DD2424 Deep Learning in Data Science - Assignment 3

Bas Straathof — btstr@kth.se

May 13, 2019

1 Introduction

The functions that were required for this assignment were all successfully implemented in Python 3, including the one to check the gradient numerically. A separate `TestMethods` class was established for testing several aspects of the data set and the methods of the `Classifier` class (with a random seed of `numpy.random.seed(0)`).

2 Gradient checking

A custom test case was created to test the similarity of arrays resulting from computing the gradients analytically and numerically by using the centered difference method. For a four layer neural network without batch normalization, the following maximum relative errors were obtained on a reduced dataset of 5 images with 30 dimensions each:

1. For layer 1 b: 8.20527e-07
2. For layer 1 W: 0.000201627
3. For layer 2 b: 7.98305e-07
4. For layer 2 W: 8.55952e-05
5. For layer 3 b: 4.69549e-07
6. For layer 3 W: 4.83805e-05
7. For layer 4 b: 4.32386e-06
8. For layer 4 W: 0.514309

For a nine layer neural network with batch normalization, the following maximum relative errors were obtained on a reduced dataset of 5 images with 30 dimensions each:

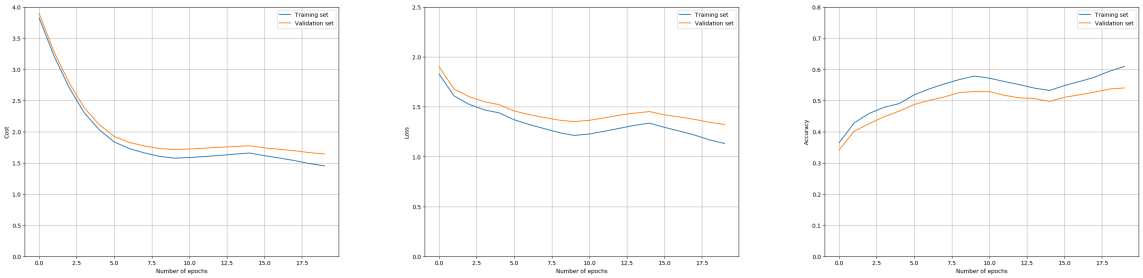
1. For layer 1 gamma: 1.91012
2. For layer 1 W: 1.99079
3. For layer 1 beta: 1.73478

4. For layer 1 b: 6.93889e-07
5. For layer 2 gamma: 1.17309e-06
6. For layer 2 W: 1.99494
7. For layer 2 beta: 9.03736e-06
8. For layer 2 b: 1.11022e-06
9. For layer 3 gamma: 0
10. For layer 3 W: 2.34264e-06
11. For layer 3 beta: 0
12. For layer 3 b: 1.8193e-07

As stated in the instructions, the relative errors for the shallower layers become quite substantial. Nevertheless, the gradient computations do not seem to be incorrect, since accuracies that are quite similar to the ones stated on the assignment sheet were obtained with the architectures containing the specified parameters.

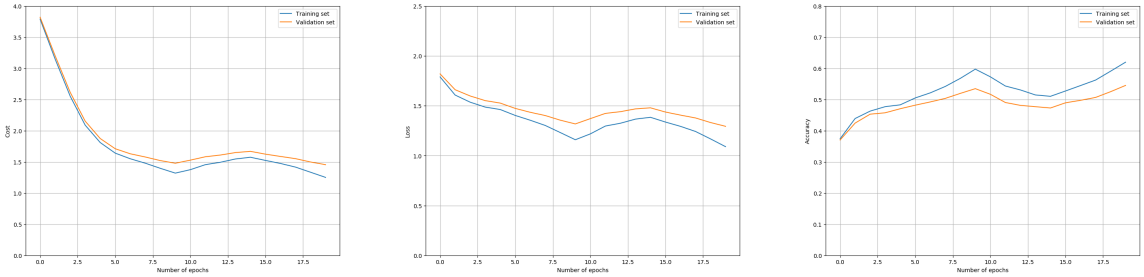
3 The evolution of the loss function for a 3-layer network

For both the three-layer network with batch normalization and the network without batch normalization, the following hyper-parameter settings were used: `eta_min = 1e-5, eta_max = 1e-1, lambda=.005, n_s=2250, batch_s=100, n_epochs=20`. See Figure 1 for the evolution of the loss function, cost function and accuracy on the network without normalization, and compare it to Figure 2, which corresponds to the implementation of a similar network with batch normalization. As can be seen, the plots are almost identical: for this type of shallow neural network, batch normalization does not seem to make a substantial contribution.



