DD2424 Deep Learning

Assignment 2

Yunpeng Ma

1. Introduction

The purpose is to train a two-layer neural network using mini-batch gradient descent which is applied to a cross-entropy loss function classifier with regulation.

2. Methodology

In order to train a multi-linear classifier, we have dataset of CIFAR-10, the output will be vector of probabilities. The following equations will be used for this training process:

$$s = W_1 x + b_1$$

$$\mathrm{h} = \max(0, s_1)$$

$$s = W_2 x + b_2$$

$$p = \frac{\exp(s)}{1^T \exp(s)}$$

During the training process, we will learn the parameters of W and b of the cost function:

$$W^*, b^* = \arg\min_{W,b} J(D, \lambda, W, b)$$

J is a cost function of cross-entropy with a regularization term:

$$J(D, \lambda, W, b) = \frac{1}{|D|} \sum_{x,y \in D} \log(y^T p) + \lambda \sum_{i,j} W_{ij}^2$$

We use mini-batch gradient descent method to learn W and b.
In this assignment, we use cyclic learning rate to update W and b, the formula is:

$$\eta_t = \eta_{\min} + \frac{t - 2l\,\mathrm{n}_s}{n_s}(\eta_{\max} - \eta_{\min})$$

when $2l \cdot \mathrm{n}_s \le t \le (2l+1)n_s$ for some $l \in \{0,1,2,...\}$;

$$\eta_t = \eta_{\max} - \frac{t - (2l+1)\,\mathrm{n}_s}{n_s}(\eta_{\max} - \eta_{\min})$$

when $(2l+1) \cdot \mathrm{n}_s \le t \le 2(l+1)n_s$ for some $l \in \{0,1,2,...\}$;

3. Exercise

3.1 Initialize the parameters of the network

To normalize the training, validation and test data with respect to the mean and standard deviations. The following codes are used:

```
[train_X, train_Y, train_y] = LoadBatch('data_batch_1.mat');
[validation_X, validation_Y, validation_y]=LoadBatch('data_batch_2.mat');
[testX,testY,testy]=LoadBatch('test_batch.mat');

mean_X = mean (train_X,    2);
std_X = std(train_X, 0, 2);
train_X = train_X - repmat(mean_X,[1, size(train_X,2)]);
train_X = train_X./ repmat(std_X,[1, size(train_X,2)]);

validation_X = validation_X - repmat(mean_X,[1, size(validation_X,2)]);
validation_X = validation_X./ repmat(std_X,[1, size(validation_X,2)]);

testX = testX - repmat(mean_X,[1, size(testX,2)]);
testX = testX./ repmat(std_X,[1, size(testX,2)]);
```

In this assignment, we have two layers and each layer has different sizes of parameters, we set bias to zero and weight matrices randomly Gaussian distributed with mean 0 and standard deviation 1/sqrt(d) for layer 1 and 1/sqrt(m) for layer 2.the function is:

```
function [W,b] = Initializing_network(d, m, K)
W1 = 1/sqrt(d).*randn(m, d);
W2 = 1/sqrt(m).*randn(K,m);
b1 = zeros(m,1);
b2 = zeros(K,1);
W = {W1, W2};
b = {b1, b2};
end
```

3.2 Compute the gradients for the network parameters

The codes for computing final p values and mini-batch gradients can be seen in the m file. By computing the relative error between a numerically computed gradient value g_n and an analytically computed gradient value g_a

$$\frac{|g_a - g_n|}{\max(eps, |g_a| + |g_n|)}$$

If this value is small, we can believe the calculation is correct.
During the training process, I get relative error of W is less than 10^{-6}, relative error of b is less than 10^{-6}.

