Course: DD2424- Assignment 2

Sevket Melih Zenciroglu – smzen@kth.se

Question-1: The code for your assignment assembled into one file.

Answer-1: Find it in the end of the document.

Question-2.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.

Answer-2.i: We calculated the gradients numerically as it was suggested in the assignment and compared these values with the ones I derived analytically. The MEAN of the differences for W and b values were smaller than 1e-11 which means that the error is very small. According to the reference given from Standford, having error values smaller than 1e-7 should make us happy.

Parameter	Value	Description
n_epochs	200	number of times we iterate on the entire data
batch_size	2	the size of the mini batch. in other words, number of images in 1 mini-batch. (in this specific example, we only used 2 images - it was suggested to use 1 image in the assignment)
eta	0.001	learning rate (step-size)
lambda_cost	0	regularization coefficient (punishment)
d	3072	dimension of X_train (input)... 3072 = 32 x 32 x 3 (20 suggested for this exercise but since the calculations are fast enough, we used all dimensions)
m	50	number of nodes in the hidden layer
h	1e-5	Precision value

Table-1: Parameters used

Another reason for not using dimension as 20 was the below results. Somehow, the small dimension couldn't help us to get the results that we are after:
d = 20, N=100 >> Cost from 2.54 to 2.38 >> Accuracy from 0.08 to 0.15 >> time: 0.15 seconds
d = 3072, N=100 >> Cost from 2.439 to 1.199 >> Accuracy from 0.15 to 0.74 >> time: 1.23 seconds
d = 3072, N=10000 >> Cost from 2.347 to 1.323 >> Accuracy from 0.1906 to 0.543 >> time: 3.08 minutes
N: Number of images used to train

	abs_MEAN (grad_numerical – grad_analytic)
grad_W1, grad_W1_num	7.448042807619361e-12
grad_W2, grad_W2_num	7.532225861799947e-12
grad_b1, grad_b1_num	8.697536371671256e-12
grad_b2, grad_b2_num	1.3584279534573085e-11

Table-2: Mean of errors

Moreover, we have done the Gradient Sanity check:

We were able to achieve an overfitting with a very small loss (J_epochs_train) (`1.12059233`) and a very high accuracy (`0.82`) on the training data.

Only 1 batch (100 images) is used:

Parameter	Value
n_epochs	200
batch_size	100
eta	0.001
lambda_cost	0
d	3072
m	50

Table-3: Parameters used

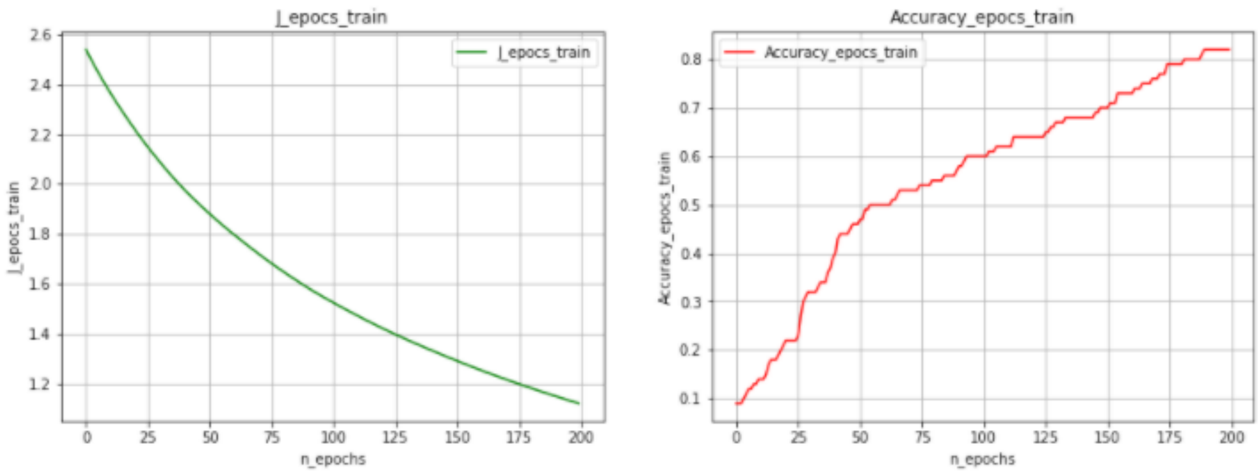


Figure-1: Overfitting

Question-2.ii: The curves for the training and validation loss/cost when using the cyclical learning rates with the default values, that is replicate figures 3 and 4. Also comment on the curves.

