EEL5840 Fundamental Machine Learning Project 2 Code

Hudanyun Sheng

MATLAB code used in this report

The **main script**

```
dbstop if error

clc

clear

close all

% load data

trainImg = loadMNISTImages('train-images.idx3-ubyte'); % 60000*784

trainLbl = loadMNISTLabels('train-labels.idx1-ubyte'); % 60000*1

testImg = loadMNISTImages('t10k-images.idx3-ubyte');

testLbl = loadMNISTLabels('t10k-labels.idx1-ubyte');


% original size

% trImg = trainImg;

% teImg = testImg;


% pca

d = 100; % dimension being kept

[ temp, w ] = PCA( trainImg', d);

trImg = temp';

teImg = w'*testImg;


trI = trImg(:,1:50000);

trL = trainLbl(1:50000,:);

vI = trImg(:,50001:60000);
```

```
vL = trainLbl(50001:60000,:);


%% train
epoch = 50;
eta = 0.01;
alpha = 0.5;
N_list = [50 100 200 300 400 500 600 700 800 900 1000 1500 2000];
for i = 1:length(N_list)
N1 = N_list(i);
for exp = 1:10
tic
[W1, B1, W2, B2, cost_train(i,exp), accuracy_train(i,exp)] =...
 mlpTrain11( trI, trL, N1, eta,epoch,alpha );
toc
y_vali = mlpTest1(vI, W1, W2, B1, B2);
errorv = 0;
viLabel = convertLabel( vL );
for m = 1:size(vI,2)
        errorv = errorv + (norm(viLabel(:,m)-y_vali(m,:)',2))^2;
end
cost_vali(i,exp) = errorv/(2*size(vI,2));
predicted_vali = classAssignment(y_vali);
CONFU_vali = confusionmat(vL,predicted_vali );
accuracy_vali(i,exp) = sum(diag(CONFU_vali))/sum(sum(CONFU_vali));
end
end


mean_cTrain = mean(cost_train,2);
std_cTrain = std(cost_train,0,2);
```

```
mean_cVali = mean(cost_vali,2);
std_cVali = std(cost_vali,0,2);

mean_aTrain = mean(accuracy_train,2);
std_aTrain = std(accuracy_train,0,2);
mean_aVali = mean(accuracy_vali,2);
std_aVali = std(accuracy_vali,0,2);

figure, subplot(1,2,1), errorbar(N_list, mean_cTrain, std_cTrain, 'Linewidth',2), hold on,...
 errorbar(N_list, mean_cVali, std_cVali, 'Linewidth',2)
legend('train', 'validation'), xlabel('number of hidden units'),...
 ylabel('cost function value'), title('Error bar for cost function')
subplot(1,2,2), errorbar(N_list, mean_aTrain, std_aTrain, 'Linewidth',2), hold on,...
 errorbar(N_list, mean_aVali, std_aVali, 'Linewidth',2)
legend('train', 'validation'), xlabel('number of hidden units'), ylabel('accuracy'),...
 title('Error bar for accuracy')

function [ projected_data, w ] = PCA( X, d)
%X: data
%d: the desired dimension of the output data
%   Detailed explanation goes here
mu = mean(X);
X_std = X - mu;
cov_mat = cov(X_std);
[eigenVecs, eigenVals] = eig(cov_mat);
w = eigenVecs(:,(end-d+1):end);
projected_data = X * w;
end
```

The **multilayer perceptron training function**

```
function [W1, B1, W2, B2,  accuracy_train] = mlpTrain1( trainData, trainLabel,N1, eta,...
 epoch, alpha )
%mlp train when there is 1 hidden layer
%   Input: train Data - sample in column data (D * trainSize)
%          train Label
%          N1 - number of units in 1st hidden layer
%          eta - learning rate

trLabel = convertLabel( trainLabel );
viLabel = convertLabel( valiLabel );
trainSize = size(trainData,2);
inputD = size(trainData,1);
outputD = size(trLabel, 1);

% initialize
W1 = rand(inputD, N1)./inputD;
B1 = rand(1, N1)./N1;
W2  = rand(N1,outputD)./N1;
B2 = rand(1,outputD)./outputD;

%% online training
update2 = 0;
update1 = 0;
updateB2 = 0;
updateB1 = 0;

for numEpoch = 1:epoch
    colrank = randperm(trainSize);
```

```
    trainData = trainData(:,colrank);
    trLabel = trLabel(:, colrank);
        trainLabel = trainLabel(colrank,:);
    correct = 0;
    for iter = 1:trainSize
        inputVec = trainData(:,iter);
        %% Forward pass
        % 1st hidden layer
        [Y1, dY1] = ReLU(inputVec'*W1+B1);
        % output layer
        [Y2, dY2] = ReLU(Y1*W2 +B2);
        %% Backward pass
        % output layer
        error(iter) = norm(trLabel(:,iter)'-Y2,2); %record every error
        lb_train = classAssignment(Y2);
        if lb_train == trainLabel(iter)
            correct = correct+1;
        end
        delta2 = (trLabel(:,iter)'-Y2).*dY2;
        delta1 = dY1.*(delta2*W2');

        % SGD
%         W2 = W2 + eta*Y1'*delta2;
%         W1 = W1 + eta*inputVec*delta1;
%         B2 = B2 + eta*delta2;
%         B1 = B1 + eta*delta1;

        % SGD with momentum
        update2 = alpha*eta*update2 + eta*Y1'*delta2;
```

```
        update1 = alpha*eta*update1 + eta*inputVec*delta1;
        updateB2 = alpha*eta*updateB2 + eta*delta2;
        updateB1 = alpha*eta*updateB1 + eta*delta1;
        W2 = W2 + update2;
        W1 = W1 + update1;
        B2 = B2 + updateB2;
        B1 = B1 + updateB1;
    end
    outputweight(:,numEpoch) = reshape(W2, [N1*outputD, 1]);
    outputbias(:,numEpoch) = reshape(B2, [outputD, 1]);
    cost_train(1,numEpoch) = mean(error.^2)/2;%MSE for train
    accuracy_train(1,numEpoch) = correct/trainSize;
    y_vali = mlpTest1(valiData, W1, W2, B1, B2);
    for i = 1:size(valiData,2)
%         errorv = errorv + (norm(viLabel(:,i)-y_vali(i,:)',2))^2;
        errorv(i) = norm(viLabel(:,i)-y_vali(i,:)',2);
    end
    cost_vali(1,numEpoch) = mean(errorv.^2)/2;%MSE for validate
    lb_vali = classAssignment(y_vali);
    accuracy_vali(1,numEpoch) = length(find((lb_vali-valiLabel')==0))/size(valiData,2);

end
end

function outputLabel = convertLabel( inputLabel )
% function to convert label as a number to a label vector
for i = 1:size(inputLabel,1)
    outputLabel(inputLabel(i,1)+1, i) = 1;
end
```

```
end


function label = classAssignment( Y )
% function to assign an output vector to a corresponding class (0~9)
for n = 1:size(Y,1)
    max = 0;
    class = 1;
    y = Y(n,:);
    for i = 1: size(y, 2)
        if y(i) > max
            max = y(i);
            class = i-1;
        end;
    end;
    label(n) = class;
end
end


function [Y] = mlpTest1( testData, W1, W2, B1, B2  )
%Given the weights and biases, calculation the output, with only 1 hidden layer
%   Input: testData - sample in the columns
%   W1/B1: weight/bias from input layer to 1st hidden layer
%   W2/B2: weight/bias from 1st hidden layer to output layer
N1 = size(W1, 2);
testSize = size(testData,2);
for n = 1:testSize
        % 1st hidden layer
    inputVec = testData(:,n);
    net1 = inputVec'*W1+B1;
```

```
        [Y1, ~] = ReLU( net1 );

        % output layer
        net2 = Y1*W2+B2;
        [Y(n,:), ~] = ReLU( net2 );
end
end

function [fx, dfx] = ReLU( x )
%ReLU fuction
%   Detailed explanation goes here
% fx = x.*(x>0) + 0.01*x.*(x<=0); % leaky relu
% dfx = 1*(x>0) + 0.01*(x<=0);
fx = x.*(x>0);
dfx = 1*(x>0);
end
```