DD2424 - Assignement 1

Pierre Rudin

May 5, 2017

1 Introduction

The aim of this assignment is to train a network using a mini-batch gradient descent 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 validation batch, each batch holding 10000 images.

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 $\mathbf{W}$ (K x d) and $\mathbf{b}$ (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. Update the network.

5. Repeat from step 2 until end of epochs or other condition is satisfied.

2 Results

This section will present results from the four test runs. Each run will be presented with images of the cost and accuracy evolution during the learning process as well with a image representation of the weights and the final cost and accuracy on both validation and learning data.

2.1 Run #1

Parameters:

$\lambda = 0$

$n\_epochs = 40$

$n\_batch = 100$

$\eta = .1$

	Training	Validation
Cost	7.336	8.052
Accuracy	26.52 %	24.23 %

Table 1: Final test results

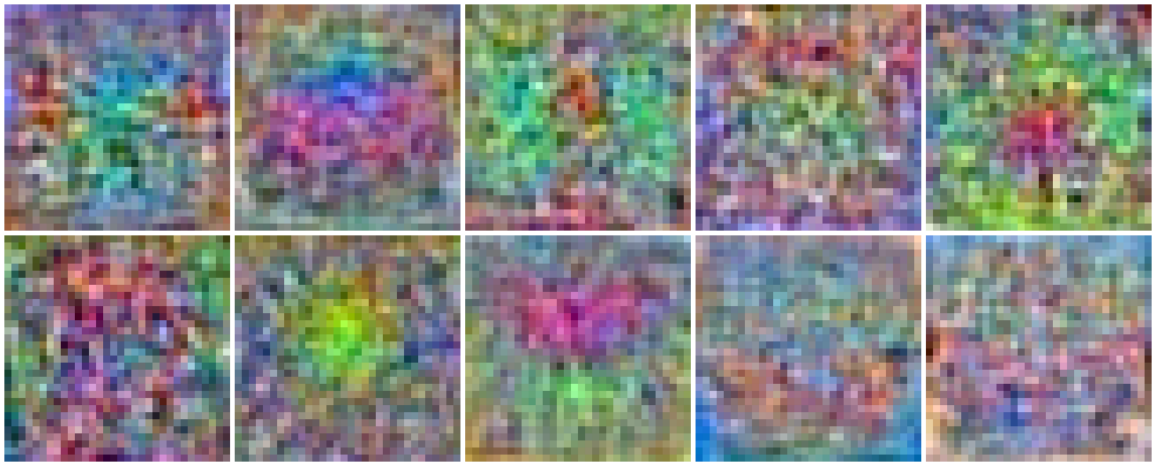

Figure 1: Image representation of the weights

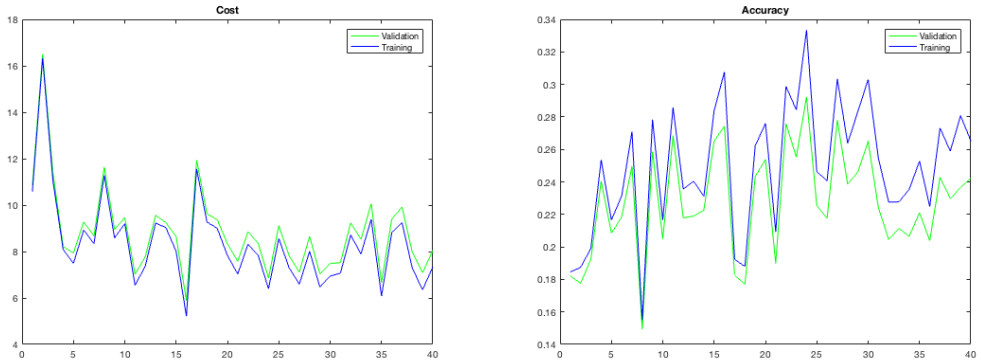

Figure 2: Graphs displaying cost and accuracy development for each epoch.