Answer-2.ii: We trained our algorithm to evaluate the cost with 3 different methods

a) the entire training data:
   all 10.000 images were used in 'Dataset/data_batch_1' for **training** and
   all 10.000 images in 'Dataset/data_batch_2' were used for **validation** to evaluate the cost

b) batch aggregation: Basically, this is kind of the average of batch costs.
   Each time, a cost was calculated per batch (100 images). Then, this was added to the previous cost calculated. Finally, the sum was divided by the number of batches added until now. For a better understanding, you might check the function: Train_Cyclical

   J_train = network1.Cost(X_batch, Y_batch, W, b, lambda_cost)
   J_train_sum += J_train
   smooth_cost = J_train_sum/(cost_record + 1)

c) batch aggregation ratio: Basically, this will add the new batch's cost to the previous cost with a ratio. For a better understanding, you might check the function: Train_Cyclical

```
J_train = network1.Cost(X_batch, Y_batch, W, b, lambda_cost)
if n_records == 0:
  smooth_cost = J_train
else:
  #smooth_cost = 0.999*smooth_cost + 0.001 * J_train
  smooth_cost = 0.99*smooth_cost + 0.01 * J_train
```

Cost Method	batch_size	n_cycles	lambda_cost	Record Per cycle	m	eta_min	eta_max	n_steps	Test Accuracy	Calculation Time
ALL_Data	100	1	0.01	100	50	1e-5	1e-1	500	0.454	2 min 39 sec
Batch_aggregated	100	1	0.01	100	50	1e-5	1e-1	500	0.4612	1 min 23 sec
Batch_aggregated_ratio	100	1	0.01	100	50	1e-5	1e-1	500	0.4526	1 min 25 sec
ALL_Data	100	3	0.01	100	50	1e-5	1e-1	800	0.4599	7 min 59 sec
Batch_aggregated	100	3	0.01	100	50	1e-5	1e-1	800	0.4641	4 min 14 sec
Batch_aggregated_ratio	100	3	0.01	100	50	1e-5	1e-1	800	0.4636	4 min 15 sec
Batch_aggregated	100	3	0.01	100	100	1e-5	1e-1	800	0.4763	4 min 47 sec

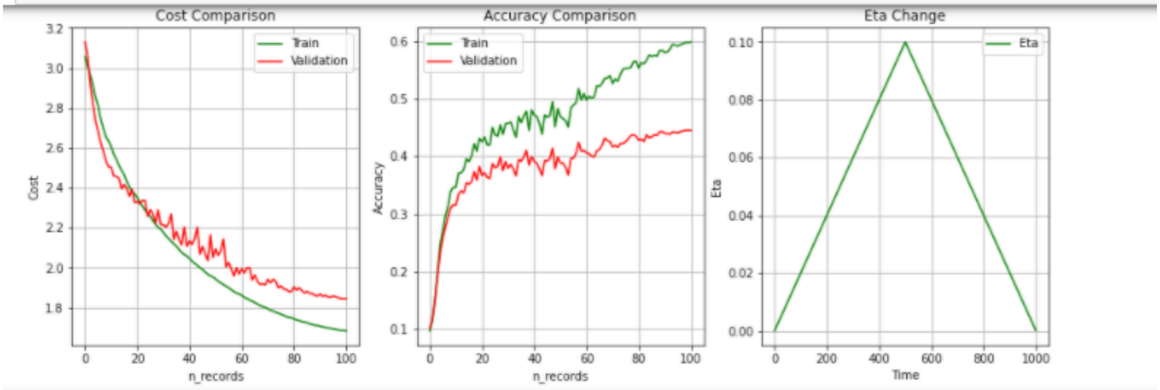
Table-4: Method and Parameters used

n_epochs = int(2 * n_cycles * (n_steps / total_batch))
d = 3072 for all
Validation data = all data in 'Dataset/data_batch_2'
'Dataset/test_batch' was used to calculate the test data set's accuracy.

