DD2424 - Assignement 2

Pierre Rudin

May 24, 2017

1 Introduction

The aim of this assignment is to train a two layer network using a mini-batch gradient descent with momentum, so that the network can classify images from the CIFAR-10 data set. This will be done by implementing the calculations and algorithms given by the instructions for this assignment in Matlab.

1.1 Data set

CIFAR-10 is a data set of 60000 labeled 32 x 32 pixel coloured images separated in 5 training batches and 1 test batch, each batch holding 10000 images. In this assignment 2 training batches and the test batch i used.

1.2 Algorithm

This algorithm make use of some pretty simple linear algebraic equations. Since the equations are given by the instructions this section will contain my implementation step by step.

1. The data sets are loaded and the network initialized. The data set loads to $\mathbf{X}$ (size d x N), $\mathbf{Y}$ (K x N) and $\mathbf{y}$ (1 x N). The network is initialized by creating $\mathbf{W_1}$ ($nodesInHidden$ x d), $\mathbf{W_2}$ (K, $nodesInHidden$), $\mathbf{b_1}$ ($nodesInHidden$, 1) and $\mathbf{b_2}$ (K x 1) which will contain randomly generated numbers. This is also where we create the mini-batches (**XBatches** and **YBatches**).
2. Take the j:th batches from **XBatches** and **YBatches** and make label predictions on each image in **XBatches**.
3. Compute gradients on the predictions and batches from previous step.
4. Add momentum
5. Update the network.
6. Repeat from step 2 until end of epochs or other condition is satisfied.

2 Results

2.1 Analyticall gradient checking

The analytically computed gradients where compared to numerically computed gradients, to compute the numerical gradients the code given with the assignment was used. The gradients are checked on a two layer network

	Sum	Mean	Min	Max
b1	6.191410e-06	1.238282e-07	6.644981e-13	6.188640e-06
b2	1.374946e-05	2.749892e-08	2.733445e-08	2.767633e-08
W1	4.240115e-03	2.760491e-08	2.723168e-08	1.127854e-06
W2	4.500134e-06	4.500134e-07	4.499698e-07	4.501177e-07

Table 1: Absolute difference between numerically and analytically computed gradients

Since the biggest difference between the numerically and analytically computed gradients are smaller than 1e-5, it's safe to say that the analytically computed gradients are accurate.

2.2 Momentum

To test the effects of momentum, a small sub-set of the training data is selected to perform many training epochs in a reasonable time. It is then possible to compare the output graphs to determine the effects of the added momentum.

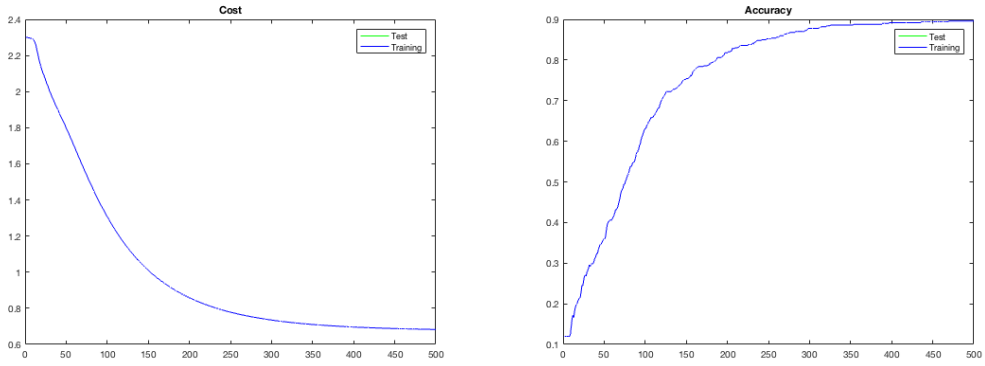

Figure 1: Graphs displaying cost and accuracy development for each epoch, when $\rho = .2$.

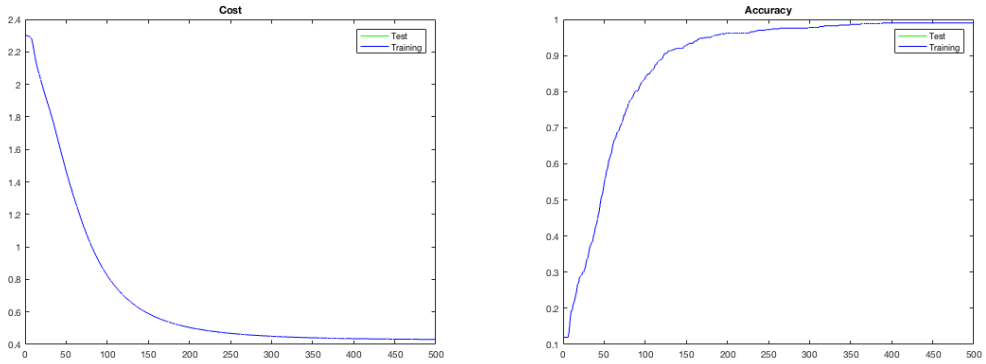

Figure 2: Graphs displaying cost and accuracy development for each epoch, when $\rho = .45$.

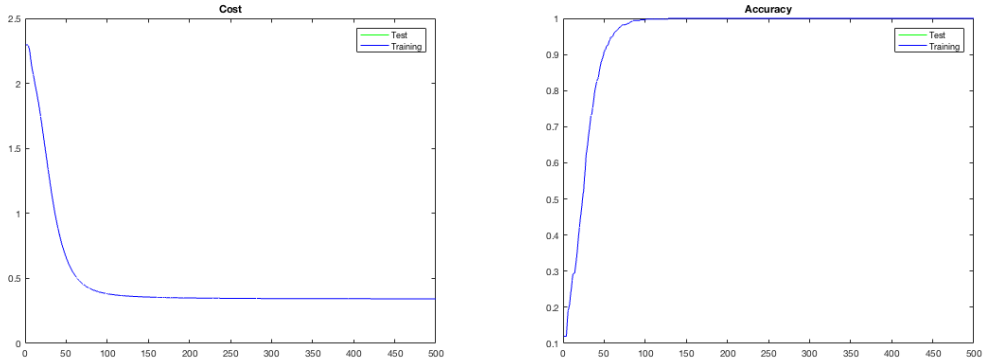

Figure 3: Graphs displaying cost and accuracy development for each epoch, when $\rho = .7$.

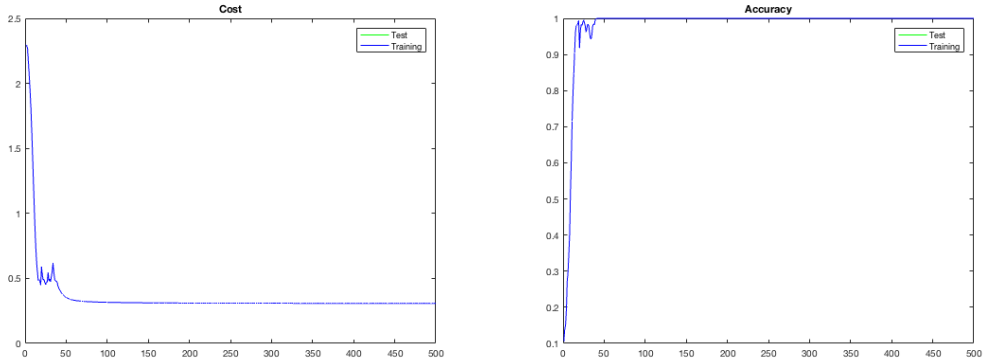

Figure 4: Graphs displaying cost and accuracy development for each epoch, when $\rho = .95$.

As shown in the graphs the accuracy curve grows faster and the cost curve descends faster when the momentum increases.

2.3 Coarse search

The initial search for our parameters starts with these values;

$\eta = [.01, .025, .05, .1, .25, .5, .75]$

$\lambda = [0, .001, .0025, .005, .01, .025, .05, .1]$

Which combined adds up to 56 different combination. Each combination is used to train the network for 5 epochs.

