

DD2424 Assignment 3

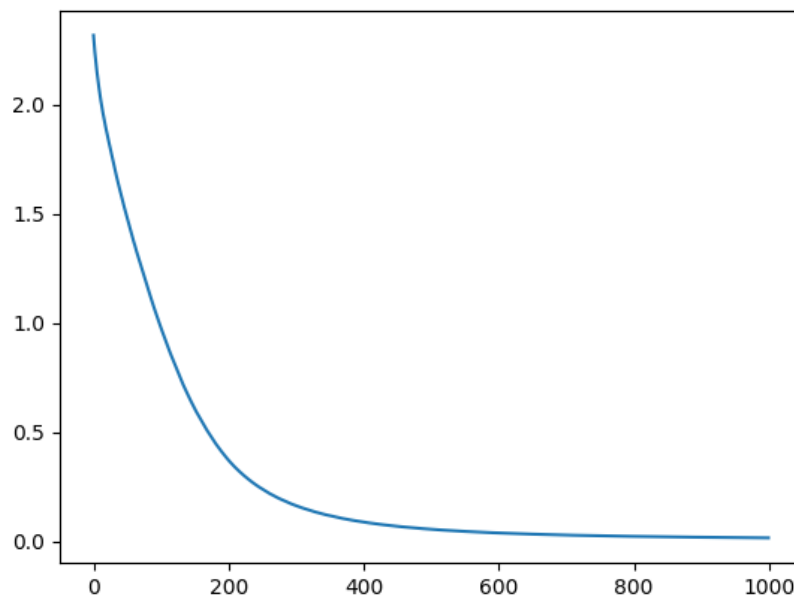
Jacob Heden Malm

May 9, 2022

1 Analytic gradient computations

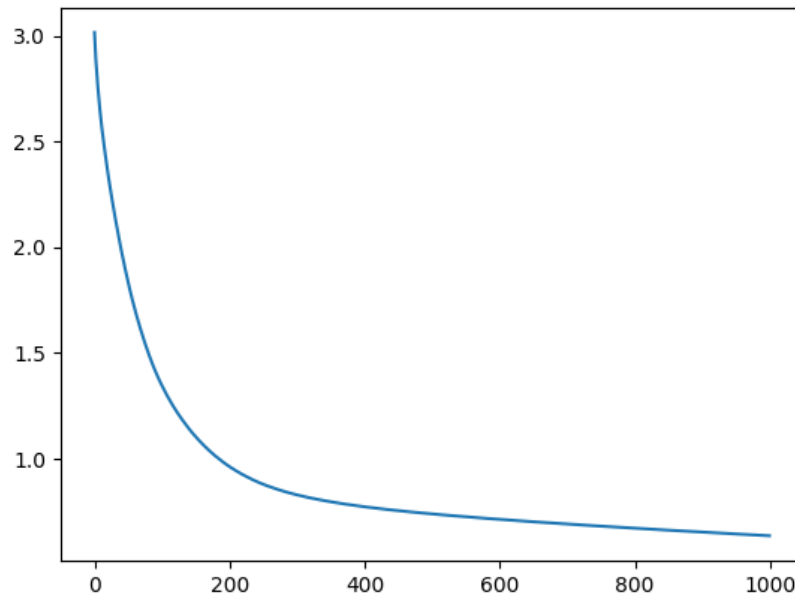
Since I wrote my code in python I could not use the provided `ComputeGradsNum()` method. Instead I performed a sanity check by attempting to over fit my network to a small amount of training examples and monitoring the development of the loss value. I wrote a method called `sanity_check()` where I passed in 100 data points and attempted to get my loss values as low as possible. I did this by training on the entire batch of data passed in for 1000.

Here is an example of the development of the loss values through training on this very limited data set.



The shape of the development of the loss function is almost exactly what we'd expect a nice satisfying $1/x$ curve. This suggests, since the derivative

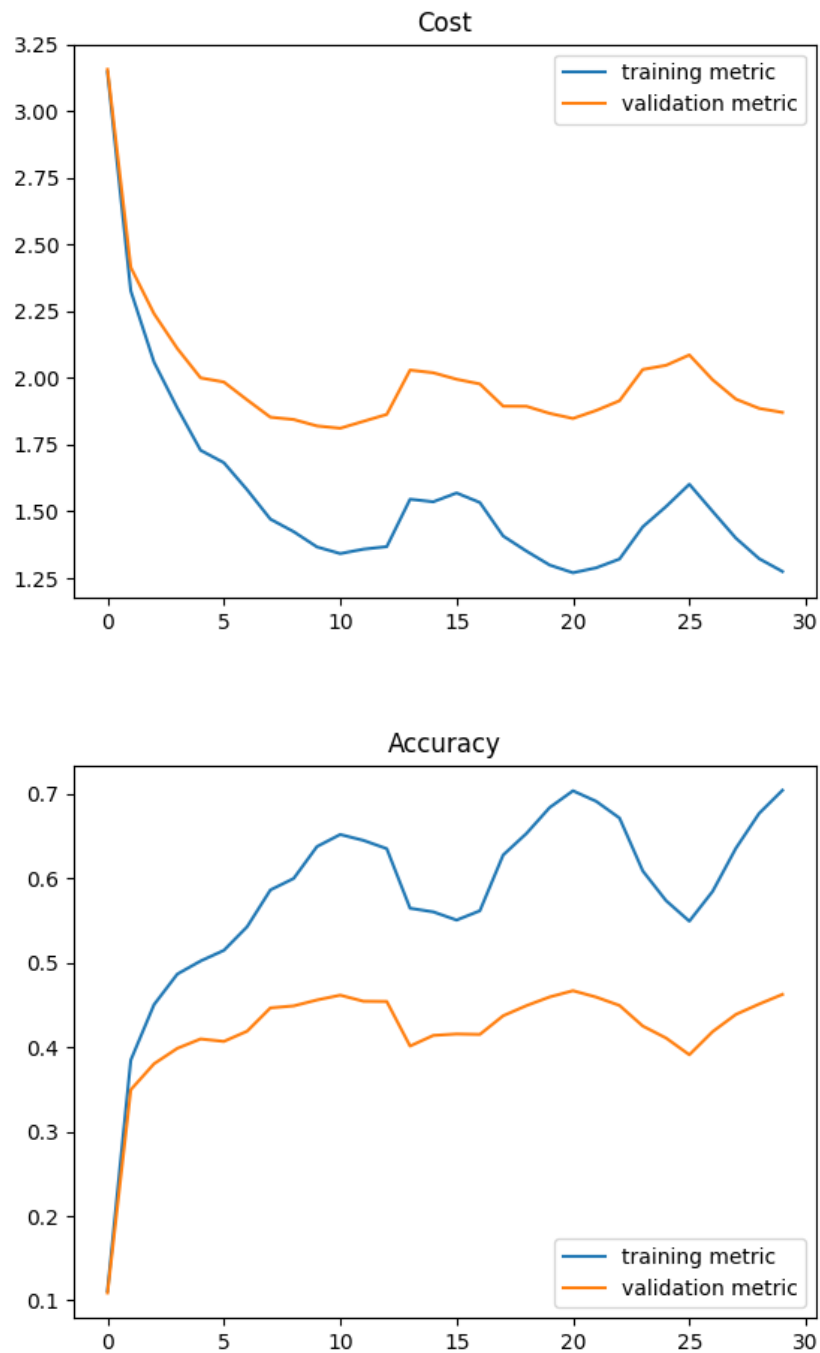
of this curve is logarithmic, that the rate of improvement decreases the more we have already learned, which makes sense. We also manage to get almost arbitrarily close to 0, suggesting that the network learns to recognize the training set perfectly, which allows us to conclude that the gradient computations are in fact working.



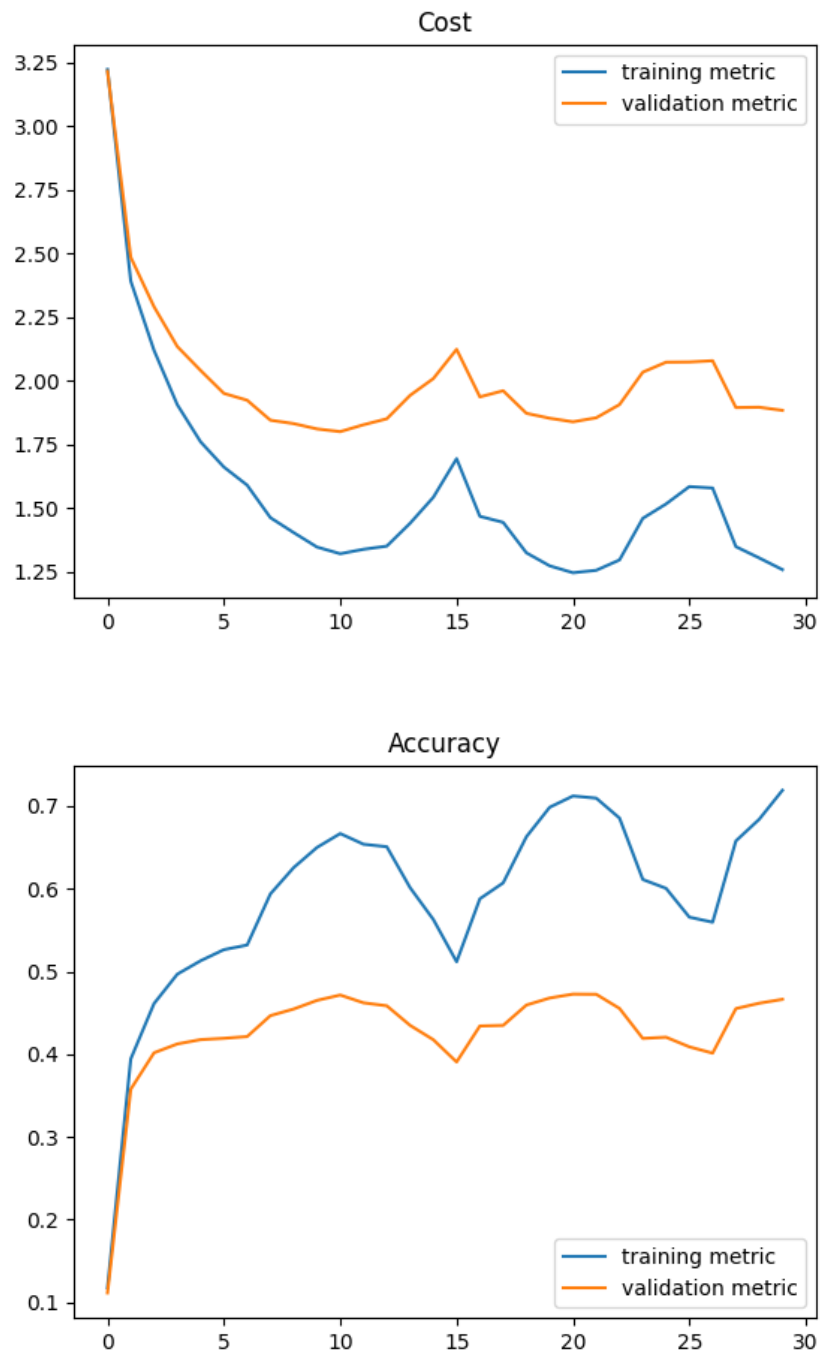
When we set λ to 0.01 we can see that we do not reach quite as low a value in our loss development, as expected, however we are still able to fit our net arbitrarily well.

2 Training a k-layer network

I began by training a 2 layer network with 1 hidden layer of 50 nodes.