As seen in the cost and accuracy graphs, they really are all over the place. This run is very unstable and the network appears to behave almost random.

2.2 Run #2

Parameters:

$\lambda = 0$

$n\_epochs = 40$

$n\_batch = 100$

$\eta = .01$

	Training	Validation
Cost	1.679	1.800
Accuracy	41.68 %	36.89 %

Table 2: Final test results

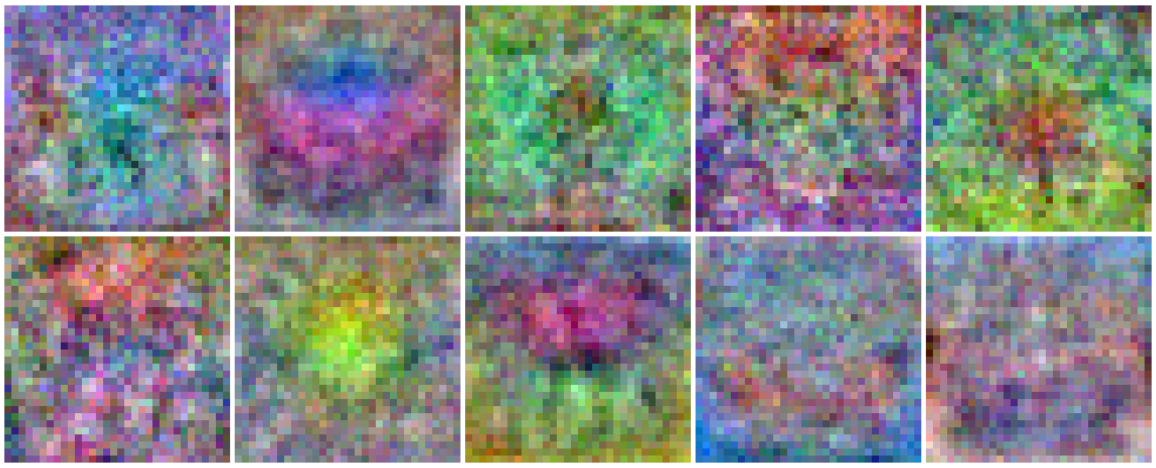

Figure 3: Image representation of the weights

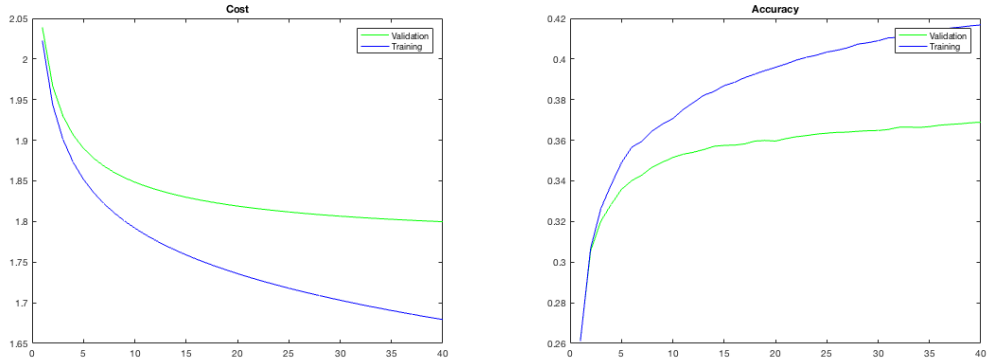

Figure 4: Graphs displaying cost and accuracy development for each epoch.

In comparison to the first run, the network now seems to learn something during each epoch and is much more stable. The big gap between the training and validation graphs might suggest that the network is over fitted.

