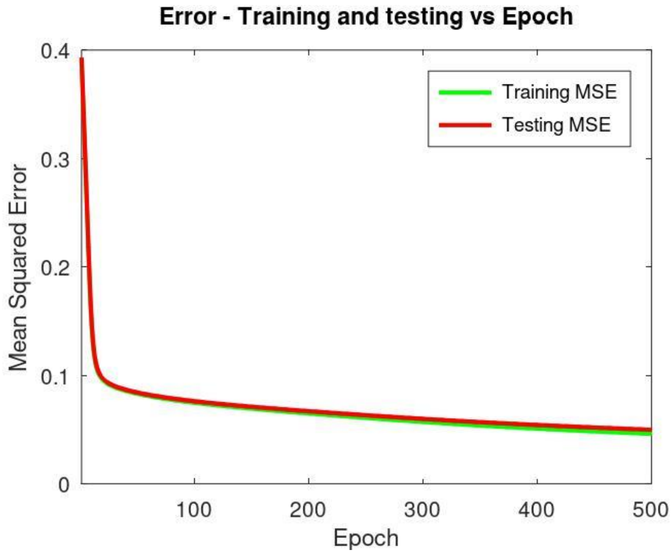
Spr 2023: CSCI 4/5588 Programming Assignment #3

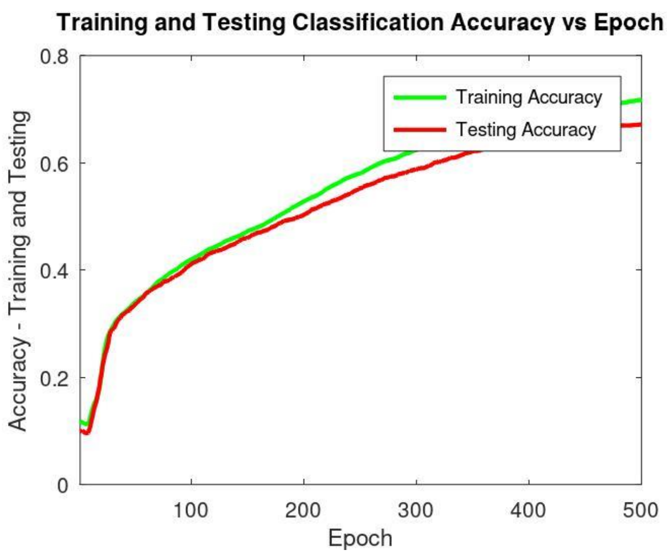
Nane: Aviral Kandel

ID: 2630609

MSE and Classification accuracy:

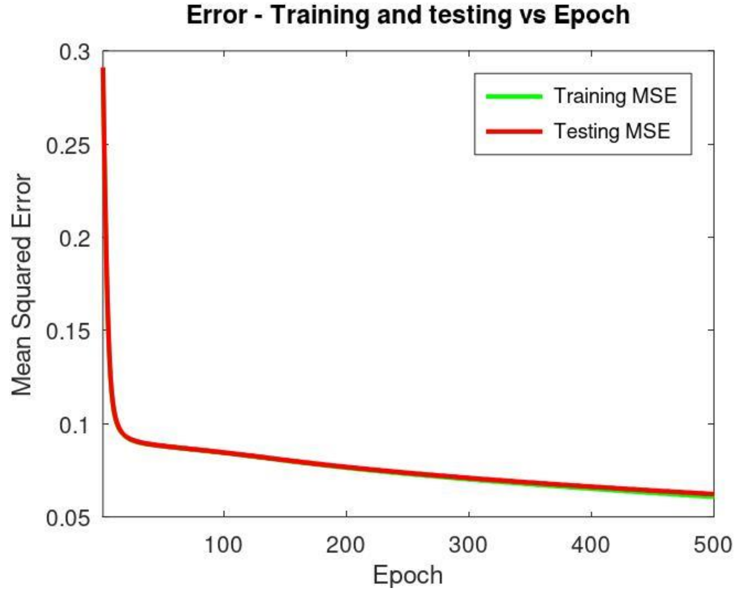
a. ANN with 1 hidden layer.
   L = [25, 49, 10]

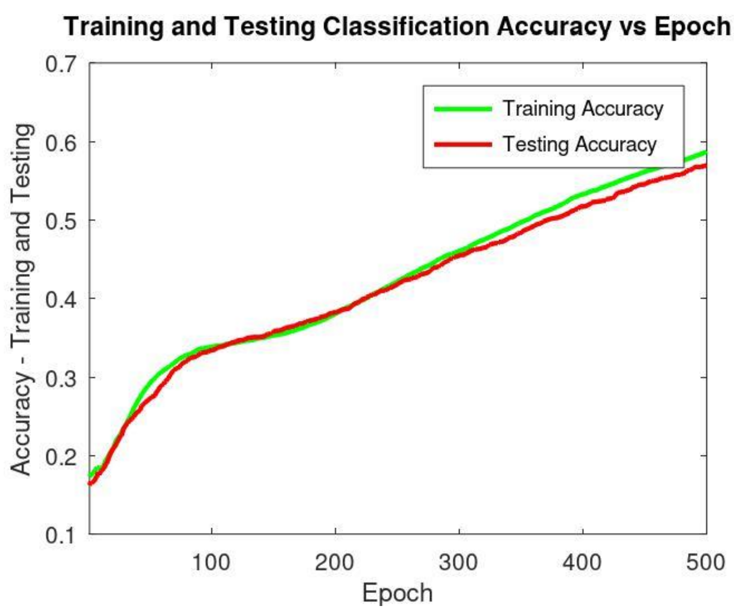




ANN with 1 hidden layer	
Minimum Error	0.0257
Minimum Error Epoch	2000
Final Training accuracy	89.26%
Final Testing accuracy	85.15%
Best Training accuracy	89.26%
Best Testing accuracy	85.15%

b. ANN with2 hidden layers.
L = [25, 67, 34 10]

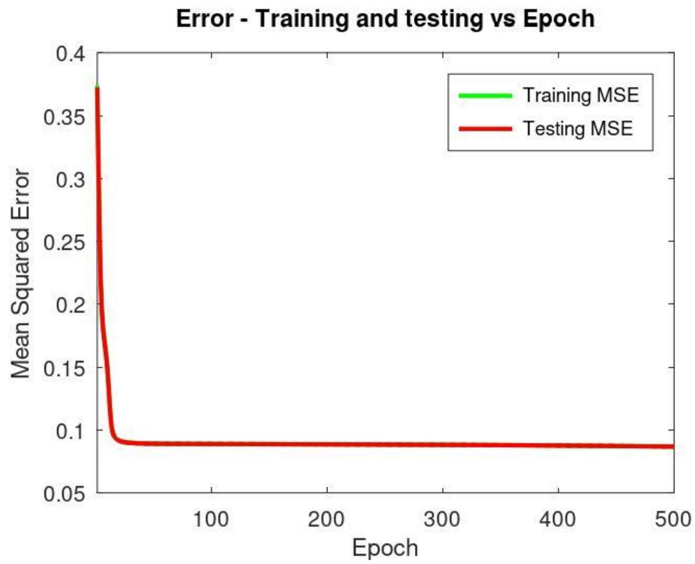


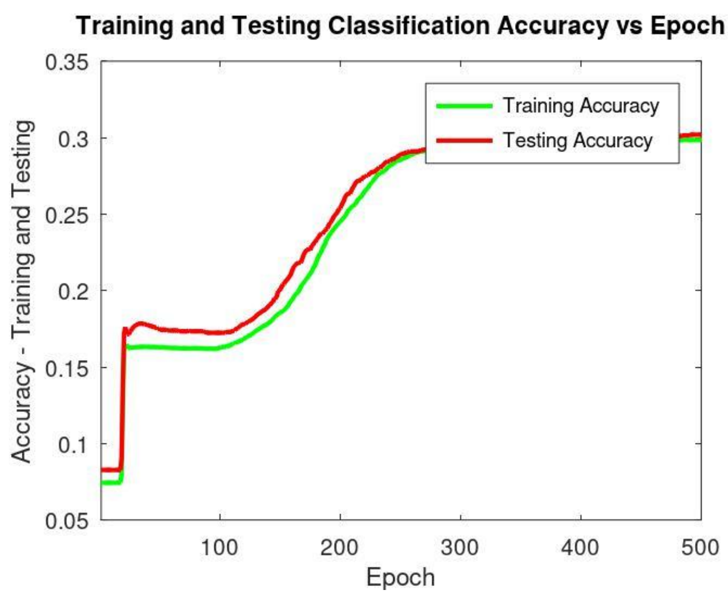


ANN with 2 hidden layers	
Minimum Error	0.0.03162
Minimum Error Epoch	2000
Final Training accuracy	87.64 %
Final Testing accuracy	82.81%
Best Training accuracy	87.64%
Best Testing accuracy	82.81%

c. ANN with 5 hidden layers.

L = [256 38 41 17 89 43 10]

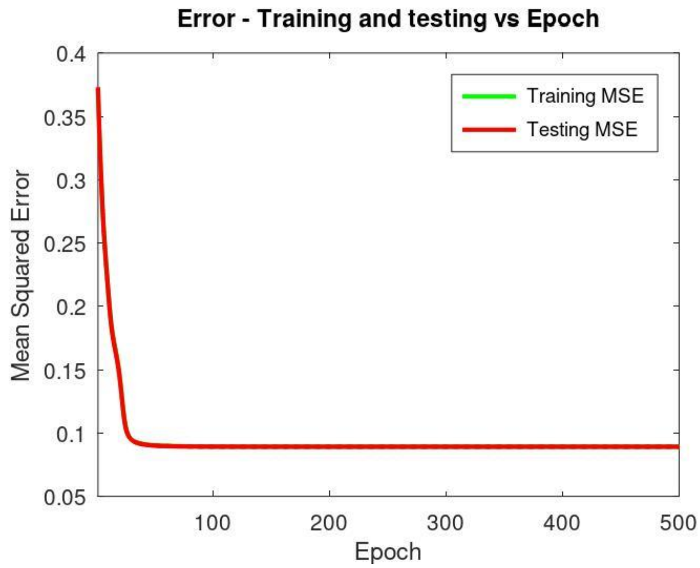


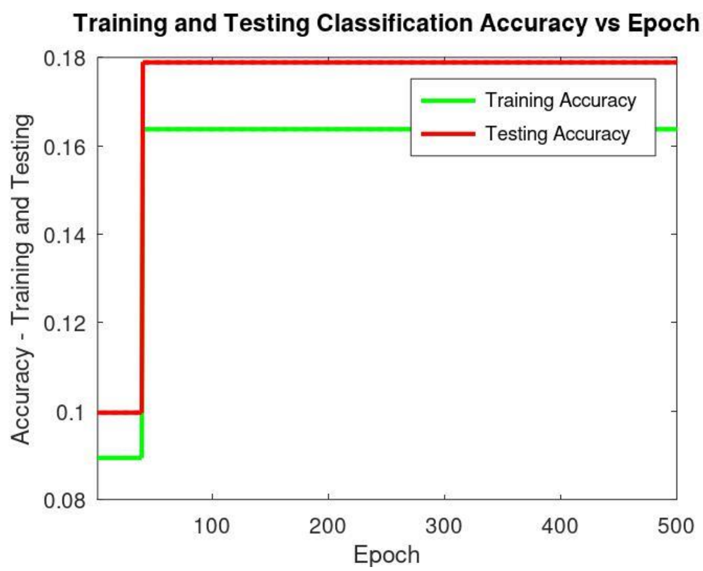


ANN with 5 hidden layers	
Minimum Error	0.067
Minimum Error Epoch	2000
Final Training accuracy	46.96%
Final Testing accuracy	45.39%
Best Training accuracy	46.96%
Best Testing accuracy	45.39%

d. ANN with 10 hidden layers.

L = [256 78 19 9 34 55 67 53 39 61 29 10]





ANN with 10 hidden layers	
Minimum Error	0.0891
Minimum Error Epoch	2000
Final Training accuracy	16.38%
Final Testing accuracy	17.89%
Best Training accuracy	16.38%
Best Testing accuracy	17.89%

Program code:

1. **ANN with 1 hidden layer:**

```
% Initialization
L = [256 49 10]; % Defining the number of layers
alpha = 0.2;
target_mse = 0.0001;
Max_Epoch = 2000;
batch_size = 100;
Min_Error = Inf;
Min_Error_Epoch = -1;
epoch = 0;
mse = Inf;
Err_train = [];
Err_test = [];
Epo = [];
accuracy_train = [];
accuracy_test = [];

% Loading Train datasets
train_data = load('train.txt');
X = train_data(:, 2:end);
[Nx, P] = size(X);
Y = train_data(:, 1);
K = max(Y) + 1;

% One-hot encoding for train labels
Y = eye(K)(Y + 1, :);

% Loading Test datasets
test_data = load('test.txt');
X_test = test_data(:, 2:end);
[Nx_test, P_test] = size(X_test);
Y_test = test_data(:, 1);

% One-hot encoding for test labels
Y_test = eye(K)(Y_test + 1, :);

% Verify the loaded sample size/dimensions
if Nx ~= size(Y, 1)
  error('The input/output sample sizes do not match');
end

if L(1) ~= P
  error('The number of input nodes must be equal to the size of the features');
end

if L(end) ~= K
```

```
    error('The number of output node should be equal to K');
end

% Rest of the code remains unchanged

% Initializing Beta matrices
B = cell(length(L) - 1, 1);
for i = 1:length(L) - 1
    B{i} = [1.4 .* rand(L(i) + 1, L(i + 1)) - 0.7];
end

% Allocating space for Term, T
T = cell(length(L), 1);
for i = 1:length(L)
    T{i} = ones(L(i), 1);
end

% Allocating space for activation, Z

Z = cell(length(L), 1);
for i = 1:length(L) - 1
    Z{i} = zeros(L(i) + 1, 1);  # L(I)+1 because we use one for bias term
end
Z{end} = zeros(L(end), 1); % this is for output layer. We don't use bias in the output layer.

% Allocating space for error term delta, d
d = cell(length(L), 1);
for i = 1:length(L)
    d{i} = zeros(L(i), 1);
end

% Batch version of loading all the testing datasets
% its just for the inputs. Z{1} is done to load the inputs
Z_test = Z;
Z_test{1} = [X_test ones(Nx_test, 1)]'; # concatinating ones for bias term
Y_test = Y_test';

% Batch version of loading all the training datasets
Z{1} = [X ones(Nx, 1)]';
Y = Y';

while ((mse > target_mse) && (epoch < Max_Epoch))
    CSqErr = 0;
    CSqErr_test = 0;

    % Test forward propagation
    for i = 1:length(L) - 1
        T_test{i + 1} = B{i}' * Z_test{i};
```

```
    if (i + 1) < length(L)
        Z_test{i + 1} = [(1 ./ (1 + exp(-T_test{i + 1}))); ones(Nx_test, 1)'];
    else
        Z_test{i + 1} = (1 ./ (1 + exp(-T_test{i + 1})));
    end
end
CSqErr_test = CSqErr_test + sum(sum(((Y_test - Z_test{end}) .^ 2), 1));
CSqErr_test = CSqErr_test / L(end);

% Train forward propagation
for i = 1:length(L) - 1
    T{i + 1} = B{i}' * Z{i};

    if (i + 1) < length(L)
        Z{i + 1} = [(1 ./ (1 + exp(-T{i + 1}))); ones(Nx, 1)'];
    else
        Z{i + 1} = (1 ./ (1 + exp(-T{i + 1})));
    end
end
CSqErr = CSqErr + sum(sum(((Y - Z{end}) .^ 2), 1));
CSqErr = CSqErr / L(end);

% Compute error term delta 'd' for each of the node except the input unit
d{end} = (Z{end} - Y) .* Z{end} .* (1 - Z{end}); % Delta error term for the output layer.

for i = length(L) - 1:-1:2
    W = Z{i}(1:end - 1, :) .* (1 - Z{i}(1:end - 1, :));
    D = d{i + 1}';
    for m = 1:Nx
        d{i}(:, m) = W(:, m) .* sum((D(m, :) .* B{i}(1:end - 1, :)), 2);
    end
end

% // Now we will update the parameters/weights
for i=1:length(L)-1
    W = Z{i}(1:end-1,:);   V1 = zeros(L(i),L(i+1));   V2 = zeros(1,L(i+1));    D = d{i+1}';
      for m = 1:Nx
          V1 = V1 + (W(:,m)*D(m,:));   V2 = V2 + D(m,:);
      end

    B{i}(1:end-1,:)=B{i}(1:end-1,:)-(alpha/Nx).*V1;
    B{i}(end,:) = B{i}(end,:)-(alpha/Nx).*V2;
end

CSqErr = CSqErr / Nx;
mse = CSqErr;
epoch = epoch + 1;

Err_train = [Err_train mse];
```

```
    Epo = [Epo epoch];

    CSqErr_test = CSqErr_test / Nx_test;
    mse_test = CSqErr_test;

    Err_test = [Err_test mse_test];

    % Calculate training accuracy
    [~, pred_train] = max(Z{end}, [], 1);
    [~, true_train] = max(Y, [], 1);
    correct_train = sum(pred_train == true_train);
    acc_train = correct_train / Nx;
    accuracy_train = [accuracy_train acc_train];

    % Calculate testing accuracy
    [~, pred_test] = max(Z_test{end}, [], 1);
    [~, true_test] = max(Y_test, [], 1);
    correct_test = sum(pred_test == true_test);
    acc_test = correct_test / Nx_test;
    accuracy_test = [accuracy_test acc_test];

    % Update the best model based on minimum testing error
    if mse_test < Min_Error
        Min_Error = mse_test;
        Min_Error_Epoch = epoch;

        % Store Best Beta's for minimum test error
        for i = 1:length(L) - 1
            best_beta{i} = B{i};
        end
    end

end % While end




% Plot epoch versus training and testing MSE
figure;
plot(Epo(1:500), Err_train(1:500), 'g', 'LineWidth', 2);
hold on;
plot(Epo(1:500), Err_test(1:500), 'r', 'LineWidth', 2);
hold off;
title('Error - Training and testing vs Epoch');
xlabel('Epoch');
ylabel('Mean Squared Error');
legend('Training MSE', 'Testing MSE');
xlim([1 500]);
```

```
% Plot epoch versus training and test classification accuracy
figure;
plot(Epo(1:500), accuracy_train(1:500), 'g', 'LineWidth', 2);
hold on;
plot(Epo(1:500), accuracy_test(1:500), 'r', 'LineWidth', 2);
hold off;
title('Training and Testing Classification Accuracy vs Epoch');
xlabel('Epoch');
ylabel('Accuracy - Training and Testing');
legend('Training Accuracy', 'Testing Accuracy');
xlim([1 500]);

Min_Error
Min_Error_Epoch
L

% Save the best model to a file
save('best_model.csv', 'best_beta', 'L');

% Displaying the minimum error and its epoch
disp(['Minimum Error: ', num2str(Min_Error)]);
disp(['Minimum Error Epoch: ', num2str(Min_Error_Epoch)]);

% Displaying final training and testing accuracy
disp(['Final Training Accuracy: ', num2str(acc_train * 100), '%']);
disp(['Final Testing Accuracy: ', num2str(acc_test * 100), '%']);

% Displaying best training and testing accuracy
best_train_accuracy = max(accuracy_train);
best_test_accuracy = max(accuracy_test);
disp(['Best Training Accuracy: ', num2str(best_train_accuracy * 100), '%']);
disp(['Best Testing Accuracy: ', num2str(best_test_accuracy * 100), '%']);

pause;
```

2. ANN with 2 hidden layers

```
% Initialization
L = [256 67 34 10]; % Defining the number of layers
alpha = 0.2;
target_mse = 0.0001;
Max_Epoch = 2000;
batch_size = 100;
Min_Error = Inf;
Min_Error_Epoch = -1;
epoch = 0;
mse = Inf;
Err_train = [];
Err_test = [];
Epo = [];
accuracy_train = [];
accuracy_test = [];

% Loading Train datasets
train_data = load('train.txt');
X = train_data(:, 2:end);
[Nx, P] = size(X);
Y = train_data(:, 1);
K = max(Y) + 1;

% One-hot encoding for train labels
Y = eye(K)(Y + 1, :);

% Loading Test datasets
test_data = load('test.txt');
X_test = test_data(:, 2:end);
[Nx_test, P_test] = size(X_test);
Y_test = test_data(:, 1);

% One-hot encoding for test labels
Y_test = eye(K)(Y_test + 1, :);

% Verify the loaded sample size/dimensions
if Nx ~= size(Y, 1)
   error('The input/output sample sizes do not match');
end

if L(1) ~= P
   error('The number of input nodes must be equal to the size of the features');
end

if L(end) ~= K
   error('The number of output node should be equal to K');
end
```

```
% Rest of the code remains unchanged

% Initializing Beta matrices
B = cell(length(L) - 1, 1);
for i = 1:length(L) - 1
    B{i} = [1.4 .* rand(L(i) + 1, L(i + 1)) - 0.7];
end

% Allocating space for Term, T
T = cell(length(L), 1);
for i = 1:length(L)
    T{i} = ones(L(i), 1);
end

% Allocating space for activation, Z

Z = cell(length(L), 1);
for i = 1:length(L) - 1
    Z{i} = zeros(L(i) + 1, 1);  # L(I)+1 because we use one for bias term
end
Z{end} = zeros(L(end), 1); % this is for output layer. We don't use bias in the output layer.

% Allocating space for error term delta, d
d = cell(length(L), 1);
for i = 1:length(L)
    d{i} = zeros(L(i), 1);
end

% Batch version of loading all the testing datasets
% its just for the inputs. Z{1} is done to load the inputs
Z_test = Z;
Z_test{1} = [X_test ones(Nx_test, 1)]'; # concatinating ones for bias term
Y_test = Y_test';

% Batch version of loading all the training datasets
Z{1} = [X ones(Nx, 1)]';
Y = Y';

while ((mse > target_mse) && (epoch < Max_Epoch))
    CSqErr = 0;
    CSqErr_test = 0;

    % Test forward propagation
    for i = 1:length(L) - 1
        T_test{i + 1} = B{i}' * Z_test{i};

        if (i + 1) < length(L)
            Z_test{i + 1} = [(1 ./ (1 + exp(-T_test{i + 1}))); ones(Nx_test, 1)'];
```

```
    else
        Z_test{i + 1} = (1 ./ (1 + exp(-T_test{i + 1})));
    end
end
CSqErr_test = CSqErr_test + sum(sum(((Y_test - Z_test{end}) .^ 2), 1));
CSqErr_test = CSqErr_test / L(end);

% Train forward propagation
for i = 1:length(L) - 1
    T{i + 1} = B{i}' * Z{i};

    if (i + 1) < length(L)
        Z{i + 1} = [(1 ./ (1 + exp(-T{i + 1}))); ones(Nx, 1)'];
    else
        Z{i + 1} = (1 ./ (1 + exp(-T{i + 1})));
    end
end
CSqErr = CSqErr + sum(sum(((Y - Z{end}) .^ 2), 1));
CSqErr = CSqErr / L(end);

% Compute error term delta 'd' for each of the node except the input unit
d{end} = (Z{end} - Y) .* Z{end} .* (1 - Z{end}); % Delta error term for the output layer.

for i = length(L) - 1:-1:2
    W = Z{i}(1:end - 1, :) .* (1 - Z{i}(1:end - 1, :));
    D = d{i + 1}';
    for m = 1:Nx
        d{i}(:, m) = W(:, m) .* sum((D(m, :) .* B{i}(1:end - 1, :)), 2);
    end
end

% // Now we will update the parameters/weights
for i=1:length(L)-1
    W = Z{i}(1:end-1,:);   V1 = zeros(L(i),L(i+1));   V2 = zeros(1,L(i+1));    D = d{i+1}';
      for m = 1:Nx
          V1 = V1 + (W(:,m)*D(m,:));   V2 = V2 + D(m,:);
      end

    B{i}(1:end-1,:)=B{i}(1:end-1,:)-(alpha/Nx).*V1;
    B{i}(end,:) = B{i}(end,:)-(alpha/Nx).*V2;
end

CSqErr = CSqErr / Nx;
mse = CSqErr;
epoch = epoch + 1;

Err_train = [Err_train mse];
Epo = [Epo epoch];
```

```
    CSqErr_test = CSqErr_test / Nx_test;
    mse_test = CSqErr_test;

    Err_test = [Err_test mse_test];

    % Calculate training accuracy
    [~, pred_train] = max(Z{end}, [], 1);
    [~, true_train] = max(Y, [], 1);
    correct_train = sum(pred_train == true_train);
    acc_train = correct_train / Nx;
    accuracy_train = [accuracy_train acc_train];

    % Calculate testing accuracy
    [~, pred_test] = max(Z_test{end}, [], 1);
    [~, true_test] = max(Y_test, [], 1);
    correct_test = sum(pred_test == true_test);
    acc_test = correct_test / Nx_test;
    accuracy_test = [accuracy_test acc_test];

    % Update the best model based on minimum testing error
    if mse_test < Min_Error
        Min_Error = mse_test;
        Min_Error_Epoch = epoch;

        % Store Best Beta's for minimum test error
        for i = 1:length(L) - 1
            best_beta{i} = B{i};
        end
    end

end % While end



% Plot epoch versus training and testing MSE
figure;
plot(Epo(1:500), Err_train(1:500), 'g', 'LineWidth', 2);
hold on;
plot(Epo(1:500), Err_test(1:500), 'r', 'LineWidth', 2);
hold off;
title('Error - Training and testing vs Epoch');
xlabel('Epoch');
ylabel('Mean Squared Error');
legend('Training MSE', 'Testing MSE');
xlim([1 500]);

% Plot epoch versus training and test classification accuracy
figure;
```

```
plot(Epo(1:500), accuracy_train(1:500), 'g', 'LineWidth', 2);
hold on;
plot(Epo(1:500), accuracy_test(1:500), 'r', 'LineWidth', 2);
hold off;
title('Training and Testing Classification Accuracy vs Epoch');
xlabel('Epoch');
ylabel('Accuracy - Training and Testing');
legend('Training Accuracy', 'Testing Accuracy');
xlim([1 500]);

Min_Error
Min_Error_Epoch
L

% Save the best model to a file
save('best_model.csv', 'best_beta', 'L');

% Displaying the minimum error and its epoch
disp(['Minimum Error: ', num2str(Min_Error)]);
disp(['Minimum Error Epoch: ', num2str(Min_Error_Epoch)]);

% Displaying final training and testing accuracy
disp(['Final Training Accuracy: ', num2str(acc_train * 100), '%']);
disp(['Final Testing Accuracy: ', num2str(acc_test * 100), '%']);

% Displaying best training and testing accuracy
best_train_accuracy = max(accuracy_train);
best_test_accuracy = max(accuracy_test);
disp(['Best Training Accuracy: ', num2str(best_train_accuracy * 100), '%']);
disp(['Best Testing Accuracy: ', num2str(best_test_accuracy * 100), '%']);

pause;
```

3. ANN with 5 hidden layers

```
% Initialization
L = [256 38 41 17 89 43 10]; % Defining the number of layers
alpha = 0.2;
target_mse = 0.0001;
Max_Epoch = 2000;
batch_size = 100;
Min_Error = Inf;
Min_Error_Epoch = -1;
epoch = 0;
mse = Inf;
Err_train = [];
Err_test = [];
Epo = [];
accuracy_train = [];
accuracy_test = [];

% Loading Train datasets
train_data = load('train.txt');
X = train_data(:, 2:end);
[Nx, P] = size(X);
Y = train_data(:, 1);
K = max(Y) + 1;

% One-hot encoding for train labels
Y = eye(K)(Y + 1, :);

% Loading Test datasets
test_data = load('test.txt');
X_test = test_data(:, 2:end);
[Nx_test, P_test] = size(X_test);
Y_test = test_data(:, 1);

% One-hot encoding for test labels
Y_test = eye(K)(Y_test + 1, :);

% Verify the loaded sample size/dimensions
if Nx ~= size(Y, 1)
   error('The input/output sample sizes do not match');
end

if L(1) ~= P
   error('The number of input nodes must be equal to the size of the features');
end
```

```
if L(end) ~= K
   error('The number of output node should be equal to K');
end

% Rest of the code remains unchanged

% Initializing Beta matrices
B = cell(length(L) - 1, 1);
for i = 1:length(L) - 1
   B{i} = [1.4 .* rand(L(i) + 1, L(i + 1)) - 0.7];
end

% Allocating space for Term, T
T = cell(length(L), 1);
for i = 1:length(L)
   T{i} = ones(L(i), 1);
end

% Allocating space for activation, Z

Z = cell(length(L), 1);
for i = 1:length(L) - 1
   Z{i} = zeros(L(i) + 1, 1);  # L(I)+1 because we use one for bias term
end
Z{end} = zeros(L(end), 1); % this is for output layer. We don't use bias in the output layer.

% Allocating space for error term delta, d
d = cell(length(L), 1);
for i = 1:length(L)
   d{i} = zeros(L(i), 1);
end

% Batch version of loading all the testing datasets
% its just for the inputs. Z{1} is done to load the inputs
Z_test = Z;
Z_test{1} = [X_test ones(Nx_test, 1)]'; # concatinating ones for bias term
Y_test = Y_test';

% Batch version of loading all the training datasets
Z{1} = [X ones(Nx, 1)]';
Y = Y';

while ((mse > target_mse) && (epoch < Max_Epoch))
```

```
CSqErr = 0;
CSqErr_test = 0;

% Test forward propagation
for i = 1:length(L) - 1
   T_test{i + 1} = B{i}' * Z_test{i};

   if (i + 1) < length(L)
      Z_test{i + 1} = [(1 ./ (1 + exp(-T_test{i + 1}))); ones(Nx_test, 1)'];
   else
      Z_test{i + 1} = (1 ./ (1 + exp(-T_test{i + 1})));
   end
end
CSqErr_test = CSqErr_test + sum(sum(((Y_test - Z_test{end}) .^ 2), 1));
CSqErr_test = CSqErr_test / L(end);

% Train forward propagation
for i = 1:length(L) - 1
   T{i + 1} = B{i}' * Z{i};

   if (i + 1) < length(L)
      Z{i + 1} = [(1 ./ (1 + exp(-T{i + 1}))); ones(Nx, 1)'];
   else
      Z{i + 1} = (1 ./ (1 + exp(-T{i + 1})));
   end
end
CSqErr = CSqErr + sum(sum(((Y - Z{end}) .^ 2), 1));
CSqErr = CSqErr / L(end);

% Compute error term delta 'd' for each of the node except the input unit
d{end} = (Z{end} - Y) .* Z{end} .* (1 - Z{end}); % Delta error term for the output layer.

for i = length(L) - 1:-1:2
   W = Z{i}(1:end - 1, :) .* (1 - Z{i}(1:end - 1, :));
   D = d{i + 1}';
   for m = 1:Nx
      d{i}(:, m) = W(:, m) .* sum((D(m, :) .* B{i}(1:end - 1, :)), 2);
   end
end

% // Now we will update the parameters/weights
for i=1:length(L)-1
   W = Z{i}(1:end-1,:);   V1 = zeros(L(i),L(i+1));   V2 = zeros(1,L(i+1));    D = d{i+1}';
     for m = 1:Nx
```

```
            V1 = V1 + (W(:,m)*D(m,:));   V2 = V2 + D(m,:);
        end

    B{i}(1:end-1,:)=B{i}(1:end-1,:)-(alpha/Nx).*V1;
    B{i}(end,:) = B{i}(end,:)-(alpha/Nx).*V2;
end

CSqErr = CSqErr / Nx;
mse = CSqErr;
epoch = epoch + 1;

Err_train = [Err_train mse];
Epo = [Epo epoch];

CSqErr_test = CSqErr_test / Nx_test;
mse_test = CSqErr_test;

Err_test = [Err_test mse_test];

% Calculate training accuracy
[~, pred_train] = max(Z{end}, [], 1);
[~, true_train] = max(Y, [], 1);
correct_train = sum(pred_train == true_train);
acc_train = correct_train / Nx;
accuracy_train = [accuracy_train acc_train];

% Calculate testing accuracy
[~, pred_test] = max(Z_test{end}, [], 1);
[~, true_test] = max(Y_test, [], 1);
correct_test = sum(pred_test == true_test);
acc_test = correct_test / Nx_test;
accuracy_test = [accuracy_test acc_test];

% Update the best model based on minimum testing error
if mse_test < Min_Error
    Min_Error = mse_test;
    Min_Error_Epoch = epoch;

    % Store Best Beta's for minimum test error
    for i = 1:length(L) - 1
        best_beta{i} = B{i};
    end
end
```

```
end % While end



% Plot epoch versus training and testing MSE
figure;
plot(Epo(1:500), Err_train(1:500), 'g', 'LineWidth', 2);
hold on;
plot(Epo(1:500), Err_test(1:500), 'r', 'LineWidth', 2);
hold off;
title('Error - Training and testing vs Epoch');
xlabel('Epoch');
ylabel('Mean Squared Error');
legend('Training MSE', 'Testing MSE');
xlim([1 500]);

% Plot epoch versus training and test classification accuracy
figure;
plot(Epo(1:500), accuracy_train(1:500), 'g', 'LineWidth', 2);
hold on;
plot(Epo(1:500), accuracy_test(1:500), 'r', 'LineWidth', 2);
hold off;
title('Training and Testing Classification Accuracy vs Epoch');
xlabel('Epoch');
ylabel('Accuracy - Training and Testing');
legend('Training Accuracy', 'Testing Accuracy');
xlim([1 500]);

Min_Error
Min_Error_Epoch
L

% Save the best model to a file
save('best_model.csv', 'best_beta', 'L');

% Displaying the minimum error and its epoch
disp(['Minimum Error: ', num2str(Min_Error)]);
disp(['Minimum Error Epoch: ', num2str(Min_Error_Epoch)]);

% Displaying final training and testing accuracy
disp(['Final Training Accuracy: ', num2str(acc_train * 100), '%']);
disp(['Final Testing Accuracy: ', num2str(acc_test * 100), '%']);
```

```
% Displaying best training and testing accuracy
best_train_accuracy = max(accuracy_train);
best_test_accuracy = max(accuracy_test);
disp(['Best Training Accuracy: ', num2str(best_train_accuracy * 100), '%']);
disp(['Best Testing Accuracy: ', num2str(best_test_accuracy * 100), '%']);

pause;
```

4. ANN with 10 hidden layers:

```
L = [256 78 19 9 34 55 67 53 39 61 29 10]; % Defining the number of layers
alpha = 0.2;
target_mse = 0.0001;
Max_Epoch = 2000;
batch_size = 100;
Min_Error = Inf;
Min_Error_Epoch = -1;
epoch = 0;
mse = Inf;
Err_train = [];
Err_test = [];
Epo = [];
accuracy_train = [];
accuracy_test = [];

% Loading Train datasets
train_data = load('train.txt');
X = train_data(:, 2:end);
[Nx, P] = size(X);
Y = train_data(:, 1);
K = max(Y) + 1;

% One-hot encoding for train labels
Y = eye(K)(Y + 1, :);

% Loading Test datasets
test_data = load('test.txt');
X_test = test_data(:, 2:end);
[Nx_test, P_test] = size(X_test);
Y_test = test_data(:, 1);

% One-hot encoding for test labels
Y_test = eye(K)(Y_test + 1, :);

% Verify the loaded sample size/dimensions
if Nx ~= size(Y, 1)
   error('The input/output sample sizes do not match');
end

if L(1) ~= P
   error('The number of input nodes must be equal to the size of the features');
end
```

```
if L(end) ~= K
    error('The number of output node should be equal to K');
end

% Rest of the code remains unchanged

% Initializing Beta matrices
B = cell(length(L) - 1, 1);
for i = 1:length(L) - 1
    B{i} = [1.4 .* rand(L(i) + 1, L(i + 1)) - 0.7];
end

% Allocating space for Term, T
T = cell(length(L), 1);
for i = 1:length(L)
    T{i} = ones(L(i), 1);
end

% Allocating space for activation, Z

Z = cell(length(L), 1);
for i = 1:length(L) - 1
    Z{i} = zeros(L(i) + 1, 1);  # L(I)+1 because we use one for bias term
end
Z{end} = zeros(L(end), 1); % this is for output layer. We don't use bias in the output layer.

% Allocating space for error term delta, d
d = cell(length(L), 1);
for i = 1:length(L)
    d{i} = zeros(L(i), 1);
end

% Batch version of loading all the testing datasets
% its just for the inputs. Z{1} is done to load the inputs
Z_test = Z;
Z_test{1} = [X_test ones(Nx_test, 1)]'; # concatinating ones for bias term
Y_test = Y_test';

% Batch version of loading all the training datasets
Z{1} = [X ones(Nx, 1)]';
Y = Y';

while ((mse > target_mse) && (epoch < Max_Epoch))
```

```
CSqErr = 0;
CSqErr_test = 0;

% Test forward propagation
for i = 1:length(L) - 1
   T_test{i + 1} = B{i}' * Z_test{i};

   if (i + 1) < length(L)
      Z_test{i + 1} = [(1 ./ (1 + exp(-T_test{i + 1}))); ones(Nx_test, 1)'];
   else
      Z_test{i + 1} = (1 ./ (1 + exp(-T_test{i + 1})));
   end
end
CSqErr_test = CSqErr_test + sum(sum(((Y_test - Z_test{end}) .^ 2), 1));
CSqErr_test = CSqErr_test / L(end);

% Train forward propagation
for i = 1:length(L) - 1
   T{i + 1} = B{i}' * Z{i};

   if (i + 1) < length(L)
      Z{i + 1} = [(1 ./ (1 + exp(-T{i + 1}))); ones(Nx, 1)'];
   else
      Z{i + 1} = (1 ./ (1 + exp(-T{i + 1})));
   end
end
CSqErr = CSqErr + sum(sum(((Y - Z{end}) .^ 2), 1));
CSqErr = CSqErr / L(end);

% Compute error term delta 'd' for each of the node except the input unit
d{end} = (Z{end} - Y) .* Z{end} .* (1 - Z{end}); % Delta error term for the output layer.

for i = length(L) - 1:-1:2
   W = Z{i}(1:end - 1, :) .* (1 - Z{i}(1:end - 1, :));
   D = d{i + 1}';
   for m = 1:Nx
      d{i}(:, m) = W(:, m) .* sum((D(m, :) .* B{i}(1:end - 1, :)), 2);
   end
end

% // Now we will update the parameters/weights
for i=1:length(L)-1
   W = Z{i}(1:end-1,:);   V1 = zeros(L(i),L(i+1));   V2 = zeros(1,L(i+1));    D = d{i+1}';
     for m = 1:Nx
```

```
        V1 = V1 + (W(:,m)*D(m,:));   V2 = V2 + D(m,:);
      end

    B{i}(1:end-1,:)=B{i}(1:end-1,:)-(alpha/Nx).*V1;
    B{i}(end,:) = B{i}(end,:)-(alpha/Nx).*V2;
end

CSqErr = CSqErr / Nx;
mse = CSqErr;
epoch = epoch + 1;

Err_train = [Err_train mse];
Epo = [Epo epoch];

CSqErr_test = CSqErr_test / Nx_test;
mse_test = CSqErr_test;

Err_test = [Err_test mse_test];

% Calculate training accuracy
[~, pred_train] = max(Z{end}, [], 1);
[~, true_train] = max(Y, [], 1);
correct_train = sum(pred_train == true_train);
acc_train = correct_train / Nx;
accuracy_train = [accuracy_train acc_train];

% Calculate testing accuracy
[~, pred_test] = max(Z_test{end}, [], 1);
[~, true_test] = max(Y_test, [], 1);
correct_test = sum(pred_test == true_test);
acc_test = correct_test / Nx_test;
accuracy_test = [accuracy_test acc_test];

% Update the best model based on minimum testing error
if mse_test < Min_Error
    Min_Error = mse_test;
    Min_Error_Epoch = epoch;

    % Store Best Beta's for minimum test error
    for i = 1:length(L) - 1
        best_beta{i} = B{i};
    end
end
```

```
end % While end



% Plot epoch versus training and testing MSE
figure;
plot(Epo(1:500), Err_train(1:500), 'g', 'LineWidth', 2);
hold on;
plot(Epo(1:500), Err_test(1:500), 'r', 'LineWidth', 2);
hold off;
title('Error - Training and testing vs Epoch');
xlabel('Epoch');
ylabel('Mean Squared Error');
legend('Training MSE', 'Testing MSE');
xlim([1 500]);

% Plot epoch versus training and test classification accuracy
figure;
plot(Epo(1:500), accuracy_train(1:500), 'g', 'LineWidth', 2);
hold on;
plot(Epo(1:500), accuracy_test(1:500), 'r', 'LineWidth', 2);
hold off;
title('Training and Testing Classification Accuracy vs Epoch');
xlabel('Epoch');
ylabel('Accuracy - Training and Testing');
legend('Training Accuracy', 'Testing Accuracy');
xlim([1 500]);

Min_Error
Min_Error_Epoch
L

% Save the best model to a file
save('best_model.csv', 'best_beta', 'L');

% Displaying the minimum error and its epoch
disp(['Minimum Error: ', num2str(Min_Error)]);
disp(['Minimum Error Epoch: ', num2str(Min_Error_Epoch)]);

% Displaying final training and testing accuracy
disp(['Final Training Accuracy: ', num2str(acc_train * 100), '%']);
disp(['Final Testing Accuracy: ', num2str(acc_test * 100), '%']);
```

```
% Displaying best training and testing accuracy
best_train_accuracy = max(accuracy_train);
best_test_accuracy = max(accuracy_test);
disp(['Best Training Accuracy: ', num2str(best_train_accuracy * 100), '%']);
disp(['Best Testing Accuracy: ', num2str(best_test_accuracy * 100), '%']);

pause;
```