DD2424 Deep Learning in Data Science
Assignment 1 with bonus points

Rui Qu
rqu@kth.se

April 7, 2019

1 Introduction

To train and test a one layer network with multiple outputs to classify images from CIFAR-10 dataset by implementing mini-batch gradient descent applied to a cost function that computes cross-entropy loss of the classifier applied to the labelled training data and an L2 regularization term on the weight matrix.

CIFAR-10 dataset
The CIFAR-10 dataset consists of 60000 32x32x3(RGB) color images in 10 classes, with 6000 images per class. There are 50000 training images and 10000 test images.

Mini-batch gradient descent
Details in instruction of assignment 1.

2 Algorithm

Algorithm 1 Mini-batch gradient descent

```
1: Data preprocess, dataset loading to data($size : N \times d$), one-hot($size : N \times K$), label($size : N \times 1$)
2: Generate $weights(size : K \times d)$, $bias(size : K \times 1)$ and the mini-batches Xbatch and Ybatch
3: repeat
4:     Take $j_{th}$ batch in mini-batches predict the label of each image in Xbatch
5:     Compute gradient and cross-entropy cost
6:     Update weights and bias
7: until n-epochs
       return Best epoch &cost; plotting of training &validation cost and representation of weight matrix
```

The code is written in Python. Pseudo-code is given above.

3 Gradient Check

With understanding of numerical &analytic derivative, it's easy to compute numerical &analytic gradient. First numerical derivative:

$$f'(x') = \frac{\partial f(x)}{\partial x}|_{x=x'} \tag{1}$$

First analytic derivative:

$$f'(x') \approx \frac{f(x'+\delta) - f(x'-\delta)}{2\delta} \tag{2}$$

Then compare the two gradient via relative error:

```
def rel_err(x, y):
 e=np.max(np.abs(x-y)/np.maximum(1e-8,np.abs(x)+np.abs(y)))
 return e
```

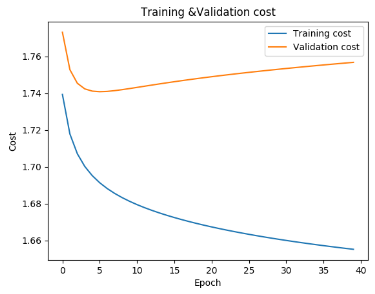
In this exercise, I set regularization lambda to 0 and epoch to 10, which improves efficiency of the computation. The relative error is very low, smaller than 1e-6. I could say that I successfully managed to write the functions to correctly compute the gradient.

4 Exercise1

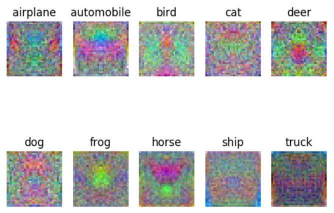
NB: Set decay factor to 1, which means there is no learning rate decay in this exercise.

4.1

lambda=0, n-epochs=40, n-batch=100, eta=0.1, decay factor=1



(a) Training &validation cost

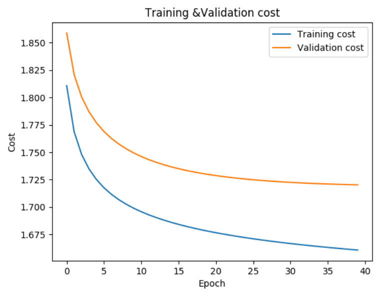


(b) Representation of weight matrix

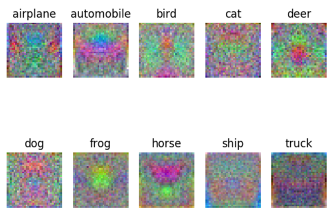
Best epoch: 5
Best cost: 1.7409
Final test accuracy:0.3944

4.2

lambda=0, n-epochs=40, n-batch=100, eta=0.01, decay factor=1



(c) Training &validation cost

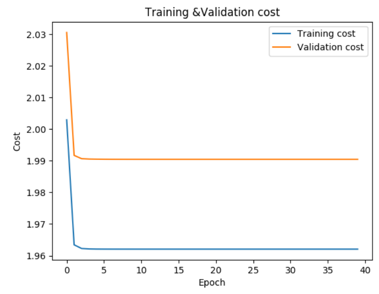


(d) Representation of weight matrix

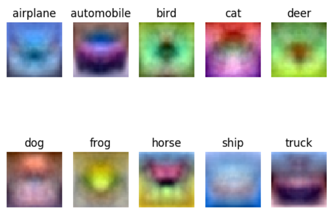
Best epoch: 39
Best cost: 1.7203
Final test accuracy:0.4196

4.3

lambda=.1, n-epochs=40, n-batch=100, eta=0.01, decay factor=1



(e) Training &validation cost

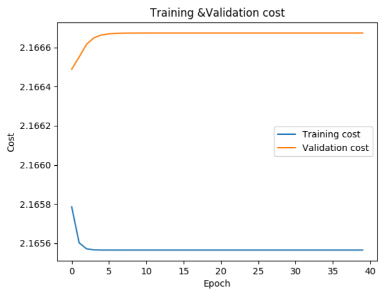


(f) Representation of weight matrix

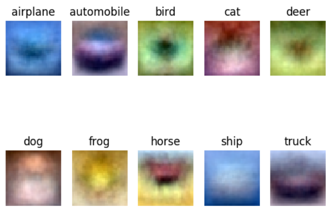
Best epoch: 17
Best cost: 1.9905
Final test accuracy:0.3710

4.4

lambda=1, n-epochs=40, n-batch=100, eta=0.01, decay factor=1



(g) Training &validation cost



(h) Representation of weight matrix

Best epoch: 1
Best cost: 2.16654
Final test accuracy:0.3061

5 Conclusions of Exercise1

4.1 and 4.2 indicate the importance of hyper-parameter, learning rate, which controls how much we are adjusting the weights of our network with respect the loss gradient. Apparently the performance of .01 is much better than that of .1 Using a low learning rate makes sure that no local minima is lost. Besides, the learning rate affects how quickly our model can converge to a local minima.

4.3 and 4.4 compares performance by taking different regularization terms. Increasing too much(up to1 in the exercise) will get bad performance on its accuracy. In some cases, however, regularization could get rid of overfitting problem.

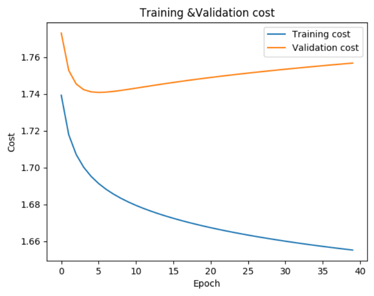
Basically, it's good experiment to understand and implement mini-batch gradient descent without importing other functional library like Tensorflow. But it can only get limited classification accuracy, approximately 30% to 40%.

6 Exercise2

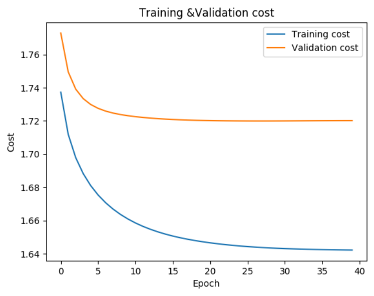
6.1 Optimize the performance of the network

Learning rate decay

Comparing to Exercise 4.1, decay factor was set to 0.9 to decay learning rate after each epoch. lambda=0, n-epochs=40, n-batch=100, eta=0.1



(i) Non-dacay

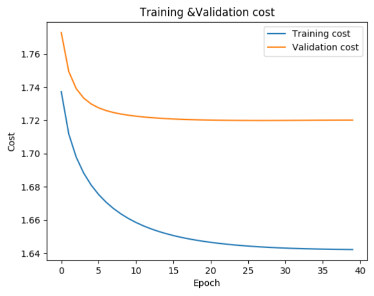


(j) Decay factor=0.9

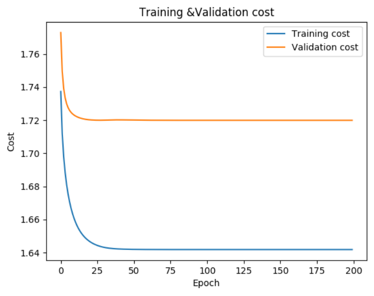
Decay factor	Final accuracy	Best cost
1(Non-decay)	0.3944	1.7409
0.9	0.4183	1.7199

k-fold cross validation

As is shown in the code, when loading the dataset I use all available training data for training and take a size-1000 validation set from all five batches. I also increase the epoch number to 200. lambda=0, n-batch=100, eta=0.1, decay factor=0.9



(k) 40 epoch



(l) 200 epoch

Epoch	Final accuracy	Best cost
40	0.4183	1.7199
200	0.4191	1.7199

With observation of each epoch ,validation loss is getting lower but in a fairly slow pace after 30 epochs.

Xavier initialization

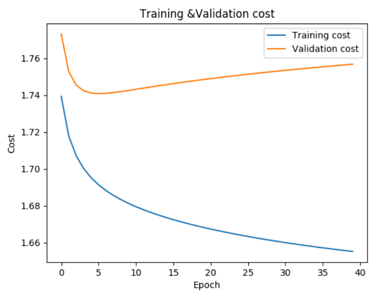
Randomly initializing weights could increase speed of network convergence and break symmetry problem. If all weights are the same(e.g. all 0 or 1 initialization), all units in hidden layer will be the same no matter what was the input. Here I take Xavier method, the variance of weights is:

$$\forall i, Var[W^i] = \frac{2}{n_i + n_{i+1}} \tag{3}$$

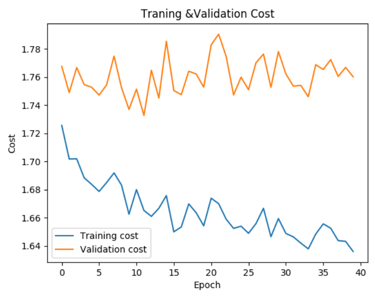
I use function (np.random.normal) to randomly generate Gaussian distribution (with mean:0 and variance:$\sqrt{\frac{2}{n}}$)in this exercise and receive a relatively good performance.

Shuffle training

Comparing to Exercise 4.1, I set shuffling step in the code to get better generalization. lambda=0, n-epochs=40, n-batch=100, eta=0.1, decay factor=1



(m) Non-shuffled



(n) Shuffled

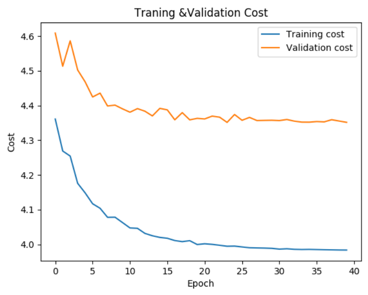
	Final accuracy	Best cost
Non-shuffled	0.3944	1.7409
Shuffled	0.4067	1.7326

6.2 SVM multi-class loss

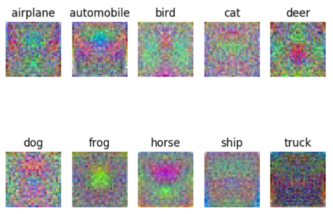
Train net by minizing SVM multi class loss instead of cross-entropy loss. SVM multi class loss has the form:

$$Given\ example(x_i, y_i), where\ x_i\ is\ image, y_i\ is\ label$$
$$L_i = \sum_{j \neq y_i} max(0, s_j - s_{y_i} + 1), where\ s = f(x_i, W) \tag{4}$$

Practically, it takes me much more time to compute SVM loss and gradient descent than that of cross entropy. Besides, it didn't receive better accuracy, as it might be influenced by settings of some sensible hyper parameters. The results with lambda=0, n-epochs=40, n-batch=100, eta=0.01, decay factor=0.9 as following:



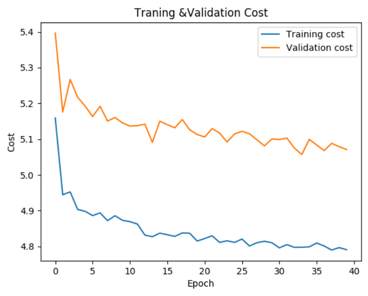
(o) Training &validation cost



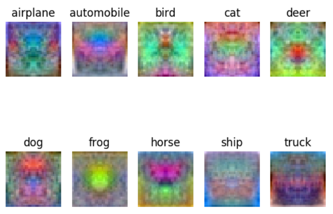
(p) Representation of weight matrix

Best cost: 4.3512 Final accuracy:0.401

Comparing to the same parameter setting in 4.2, the final accuracy decrease by approximately 2% from 0.4196 to 0.401 Another example with lambda=.1, n-epochs=40, n-batch=100, eta=0.01, decay factor=0.95 as following:



(q) Training &validation cost



(r) Representation of weight matrix

Best cost: 5.0568 Final accuracy:0.3962
Basically in exercise2,with same parameters, using cross entropy loss is better.