(a) Costplot (b) Loss plot (c) Accuracy plot

Figure 1: Training curves (cost, loss, accuracy) for a 3-layer network for two cycles of training without batch normalization. A test accuracy of 52.87% was achieved.

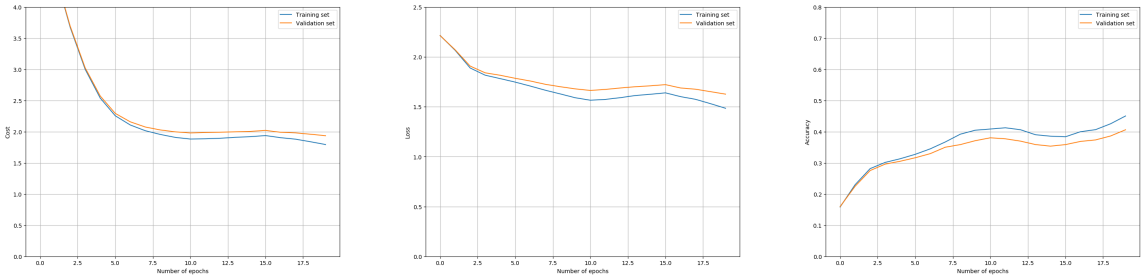


(a) Costplot (b) Loss plot (c) Accuracy plot

Figure 2: Training curves (cost, loss, accuracy) for a 3-layer network for two cycles of training wit batch normalization. A test accuracy of 53.08% was achieved.

4 The evolution of the loss function for a 9-layer network

It is only when we are training neural networks with deeper layers that batch normalization starts making a considerable contribution. The aforementioned hyper-parameter settings were used again (i.e. `eta_min = 1e-5, eta_max = 1e-1, lambda=.005, n_s=2250, batch_s=100, n_epochs=20`). See Figure 3 for the evolution of the loss function, cost function and accuracy on the network without normalization, and compare it to Figure 4, which corresponds to the implementation of a similar network with batch normalization. These plots demonstrate that for these kind of deeper neural networks, batch normalization has a considerable impact.

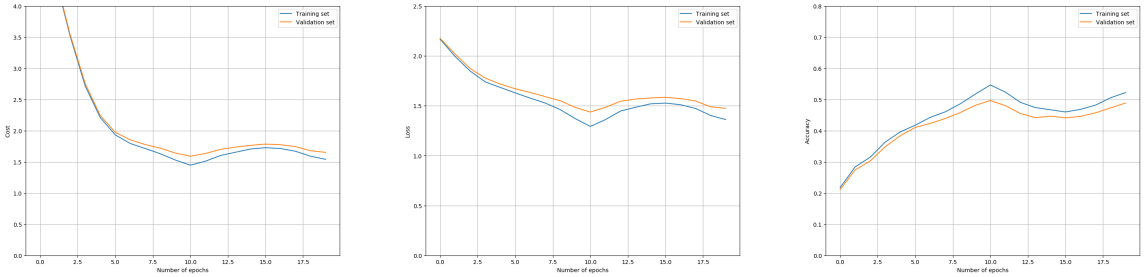


(a) Costplot (b) Loss plot (c) Accuracy plot

Figure 3: Training curves (cost, loss, accuracy) for a 9-layer network for two cycles of training without batch normalization. A test accuracy of 40.21% was achieved.

5 Lambda search for optimizing the performance of a 3-layer network

Let's first show that the 3-layer neural network with batch normalization can obtain similar performance as the one stated in the assignment sheet (53.5%) with the same hyper parameters as stated before. The following mean classification results (with $\pm$ the standard deviation) were obtained



(a) Costplot (b) Loss plot (c) Accuracy plot

Figure 4: Training curves (cost, loss, accuracy) for a 9-layer network for two cycles of training wit batch normalization. A test accuracy of 47.61% was achieved.

after running the models 5 times:

- Settings: **eta_min = 1e-5, eta_max = 1e-1, lambda=.005, n_s=2250, batch_s=100, n_epochs=20**
  - The accuracy on the training set is: 61.67% ± 0.15%
  - The accuracy on the validation set is: 54.11% ± 0.49%
  - The accuracy on the testing set is: 52.59% ± 0.31%

The above was obtained after running the model 5 times. Over a single run, sometimes testing accuracies of ca. 53% were observed.

