Assignment 2 in DD2424
Deep Learning in Data Science

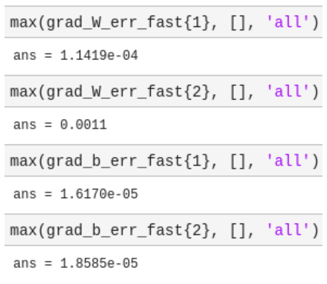
Marie-Ange Stefanos

april 2022

Question i) - Exercises 1 & 2

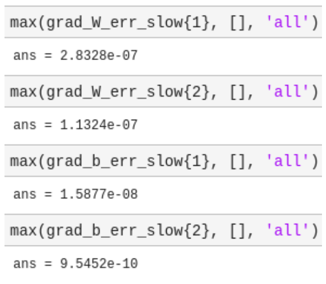
In order to check that my gradient computations are correct, I compared my analytic gradient results to the numerical gradients using the relative error formula given in the Standford course mentioned in first assignment[1], because dividing by a low ϵ as asked in the instructions gives very high errors. That is why I have chosen to work with an expression of the error that does not depend on an external parameter. Moreover, this course indicates how low you should expect your error to be in practice: for 1e-7 and lower errors, you can be happy of your implementation.

As can be seen on figure 1a, the errors obtained on W_1, W_2, b_1 and b_2 are all small (from 1e-5 to 1e-2) but still higher than the recommended value, when using the numerical fast and less accurate method (finite difference). However, this can be explained by the fact that the error between both given numerical gradients is of the same order of magnitude, as shows figure 1c. Nevertheless, I can consider that my gradient computations are correct because the comparison between my analytical gradient and the numerical method that gives slower but more accurate results gives way smaller and more acceptable results (see figure 1b), below the recommended threshold of 1e-7.



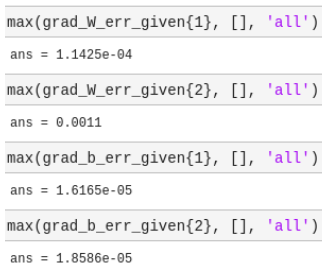
```
max(grad_W_err_fast{1}, [], 'all')
ans = 1.1419e-04
max(grad_W_err_fast{2}, [], 'all')
ans = 0.0011
max(grad_b_err_fast{1}, [], 'all')
ans = 1.6170e-05
max(grad_b_err_fast{2}, [], 'all')
ans = 1.8585e-05
```

(a) Comparison between analytical and finite difference (numerical) method



```
max(grad_W_err_slow{1}, [], 'all')
ans = 2.8328e-07
max(grad_W_err_slow{2}, [], 'all')
ans = 1.1324e-07
max(grad_b_err_slow{1}, [], 'all')
ans = 1.5877e-08
max(grad_b_err_slow{2}, [], 'all')
ans = 9.5452e-10
```

(b) Comparison between analytical and centered difference (numerical) method



```
max(grad_W_err_given{1}, [], 'all')
ans = 1.1425e-04
max(grad_W_err_given{2}, [], 'all')
ans = 0.0011
max(grad_b_err_given{1}, [], 'all')
ans = 1.6165e-05
max(grad_b_err_given{2}, [], 'all')
ans = 1.8586e-05
```

(c) Comparison between both given numerical gradients (fast vs. accurate)

Figure 1: Three gradients comparisons using relative error

Checking that my mini-batch gradient descent algorithm has been done by comparing the evolution of the cost, the loss and the accuracy through the epochs to the results given in the instructions. These results can be seen on figure 2 and show a clear overfitting since the loss and the cost on the training set are way smaller than for the validation set. The accuracy is also way higher for the training set than for the validation set.

