KTH Royal Institute of Technology

Deep Learning in Datascience
DD2424

Assignment 3

Authors:
Ali Banaei Mobarak Abadi

May 5, 2022

1 Extending the code

As the first step of this assignment, we had to generalize the code from the previous assignment to a k-layer multi-layer perceptron. In order to make the code easier to debug and more readable, I used an object-oriented approach to solve the problem. The Model class implements an MLP model by using Dense_layer class. In Dense_layer class, we have forward and backward methods. In the forward method, the layer performs the forward pass operation and stores the values it will need for calculating the gradient later. In backward method, the gradients of the cost function based on different components are calculated and stored. The gradient with respect to the inputs is returned to be used as input in the backward function of the previous layer. Also, I used He initialization for the initial values of the weights, and the initial value for γ and β are 1 and 0.

2 Checking the gradient

After implementing everything, the first step is to check if the calculation of the gradients is correctly implemented. So, all the gradients are compared to the numerically calculated versions to ensure implementation is bug-free. In order to make the comparison of values easier, the initial values of the weight vectors are changed to random values drawn from a normal distribution with a mean of 0 and a standard deviation equal to 2. By doing this, the values in the network will get big, and we will see big gradients. Also, if there is a bug in the implementation with big weights, the error will be amplified. Finally, we compare the mean, STD, and the max value of the difference between the two methods with the respective metrics of the gradient. The results of this comparison can be found in Listing 2.

```
model with no batch norm
layer 0:
W: mean=-0.5097001036303636 std=1.2106800385111829 mean_diff=9.95608131190151e-07 std_diff=1.3946208283390884e-06 max_diff=4.213600700575526e-06
b: mean=0.6897851405067021 std=0.9101289248052089 mean_diff=6.026841938538974e-07 std_diff=7.496373064288024e-07 max_diff=2.0435311541555645e-06
-----------
layer 1:
W: mean=0.5049977166998967 std=4.072281517260328 mean_diff=1.360864474285961e-05 std_diff=4.3855519461204086e-05 max_diff=0.00020874103731571836
b: mean=0.2988415389271323 std=0.4954698414899397 mean_diff=2.4581335669113246e-07 std_diff=2.3652371620016618e-07 max_diff=6.637855751723265e-07
-----------
layer 2:
W: mean=0.0043824488585420054 std=2.3629288066516163 mean_diff=1.4869869114031258e-06 std_diff=6.499394877309395e-06 max_diff=3.327843577949352e-05
b: mean=-5.551115123125783e-18 std=0.167084261083203 mean_diff=9.947598307721243e-09 std_diff=1.0942000451612945e-08 max_diff=2.7732373175659575e-08
-----------
model with batch norm
layer 0:
W: mean=-0.0839780101579187 std=0.1771482696455477 mean_diff=8.939838169269576e-08 std_diff=2.441997646496221e-07 max_diff=1.0027733415851614e-06
b: mean=-5.8286708792820724e-18 std=4.165556196590227e-17 mean_diff=-1.0658140978114796e-09 std_diff=1.4210854339113955e-09 max_diff=0.0
gamma: mean=1.897402009967686e-07 std=0.9800554661835622 mean_diff=5.112102078305703e-07 std_diff=1.2930831486309136e-06 max_diff=3.067726149552641e-06
betas: mean=0.24747773473014822 std=0.7069369348475908 mean_diff=-3.630722163472377e-08 std_diff=3.8770300560018024e-07 max_diff=7.235364460900229e-07
-----------
layer 1:
W: mean=-0.09192297948795643 std=0.19137018135060274 mean_diff=1.552730084335152e-08 std_diff=9.923212471724819e-08 max_diff=3.405460177186015e-07
b: mean=-2.9976021664879216e-17 std=7.487114753582493e-17 mean_diff=-7.105427057840785e-10 std_diff=8.702335314144054e-10 max_diff=4.4408920985006264e-17
gamma: mean=0.8582151925485121 std=0.799033358393596 mean_diff=1.1782459954264024e-07 std_diff=8.254835598673965e-08 max_diff=2.613252086947071e-07
betas: mean=0.9834085325781471 std=0.9216490070331642 mean_diff=3.147544647627565e-07 std_diff=2.7654333219880184e-07 max_diff=8.266876407869717e-07
-----------
layer 2:
W: mean=-8.750016372312291e-05 std=0.13849091546221962 mean_diff=1.7573662150298896e-08 std_diff=1.8928908162452413e-08 max_diff=9.83173368551693e-08
b: mean=1.1102230246251566e-17 std=0.1767370838506264 mean_diff=2.0250467958494307e-08 std_diff=2.177676795353542e-08 max_diff=7.629055043256727e-08
-----------
```