5.1 Coarse search

For a coarse search for `lambda`, 20 values were randomly sampled from the range $[10 \cdot 10^{-5}, 10 \cdot 10^{-1}]$. Using these values, in conjunction with the parameters `batch_s` = 100, `n_s` = 2250 and `n_epochs` = 10 the results displayed in Table 1 were obtained. Based on the accuracies obtained on the validation set, the three best regularization values obtained were `lambda` ≈ 0.0121, `lambda` ≈ 0.0127 and `lambda` ≈ 0.0143.

5.2 Fine search

For a fine search for `lambda`, the best values obtained during the coarse search were used as an indication for the fine search range. 20 values were uniformly sampled from the range $[0.0017, 0.0367]$. Using these values, in conjunction with the parameters `batch_s` = 100, `n_s` = 2250 and `n_epochs` = 20, the results displayed in Table 2 were obtained. Based on the accuracies obtained on the validation set, the three best regularization values obtained were `lambda` ≈ 0.00645, `lambda` ≈ 0.00724 and `lambda` ≈ 0.0153.

Lambda	**Training accuracy**	**Validation accuracy**
1.800743784850257e-05	0.5772666666666667	0.511
3.2945541252710264e-05	0.6228666666666667	0.5136
3.800259933649657e-05	0.6543555555555556	0.5044
5.575193640037552e-05	0.6763777777777777	0.4994
8.926244452054713e-05	0.6913333333333334	0.4942
0.00015937609509227097	0.7042888888888889	0.4906
0.0001752824533379222	0.7132	0.4896
0.00021797292259508874	0.7222222222222222	0.4858
0.0002386878857881752	0.7281111111111112	0.4864
0.00024259109093076895	0.7308666666666667	0.479
0.0004673961157889023	0.7334	0.4772
0.0009682947335794582	0.7290888888888889	0.4874
0.001166736352671299	0.7168222222222222	0.4912
0.0017126201828583333	**0.6952**	**0.5016**
0.0017742313726441265	**0.6821333333333334**	**0.517**
0.0024299266439643478	**0.6650222222222222**	**0.5306**
0.00524855191741102	**0.6304**	**0.5388**
0.012176982468910604	**0.6003555555555555**	**0.5406**
0.01272173170997605	**0.6002666666666666**	**0.545**
0.014321784844573322	**0.5982222222222222**	**0.5474**
0.03580624378621181	**0.5588666666666666**	**0.5234**
0.036640478622647465	**0.5563555555555556**	**0.5196**
0.06917781898608903	0.5288666666666667	0.5004
0.08245534839102281	0.5195111111111111	0.4954
0.08367422441145302	0.5213111111111111	0.4926

Table 1: Coarse search for the regularization parameter `lambda`.

Lambda	Training accuracy	Validation accuracy
0.0031760648204450844	0.6224	0.5256
0.006255312874316667	0.6206888888888888	0.537
0.006446305867707408	**0.6201333333333333**	**0.5388**
0.007239855718008302	**0.6190888888888889**	**0.5428**
0.009965453241512507	0.6087333333333333	0.5358
0.010080212418698475	0.6102222222222222	0.537
0.01530364217551352	**0.5936888888888889**	**0.5398**
0.01915342227779119	0.5848888888888889	0.5264
0.022609784181142737	0.5744222222222222	0.5276
0.02371237426841553	0.5719111111111111	0.5292
0.025404432084701244	0.5691333333333334	0.5296
0.028279104070800226	0.5676888888888889	0.5228
0.0318448018959893	0.5586222222222222	0.5198
0.03214061645172008	0.5588444444444445	0.5246
0.03554799107161745	0.5568	0.5172

Table 2: Fine search for the regularization parameter `lambda`.

If we now use the optimal value for `lambda`, that is `lambda=0.0072`, we obtain the following results after training the model five times.

- Settings: **eta_min = 1e-5, eta_max = 1e-1, lambda=.0072, n_s=2250, batch_s=100, n_epochs=20**
  - The accuracy on the training set is: 60.82% ± 0.08%
  - The accuracy on the validation set is: 53.87% ± 0.31%
  - The accuracy on the testing set is: 53.06% ± 0.27%

6 Sensitivity to initialization

In this section, the sensitivity of k-layer models with and without batch normalization is explored. Instead of using He initalization, the weight parameters of each layer are initialized to be normally distributed with `sig=1e-1`, `sig=1e-3` and `sig=1e-4`. The test models are 9-layer networks with the following hyper parameter settings: `eta_min = 1e-5, eta_max = 1e-1, lambda=.0072, n_s=2250, batch_s=100, n_epochs=20`. After training the models five times, the following results were obtained:

- No batch normalization: `sig=1e-1`
  - The accuracy on the training set is: 10.05% ± 0.00%
  - The accuracy on the validation set is: 9.50% ± 0.00%
  - The accuracy on the testing set is: 10.00% ± 0.00%
- Batch normalization: `sig=1e-1`

    - The accuracy on the training set is: 58.16% ± 0.14%
    - The accuracy on the validation set is: 51.34% ± 0.53%
    - The accuracy on the testing set is: 51.14% ± 0.34%

- No batch normalization: `sig=1e-3`

    - The accuracy on the training set is: 10.06% ± 0.00%
    - The accuracy on the validation set is: 9.50% ± 0.00%
    - The accuracy on the testing set is: 10.00% ± 0.00%

- Batch normalization: `sig=1e-3`

    - The accuracy on the training set is: 55.93% ± 1.76%
    - The accuracy on the validation set is: 49.58% ± 1.41%
    - The accuracy on the testing set is: 49.17% ± 1.37%

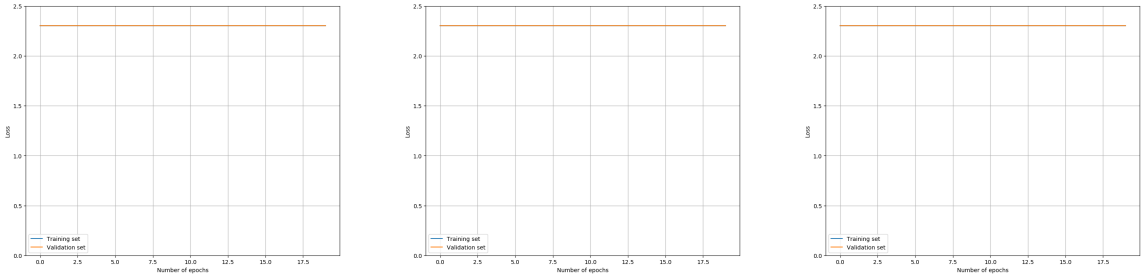
- No batch normalization: `sig=1e-4`

    - The accuracy on the training set is: 10.06% ± 0.00%
    - The accuracy on the validation set is: 9.50% ± 0.00%
    - The accuracy on the testing set is: 10.00% ± 0.00%

- Batch normalization: `sig=1e-4`

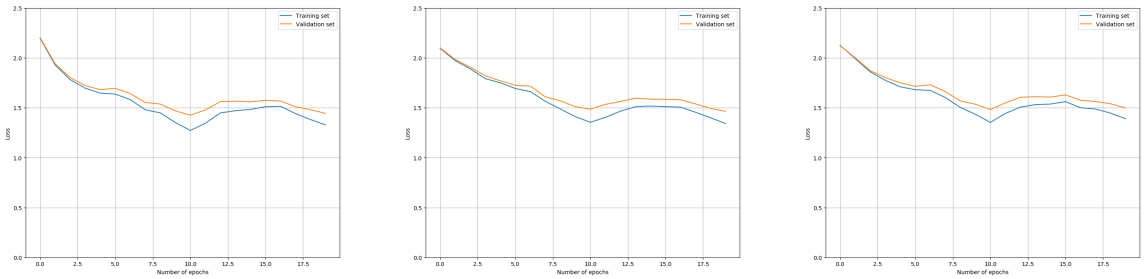
    - The accuracy on the training set is: 55.20% ± 1.94%
    - The accuracy on the validation set is: 49.56% ± 0.92%
    - The accuracy on the testing set is: 49.17% ± 1.28%

From the above results, we can conclude that deep neural networks trained with batch normalization are way less sensitive to weight initialization. The above normalizations render conventional deep architectures completely useless; although suboptimal, batch normalization ensures that the network can still learn even when the initialization is bad. See Figures 5 and 6 to see the loss function evolution for the networks with and without batch normalization. These plots show a flat line for the network without batch normalization, and a steady cyclic decrease of loss for the network trained with batch normalization.



(a) sig=1e-1 (b) sig=1e-3 (c) sig=1e-4

Figure 5: Loss function evolution plots for a 9-layer network for two cycles of training without batch normalization and weight initialization following a normal distribution with `sig=1e-1`, `sig=1e-3` and `sig=1e-4`, respectively.



(a) sig=1e-1 (b) sig=1e-3 (c) sig=1e-4

Figure 6: Loss function evolution plots for a 9-layer network for two cycles of training with batch normalization and weight initialization following a normal distribution with `sig=1e-1`, `sig=1e-3` and `sig=1e-4`, respectively.