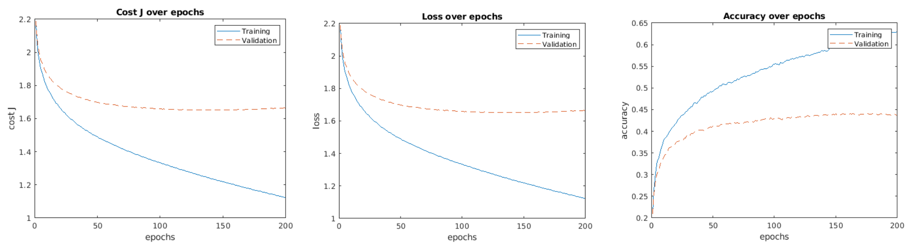


Figure 2: Training data overfitting illustration by subsampling 10 points for each epoch (all batch 1 used for training set with settings $\lambda = 0$, n_s = 200, $\eta = 0.001$)

[1]Convolutional Neural Networks for Visual Recognition

Question ii) - Exercises 3 & beginning of 4

This section is about training your network with cyclical learning rates in order to speed up training times. To check that I have a bug free implementation of the cyclic scheduled learning rate, I checked that it is working by replicating figures 3 and 4 of the instructions. All curves have been subsampled (10 points per epoch).

Replication of figure 3

Settings value:

- number of cycle: nb_cycles = 1
- eta_min = 1e-5
- eta_max = 1e-1
- stepsize: n_s = 500
- batch size: n_batch = 100
- $\lambda = 0.01$
- update step: t $\in [\![1, 2n\_s]\!] = [\![1, 1000]\!]$

Figure 3 shows the learning characteristics, meaning the evolution of the cost J, the loss and the accuracy through the epochs for both the training set (blue curves) and the validation set (orange curves). As shows figure 4, the training has been stopped after 1 cycle of learning rate evolution, meaning the first time η comes back to its minimum value. One can see that the cost and the loss decrease through the epochs, while the accuracy of both training and validation sets increases. The learnt network performance can be measured thanks to the accuracy obtained on the test set, which is about 46% (fig. 5), which is the same order of magnitude of the result obtained in the instructions.

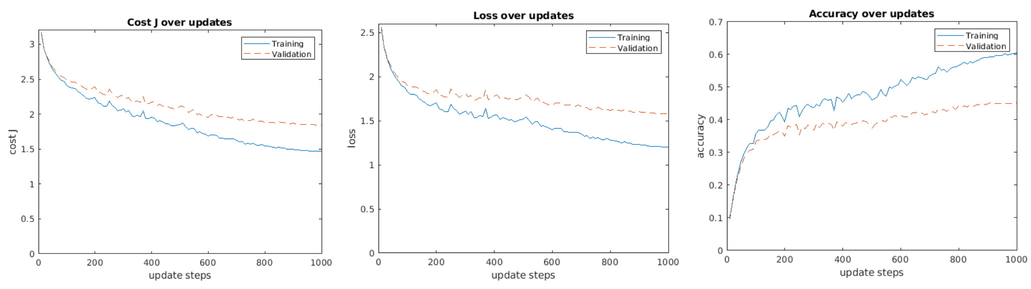


Figure 3: Replicate of figure 3 of the instructions

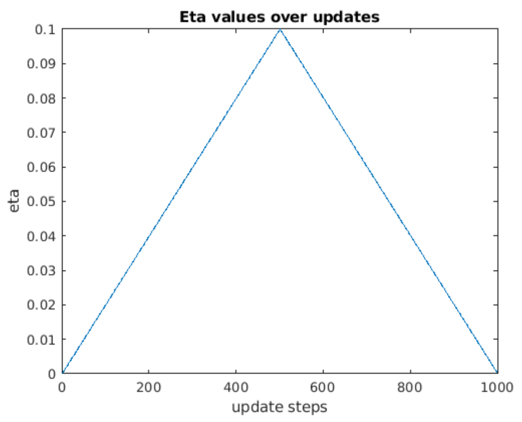
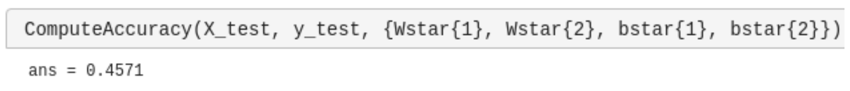


Figure 4: Evolution of etas though the updates during 1 cycle



```
ComputeAccuracy(X_test, y_test, {Wstar{1}, Wstar{2}, bstar{1}, bstar{2}})
```

```
ans = 0.4571
```

Figure 5: Accuracy obtained on the test set after 1 cycle

Replication of figure 4

Next step is to run the training algorithm for more cycles (3 here) and for a larger stepsize n_s.

Settings value:

- number of cycle: nb_cycles = 3
- eta_min = 1e-5
- eta_max = 1e-1
- stepsize: n_s = 800
- batch size: n_batch = 100
- $\lambda = 0.01$
- update step: t $\in [\![1, 6n\_s]\!] = [\![1, 4800]\!]$

Figure 6 shows the learning characteristics, meaning the evolution of the cost J, the loss and the accuracy through the epochs for both the training set (blue curves) and the validation set (orange curves). As shows figure 7, the training has been stopped after 3 cycles of learning rate evolution, meaning the third time η comes back to its minimum value. One can see that the cost and the loss decrease through the epochs on the first cycle. During the second cycle, both increase and then decrease again to lower values. The evolution is the same through the third cycle. The accuracy coherently increases during the first cycle, then decreases and increases to higher value during the second cycle, and the third cycle sees the same evolution. The learnt network performance can be measured thanks to the accuracy obtained on the test set, which is about 47% (fig. 8), which is the same order of magnitude of the result obtained in the instructions and as before.

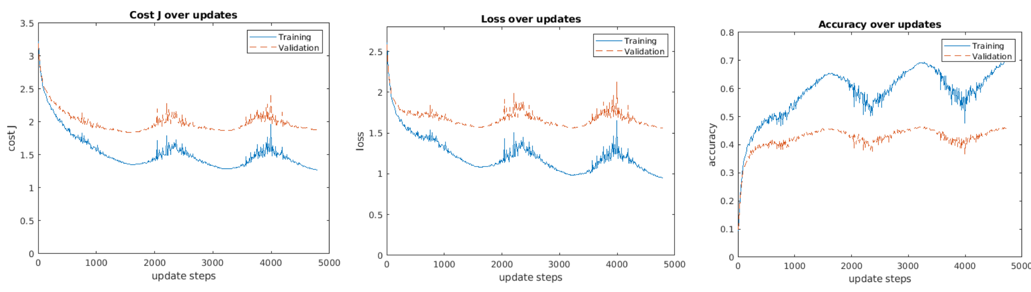


Figure 6: Replicate of figure 4 of the instructions

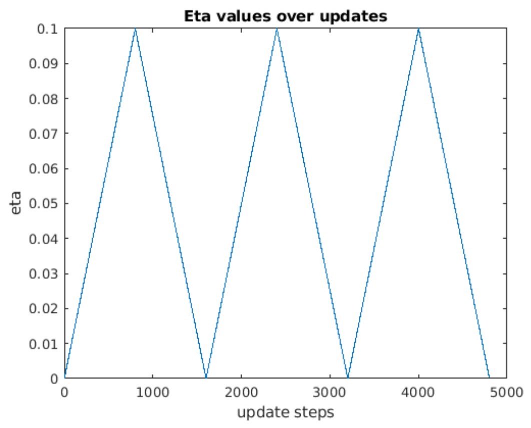
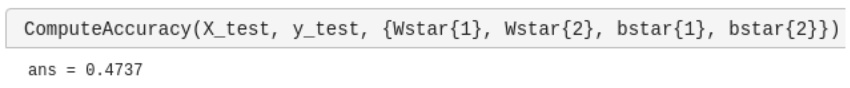


Figure 7: Evolution of etas though the updates during 3 cycles



```
ComputeAccuracy(X_test, y_test, {Wstar{1}, Wstar{2}, bstar{1}, bstar{2}})
ans = 0.4737
```

Figure 8: Accuracy obtained on the test set after 3 cycles

Question iii) - coarse search of exercise 4

For the coarse search of lambda, I first performed a searched on the recommend interval $[\![l_{min}, l_{max}]\!] = [\![10^{-5}, 10^{-1}]\!]$ with a uniform grid with eight different values and for 2 cycles of learning rate evolution, but the accuracy was decreasing on the interval so the higher value of accuracy was obtained for the smaller lambda. That's why I decided to perform a second coarse search on smaller orders of magnitude of lambda ($[\![l_{min}, l_{max}]\!] = [\![10^{-7}, 10^{-5}]\!]$) to find the best value around which I should perform the fine search. The obtained results can be summarized in table 1. The green row matches the values for which the accuracy is the greatest, the yellow ones are for the second and third greatest values of accuracy. Thus, you have the hyper-parameter settings for the 3 best performing networks I trained. The other settings have not changed from question ii).