As we can see, the differences are minimal compared to the actual values, and they are probably due to a numerical error in floating-point calculations. Note that the gradients are for a three-layer network with the input size equal to 10. A bigger input size with big weights would result in floating-point overflow in calculations. So, we can conclude the implementation for both with and without batch normalization seems to work fine.

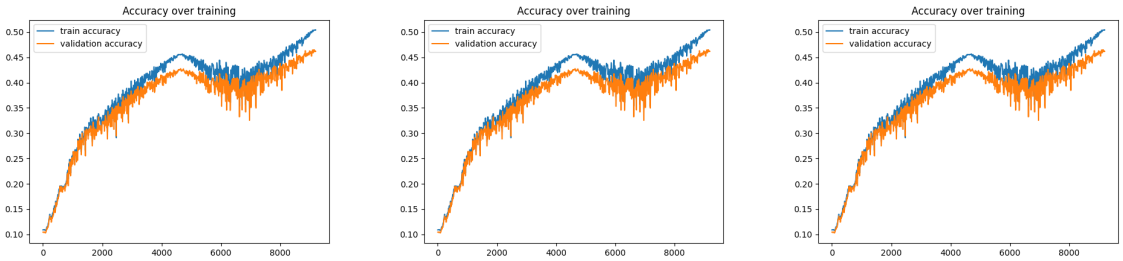
3 Training the network

In the next step, we try to train the model. The model is trained by the settings of exercise 2 with two hidden layers. Figure 1 depicts the logs during the training. After training for two cycles, we reached the accuracy of 0.536 on validation and 0.529 on test data which is the same as the reported values in assignment instructions.

(a) accuracy (b) loss (c) cost

Figure 1: Accuracy, loss and cost function for three-layer model of exercise 2.

Then we must train the 9-layer model. With the same hyper-parameter settings, this model reached the accuracy of 0.461 on validation and 0.463 on test data. Logs of the training for this function can be found in Figure 2. As expected, with increasing the number of layers, we see a notable decrease in the performance of the network.



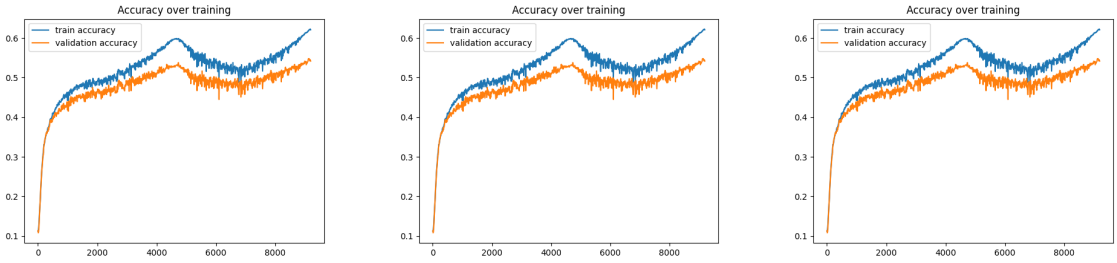
(a) accuracy (b) loss (c) cost

Figure 2: Accuracy, loss and cost function for nine-layer model of exercise 2.

4 Batch Normalization

Now that we have seen how a k-layer MLP performs on classification on CIFAR-10, it is time to test our network with Batch Normalization. We have already implemented it and tested the calculation of the gradients. For the first step, we add batch normalization to the first and second layers of the three-layer architecture of the previous section. After training with the same settings as before, the network reached the accuracy of 0.542 on validation and 0.534 on test data. Logs are also available in Figure 3. As we saw, there is an small improvement on the performance of the network.

The increase in final accuracy for the deeper model was more significant. After training, the performance of the 9-layer model was 0.524 on validation and 0.525 on test data. As we saw, it is still less than the model with fewer number of layers, but there is approximately a %7 increase in the final accuracy. Logs of trainig can be found in Figure 4.



(a) accuracy (b) loss (c) cost

Figure 3: Accuracy, loss and cost function for three-layer model of exercise 2 with batch normalization.

(a) accuracy (b) loss (c) cost

Figure 4: Accuracy, loss and cost function for nine-layer model of exercise 2 with batch normalization.

5 Searching for λ

In order to find a good value for the λ parameter, we tried different values of λ and trained the network for eight epochs (which is equal to the shorter training version in instructions). Then the validation accuracy was recorded for each setting. Table 1 shows these values. As we can see, our three-layer network best performed when we trained it with very small λ or even no regularization at all. This is interesting since, in the last assignment, we found that without using regularization, the network overfits very quickly, but here we do not see a sign of overfitting so far. This may indicate the regularization effect of batch normalization. Also, the performance of the best network with 8 epochs of training was 0.528 on the test data.

Table 1: Validation accuracy for different values of λ.

sigma	Test accuracy without BN
0.1	0.464
0.05	0.486
0.01	0.515
0.005	0.519
0.001	0.514
0.0001	0.522
0	0.522

6 Initialization sensitivity

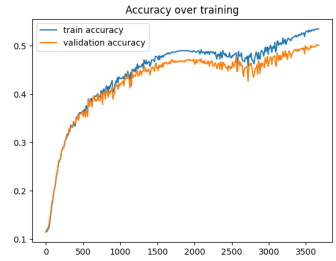
To investigate the effect of weight initialization, we used trained and tested the three-layer network with and without batch normalization with three different values for the *sigma* value of the normal distribution which we sample from. Results can be seen in [].

As we see, when we have small values of σ, since the weights and thus the resulting outputs and gradients are small, the network is not able to learn and does a random classification. However, batch normalization with scaling the output makes the network more robust to initial values and results in normal learning even when the initial values are very small. The results of the evaluation on the test set can be found in Table 2. Logs

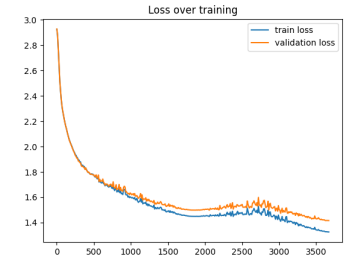
during training for model with no use of batch normalization can be found in Figure 5 and page 5 for model with batch normalization.

Table 2: Test accuracy on three-layer network with different σ values for random initialization.

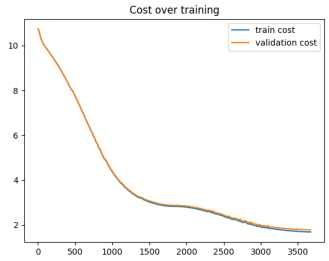
σ	Test accuracy without BN	Test accuracy with BN
0.1	0.512	0.522
1e-3	0.1	0.524
1e-4	0.1	0.527



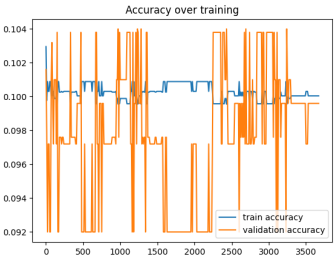
(a) accuracy $\sigma = 0.1$



(b) loss $\sigma = 0.1$

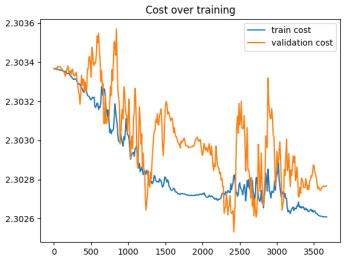


(c) cost $\sigma = 0.1$

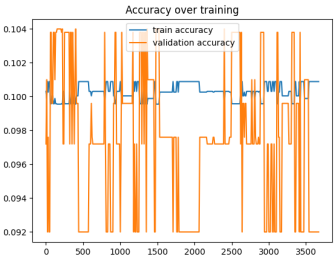


(d) accuracy $\sigma = 1e-3$

(e) loss $\sigma = 1e-3$

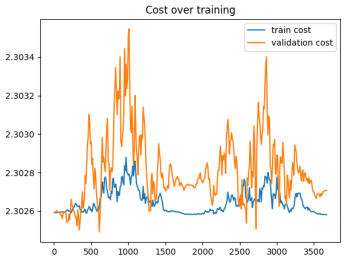


(f) cost $\sigma = 1e-3$



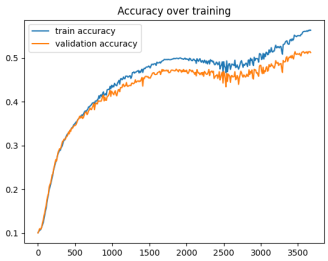
(g) accuracy $\sigma = 1e-4$

(h) loss $\sigma = 1e-4$

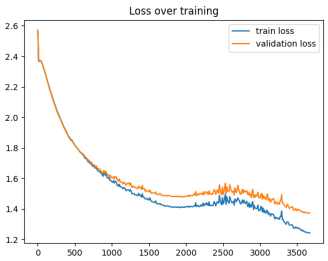


(i) cost $\sigma = 1e-4$

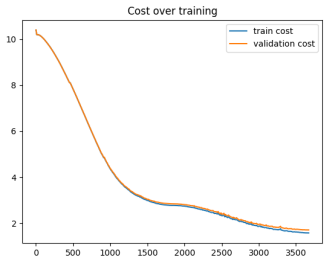
Figure 5: Accuracy, loss and cost function for three-layer model of exercise 2 without batch normalization and sifferent σ values.



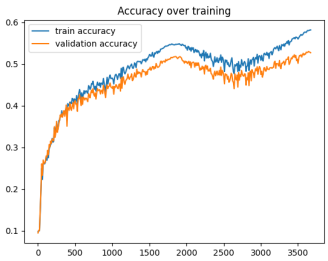
(a) accuracy $\sigma = 0.1$



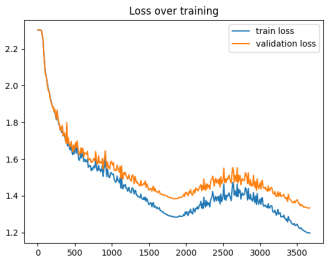
(b) loss $\sigma = 0.1$



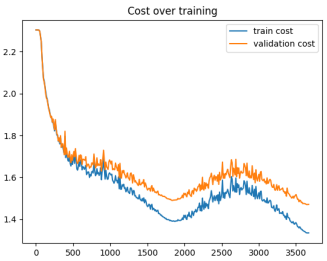
(c) cost $\sigma = 0.1$



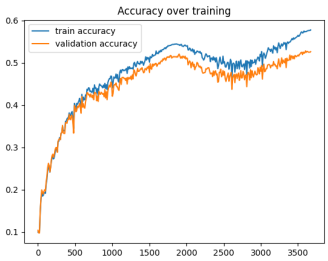
(d) accuracy $\sigma = 1e-3$



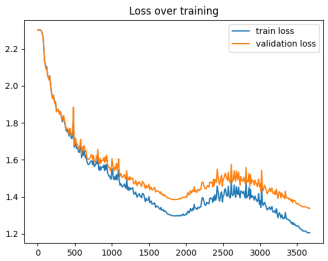
(e) loss $\sigma = 1e-3$



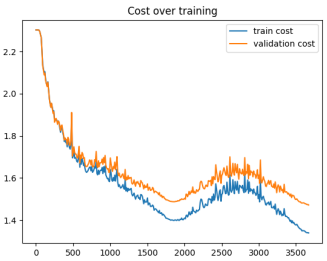
(f) cost $\sigma = 1e-3$



(g) accuracy $\sigma = 1e-4$



(h) loss $\sigma = 1e-4$



(i) cost $\sigma = 1e-4$

Figure 6: Accuracy, loss and cost function for three-layer model of exercise 2 with batch normalization and sifferent σ values.