KTH Royal Institute of Technology
DD2424 Deep Learning in Data Science

# Assignment 2

Yage Hao (yage@kth.se)

21/05/2021

## 1 Introduction

In this assignment, we intend to classify images into multiple categories by constructing a two-layer neural network. Our dataset is CIFAR-10 which has already been split into training set, validation set and testing set. Each one includes data (observations) and labels (results).

## 2 Code Explanation

In general, we applied mini-batch gradient descent algorithm on the cross-entropy loss with L2 regularization term. We implemented the forward pass which evaluate the network and the backward pass which compute the gradients. We also exploiting cyclical learning rate during the process. Full detailed mathematical formulas are provided in the 'Background' part of the assignment document.

In detail, we first defined three function during the data preprocessing step. They are 'loadBatch', 'unpickle' and 'proprocess'.

Then, we defined a class named 'Classifier', in which we realized the neural network and the mini-batch gradient descent algorithm. In this class, we defined the following methods:

- initialization: initialize weight and bias parameters.
- relu: compute ReLU activation.
- softmax: compute softmax activation.
- evaluateClassifier: output the final classification store vector after processing the two layer network.
- computeCost: Compute cost function.

- computeAccuracy: Compute the accuracy of the network's predictions.
- computeGradient: Evaluate the gradients of the cost function w.r.t. W and b analytically.
- computeGradientNum: Numerically evaluate gradients.
- minibatchGD: Model training using mini-batch gradient descent with cyclical learning rate.

Specially, intend to compare the results obtained by function 'computeGradients' and 'computeGradientsNum', we defined a class named 'TestEqualityMethods' to check whether these methods give the same results in level of 4 decimal places.

Finally, we defined two more functions to call in the main function. One is the 'replicate' function, in which we train the network using one batch file and replicate figure 3 and 4 of the assignment document. The other one is the 'fit' function, in which we adjust parameters and train the real network on the whole dataset (5 batch files).

## 3 Results

### 3.1 Gradient check

The check is done by applying methods in class 'TestEqualityMethods'. We compared the analytically and numerically computed results of gradients with respective to weight and bias and assert that they provide the same values in four decimal places. When test into 5 decimal places, there will be 56 over 500 mismatched elements with max relative difference of 5.5%.

### 3.2 Replicate figures 3 and 4

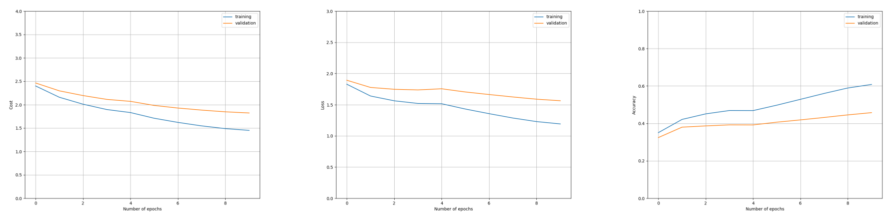

Figure 1: Replicated plots of Figure 3. From left are cost plot, loss plot and accuracy plot respectively. Training accuracy is 0.6083, validation accuracy is 0.4575, testing accuracy is 0.463.

In a single cycle, as epoch increasing, the cost and loss of both training set and validation set decrease, and the accuracy of both training set and validation set increase. It can also be noticed that in the cost and the loss plot, the validation curve is above the training curve, while in the accuracy plot, the validation curve is below the training curve, indicating out model is not overfitting.

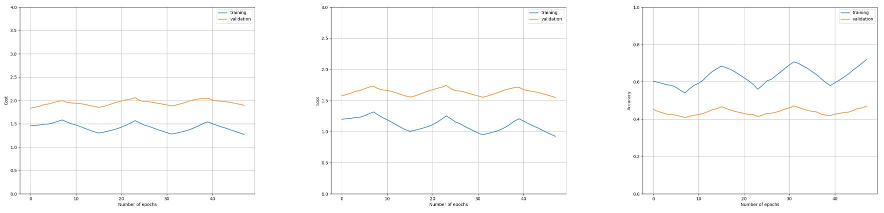

Figure 2: Replicated plots of Figure 4. From left are cost plot, loss plot and accuracy plot respectively. Training accuracy is 0.7194, validation accuracy is 0.4692, testing accuracy is 0.473.

In running three cycle with cyclical learning rate, we find our plots of cost, loss and accuracy show periodical performance with roughly three periods on every graph.

## 3.3 Coarse search

Setting: number of cycles = 2, batchsize = 100, eta_min = $10^{-5}$, eta_max = $10^{-1}$, n_s = 400, epochs = 16, 20 values of lambda are randomly sampled from the range $[10^{-5}, 10^{-1}]$. The 3 best performing networks have been bold in Table 1 based on the validation accuracy.

| lambda | training accuracy | validation accuracy | testing accuracy |
|---|---|---|---|
| 0.08054361389370582 | 0.3869183673469388 | 0.405 | 0.3918 |
| **0.006637540559630859** | **0.5351224489795918** | **0.528** | **0.5009** |
| 0.04317519755059854 | 0.4373061224489796 | 0.441 | 0.4375 |
| 0.015215986215306138 | 0.4985510204081633 | 0.494 | 0.479 |
| 0.08524250640871242 | 0.3832857142857143 | 0.396 | 0.3879 |
| 0.04817676962437736 | 0.43053061224489797 | 0.434 | 0.4308 |
| **0.004604602434093875** | **0.553530612244898** | **0.54** | **0.5086** |
| 0.07671346166478624 | 0.39132653061224487 | 0.408 | 0.3939 |
| 0.04817676962437736 | 0.4973673469387755 | 0.496 | 0.4873 |
| 0.009203651779726296 | 0.5251428571428571 | 0.521 | 0.4969 |
| 0.05822390085405982 | 0.4150408163265306 | 0.416 | 0.4179 |
| 0.09862125511511498 | 0.372469387755102 | 0.384 | 0.3755 |
| 0.0960209054005609 | 0.37359183673469387 | 0.386 | 0.3771 |
| 0.04362221547649208 | 0.4373877551020408 | 0.44 | 0.4383 |
| 0.006231786943298138 | 0.5384489795918367 | 0.516 | 0.5016 |
| 0.013585801779367307 | 0.507265306122449 | 0.51 | 0.4908 |
| 0.026701486384588247 | 0.471265306122449 | 0.473 | 0.465 |
| 0.09024890053798021 | 0.37842857142857145 | 0.393 | 0.3829 |
| **0.005388210881973321** | **0.5458571428571428** | **0.525** | **0.5038** |
| 0.09608865835627964 | 0.3745918367346939 | 0.388 | 0.378 |

Table 1: Coarse search for the regularization parameter lambda.

### 3.4 Fine search

Based on the results received from coarse search, we now limit our range of fine search in $[0.004, 0.007]$.

Setting: number of cycles = 2, batchsize = 100, eta_min = $10^{-5}$, eta_max = $10^{-1}$, n_s = 400, epochs = 16, 20 values of lambda are randomly sampled from the range $[0.004, 0.007]$. The 3 best performing networks have been bold in Table 1 based on the validation accuracy.

| lambda | training accuracy | validation accuracy | testing accuracy |
|---|---|---|---|
| 0.005450287416669631 | 0.5449387755102041 | 0.497 | 0.5028 |
| 0.005732844562318645 | 0.540795918367347 | 0.517 | 0.5063 |
| 0.005697279339168935 | 0.5406938775510204 | 0.512 | 0.5044 |
| 0.004057220289551563 | 0.5544285714285714 | 0.504 | 0.5059 |
| 0.006499487469139694 | 0.5378979591836734 | 0.508 | 0.5051 |
| 0.006893995746310631 | 0.5371428571428571 | 0.515 | 0.5007 |
| **0.004167196043587783** | **0.5560612244897959** | **0.543** | **0.5116** |
| 0.006341792216655227 | 0.5393673469387755 | 0.522 | 0.4995 |
| 0.00412204681661838 | 0.5519591836734694 | 0.511 | 0.5049 |
| 0.00698682502313402 | 0.5343877551020408 | 0.516 | 0.5001 |
| **0.004184693328057736** | **0.5520816326530612** | **0.534** | **0.5102** |
| 0.00490579584291298 | 0.5430204081632654 | 0.528 | 0.5066 |
| 0.006928710476732684 | 0.5323877551020408 | 0.522 | 0.4966 |
| 0.005817967723582414 | 0.5445510204081633 | 0.523 | 0.5043 |
| 0.004025281788201009 | 0.5541224489795918 | 0.52 | 0.5023 |
| 0.005065512694779805 | 0.5476530612244898 | 0.515 | 0.5022 |
| 0.0043834051086300225 | 0.5550612244897959 | 0.521 | 0.5039 |
| 0.005749169087504653 | 0.5368163265306123 | 0.506 | 0.5027 |
| 0.005441169766963644 | 0.5459387755102041 | 0.515 | 0.5031 |
| **0.005406041067504818** | **0.5446530612244898** | **0.535** | **0.5021** |

Table 2: Fine search for the regularization parameter lambda.

### 3.5 Best classifier

The best classifier was trained with the following parameter settings: number of cycles = 2, batchsize = 100, eta_min = $10^{-5}$, eta_max = $10^{-1}$, n_s = 400, epochs = 16, lambda = 0.004167.

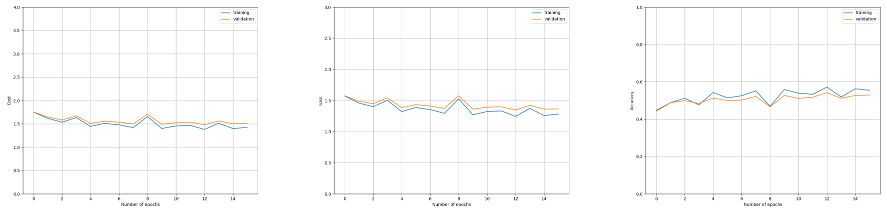

Figure 3: Plots of the best classifier. From left are cost plot, loss plot and accuracy plot respectively. Training accuracy is 0.5543877551020409, validation accuracy is 0.529, testing accuracy is 0.5069.