Assignment 3 - Option 1 - DD2424 - K-Layer Network and Batch Normalization

Tristan Perrot

April 2024

Gradient check

After completing the code for the previous assignment, I have upgraded the code to support a K-layer network. The code is now able to train a K-layer network. This model is without batch normalization and to check whether the gradient computation was good or not I used the same tests as in the previous assignment.
How I checked my analytic gradient computations with batch normalization?
As I did before, for the parameters of the network that are not in the batch normalization I computed numerically the gradient with the given function. Moreover, for the `gamma` and `beta` parameters of the batch normalization, I computed the gradient with the grad functionality of PyTorch. I compared the two gradients and they were the same.

3 layer network

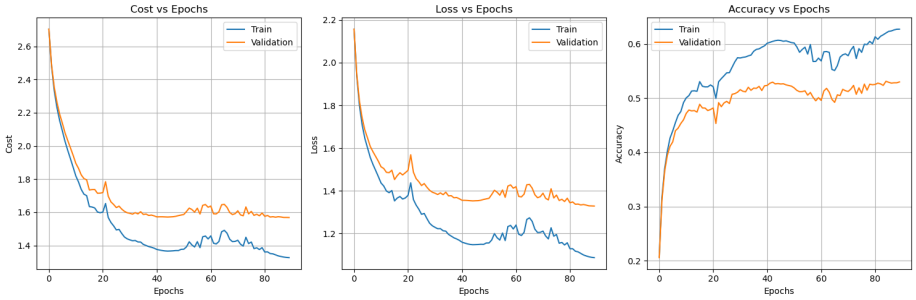

Figure 1: The training and validation loss for a 3-layer network without batch normalization with `lambda=.005`. The **test accuracy** is 53.05%. I used Xavier initialization.

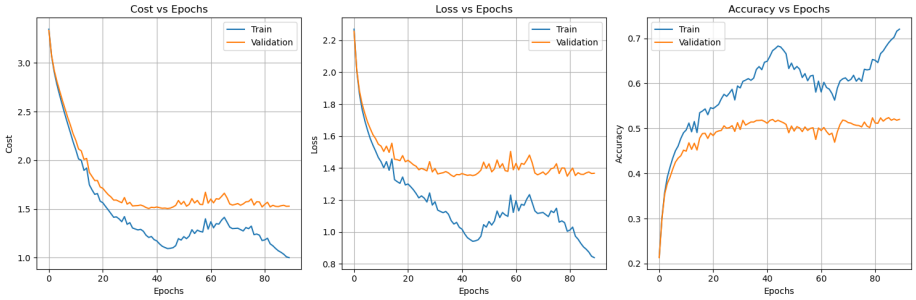

Figure 2: The training and validation loss for a 3-layer network with batch normalization with `lambda=.005`. The **test accuracy** is 51.45%. I used He initialization.

Here we can see that the batch normalization helps a lot for the training accuracy. Moreover, with a bit of tuning, the result for the test accuracy will be better. We can see that the loss and the cost are really smaller with batch normalization. The training is also faster.

9 layer network

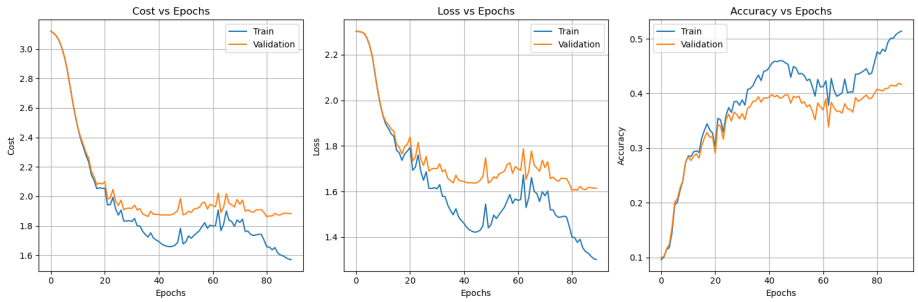

Figure 3: The training and validation loss for a 9-layer network without batch normalization with `lambda=.005`. The **test accuracy** is 42.27%. I used Xavier initialization.

