Deep Learning in Data Science

Assignment 3

Isak Gamnes Sneltvedt

23.05.2022

DD2424

Assignment 3

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 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 because 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.

3-layer network

Without batch normalization

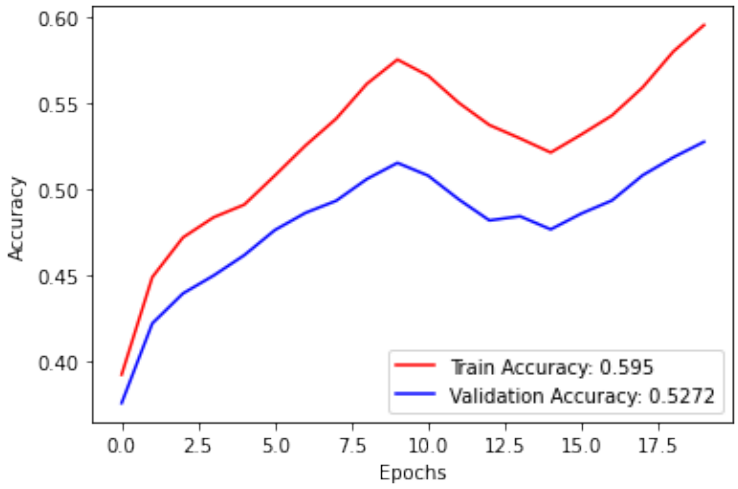

Figure 1: Accuracy without batch normalization

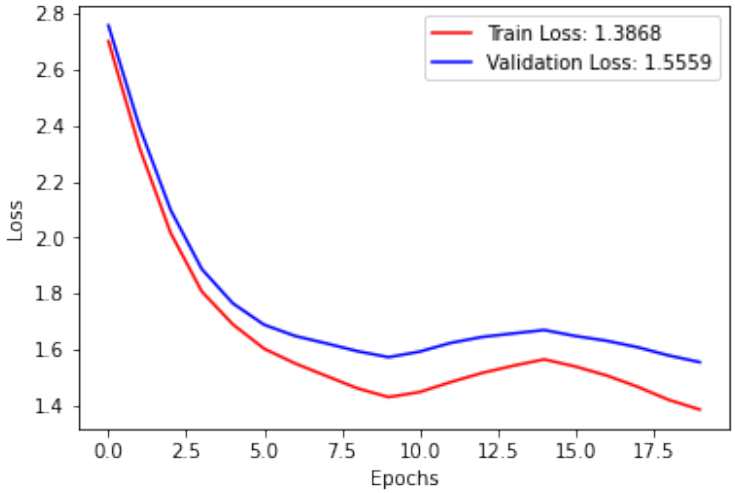

Figure 2: Loss without batch normalization

With batch normalization

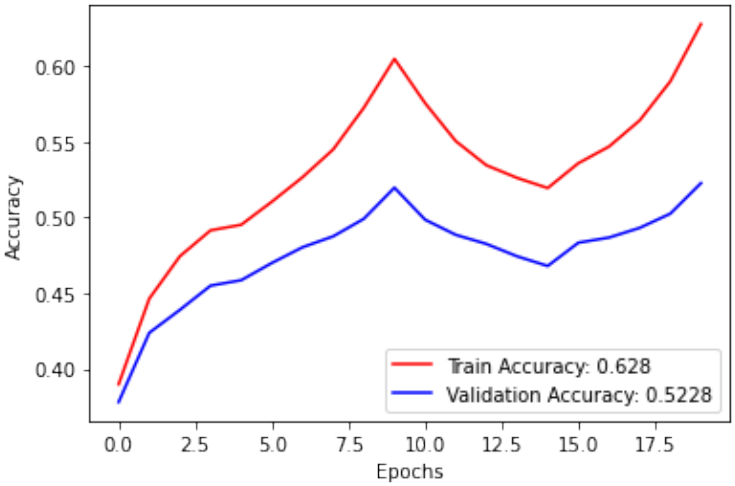

Figure 3: Accuracy with batch normalization

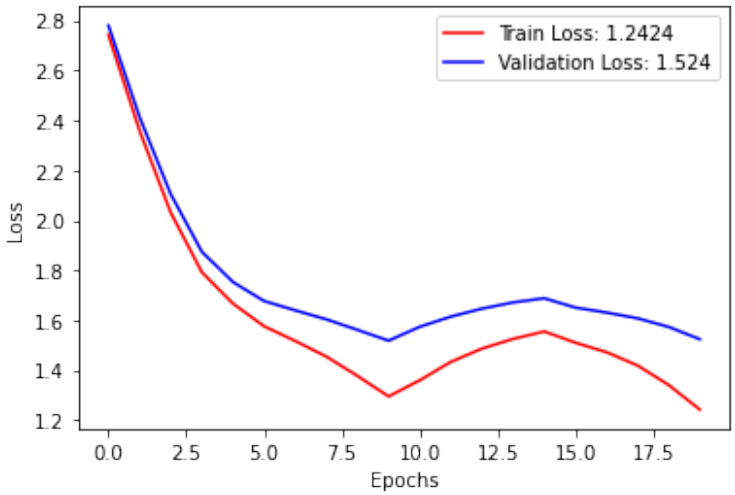

Figure 4: Loss with batch normalization