Above methods were tested because once we moved to the next exercise, the calculation was taking too much time. So, instead of using the entire training data to make the cost calculations, batch-based calculations were considered. It was observed that the calculation time reduced while the calculated values were more or less the same in each 3 methods used. That was kind of a validation of the methods used. So, for the rest of the assignment, "Batch_aggregated" method is decided to be utilized to save time.

***NOTE: The x-axis (n_records) shows how many times we calculated the cost, it does not represent the number of steps. To be able to know the number of steps, you need to multiply "n_records" by the "record_per_cycle" which is usually picked as 100.

```
#### Exercise - 3 ###
# takes around 1.5 minutes
# TRAIN = VALIDATION = TEST = 10.000
# Train_Cyclical(network1, train_X_Norm, validation_X_Norm, n_cycles, n_steps)
# param_list = [network1, train_X_Norm, validation_X_Norm, batch_size, n_cycles, lambda_cost, record_per_cycle, m,
#               eta_min, eta_max, n_steps]
param_list = [network1, train_X_Norm, validation_X_Norm, 100, 1, 0.01, 100, 50, 1e-5, 1e-1, 500]
layers1, W1, b1, eta_train = Train_Cyclical(param_list, 'Batch_aggregated')
```

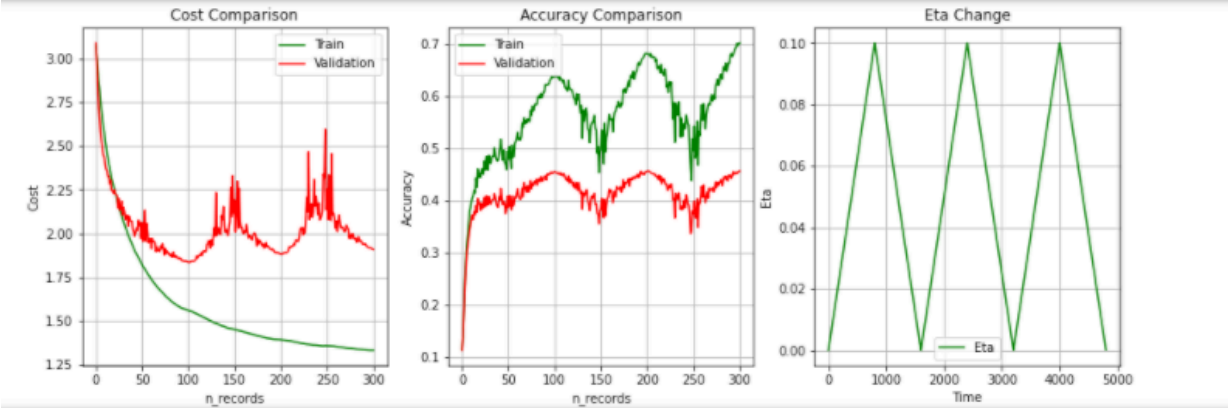



```
P_test, H_test = network1.EvaluationClassifier(layers1, test_X_Norm, W1, b1)
k_test = np.argmax(P_test, axis=0)
A_test = network1.ComputeAccuracy(k_test, network1.test_y)
print(A_test)
```

```
0.4612
```

Figure-2: Cost comparison and accuracy change for 1 cycle

```
# param_list = [network1, train_X_Norm, validation_X_Norm, batch_size, n_cycles, lambda_cost, record_per_cycle, m,
#               eta_min, eta_max, n_steps]
param_list = [network1, train_X_Norm, validation_X_Norm, 100, 3, 0.01, 100, 50, 1e-5, 1e-1, 800]
layers2, W2, b2, eta_train = Train_Cyclical(param_list, 'Batch_aggregated')
```

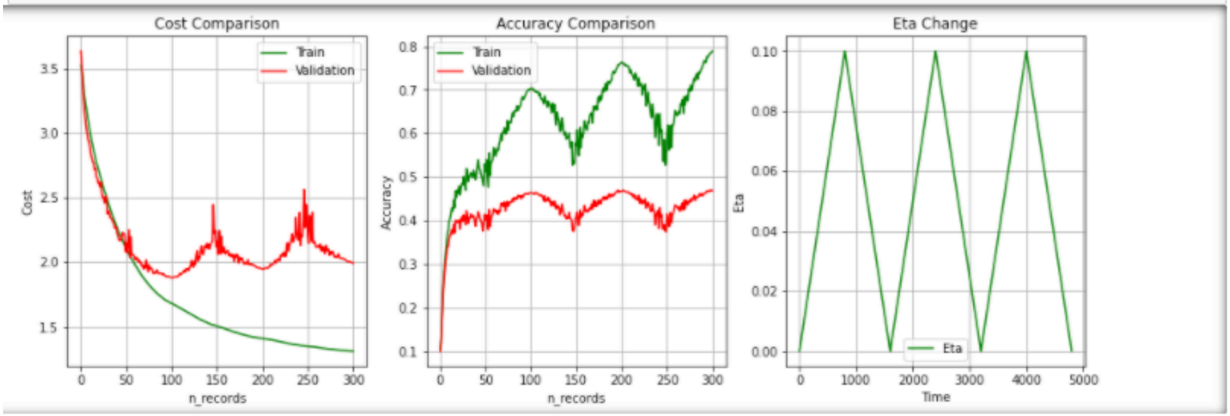



```
P_test, H_test = network1.EvaluationClassifier(layers2, test_X_Norm, W2, b2)
k_test = np.argmax(P_test, axis=0)
A_test = network1.ComputeAccuracy(k_test, network1.test_y)
print(A_test)
```

```
0.4641
```

Figure-3: Cost comparison and accuracy change for 3 cycles

```
# param_list = [network1, train_X_Norm, validation_X_Norm, batch_size, n_cycles, lambda_cost, record_per_cycle, m,
#               eta_min, eta_max, n_steps]
param_list = [network1, train_X_Norm, validation_X_Norm, 100, 3, 0.01, 100, 100, 1e-5, 1e-1, 800]
layers2, W2, b2, eta_train = Train_Cyclical(param_list, 'Batch_aggregated')
```




```
P_test, H_test = network1.EvaluationClassifier(layers2, test_X_Norm, W2, b2)
k_test = np.argmax(P_test, axis=0)
A_test = network1.ComputeAccuracy(k_test, network1.test_y)
print(A_test)
```

```
0.4763
```

Figure-4: Cost comparison and accuracy change for 3 cycles & 100 hidden layers

*** NOTE: The line for Training cost is not representing the 3 cycles properly since the calculation is done in an aggregated way but not using the ALL_Data each time. If we use ALL_Data as in the below figure (Figure-5), the cycles are easily observed:

```
# param_list = [network1, train_X_Norm, validation_X_Norm, batch_size, n_cycles, lambda_cost, record_per_cycle, m,
#               eta_min, eta_max, n_steps]
param_list = [network1, train_X_Norm, validation_X_Norm, 100, 3, 0.01, 100, 50, 1e-5, 1e-1, 800]
layers2, W2, b2, eta_train = Train_Cyclical(param_list, 'ALL_Data')
```

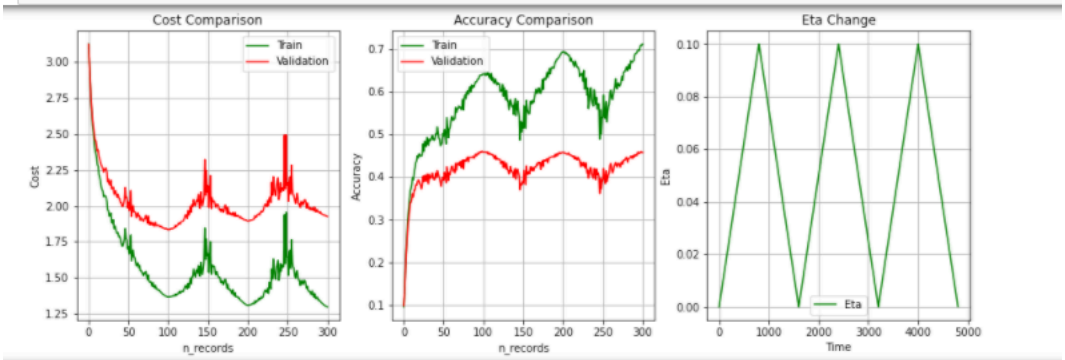


Figure-5: Cost comparison and accuracy change for 3 cycles & 50 hidden layers by using ALL_Data for cost calculations

Adding more cycles has a positive impact on the accuracy & cost for training but it doesn't have a big impact on the validation and test data. The impact was more significant once we increased the number of nodes in the hidden layer in comparison to increasing the number of cycles.

Question-2.iii: State the range of the values you searched for lambda, the number of cycles used for training during the coarse search and the hyper-parameter settings for the 3 best performing networks you trained.

Answer-2.iii: Starting from this question, 45.000 images are used for **training**, 5.000 for **validation** and 10.000 for **testing**.

As it was suggested in the assignment, the test accuracy was tested by searching for lambda on a log scale by generating one random sample in the range of 1e-5 and 1e-1 (10^l_min to 10^l_max):

l = l_min + (l_max - l_min) * np.random.uniform(0,1)
lambda_coarse = pow(10, l)

lambda_cost	Test Accuracy	batch_size	n_cycles	Record Per cycle	m	eta_min	eta_max
0.003772863865559457	0.499	100	1	100	50	1e-5	1e-1
0.0004930704361181606	0.4996	100	1	100	50	1e-5	1e-1
0.0033665965785468826	0.4975	100	1	100	50	1e-5	1e-1
0.001305094636607304	0.5022	100	1	100	50	1e-5	1e-1
0.01757893078506135	0.4828	100	1	100	50	1e-5	1e-1
0.0007012053067447107	0.499	100	1	100	50	1e-5	1e-1
9.177497906795191e-05	0.502	100	1	100	50	1e-5	1e-1
0.000310949509136683	0.4969	100	1	100	50	1e-5	1e-1

Table-5: Parameters used for the coarse search

The value of lambda starting from 1e-3 to 1e-5 have better test accuracy values. Besides those numbers above, more tests were conducted, and we decided to narrow the search down between 1e-4 to 1e-5 in the next step.

Question-2.iv: State the range of the values you searched for lambda, the number of cycles used for training during the fine search and the hyper-parameter settings for the 3 best performing networks you trained.

Answer-2.iv: Here, you will have the results for the narrowed down lambda values. This time we executed for 16 lambda values:

lambda_cost	Test Accuracy	batch_size	n_cycles	Record Per cycle	m	eta_min	eta_max
9.862809522185001e-05	0.498	100	1	100	50	1e-5	1e-1
9.054812798313612e-05	00.5001	100	1	100	50	1e-5	1e-1
1.9651064567590842e-05	0.4968	100	1	100	50	1e-5	1e-1
3.544673306817798e-05	**0.5056**	**100**	**1**	**100**	**50**	**1e-5**	**1e-1**
2.653016524046826e-05	0.4986	100	1	100	50	1e-5	1e-1
2.055764159216546e-05	0.4959	100	1	100	50	1e-5	1e-1

