Lukas Fu Homework 2
One Layer Perceptron

One-Layer Perceptron

Main

```
clc
%input
[inputTrain,targetTrain,inputValid,targetValid] = DataLoadAndNorm('
    training_set.csv','validation_set.csv');
%initialize constants
patterns = size(inputTrain,2);
M = 15; %hidden layer size
eta = 0.005;
%initialize weights and thresholds
[w1Mat,w2Mat,th1,th2] = InitializeNetwork(2,M,1);

%while error C is below 0.12
C = 1;
myTrys = zeros(1000,2);
iteration = 0;
while C > 0.12
    iteration = iteration + 1;
    for i = 1:patterns
        iChosen = fix(rand*patterns) + 1;
        target = targetTrain(iChosen);
        X = inputTrain(:,iChosen);
        V = GetNeuronValue(X,w1Mat,th1);
        O = GetNeuronValue(V,w2Mat,th2);
        [w1Mat,w2Mat,th1,th2] = UpdateWeightsAndTheta(eta,w1Mat,w2Mat,th1,th2,
            X,V,O,target);
    end
    C = ClassificationError(w1Mat,w2Mat,th1,th2,inputValid,targetValid);
    myTrys(iteration,1) = iteration;
    myTrys(iteration,2) = C;
    if iteration == 1000
        break
    end
end
C = ClassificationError(w1Mat,w2Mat,th1,th2,inputValid,targetValid,'plot')

%write csv files for w1, w2, t1, t2
% writematrix(w1Mat,'w1.csv')
% writematrix(w2Mat,'w2.csv')
% writematrix(th1,'t1.csv')
```

```
% writematrix(th2,'t2.csv')
```

Load and normalize data

```
function [inputTraining,targetTraining,inputValidation,targetValidation] =
    DataLoadAndNorm(stringTrain,stringValid)
    %import
    dataTrain = load(stringTrain);
    dataValid = load(stringValid);
    %split
    x1Train = dataTrain(:,1);
    x1Valid = dataValid(:,1);
    x2Train = dataTrain(:,2);
    x2Valid = dataValid(:,2);
    %mean
    mean1 = mean(x1Train);
    mean2 = mean(x2Train);
    %std
    std1 = std(x1Train);
    std2 = std(x1Train);
    %normalize
    x1Train = (x1Train - mean1)/std1;
    x1Valid = (x1Valid - mean1)/std1;
    x2Train = (x2Train - mean2)/std2;
    x2Valid = (x2Valid - mean2)/std2;
    %(input1,input2,target)
    inputTraining = transpose([x1Train,x2Train]);
    targetTraining = dataTrain(:,3);

    inputValidation = transpose([x1Valid,x2Valid]);
    targetValidation = dataValid(:,3);
end
```

Initialize Network

```
function [W1,W2,TH1,TH2] = InitializeNetwork(inputSize,hiddenLayerSize,
    outputSize)
    variance1 = 1/sqrt(inputSize);
    variance2 =1/sqrt(hiddenLayerSize);
    W1 = normrnd(0,variance1,hiddenLayerSize,inputSize);
    W2 = normrnd(0,variance2,outputSize,hiddenLayerSize);
    TH1 = zeros(hiddenLayerSize,1);
    TH2 = zeros(outputSize,1);
end
```

Classification Error

```
function C = ClassificationError(wMat1,wMat2,theta1,theta2,input,target,plot)
    patterns = size(target,1);
    C = 0;
    outputVec = zeros(patterns,1);
    for i = 1:patterns
        inputP = input(:,i);
        V = GetNeuronValue(inputP,wMat1,theta1);
        O = GetNeuronValue(V,wMat2,theta2);
```

```
        outputVec(i) = O;
        C = C + abs(sign(O)-target(i));
    end
    if nargin == 7
        PlotTraining(input,outputVec,target)
    end
    C = 1/(2*patterns) * C;
end
```

Get Neuron Values

```
function neuronValue = GetNeuronValue(input, weights, theta)
    neuronValue = tanh(weights*input - theta);
end
```

Update Weights and Theta

```
function [W1,W2,TH1,TH2] = UpdateWeightsAndTheta(eta,weight1,weight2,theta1,
    theta2,I,V,O,T)
    %error2 = (T - O) * g'(B), g'(B) = 1 - tanh(B)^2, tanh(B) = O
    %tanh(b) = V
    error2 = (T-O).*(1-O.^2);
    error1 = error2'*weight2(:,1)*(1-V.^2);

    W2 = weight2 + eta*error2*transpose(V);
    TH2 = theta2 - eta*error2;

    W1 = weight1 + eta*error1*transpose(I);
    TH1 = theta1 - eta*error1;

end
```

Plot Training

```
function PlotTraining(input,output,target)
    pointSize = 10;
    subplot(1,2,1)
    scatter(input(1,:),input(2,:),pointSize, output);
    subplot(1,2,2)
    scatter(input(1,:),input(2,:),pointSize, target);
    sgtitle('Output vs target')
end
```