9-layer network

Without batch normalization

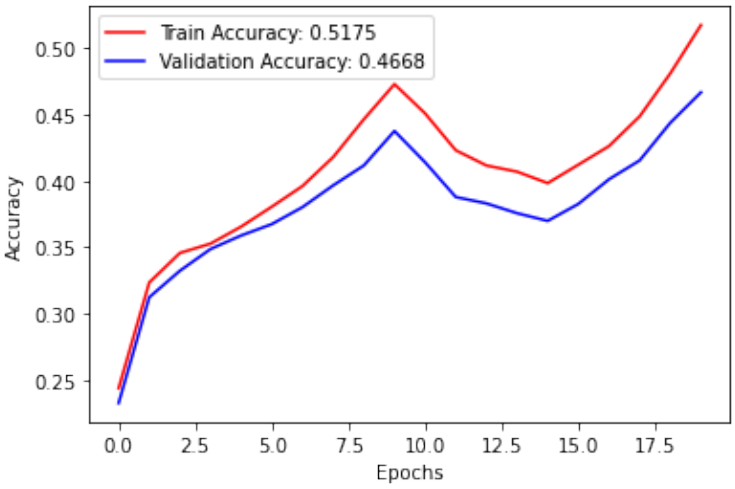

Figure 5: Accuracy without batch normalization

With batch normalization

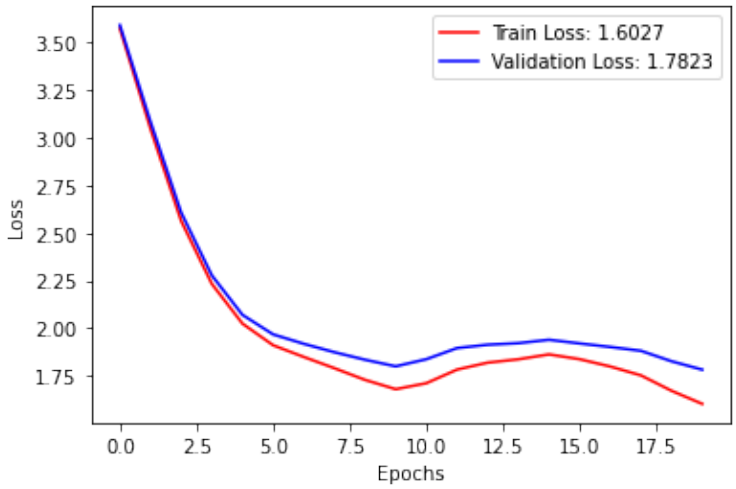

Figure 6: Loss without batch normalization

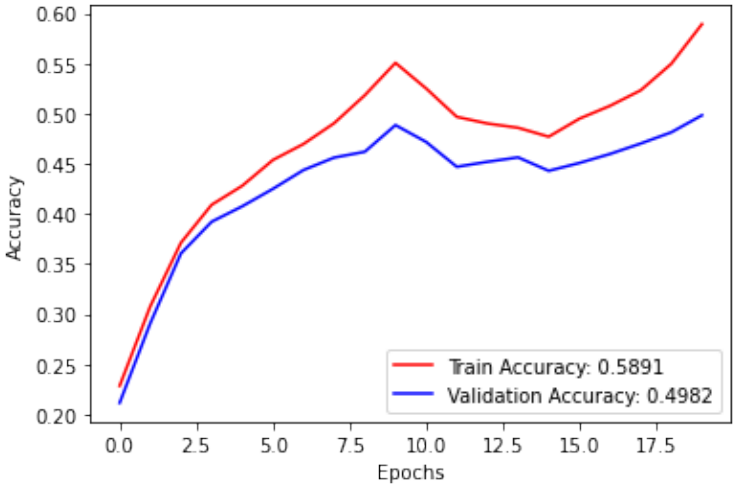

Figure 7: Accuracy with batch normalization

Lambda search

First search

After running the first search I see that the best values of lambda are placed quite close to the default lambda parameter value. The next iteration I therefore narrowed the values to be around these values:

```
list_of_lambdas = [0.0015, 0.002, 0.0025, 0.003, 0.0035, 0.004, 0.0045,
    0.005, 0.0055, 0.006]
```