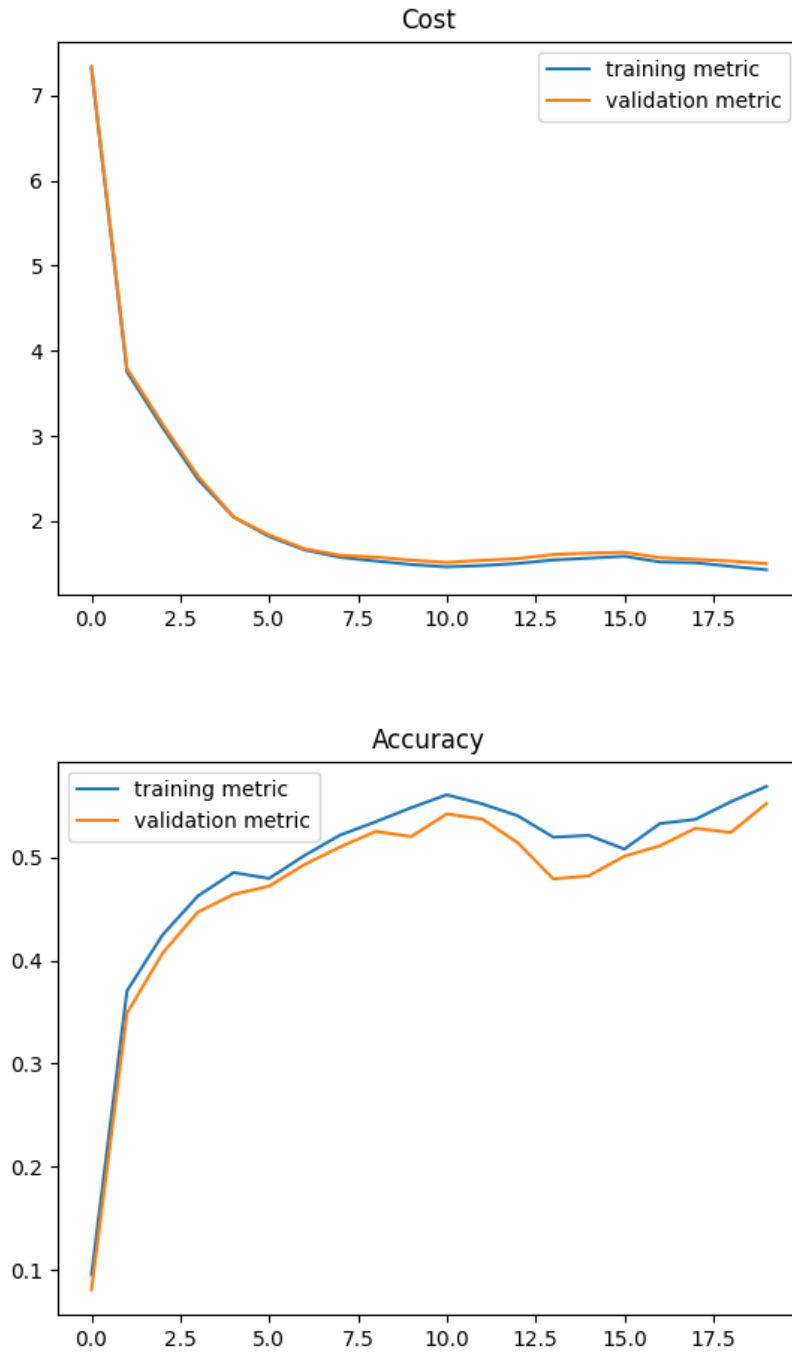


We can compare this with the results we got for a similar network from assignment 2



The results look pretty similar.

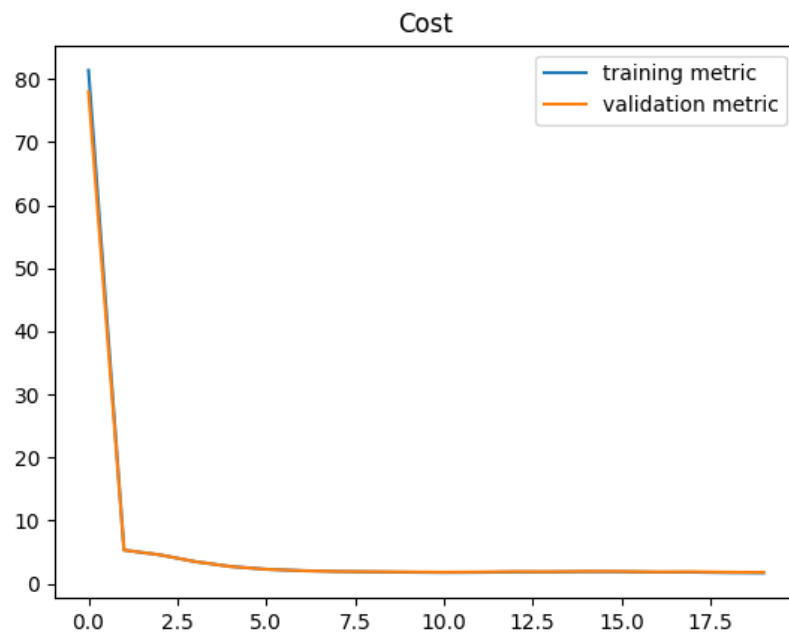
If we add a second hidden layer also of 50 nodes, our metrics look as follows

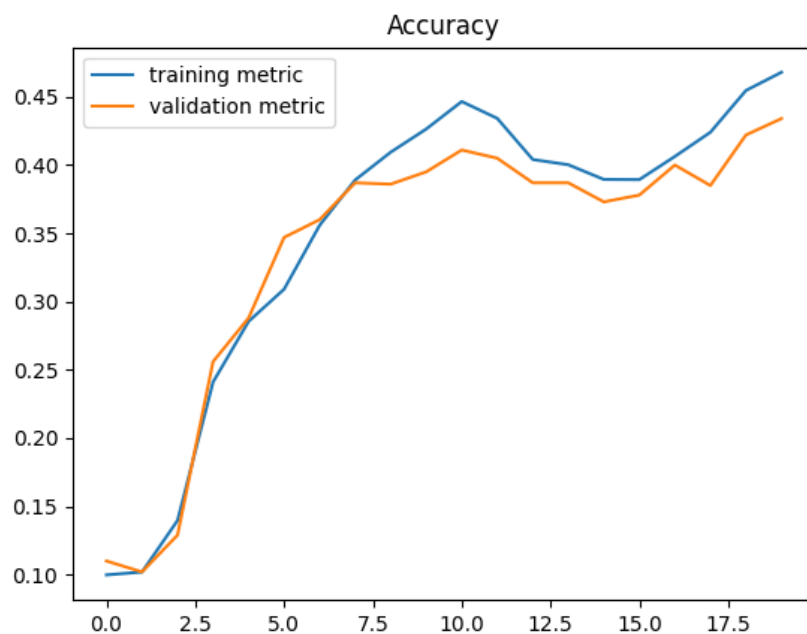


These graphs look pretty nice, the decrease in over fitting can be attributed

to me switching to using the full training set.

The accuracy this 3 layer network exhibits on the test set is 0.5253333333333333. This test accuracy is however higher than my most performant network from assignment 2, suggesting that there's some value in increasing the depth of the network. If we investigate this assumption by attempting to train a 9 layer network, we see the following results

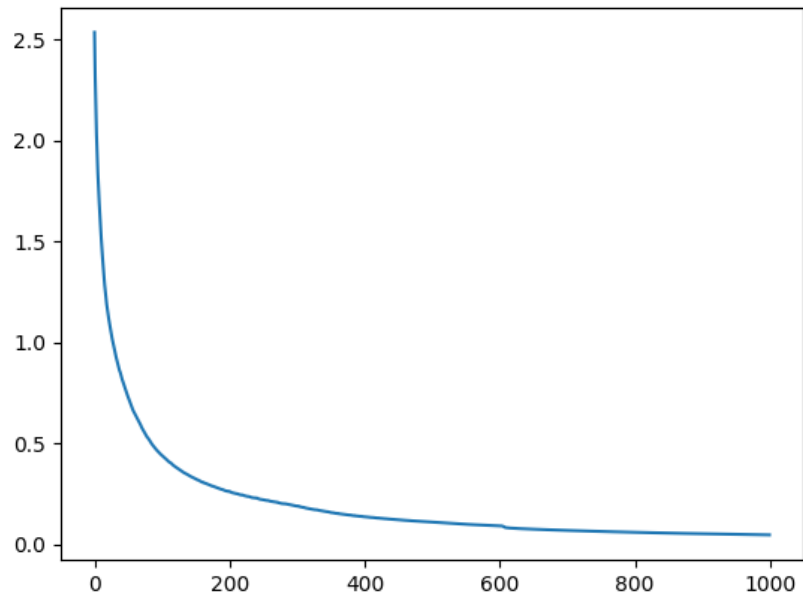




The test accuracy of this 9 layer network was 0.4488888888888889, almost an 8% point decrease from our 3 layer network, no bueno.

3 Implementing batch normalization

I performed the same sanity check as before, but on a network with batch normalization.



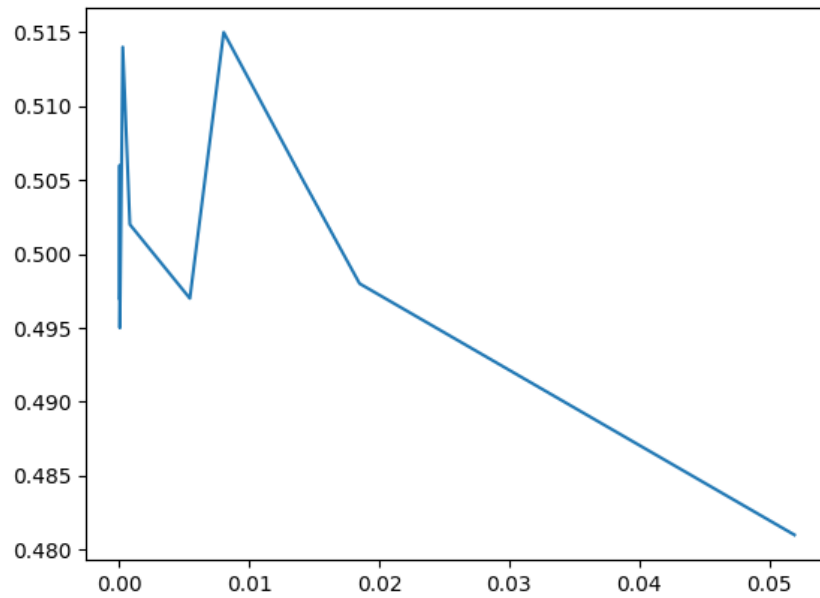
We can see that the loss development looks the same as previously, suggesting that our gradient computations with batch normalization work well.

4 Training a k-layer network with batch normalization

Test accuracy on 2 layer network: 0.5173333333333333 Test accuracy on 3 layer network: 0.5361111111111111

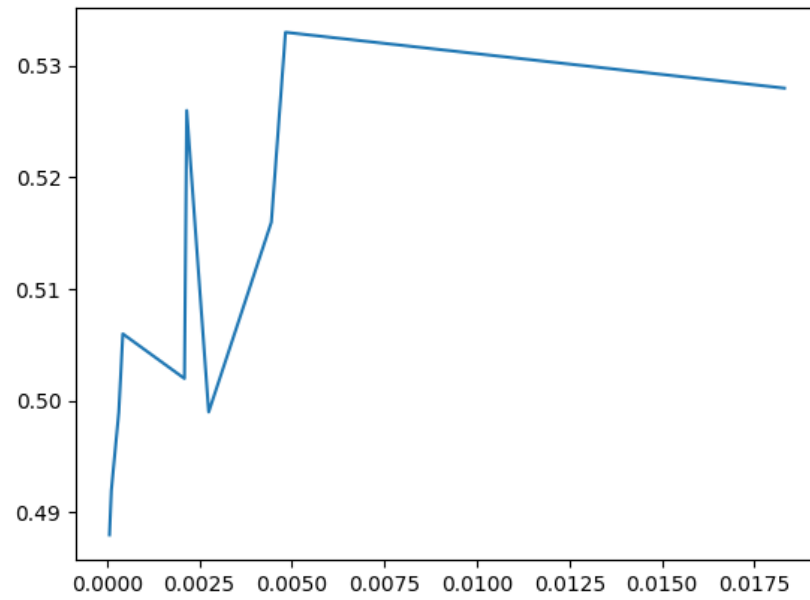
4.1 Search for optimal lambda value

I began by performing a coarse search with 10 uniformly sampled lambda values from the range $1E-5$, $1E-5$. The validation accuracy is plotted here.



As we can see the area of interest is roughly 0-0.015, with more emphasis to be placed on the upper end of this range.

Thus I performed a fine search by pulling 30 samples from a uniform distribution with domain $[1E-4, 1E-2]$.



Test accuracy of 3 layer network with optimal lambda: