Code for One-layer perceptron task. Note! Two function files is below the main code

```
% Written by Axel Q
clear all
close all
clc

% Loading the data sets
trainingDataSet = load('training_set.csv');
validationDataSet = load('validation_set.csv');

% Training patterns and targets
inputPatterns = trainingDataSet(:,[1, 2]);
targets = trainingDataSet(:,3);

% Validation patterns and targets
validationPatterns = validationDataSet(:,[1 2]);
validationTargets = validationDataSet(:,3);

% Standardization of the data
meanXTrain = mean(inputPatterns);
stdXTrain = std(inputPatterns);
inputPatterns = inputPatterns - meanXTrain; % scaling
inputPatterns = inputPatterns./stdXTrain;
validationPatterns = validationPatterns - meanXTrain;
validationPatterns = validationPatterns./stdXTrain;



M1 = 10;                                % Number of neurons in hidden layer
pVal = length(validationDataSet);       % Number of patterns in validation set
nPatterns = length(inputPatterns);      % Number of patterns in training set
eta = 0.01;                             % Learning rate

% initial weights and threshold
weights1 = normrnd(0,1, [M1, 2]);
weights2 = normrnd(0,1,[M1, 1]);
threshold1 = zeros(M1,1);
threshold2 = 0;
nOfepoch = 0;
classificationError = 10;              % Assign a value higher than the tolerance 0.12

% Run the algorithm until the classification error is below 12%
while classificationError > 0.12

    for i = 1:nPatterns

        % choose random input
    patternIndex = randi(nPatterns);
    inputNeurons = inputPatterns(patternIndex,:)';
    target = targets(patternIndex,:);

    % Calclulating the states of the hidden layer neurons
    b1 = LocalField(weights1, inputNeurons, threshold1);
    hiddenLayerNeurons = tanh(b1);

    % Calculating the states of the output neuron
    b2 = LocalField(weights2', hiddenLayerNeurons, threshold2);
    outputLayerNeuron = tanh(b2);

    % Calculating the output error
    delta2 = activationPrime(b2).*(target - outputLayerNeuron);

    % Error backpropagation
    delta1 = delta2 .* weights2 .* activationPrime(b1);

    % Update weights and thresholds
    deltaWeights2 = eta * delta2 * hiddenLayerNeurons;
    deltaWeights1 = eta * delta1 * inputNeurons';
    deltaThreshold2 = -eta .* delta2;
    deltaThreshold1 = -eta .* delta1;

    weights2 = weights2 + deltaWeights2;
    weights1 = weights1 + deltaWeights1;
    threshold2 = threshold2 + deltaThreshold2;
    threshold1 = threshold1 + deltaThreshold1;
end

% Check classification error
classificationErrorSum = 0;
for j = 1:pVal

    % Calulating the hidden layer neurons for the validation data
    b1Val = LocalField(weights1,validationPatterns(j,:)',threshold1);
    hiddenLayerNeuronsVal = tanh(b1Val);

    % Calculating the output neuron for the validation data
    b2Val = LocalField(weights2',hiddenLayerNeuronsVal,threshold2);
    outputLayerNeuronVal = tanh(b2Val);

    % Calculating the sum in the classification error formula
    classificationErrorSum = classificationErrorSum +...
        abs(sign(outputLayerNeuronVal) - validationTargets(j));
end

% Calculation the classification error and print it for each epoch
```

```
    classificationError = 1/(2*pVal) * classificationErrorSum;
    fprintf('Epoch: %0.f The classification error is: %f\n',nOfepoch,classificationError)
    nOfepoch = nOfepoch + 1;


end

% Writing to csv
csvwrite('w1.csv',weights1);
csvwrite('w2.csv',weights2);
csvwrite('t1.csv',threshold1);
csvwrite('t2.csv',threshold2);
```

Function file for local field:

```
function output = LocalField(weightVector, inputNeurons, threshold)

   output = weightVector * inputNeurons - threshold;
end
```

Function file for the derivative of the activation function:

```
function output = activationPrime(localField)

    output = 1 - tanh(localField).^2;

end
```