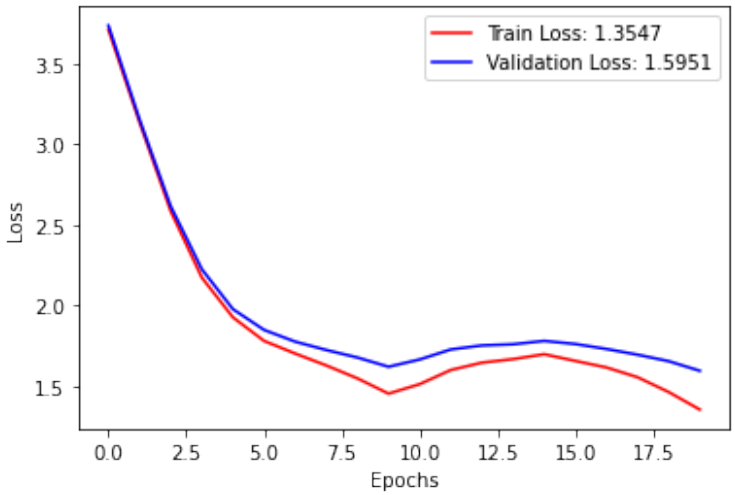

Figure 8: Loss with batch normalization

	Lambda	Accuracy
4	0.000013	0.4960
5	0.000118	0.4980
7	0.000032	0.4996
0	0.000047	0.5022
2	0.000395	0.5034
1	0.049369	0.5056
3	0.001223	0.5110
6	0.020373	0.5180
9	0.003140	0.5194
8	0.004764	0.5252

As seen here, the best lambda values are still quite close to the default lambda value. The best performance came from 0.0045, and that is the one that I chose to continue with as the best performing lambda. The results after running the 3-layer network with this lambda value are as follows:

As can be seen here, The final accuracy is not very different from the accuracy obtained using the default values.

	Lambda	Accuracy
3	0.0030	0.5178
2	0.0025	0.5184
4	0.0035	0.5188
0	0.0015	0.5202
1	0.0020	0.5230
9	0.0060	0.5262
7	0.0050	0.5264
5	0.0040	0.5272
8	0.0055	0.5306
6	0.0045	0.5314

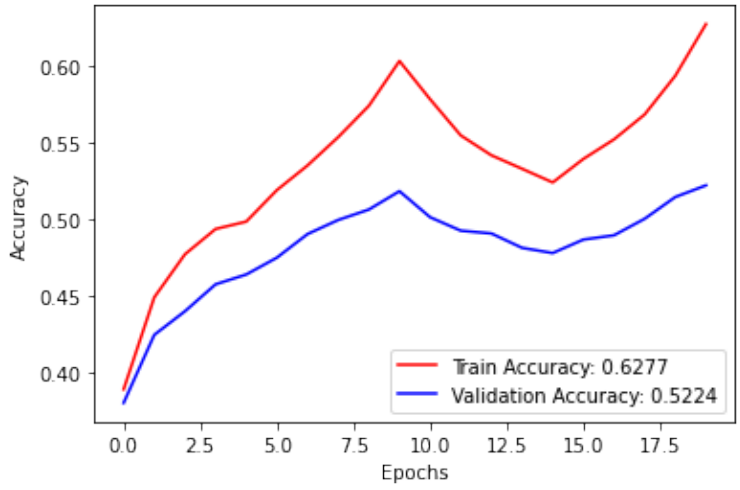

Figure 9: Accuracy using the best lambda value

No sensitivity initialization

```
Loss without batch normalisation and sigma = 0.1: 1.5216
Loss with batch normalisation and sigma = 0.1: 1.5543
Loss without batch normalisation and sigma = 0.001: 2.1213
Loss with batch normalisation and sigma = 0.001: 1.5411
Loss without batch normalisation and sigma = 0.0001: 2.3028
Loss with batch normalisation and sigma = 0.0001: 1.5367
```

The first thing I noticed is that for $\sigma = 1e-1$, the batch normalization has minimal effect. The loss is quite similar for both with and without batch normalization. When the σ is changed to $1e-3$ and $1e-4$ however, it is possible to see a difference when using batch

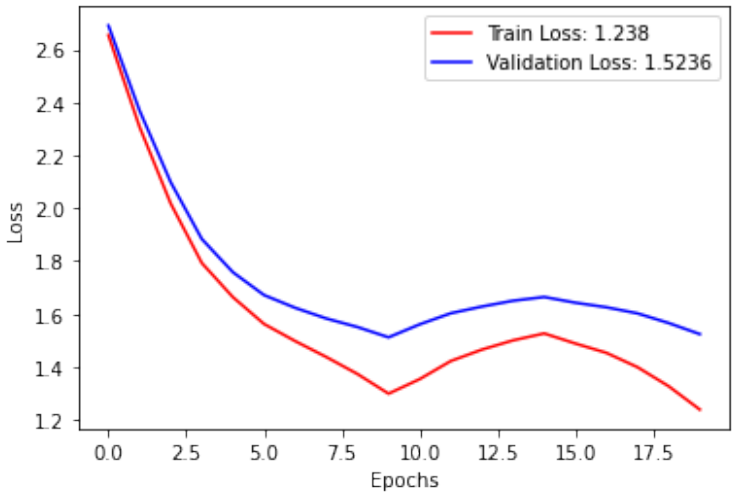

Figure 10: Loss using the best lambda value

normalization and not. Based on these outputs, I see that using batch normalization makes the model less sensitive to the initialisation of the weight parameters.