Report for Assignment 2

DD2424 Deep Learning in Data Science

Jiaying Yang
950826-9124

In this assignment, I deal with a problem to train, validate and test a two layer network with multiple outputs to classify images from the CIFAR-10 dataset. The whole assignment is realized in these steps: loading data, initialization, computing cost and gradient, realizing mini-batch, adding momentum and training the network.

1 Exercise1~4: Mandatory Part

During the whole process, there are several aspects to be analyzed:

i) State how you checked your analytic gradient computations and whether you think that your gradient computations were bug free. Give evidence for these conclusions.

I determine whether I have calculated the gradients correctly by comparing the gradients calculated by my function with the ones calculated by the give function "ComputeGradsNumSlow". My function is designed analytically according to the formulas given in the assignment guidance, while the given function "ComputeGradsNumSlow" computes the gradients numerically.

Their results are very similar. I calculate the biggest difference among all elements of the gradient between the two grad_W1, grad_W2 (the code is "max(max(ngrad_b-grad_b))") and the two grad_d1, grad_d2, and find out their values are 1.70e-6, 1.14e-10, 7.29e-6 and 4.12e-11 respectively, which are small enough for us to regard the results of two functions are the same.

ii) Comment on how much faster the momentum term made your training.

I compare the value loss with respective to each epoch in both two cases (with and without momentum). The result is shown in the image below:

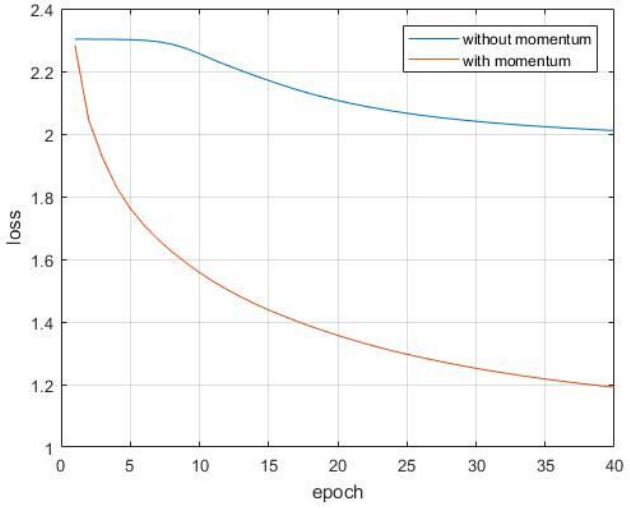

Figure 1. Comparition between loss with and without momentum

Here for both cases my parameter settings are: n_epochs = 40; rho=0.9; lambda=0; eta=0.01.
We can find out that with momentum, the speed of loss convergence is much more faster, which means the training is much faster..

iii) State the range of the values you searched for lambda and eta, the number of epochs used for training during the coarse search and the hyper-parameter settings for the 3 best performing networks you trained.

The range of lambda I set is from 10^-6 to 10^-1, and the range of eta I set is from 10^-4 to 10^1. The number of epochs used is 20. The number of iterations for generating random parameters is set to 100. The best three performing networks and their hyper-parameter settings are:

	lambda	eta	Validation Accuracy
1	0.00483(10^-2.316)	0.0430(10^-1.367)	44.66%
2	0.00229(10^-2.641)	0.0203(10^-1.692)	44.35%
3	0.0000198(10^-4.704)	0.0178(10^-1.750)	44.04%

Table 1. The three best validation accuracy and parameter settings during coarse search

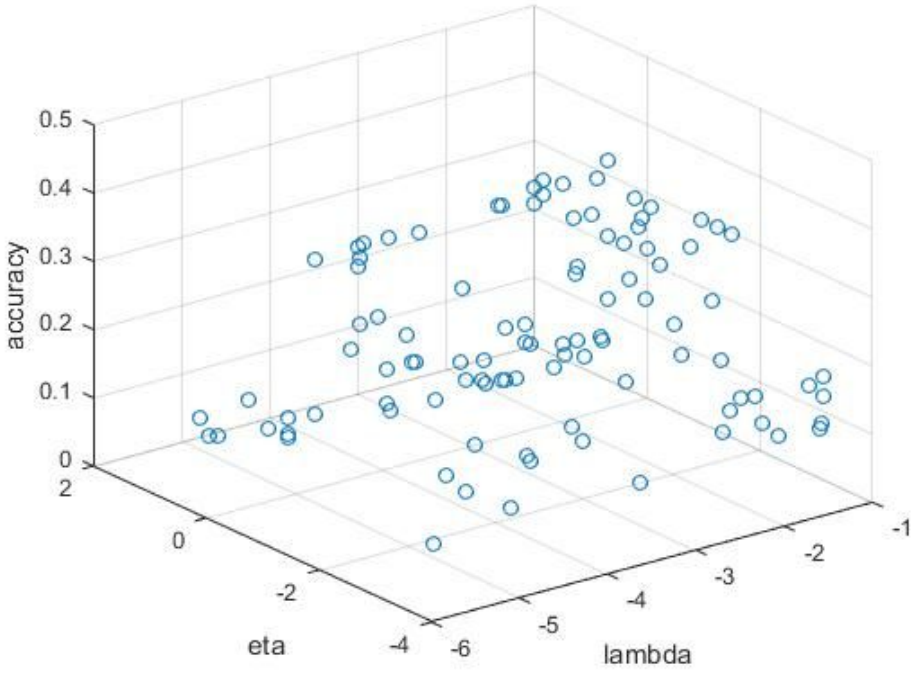

Figure 2. Validation value with respective to lambda and eta after coarse search

iv) State the range of the values you searched for lambda and eta, the number of epochs used for training during the fine search, and the hyper-parameter settings for the 3 best performing networks you trained.

The range of lambda I set is from 10^-6 to 10^-1, and the range of eta I set is from 10^-2.5 to 10^0.5. The number of epochs used is 30. The number of iterations for generating random parameters is set to 100. The best three performing networks and their hyper-parameter settings are:

	lambda	eta	Validation Accuracy
1	0.003475(10^-2.459)	0.04529(10^-1.344)	46.10%
2	0.00331(10^-2.480)	0.0364(10^-1.439)	45.71%
3	0.00285(10^-2.545)	0.0282(10^-1.550)	45.38%

Table 2. The three best validation accuracy and parameter settings during finesearch

v) For your best found hyper-parameter setting (according to performance on the validation set), train the network on all the training data (all the batch data), except for 1000 examples in a validation set, for ~30 epochs. Plot the training and validation cost after each epoch of training and then report the learnt network's performance on the test data.

Figure 3 is generated when using all available data and setting parameters according to the best result in my fine search, which is lambda=0.003475, and eta=0.04529. Table 3 refers to the accuracy of the system.

