Deep Learning in Data Science

Assignment 1

Isak Gamnes Sneltvedt

April 4th, 2022

DD2424

Assignment 2

Gradient comparison

For the gradient comparison I checked the absolute differences between the numerical and analytical computed vectors. This method check that the relative difference between the numerical and the analytical is less than a certain threshold (10^{-5} in my case).

The output from the terminal after the comparison is as follows:

```
Output:
        Grads test on W1 returned: True, with nominator: 4.724876676739617
   e-05 and denominator: 47.678748314893035
        Grads test on W2 returned: True, with nominator: 4.779799141510413
   e-07 and denominator: 2.925975628886696
        Grads test on b1 returned: True, with nominator:
   2.2119807889667623e-06 and denominator: 1.970798722821977
        Grads test on b2 returned: True, with nominator: 4.499112193947965
   e-06 and denominator: 0.8364133132592686
        9.90981693884725e-07
        1.6335744885643762e-07
        1.1223778275030734e-06
        5.3790537795437325e-06
```

This shows both the nominator and denominator. The nominator is calculated from the absolute different between the two gradients. The denominator is calculated from the absolute sum of the two gradients. The reason why I use the denominator is because it will yield the *relative error* rather than just the error. This is important as an error of, for instance, 0.0005 is considered small if we have a gradient of 3. Yet, if we have a gradient of 0.000001, then the error of 0.0005 is considered huge and we are not able to use it. So by using the *relative error*, we find the error in regards to the total gradient. The two gradient checks that I chose arbitrary earlier will now be: $\frac{0.0005}{3}$ which is a very small number, and $\frac{0.0005}{0.000001}$, which is a very large number. Based on this, I believe my gradient check and calculation to be correct.

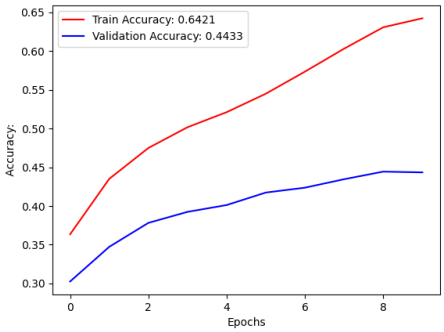

Figure 1: Accuracy with CLR

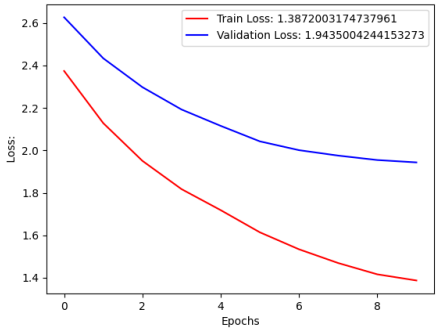

Figure 2: Loss with CLR

CLR

As can be seen on the plots, the model is being a bit over fitted. This can be seen by the accuracy. The training yields 20% higher accuracy compared to the validation set. Also on the loss we are able to see that the loss on the training data is lower than the loss on the validation data. Figure 3 was added to show correct implementation of the CLR.

Coarse lambda search

For the coarse search for a lambda value I used the recommended numbers given in the assignment. I generated a total of 10 random numbers that was used to find a good search area for a lambda value.

```
    l_min = -5
    l_max = -1
    l = l_min + (l_max-l_min)*np.random.rand(10)
```

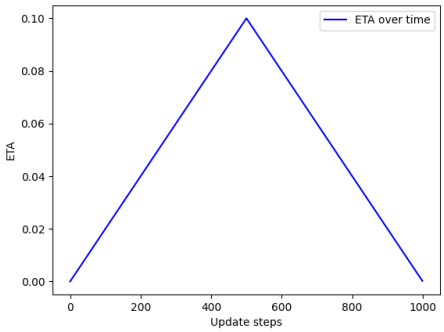

Figure 3: The change of ETA in one cycle

```
4     list_of_lambdas = 10**l
```

np.random.rand(10) will generate a list of 10 elements ranging from 0-1. From the expression ($l_{max} - l_{min}$ we get -4. The extreme cases are when the random generated number is 0 or 1. When the number is 0 we will end up with $\lambda = 10^{-1}$. Whereas if the random generated number is 1 we get that $\lambda = 10^{-5}$. This implies that the initial search range for my lambda values ranges from 10^{-1} to 10^{-5}.

During the lambda search I used the hyper parameters that was presented during the assignment. I.e. n_s = 800, batch size = 100, eta_min = 1e-5, eta_max = 1e-1 and two cycles.

Lambda	Accuracy
0.000044	0.4486
0.046993	0.4536
0.027695	0.4654
0.000055	0.4692
0.000020	0.4774
0.000697	0.4810
0.000594	0.4850
0.000121	0.4914
0.000253	0.4932
0.003422	0.4944

Table 1: Accuracy for each lambda value sorted after accuracy.

From the initial search I found that $\lambda = 0.003422$ gave the best results.

Fine search

For the fine search I ended up running a total of 3 searches. The first one had the lambda values of:

```
list_of_lambdas = [0.002, 0.0025, 0.003, 0.0035, 0.004, 0.0045, 0.005]
```

The results from this search was:

Lambda	Accuracy
0.0020	0.5160
0.0025	0.5192
0.0030	0.5250
0.0040	0.5250
0.0035	0.5262
0.0045	0.5272
0.0050	0.5328

As the results seemed to increase with an increasing lambda, I decided to increase the values in my search. The next lambda values were therefore:

```
list_of_lambdas = [0.0045, 0.00475, 0.005, 0.00525, 0.0055, 0.00575,
   0.006, 0.00625]
```

The results from this search were:

Again I ended up with the best performance being the highest lambda value. Therefore I increased the values once again to find the area where the highest accuracy was in the region of my lambda values.

Lambda	Accuracy
0.00475	0.5250
0.00450	0.5254
0.00500	0.5262
0.00575	0.5268
0.00525	0.5276
0.00550	0.5280
0.00600	0.5332
0.00625	0.5338

```
list_of_lambdas = [0.00625, 0.0065, 0.00675, 0.007, 0.00725, 0.0075,
    0.00775, 0.008]
```

From this test I got the following results:

Lambda	Accuracy
0.00625	0.5186
0.00650	0.5228
0.00675	0.5272
0.00750	0.5274
0.00775	0.5278
0.00700	0.5284
0.00800	0.5308
0.00725	0.5320

After performing all of these tests I realized that the accuracies were very similar, as well as changing from each training. This is, among other things, due to the random values in the weights and biases and I the concluded that the best value for lambda was around 0.006, as this gave quite good results with a accuracy consistently above 50%

Final test

After running the final test for a total of 48 epochs (3 cycles) I got eh following plots:

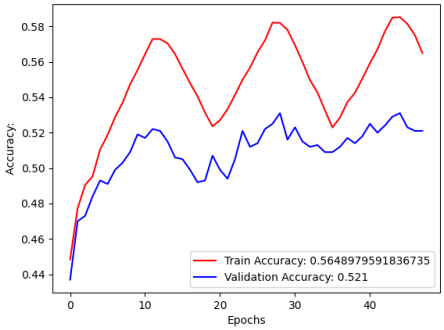

Figure 4: Accuracy in the final test

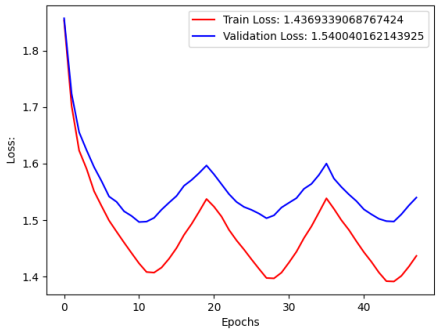

Figure 5: Loss in the final test

The test accuracy came out to be:

```
The test accuracy from the final test is: 0.5055 (50.55\%)
```