lambda	accuracy
1e-7	0.511
3.4e-6	0.5138
6.7e-6	0.5134
1e-5	0.5108
0.042	0.449
0.1	0.32832

Table 1: Final accuracy obtained on the validation set for the coarse search of lambdas

Question iv) - fine search of exercise 4

Once I got the hyper-parameter of the 3 best performing I trained, I can now perform a fine search around the lambda that matches the highest accuracy and using the second and third values as the limits: $[\![l_{min}, l_{max}]\!] = [\![10^{-7}, 6.7e - 6]\!]$. The 3 best performing networks have the lambda values showed in table 2. I got a good accuracy of 52% so I stopped the search at this point.

lambda	accuracy
1e-7	0.511
1.043e-6	0.5214
1.96e-6	0.518

Table 2: Final accuracy obtained on the validation set for the fine search of lambdas

Question v)

My best performing network has a lambda value equal to 1.043e-6 as showed in previous question. In this section, I trained the network on all the training data except for 1000 examples in a validation set, for 3 cycles. The results can be seen figure 9. The evolution of the cost J, the loss and the accuracy are similar to what I obtained in question ii) about ”figure 4” replication, with the exception that the blue curve and the orange curve are closer than before. That means that the accuracy of the validation set is closer to the accuracy of the training set, corresponding to even better performance. This can be verified thanks to the accuracy obtained on the test set of 51% which is great compared to what we got previously on the test set (maximum 46%), as showed on figure 11.

All the other settings have not been changed, the eta values evoluting through 3 cycles with $n_s = 1000$, as shows figure 10.

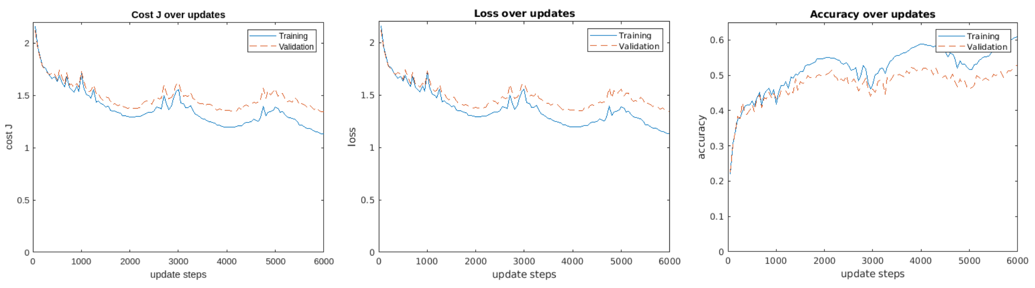


Figure 9: Training data illustration (cost, loss, accuracy through the epochs)

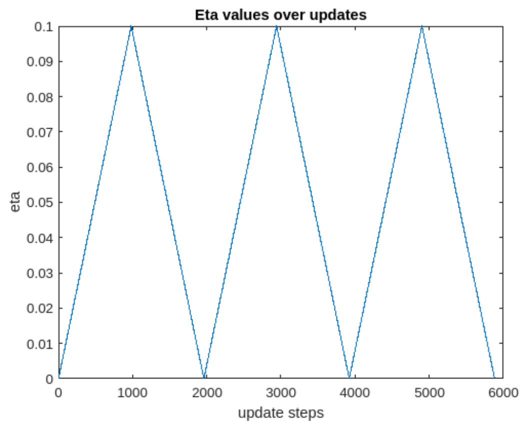
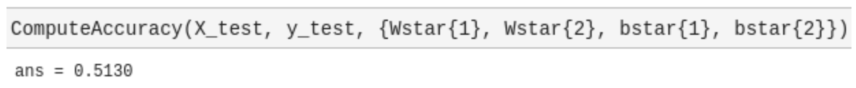


Figure 10: Evolution of etas though the updates during 3 cycles



```
ComputeAccuracy(X_test, y_test, {Wstar{1}, Wstar{2}, bstar{1}, bstar{2}})
ans = 0.5130
```

Figure 11: Learnt network’s performance obtained on the test set for the best lambda settings

Conclusion

I have experienced through this assignment the influence of different parameters, such as the stepsize and the number of cycles in the cyclical evolution of the learning rate eta in order to speed up training. Searching for a good value of lambda can also be worth to improve the network performance even though coarse-to-fine search can be time consuming.