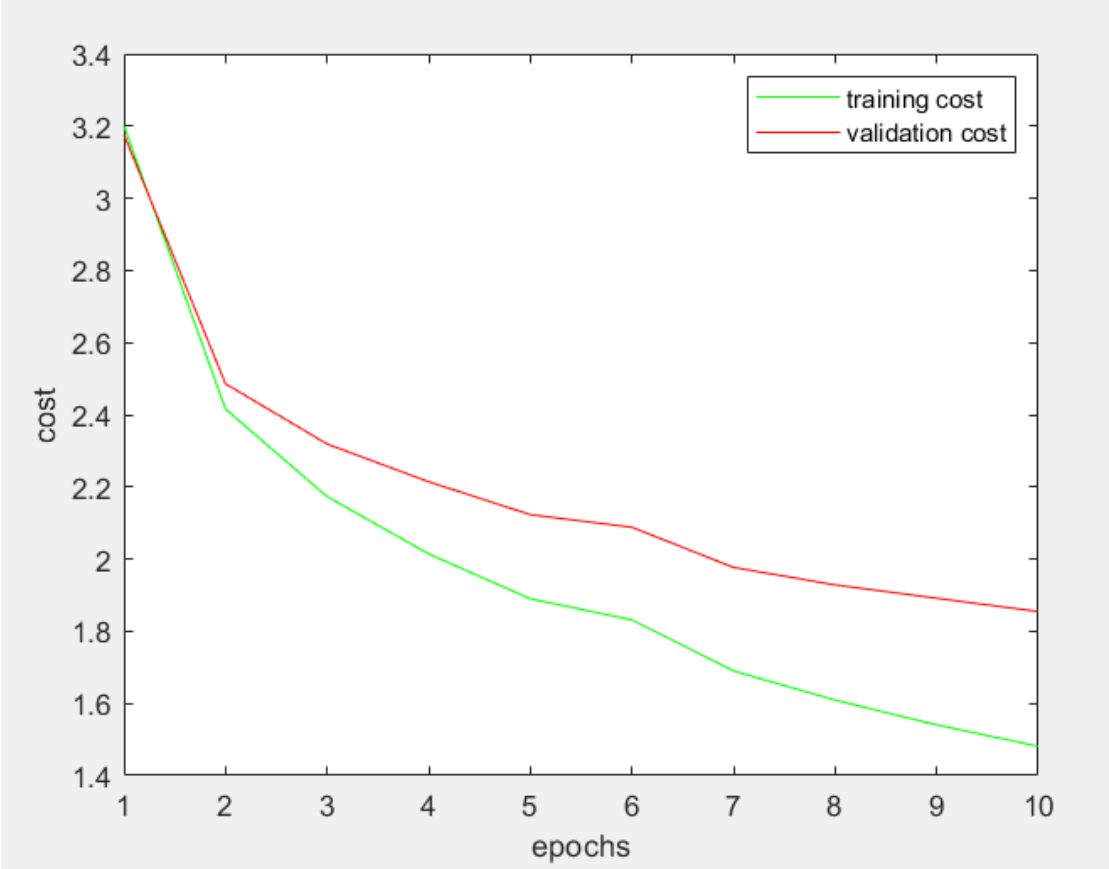
Parameters	Relative Error
W1	$5.1804*10^{-10}$
W2	$2.3393*10^{-10}$
b1	$4.6837*10^{-10}$

b2	$1.5225*10^{-10}$

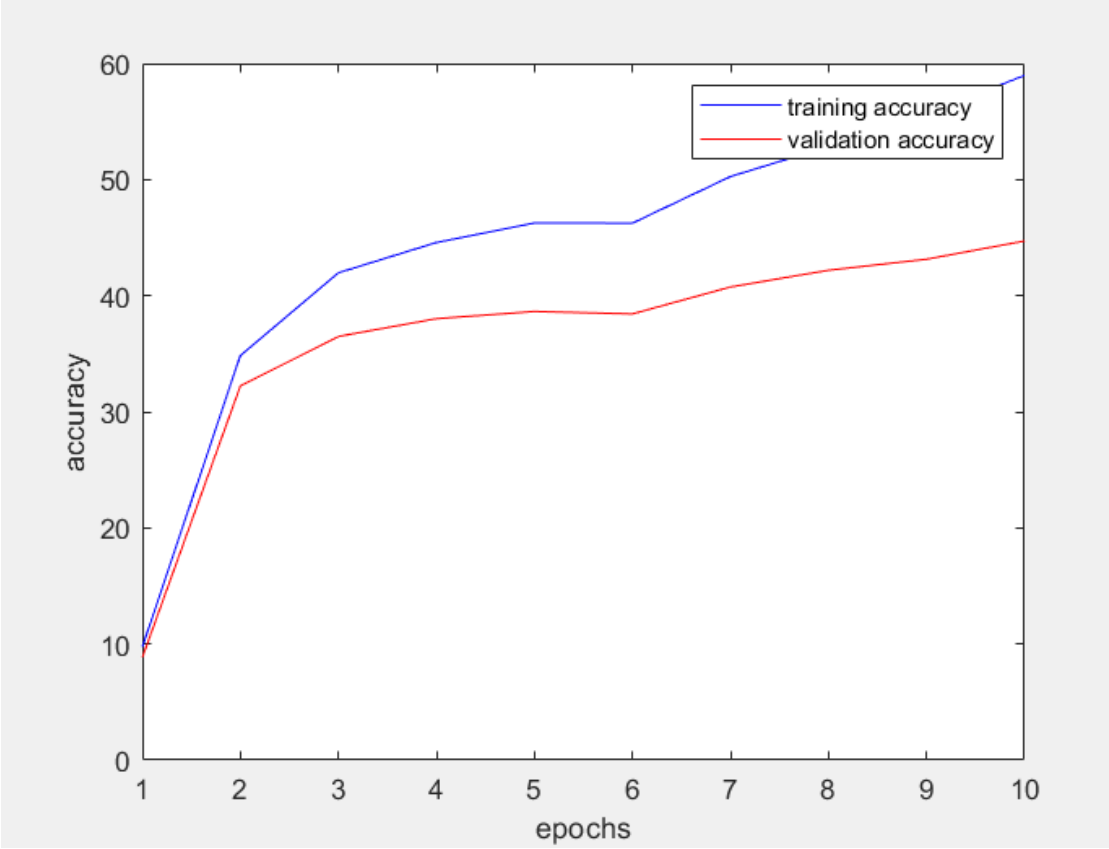
3.3 Train your network with cyclical learning rates

In this exercise, I wrote a function to calculate eta, and put every eta with respect to every step in an array and call this array when I update the weights. I gave parameters as following:

eta_min = 10^{-5}, eta_max= 10^{-1}, n_s=500 and the batch_size= 100, lambda=0.01, t=1 to t=2n_s, and get the cost curve:



The cost of training data and validation data reduce with increasing of epochs. The accuracy curve is shown as following:

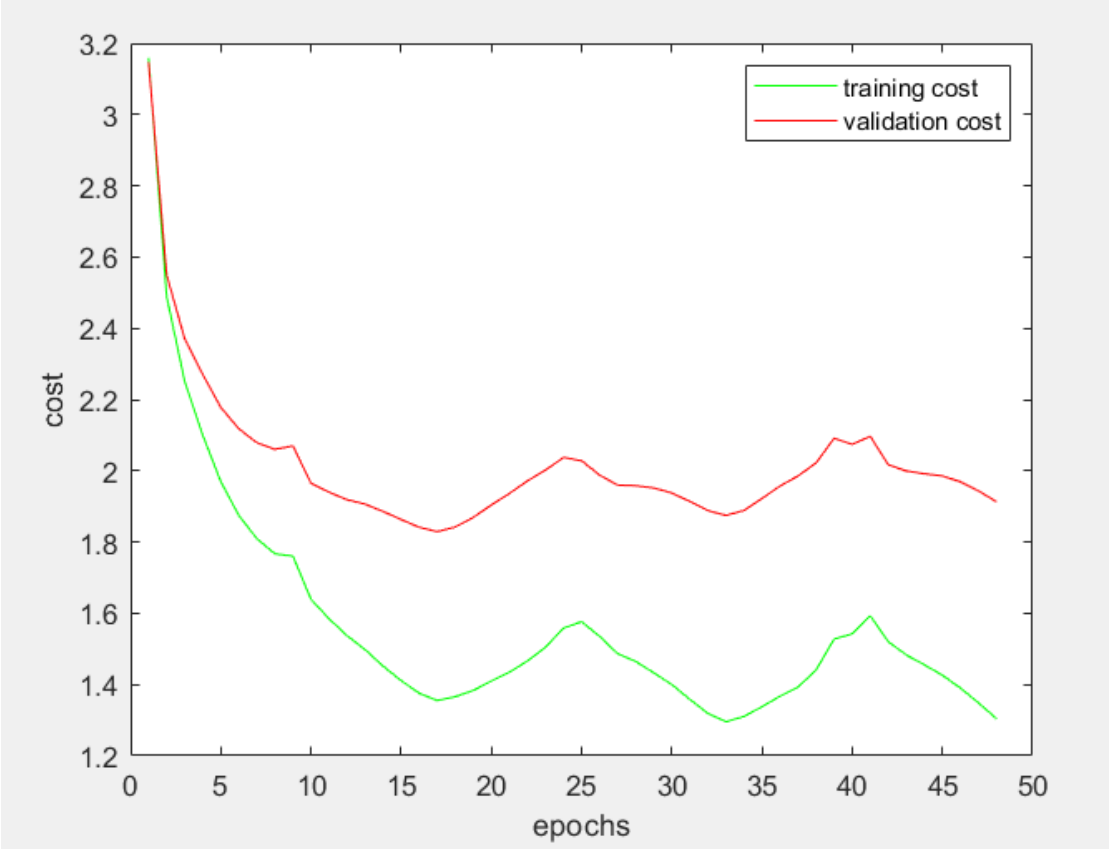


We can see training and validation accuracy increase with increasing of epochs; I also calculated the accuracy of training data and test data after 1000 update steps which is 60.71% for training data and 45% for testing data.

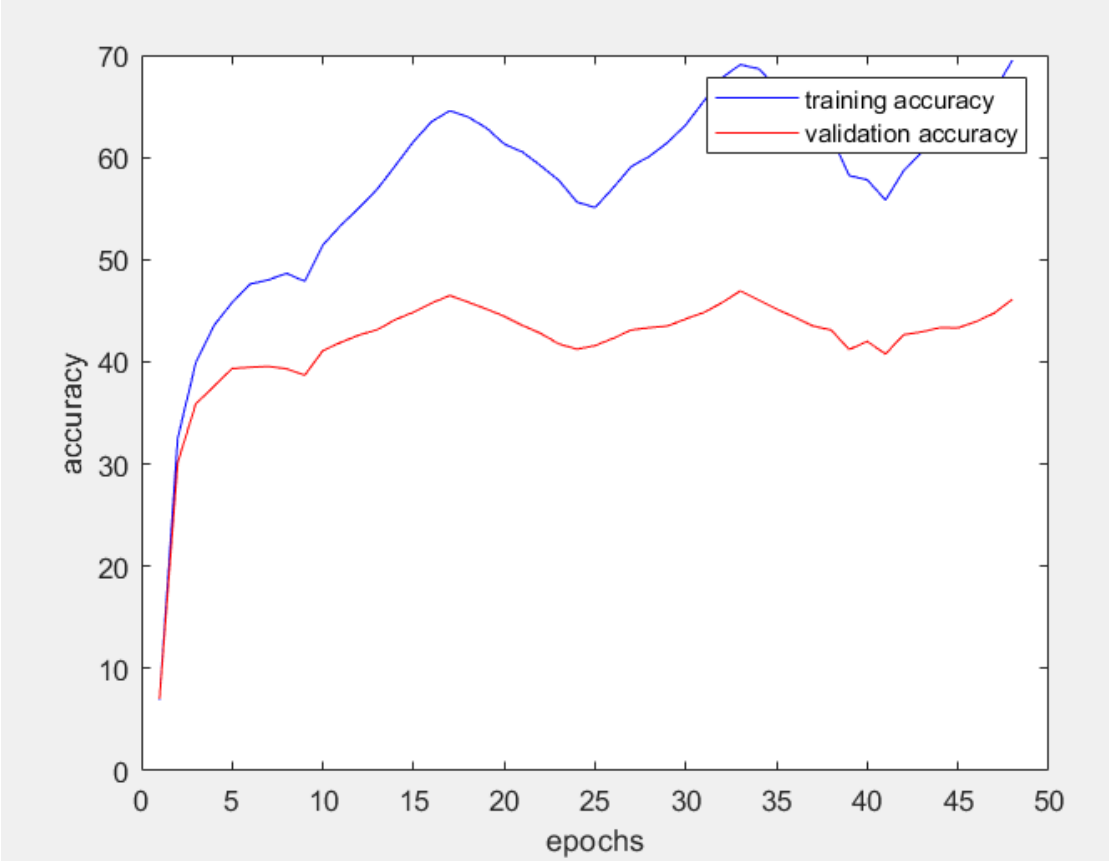
3.4 Train your network for real

3.4.1 Run 3 cycles without optimizing regularization

In this exercise, to run 3 cycles of I set parameters as following:
eta_min = 10^{-5}, eta_max= 10^{-1}, n_s=800 and the batch_size= 100, lambda=0.01, t=1 to t=6n_s, and get the cost curve:



The accuracy curve:



After 3 cycles' running, the training accuracy is 70.84%, and the test accuracy is 46.76%

comparing 45% of running 1 cycle. We can conclude that when we run more cycles the training data is better trained (3 cycles:70.84%, 1 cycle: 60.71%), test accuracy also increases, but not much as training data, that is mainly because we have a small size of training data, after 3 cycles' training, it get overfitting when we do not optimize regularization term. Nevertheless, the performance of test data still increases comparing one cycle.

3.4.2 Coarse-to-fine random search to set lambda

In this exercise, because of my laptop memory limits, I can only train 3 batches of datasets. I use 5000 images as validation data which is chosen in batch 2, the rest 25000 images are used as training data.

For the coarse search, I ran 15 random samples, the range of best performance for coarse search is from 0.005 to 0.03 and the lambdas which performs best are $\lambda = 1.3564 \cdot 10^{-2}$, $\lambda = 9.565 \cdot 10^{-3}$, $\lambda = 1.2538 \cdot 10^{-2}$ corresponding to the test accuracy 48.26%, 48.75%, 49.01%.

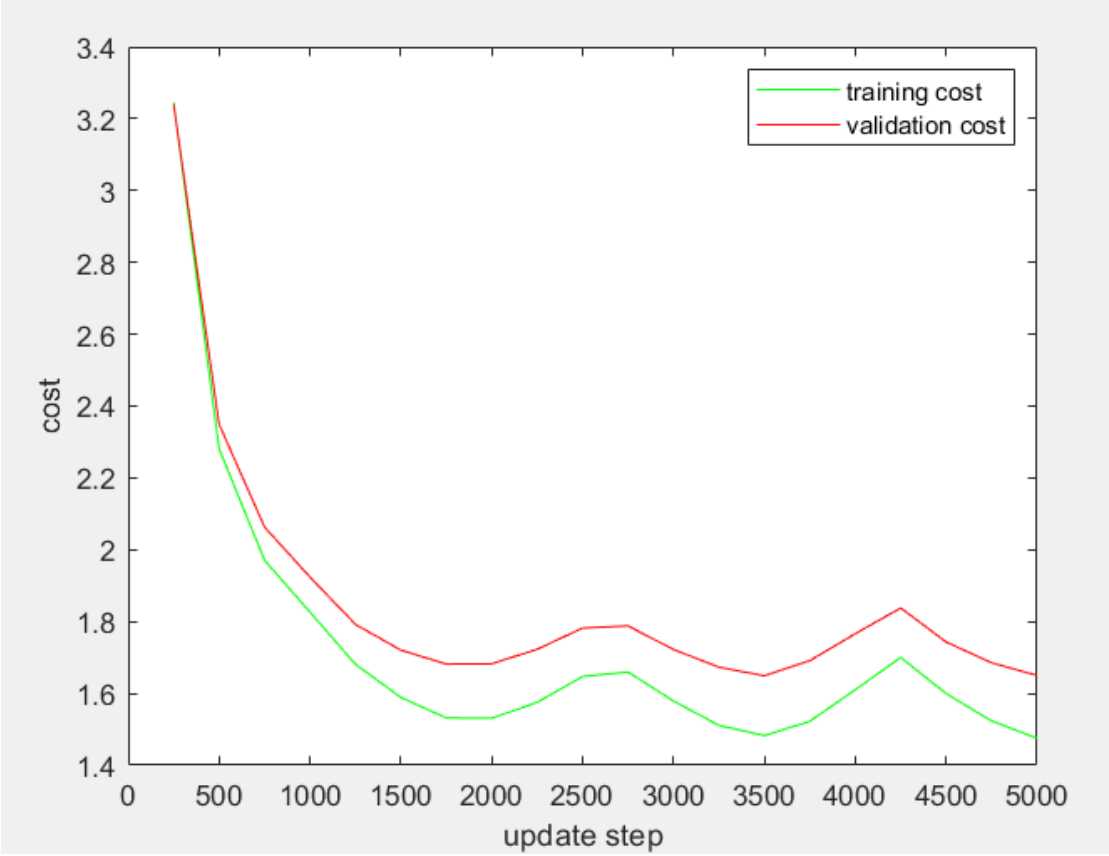
For the fine search, I shrink the range and ran the code when lambda changes between 0.007 to 0.022, and found the best three lambda $\lambda = 0.01156$, $\lambda = 0.01155$, $\lambda = 0.0115$, which corresponds to test accuracy 49.89%, 49.51%, 49.51%.

I tried a lot of lambdas and could not find the lambda which get test accuracy more than 50%, the best performance is 49.89%. I suppose that is because I just use 3 batches of data. If I increase the training data size to 5 batches, I think it would be easier to get more than 50% test accuracy.

The following is the best performance which the parameters are:

eta_min = 10^{-5}, eta_max= 10^{-1}, n_s=800 and the batch_size= 100, lambda=0.01156, t=1 to t=6n_s. I use 25000 images as training data(I cannot use more than that for the laptop memory limits), and 5000 images as validation data. The test accuracy is 49.89% which is the highest test accuracy I can get.

Cost curve:



Accuracy curve: