_,hidden,mean_cv)
        pause(0.00001)
    end
end
%%
[minimum,idx] = min(metrics,[],'all','linear');
[act_idx,hidden_idx] = ind2sub(size(metrics),idx);
transfer_fun = tsfn_fun(act_idx);
hidden_unit = hidden_units(hidden_idx);
fprintf("Best TSFN Function %s - Best # Hidden Units %d\n",transfer_fun,hidden_unit)
%%
net = define_NN(transfer_fun,hidden_unit);
trained_net = train(net,data,label);

y_pred = trained_net(Xtest(1,:));
figure(1)
subplot(1,2,2)
hold on
plot(Xtest(1,:),y_pred,'r*')
MSE_FINAL = perform(net,y_pred,Xtest(2,:))


%% Show validation results
figure
i = 1:length(hidden_units);
bar(i,metrics','grouped');
g = gca;
xticklabels = {'2','5','10','15','25','50'}';
g.XTickLabel = xticklabels;
legend(["Feature 1","Feature 2",]);
ylabel("Mean Square Error")
title("MSE vs # of Hidden Nuerons")
```

```
%% Generate Data Question 2
clear all; close all; clc

% Generate samples from a 4-component GMM
alpha_true = [0.1,0.2,0.3,.4]; %priors
mu_true = [-8 0 8 0;0 0 0 9; 2 3 2 3]';
Sigma_true(:,:,1) = [7 1 1;1 8 1; 1 1 4];
Sigma_true(:,:,2) = [7 1 1;1 2 1; 1 1 3];
Sigma_true(:,:,3) = [4 1 1;1 8 1; 1 1 2];
Sigma_true(:,:,4) = [4 1 1;1 8 1; 1 1 5];


%Creat GMM
gm = gmdistribution(mu_true, Sigma_true, alpha_true);
%rng('default');
N = 1000;
[Y, compIdx] = random(gm,N);

% True GMM visualization
figure(1)
scatter3(Y(:,1),Y(:,2),Y(:,3),3,compIdx);
title('GMM');
xlabel('Feature 1')
ylabel('Feature 2')
zlabel('Feature 3')


%%
pc1_x = mvnpdf(Y, mu_true(1,:), Sigma_true(:,:,1)) * alpha_true(1);
pc2_x = mvnpdf(Y, mu_true(2,:), Sigma_true(:,:,2)) * alpha_true(2);
pc3_x = mvnpdf(Y, mu_true(3,:), Sigma_true(:,:,3)) * alpha_true(3);
pc4_x = mvnpdf(Y, mu_true(4,:), Sigma_true(:,:,3)) * alpha_true(4);
[~,predictions] = max([pc1_x,pc2_x,pc3_x,pc4_x],[],2);


%% a
tabulate(compIdx);
C = confusionmat(compIdx,predictions);
figure(2)
confusionchart(C)
sum(compIdx~=predictions);
sum(compIdx~=predictions)/length(Y);
%% Function to plot MAP estimate errors
figure(3)

data = Y;
classIndex =  compIdx;
nClass = 4;
my_inference = predictions;
```

```
markerStrings = 'xo.+';
errors = my_inference~=classIndex;
legendStrings = [];
for idxClass = 1:nClass
    idx_correct = (~errors) & (classIndex==idxClass);
    dataClass = data(idx_correct,:);
    scatter3(dataClass(:,1), dataClass(:,2) ,dataClass(:,3));
    hold on
    if ~isempty(dataClass)
        legendStrings = [legendStrings "Class "+num2str(idxClass)+" Correct"];
    end

    idx_incorrect = (errors) & (classIndex==idxClass);
    dataClass = data(idx_incorrect,:);
    scatter3(dataClass(:,1), dataClass(:,2) , dataClass(:,3),
'MarkerEdgeColor','k','MarkerFaceColor',[1 1 0] );
    hold on
    if ~isempty(dataClass)
        legendStrings = [legendStrings "Class "+num2str(idxClass)+" Wrong"];
    end

    %
end
xlabel('Feature 1');
ylabel('Feature 2');
zlabel('Feature 3');

legend(legendStrings);
title("MAP CLASSIFIER | Pe = "+num2str(1-(1-sum(errors)/length(data))))

%% Part 3

%Creat GMM 100, 1000, 10000 Samples
gm_100 = gmdistribution(mu_true, Sigma_true, alpha_true);
gm_1000 = gmdistribution(mu_true, Sigma_true, alpha_true);
gm_10000 = gmdistribution(mu_true, Sigma_true, alpha_true);

[Y_100, compIdx_100] = random(gm,100);
[Y_1000, compIdx_1000] = random(gm,1000);
[Y_10000, compIdx_10000] = random(gm,10000);

% True GMM visualization
figure(4)
scatter3(Y_100(:,1),Y_100(:,2),Y_100(:,3),3,compIdx_100);
title('GMM 100'); xlabel('Feature 1'); ylabel('Feature 2'); zlabel('Feature 3')

figure(5)
scatter3(Y_1000(:,1),Y_1000(:,2),Y_1000(:,3),3,compIdx_1000);
title('GMM 1000'); xlabel('Feature 1'); ylabel('Feature 2'); zlabel('Feature 3')

figure(6)
scatter3(Y_10000(:,1),Y_10000(:,2),Y_10000(:,3),3,compIdx_10000);
title('GMM 10000'); xlabel('Feature 1'); ylabel('Feature 2'); zlabel('Feature 3')


%% Organize data
data = cell(1,3);
data{1} = Y_100;
data{2} = Y_1000;
data{3} = Y_10000;
```

```
label = cell(1,3)
label{1} = compIdx_100;
label{2} = compIdx_1000;
label{3} = compIdx_10000;

%% K fold cross validation

for sets = 1:3
    indices = crossvalind('Kfold',label{sets},10);
    tsfn_fun = "logsig";
    hidden_units = [2 5 10 15 25 50];
    dataset = data{sets};
    datalabels = label{sets};
    for j = 1:length(hidden_units)
        hidden = hidden_units(j);
        net = setupNN(tsfn_fun,hidden);
        mean_cv = 0;
        for k = 1:10
            test_data = (indices == k);
            train_data = ~test_data;
            trained_net =
train(net,dataset(train_data,:)',ind2vec(datalabels(train_data)',4));
            y_pred = trained_net(dataset(test_data,:)');
            CE = perform(trained_net,ind2vec(datalabels(test_data)',4),y_pred);
            cv(k) = CE;
            if k == 10
                mean_cv = mean(cv);
            end
        end
        metrics(sets,j) = mean_cv;
        fprintf("Set %d - # Hidden Units %d - CE %f\n",sets,hidden,mean_cv)
        pause(0.00001)
    end
end


%% Find best Dataset and # neurons
[minimum,idx] = min(metrics,[],'all','linear');
[set,hidden_idx] = ind2sub(size(metrics),idx);
hidden_unit = hidden_units(hidden_idx);
fprintf("Best Dataset %d - Best # Hidden Units %d\n",set,hidden_unit)

%% Apply trained neural network to test dataset
net = setupNN(tsfn_fun,hidden_unit);
dataset = data{set}';
datasetlabel = label{set}';
trained_net = train(net,dataset,ind2vec(datasetlabel,4));

y_pred = trained_net(Y');
perf = perform(trained_net,ind2vec(compIdx',4),y_pred);
classes = vec2ind(y_pred);

%% P(error)
p_error = sum(classes'~=compIdx)/N
%% Show validation results
figure
i = 1:length(hidden_units);
bar(i,metrics','grouped');
g = gca;
xticklabels = {'2','5','10','15','25','50'}';
g.XTickLabel = xticklabels;
legend(["100","1000","10000"]);
ylabel("Cross Entropy")
```

```
title("Cross Entropy vs # of Hidden Nuerons")
```