3.614606936334905e-05	0.5019	100	1	100	50	1e-5	1e-1
2.6196012000341928e-05	0.5001	100	1	100	50	1e-5	1e-1
2.406038940258964e-05	0.4992	100	1	100	50	1e-5	1e-1
5.9831933161984584e-05	0.495	100	1	100	50	1e-5	1e-1
3.516860763871908e-05	0.4995	100	1	100	50	1e-5	1e-1
4.459893452752506e-05	0.4944	100	1	100	50	1e-5	1e-1
1.1067150412880694e-05	0.5033	100	1	100	50	1e-5	1e-1
1.4001893917970237e-05	0.4947	100	1	100	50	1e-5	1e-1
1.2967366157634313e-05	0.4972	100	1	100	50	1e-5	1e-1
2.7575325697602636e-05	0.5029	100	1	100	50	1e-5	1e-1

Table-6: Parameters used for the fine search

Question-2.v:

For your best found **lambda** 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 ~3 cycles. Plot the training and validation loss plots and then report the learnt network's performance on the test data.

Answer-2.v:

3.544673306817798e-05 gave us the best accuracy result in the fine search, so we will use it as lambda. Once we used higher number of nodes in the hidden layer (m=200), we achieved a higher accuracy of 0.5336:

The best parameter setting:

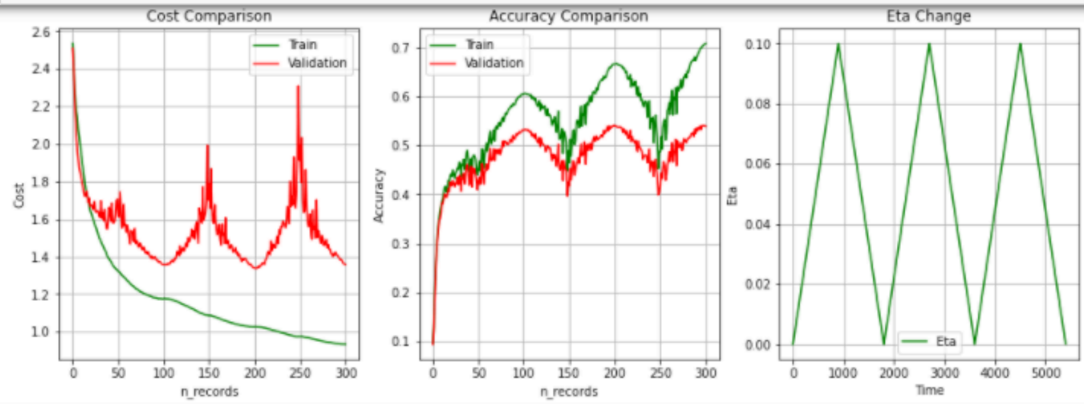
lambda_cost	Test Accuracy	batch_ size	n_cycles	Record Per cycle	m	eta_min	eta_max
3.544673306817798e-05	**0.5336**	200	3	100	200	1e-5	1e-1

Results for some other settings changing the number of nodes in the hidden layer:

lambda_cost	Test Accuracy	batch_ size	n_cycles	Record Per cycle	m	eta_min	eta_max
3.544673306817798e-05	0.5186	100	3	100	200	1e-5	1e-1
3.544673306817798e-05	0.5051	50	3	100	200	1e-5	1e-1
3.544673306817798e-05	0.4141	10	3	100	200	1e-5	1e-1

```
lambda_coarse = 3.544673306817798e-05  #1.8544671883635666e-05
# N_CYCLEs changed >> from 1 to 3
# param_list = [network1, train_X_Norm, validation_X_Norm, batch_size, n_cycles, lambda_cost, record_per_cycle, m,
#               eta_min, eta_max]

param_list = [network1, train_X_Norm, validation_X_Norm, 100, 3, lambda_coarse, 100, 200, 1e-5, 1e-1]
layers, W, b, eta_train = Train_Cyclical_Coarse_lambda_smooth_aggregated_2(param_list)
```




```
P_test, H_test = network1.EvaluationClassifier(layers, test_X_Norm, W, b)
k_test = np.argmax(P_test, axis=0)
A_test = network1.ComputeAccuracy(k_test, network1.test_y)
print(A_test)
```

```
0.5336
```