2.3.1 #1

Parameters:

$\eta = .025$

$\lambda = .001$

Accuracy test: .4243 Accuracy train: .4884

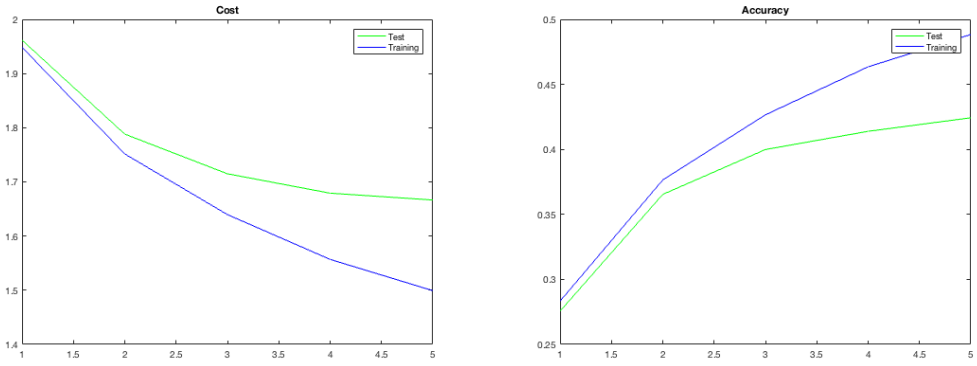

Figure 5: Graphs displaying cost and accuracy development for each epoch, when $\eta = .025$ and $\lambda = .001$.

2.3.2 #2

Parameters:

$\eta = .025$

$\lambda = 0$

Accuracy test: .4243 Accuracy train: .4938

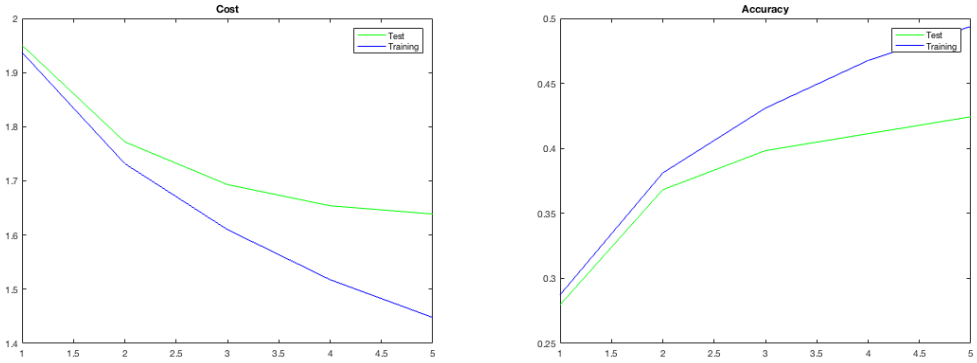

Figure 6: Graphs displaying cost and accuracy development for each epoch, when $\eta = .025$ and $\lambda = 0$.

2.3.3 #3

Parameters:

$\eta = .025$

$\lambda = .0025$

Accuracy test: .4146 Accuracy train: .4764

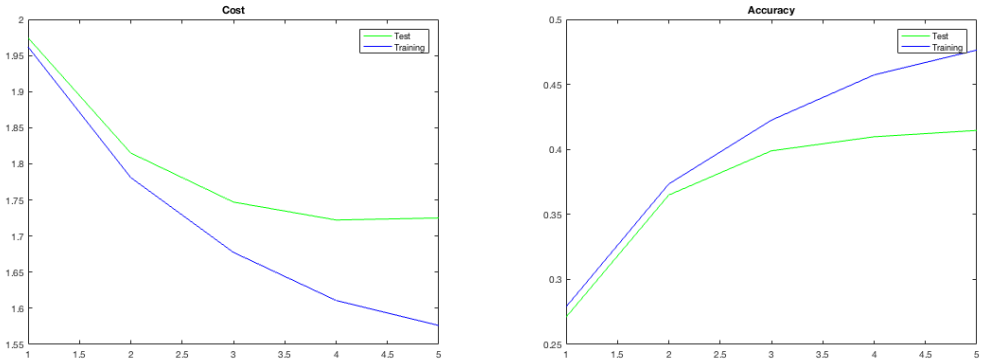

Figure 7: Graphs displaying cost and accuracy development for each epoch, when $\eta = .025$ and $\lambda = .0025$.

2.4 Fine search

With the result from the Coarse search in mind, the following parameters where chosen for the Fine search;

η = (.04-.01) .* rand(1,8) + .01

λ = (.005) .* rand(1,8)

Which combined adds up to 64 different combination. Each combination is used to train the network for 10 epochs.

2.4.1 #1

Parameters:

η = .016228

λ = .002828

Accuracy test: .4413, Accuracy train: .5242

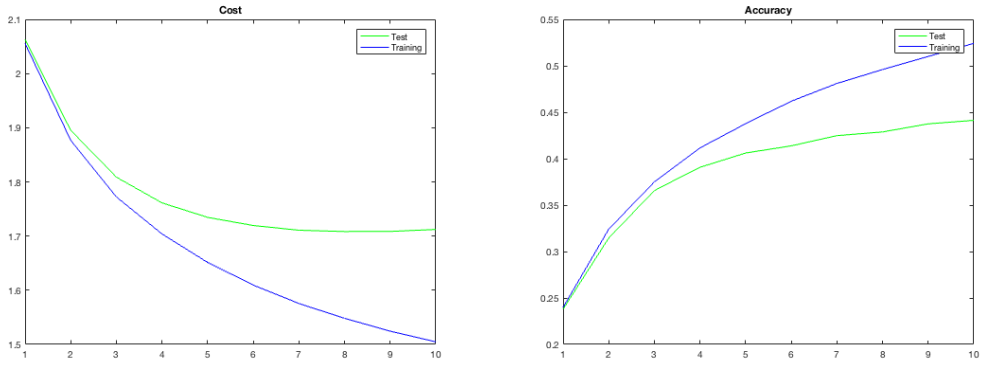

Figure 8: Graphs displaying cost and accuracy development for each epoch, when η = .016228 and λ = .002828.

2.4.2 #2

Parameters:

η = .022175

λ = .001514

Accuracy test: .4376, Accuracy train: .5532

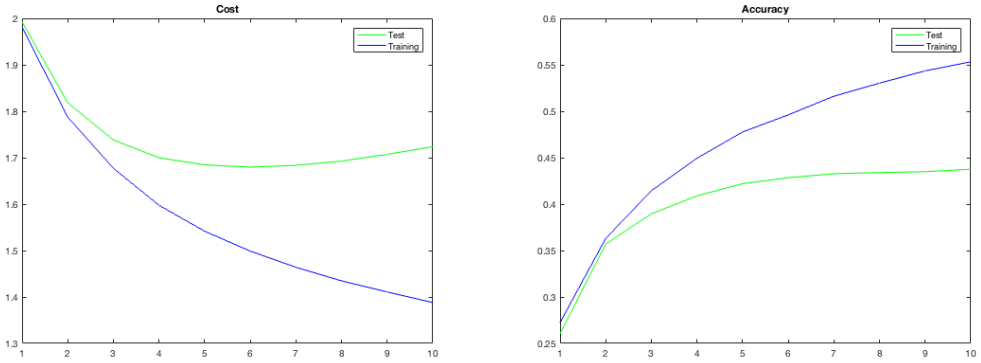

Figure 9: Graphs displaying cost and accuracy development for each epoch, when $\eta = .022175$ and $\lambda = .001514$.

2.4.3 #3

Parameters:

$\eta = .022175$

$\lambda = .003633$

Accuracy test: .4354, Accuracy train: .5253

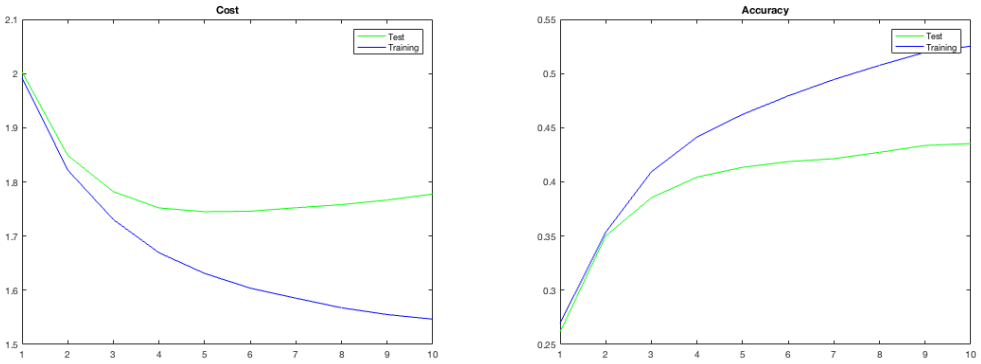

Figure 10: Graphs displaying cost and accuracy development for each epoch, when $\eta = .022175$ and $\lambda = .003633$.

2.5 Best parameters

When the best parameters are found, a further run is done where the network is trained for 30 epochs on the full training set except for the last 1000 images that are used for validation. Parameters:

$\eta = .016228$

$\lambda = .002828$

Accuracy test: .742, Accuracy train: .648222

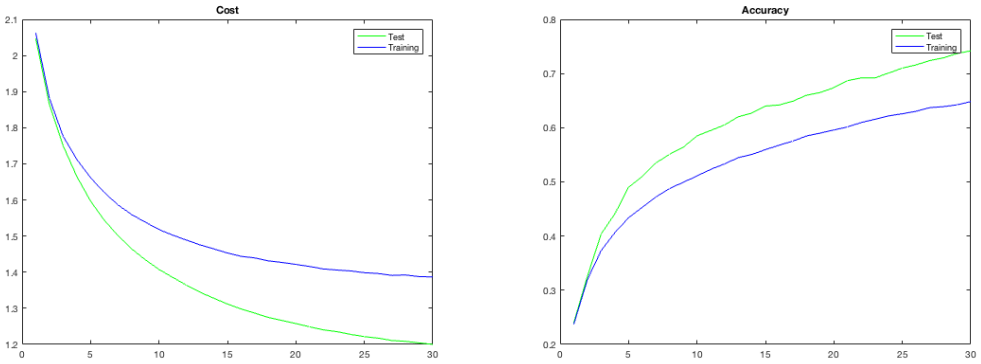

Figure 11: Graphs displaying cost and accuracy development for each epoch, when $\eta = .016228$ and $\lambda = .002828$.

3 Code

3.1 Main

```
clear all;
clc;
close all;
format longEng

addpath Datasets/cifar-10-batches-mat;

% Parameters
n_batch = 50; %
n_epochs = 5; %
h = 1e-5; %
nodes_in_hidden_layer = 100; % Number of nodes in hidden layer

```

```
% Hyper parameters
eta = .044576; % learning rate
lambda = .00998; % regularization
decay_rate = .998; % decay in learning rate
rho = .8; % Momentum

% Data setup
[X,Y,y,mean_X] = LoadBatch('data_batch_1.mat');
[XValid, YValid, yValid] = LoadData('data_batch_2.mat', mean_X);
[XTest,YTest,yTest] = LoadData('test_batch.mat', mean_X);
[XBatches, YBatches] = GetMiniBatches(X, Y, n_batch);

[W, b] = InitModel(X,nodes_in_hidden_layer);

[W,b,costs_train,costs_test,accs_train,accs_test] = trainingLoop(
    XBatches,YBatches,W,b,n_epochs,eta,lambda,rho,decay_rate,
    nodes_in_hidden_layer,X,Y,y,XTest,YTest,yTest);

PTest = EvaluateClassifier(XTest, W, b);
acc_test = ComputeAccuracy(PTest, yTest);

fprintf('Accuracy test\t%f\tCost test\t', acc_test);
for n = 1:length(costs_test)
    fprintf('%f\t ', costs_test(n));
end;

P = EvaluateClassifier(X, W, b);
acc_train = ComputeAccuracy(P, y);

fprintf('Accuracy train\t%f\tCost train\t', acc_train);
for n = 1:length(costs_train)
    fprintf('%f\t ', costs_train(n));
end;


% Plots evolution of the cost
figure(2);
x = 1:1:n_epochs;
plot(x,costs_test,'g',x,costs_train,'b');
title('Cost')
legend('Test','Training')

```

```
% Plots evolution of the accuracy
figure(3);
x = 1:1:n_epochs;
plot(x,accs_test,'g',x,accs_train,'b');
title('Accuracy')
legend('Test','Training')
```

3.2 LoadBatch()

```
function [X, Y, y, mean_X] = LoadBatch(filename)
    A = load(filename);
    X = double(A.data') / 255;
    y = A.labels';
    y = y + uint8(ones(1,length(y))); % Add one to simplify
        indexing

    mean_X = mean(X, 2);
    X = X - repmat(mean_X, [1, size(X, 2)]);

    % Create image label matrix
    Y = zeros(10, length(X));
    for i = 1:length(Y)
        Y(y(i),i) = 1;
    end;
```

3.3 LoadData()

```
function [X, Y, y] = LoadData(filename, mean_X)
    A = load(filename);
    X = double(A.data') / 255;
    y = A.labels';
    y = y + uint8(ones(1,length(y))); % Add one to simplify
        indexing

    X = X - repmat(mean_X, [1, size(X, 2)]);

    % Create image label matrix
    Y = zeros(10, length(X));
    for i = 1:length(Y)
        Y(y(i),i) = 1;
    end;
```

3.4 InitModel()

```
function [W, b] = InitModel(X)
    [d, ~] = size(X);
    W = normrnd(0,.01,10,d);
    b = normrnd(0,.01,10,1);
```

3.5 GetMiniBatches()

```
function [XBatches, YBatches] = GetMiniBatches(Xtrain, Ytrain,
    n_batch)
    [d,N] = size(Xtrain);
    [K,~] = size(Ytrain);

    XBatches = zeros(N/n_batch,d,n_batch);
    YBatches = zeros(N/n_batch,K,n_batch);

    for j=1:N/n_batch
        j_start = (j-1)*n_batch + 1;
        j_end = j*n_batch;
        Xbatch = Xtrain(:, j_start:j_end);
        Ybatch = Ytrain(:, j_start:j_end);

        XBatches(j,:,:) = Xbatch;
        YBatches(j,:,:) = Ybatch;
    end

    % Permute to simplify picking out image representations from
        the
    % matrices.
    XBatches = permute(XBatches,[3 2 1]);
    YBatches = permute(YBatches,[3 2 1]);
```

3.6 trainingLoop()

```

function [W,b,costs_train,costs_test,accs_train,accs_test] =
    trainingLoop(XBatches,YBatches,W,b,n_epochs,eta,lambda,rho,
    decay_rate,nodes_in_hidden_layer,X,Y,y,XTest,YTest,yTest)
    accs_train = double.empty();
    accs_test = double.empty();
    costs_train = double.empty();
    costs_test = double.empty();

    %Momentum
    moment_W = {zeros(size(W{1})), zeros(size(W{2}))};
```

```
10      moment_b = {zeros(size(b{1})), zeros(size(b{2}))};
11
12      [~,~,l] = size(XBatches);
13
14      for i = 1:n_epochs
15          for j = 1:l
16              % Get j:th batch
17              XBatch = XBatches(:,:,j)';
18              YBatch = YBatches(:,:,j)';
19
20              %hidden_layer = MakeHiddenLayer(XBatch, W{1}, b{1});
21              P = EvaluateClassifier(XBatch, W, b);
22
23              [grad_W, grad_b] = ComputeGradients(XBatch, YBatch, P
                    , W, b, lambda, nodes_in_hidden_layer);
24
25              % Calculate momentum
26              moment_W{1} = eta * grad_W{1} + rho * moment_W{1};
27              moment_b{1} = eta * grad_b{1} + rho * moment_b{1};
28              moment_W{2} = eta * grad_W{2} + rho * moment_W{2};
29              moment_b{2} = eta * grad_b{2} + rho * moment_b{2};
30
31              % Update W's and b's
32              W{1} = W{1} - moment_W{1};
33              b{1} = b{1} - moment_b{1};
34              W{2} = W{2} - moment_W{2};
35              b{2} = b{2} - moment_b{2};
36
37          end;
38          eta = decay_rate * eta;
39
40          %hidden_layer = MakeHiddenLayer(X, W{1}, b{1});
41          P = EvaluateClassifier(X, W, b);
42          acc_train = ComputeAccuracy(P, y);
43          cost_train = ComputeCost(X, Y, W, b, lambda);
44
45          %hidden_layer = MakeHiddenLayer(XTest, W{1}, b{1});
46          PTest = EvaluateClassifier(XTest, W, b);
47          acc_test = ComputeAccuracy(PTest, yTest);
48          cost_test = ComputeCost(XTest, YTest, W, b, lambda);
49
50          accs_train = [accs_train acc_train];
```

```
51          accs_test = [accs_test acc_test ];
52
53          costs_train = [costs_train cost_train];
54          costs_test = [costs_test cost_test ];
55
56          % Displays the cost and accuracy of each epoch
57          fprintf('Cost train: %f\n', cost_train);
58          fprintf('Accuracy train: %f\n', acc_train);
59          fprintf('Cost test: %f\n', cost_test);
60          fprintf('Accuracy test: %f\n\n', acc_test);
61
62          fprintf('Epoch %f is done.\n', i);
63      end;
```

3.7 EvaluateClassifier()

```
1  function P = EvaluateClassifier(X, W, b)
2      hiddenLayer = MakeHiddenLayer(X, W{1}, b{1});
3      s2 = W{2}*hiddenLayer+b{2};
4      P = exp(s2) ./ sum(exp(s2));
```

3.8 MakeHiddenLayer()

```
1  function hiddenLayer = MakeHiddenLayer(X, W, b)
2      s = W * X + b;
3      hiddenLayer = max(0,s);
```

3.9 ComputeGradients()

```
1  function [grad_W, grad_b] = ComputeGradients(X, Y, P, W, b,
       lambda, nodes_in_hidden_layer)
2      [hY, ~] = size(Y);
3      [hX, lX] = size(X);
4
5      grad_W = {zeros(nodes_in_hidden_layer, hX), zeros(hY,
           nodes_in_hidden_layer)};
6      grad_b = {zeros(nodes_in_hidden_layer, 1), zeros(hY, 1)};
7
8      hidden_layer = MakeHiddenLayer(X, W{1}, b{1});
9
10     for i = 1:lX
11         yy = Y(:,i);
12         xx = X(:,i);
13
```

```
        p = P(:,i);
        [hp,~] = size(p);
        diag_p = eye(hp) .* p;
        g = -((yy' / (yy' * p)) * (diag_p - p * p'));

        grad_b{2} = grad_b{2} + g';
        grad_W{2} = grad_W{2} + g' * hidden_layer(:,i)';

        s1 = W{1} * xx + b{1};
        [hs1, ~] = size(s1);

        g = g * W{2};
        diag_s1 = eye(hs1) .* (s1>0);
        g = g * diag_s1;



        grad_b{1} = grad_b{1} + g';
        grad_W{1} = grad_W{1} + g' * xx';
    end;

    grad_b{1} = grad_b{1} ./ lX;
    grad_W{1} = grad_W{1} ./ lX + 2 * lambda .* W{1};

    grad_b{2} = grad_b{2} ./ lX;
    grad_W{2} = grad_W{2} ./ lX + 2 * lambda .* W{2};
```

3.10 ComputeCost()

```
function J = ComputeCost(X, Y, W, b, lambda)
    P = EvaluateClassifier(X, W, b);
    c = 0;
    [~,x] = size(X);
    for i = 1:x
        y = Y(:,i);
        p = P(:,i);
        c = c + -log(y'*p);
    end;
    J = c/x + lambda * sum(sum(W{1}.^2)) + lambda * sum(sum(W
        {2}.^2));
```

3.11 ComputeAccuracy()

```
function acc = ComputeAccuracy(P, y)
```

```
2       [~,prediction] = max(P);
3       acc = sum(prediction==y) / length(P);
```