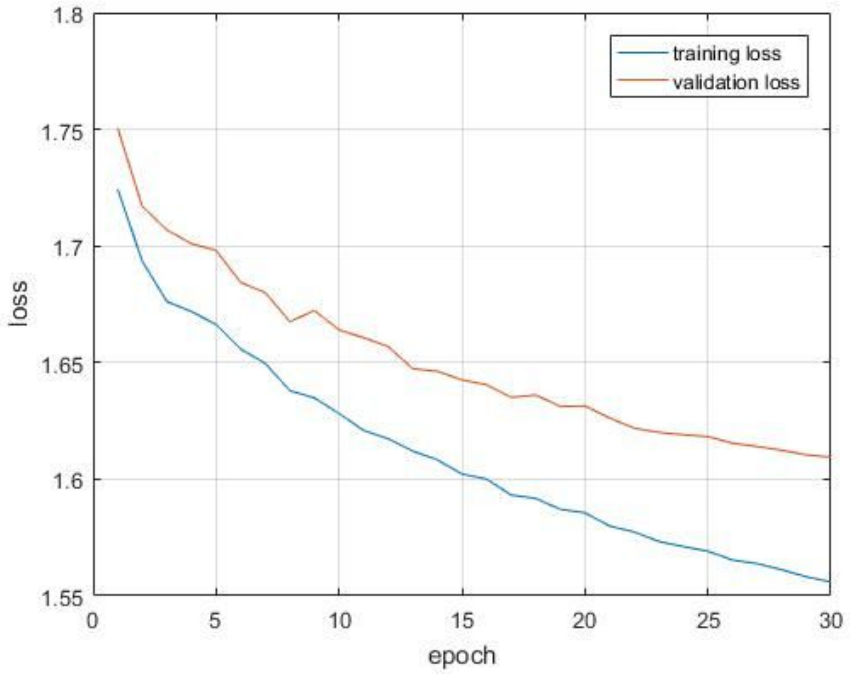

Figure 3. **Training and validation cost after each epoch**

Training Accuracy	Validation Accuracy	Test Accuracy
53.33%	49.50%	50.81%

Table 3. The **performance on the test data**

2 Exercise 5.1: Optimize the Performance of the Network

1. Results

	Training Accuracy	Validation Accuracy	Test Accuracy
Before Improvement	53.33%	49.50%	50.81%
After Improvement	57.18%	53.20%	53.02%

Table 4. 3 kinds of accuracy before and after improvement

he test accuracy increased from 50.81% to 53.02% after all the improvement methods, which is a 5.66% increase with respect to the original accuracy.

2. Improvements chosen

1) ***Use all the available data from training, train for more update steps and use your validation set to make sure you don't overfit or keep a record of the best model before you begin to overfit***
   All of the 5 data batch are used. The first four data batch and the first 9000 data of the 5th data batch, which in total is 49000 data, is used as training data. The last 1000 data of the 5th data batch is used as validation data.

2) ***Switch to He initialization and see the effect it has on training.***
   This is an two layer network with nodes. Thus according to the He Initialization, the weights are initialized by multiplying $\sqrt{\frac{2}{n}}$. n is the number of nodes, whose value is 210, the sum of 200 (nodes in the middle layer) and 10 (nodes in the output layer).

3) ***Do a more exhaustive random search to find good values for the amount of regularization and the learning rate.***
   After the fine research, I do a more exhaustive research on parameter settings. This time I set the range of lambda I set is from 10^-3 to 10^-2, and the range of eta I set is from 10^-2 to 10^-1, and then set the number of epoches into a bigger value, which is 40. I also set the number of iterations into 100. The best result of this research is when lambda=0.003758(10^-2.425), eta=0.0899(10^-1.0463).

4) ***Play around with different approaches to anneal the learning rate. For example you can keep the learning rate fixed over several epochs then decay it by a factor of 10 after n epochs.***
   I finished this step by decaying the learning rate by a factor of 10 for every 10 epochs, and the whole process lasts 100 epoches.

5) ***Apply dropout to your training if you have a high number of hidden nodes and you feel you need more regularization.***
   Here I set the number of hidden nodes to be 200 instead of using the previous value m=50.

3. Analysis

I think that the 3nd improvement I chosen, which is to do a random search on the parameter settings, helps improve the accuracy best. This is because it is the third time for me to do the random research, and the range of parameter generating is set to be relatively small, which is much more likely to find better settings. Also, parameters chosen randomly are more efficient than parameters chosen by grid search.

3 Exercise 5.2: Train network using a different activation to ReLu

The different activation I choose is the Leaky ReLu, which follows the equation: Leaky ReLu(x) = max(.01x, x). For both of the two cases, I set the parameters to the same value (lambda=0.003475, eta=0.04529, rho=0.9, n_epoch=100). Both use all the data available, and also use the improvements 1~4 mentioned in 5.1. The result is shown in below Table 5. We can find out that the performance of Leaky ReLu isa little better than that of the ReLu, almost the same.

	Training Accuracy	**Validation Accuracy**	**Test Accuracy**
ReLu	54.49%	52.30%	51.75%
Leaky ReLu	54.64%	52.30%	51.86%

Table 5. The comparison of accuracy when using ReLu and Leaky ReLu