2.3 Run #3

Parameters:

$\lambda = .1$

$n\_epochs = 40$

$n\_batch = 100$

$\eta = .01$

	Training	Validation
Cost	1.982	2.014
Accuracy	34.16 %	33.42 %

Table 3: Final test results

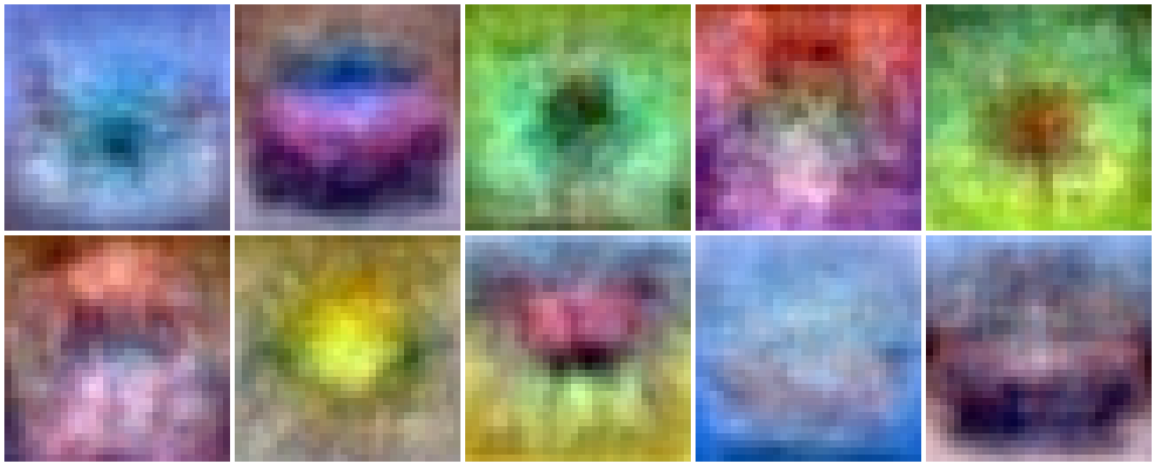

Figure 5: Image representation of the weights

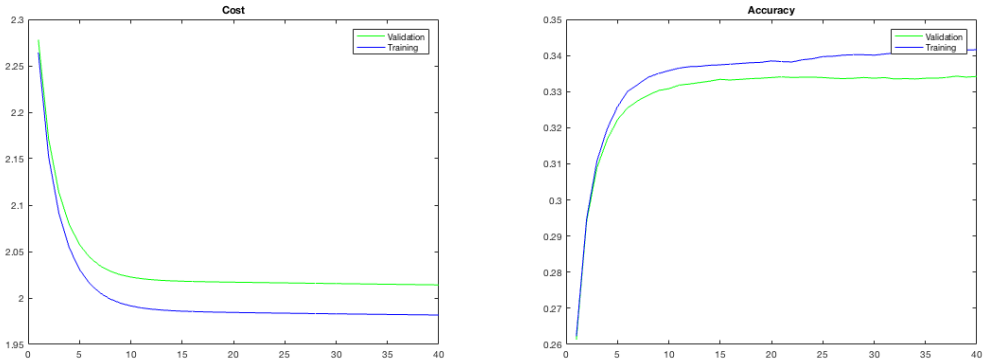

Figure 6: Graphs displaying cost and accuracy development for each epoch.

In comparison to the second run, the accuracy is lower but the lines in the graph are closer. Which suggests that this network ain't over fitted. It's also interesting to see that the weight images got clear clusters of colors and some shapes.

2.4 Run #4

Parameters:

$\lambda = 1$

$n\_epochs = 40$

$n\_batch = 100$

$\eta = .01$

	Training	Validation
Cost	2.195	2.205
Accuracy	22.30 %	21.86 %

Table 4: Final test results

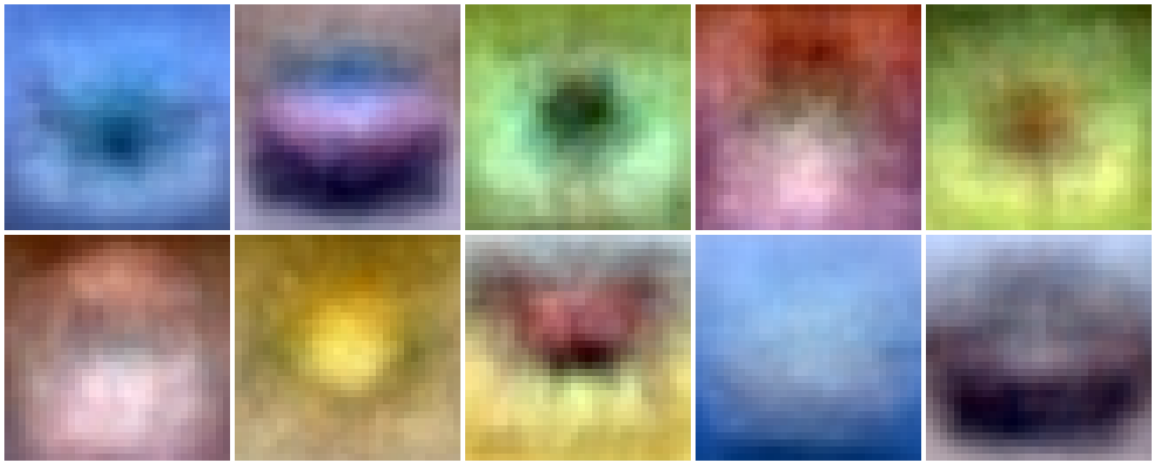

Figure 7: Image representation of the weights

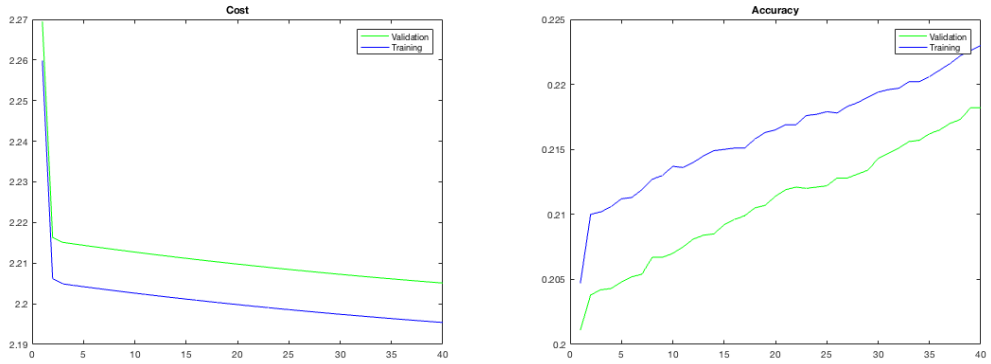

Figure 8: Graphs displaying cost and accuracy development for each epoch.

In this run the gap between training and validation is even smaller then in run #3. This comes with the cost that the accuracy is even lower than before. However it doesn't look like the network has converged and it is possible that with further learning that the final result might get better.

3 Conclusion

3.1 Gradient

When comparing my gradients with the gradients generated by the two scripts given, the difference where in almost all tests I did smaller than 1e-6 and always smaller than 1e-5. So I would say that I managed to compute the gradients.

3.2 Regularization

The two most obvious effects of regularization is that the accuracy becomes lower but the performance on booth training and validation data is more similar, thus less over fitted to training data. When looking at the weight matrix images, it looks like the regularization has the effects of noise reduction.

3.3 Performance

Best case here seems to be a accuracy of about 35-40 %. This is of course a lot better then random, but maybe not as good as you would want for it to be useful in some scenarios, e.g. I don't think many people would use Google image search if the accuracy wasn't higher than this. The low accuracy might be because of the algorithm, but it could also be because of the data set (to little data or too small/large batches).

4 Code

4.1 Main

```
clear all;
clc;
close all;

addpath Datasets/cifar-10-batches-mat/;

% Creates empty arrays to store results.
costs_test = double.empty();
costs_train = double.empty();
accs_train = double.empty();
accs_test = double.empty();

% Parameters
lambda = 0.1;
n_batch = 100;
n_epochs = 40;

% Learning rate with linear step size
eta = .01;
final_eta = .01;
eta_step = (eta - final_eta)/n_epochs;

% Data setup
[X,Y,y] = LoadBatch('data_batch_1.mat');
[Xtest,Ytest,ytest] = LoadBatch('test_batch.mat');
[W, b] = InitModel(X);
[XBatches, YBatches] = GetMiniBatches(X, Y, n_batch);
[~,~,l] = size(XBatches);

% Learning loop
for i = 1:n_epochs
    for j = 1:l
        % Get j:th batch
        XBatch = XBatches(:,:,j)';
        YBatch = YBatches(:,:,j)';

        % Make prediction and calculate gradients
        P = EvaluateClassifier(XBatch, W, b);
```

```
39          [grad_W, grad_b] = ComputeGradients(XBatch, YBatch, P, W,
                lambda);
40
41          % Update W and b
42          W = W - eta * grad_W;
43          b = b - eta * grad_b;
44      end;
45      eta = eta - eta_step;
46
47      % Calculate predictions, cost and accuracy of each epoch
48      P_train = EvaluateClassifier(X, W, b);
49      P_test = EvaluateClassifier(Xtest, W, b);
50      cost_test = ComputeCost(Xtest, Ytest, W, P_test, lambda);
51      cost_train = ComputeCost(X, Y, W, P_train, lambda);
52      acc_train = ComputeAccuracy(P_train, y);
53      acc_test = ComputeAccuracy(P_test, ytest);
54
55      % Displays the cost and accuracy of each epoch
56      fprintf('Cost train: %f\n', cost_train);
57      fprintf('Accuracy train: %f\n', acc_train);
58      fprintf('Cost test: %f\n', cost_test);
59      fprintf('Accuracy test: %f\n\n', acc_test);
60
61      % Store cost and accuracy of each epoch
62      costs_test = [costs_test cost_test];
63      costs_train = [costs_train cost_train];
64      accs_train = [accs_train acc_train];
65      accs_test = [accs_test acc_test ];
66  end;
67
68  % Plots the weights
69  figure(1);
70  for i=1:10
71      im = reshape(W(i, :), 32, 32, 3);
72      s_im{i} = (im - min(im(:))) / (max(im(:)) - min(im(:)));
73      s_im{i} = permute(s_im{i}, [2, 1, 3]);
74  end
75  montage(s_im, 'Size', [2,5]);
76
77  % Plots evolution of the cost
78  figure(2);
79  x = 1:1:n_epochs;
```

```
plot(x,costs_test,'g',x,costs_train,'b');
title('Cost')
legend('Validation','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('Validation','Training')
```

4.2 LoadBatch()

```
function [X, Y, y] = 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

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

4.3 InitModel()

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

4.4 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]);
```

4.5 EvaluateClassifier()

```
function P = EvaluateClassifier(X, W, b)
    s = W*X+b;
    P = exp(s) ./ sum(exp(s));
```

4.6 ComputeGradients()

```
function [grad_W, grad_b] = ComputeGradients(X, Y, P, W, lambda)
    [hY, lY] = size(Y);
    [hX, lX] = size(X);
    grad_W = zeros(hY,hX);
    grad_b = zeros(hY,1);

    for i = 1:lX
        yy = Y(:,i);
        x = X(:,i);
        p = P(:,i);

        diag_p = eye(10) .* p;
        g = -((yy' / (yy' * p)) * (diag_p - p * p'));

        grad_b = grad_b + g';
        grad_W = grad_W + g' * x';
    end;

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

4.7 ComputeCost()

```
function J = ComputeCost(X, Y, W, P, lambda)
    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.^2));
```

4.8 ComputeAccuracy()

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