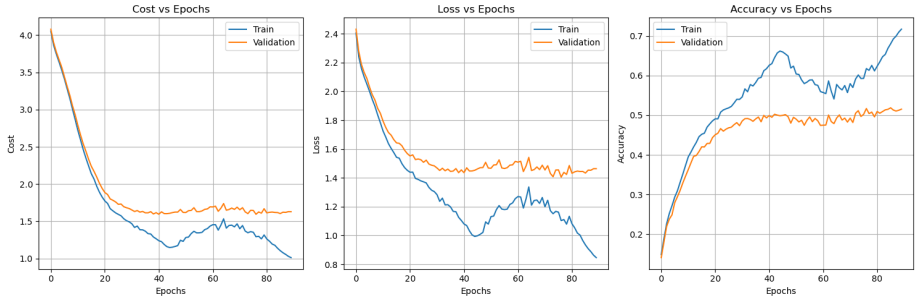

Figure 4: The training and validation loss for a 9-layer network with batch normalization with `lambda=.005`. The **test accuracy** is 51.03%. I used He initialization.

Here we can see that the batch normalization helps a lot for the training accuracy. Moreover, the final test accuracy difference is huge ! Almost 10% of difference. The training is also faster. We can see here the powerfulness of the batch normalization.

Lambda tuning

I re-used the technique of the previous assignment to tune the lambda parameter. I used the 3-layer network with batch normalization. I tried different values of lambda with `l_min=-4` and `l_max=-0.3` for the exponential search. I used an higher lambda because we were having great result with `lambda=.005`. I

compared the validation accuracy and each time take the best and the worst lambda from the 5 best training for the next fine tuning.
At the end of 3 rows of fine tuning. I get this results:

- `lambda=0.01634`, Accuracy: 0.5488
- `lambda=0.02036`, Accuracy: 0.5468
- `lambda=0.0226`, Accuracy: 0.5464

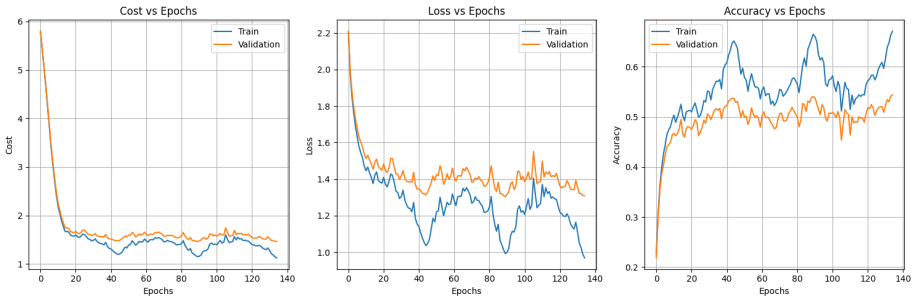

Figure 5: The training and validation loss for a 3-layer network with batch normalization with `lambda=.01634`. The **test accuracy** is 52.58%. I used He initialization.

There we can see that the test accuracy is 1.13% better than the previous same model with batch normalization. We can see that the lambda tuning can be really important.

And then I used the best lambda and used it for a 9-layer network. The result is:

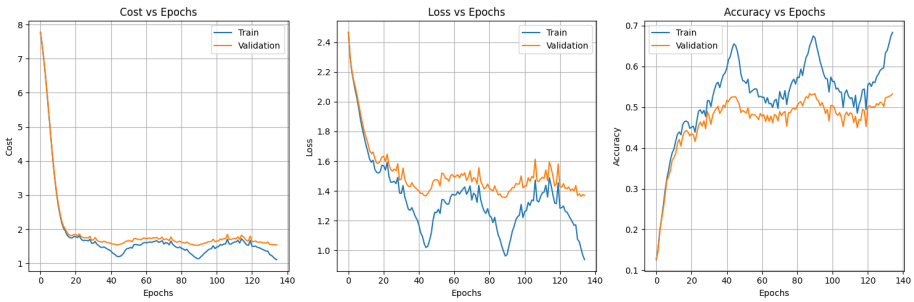

Figure 6: The training and validation loss for a 9-layer network with batch normalization with `lambda=.01634`. The **test accuracy** is 52.77%. I used He initialization.

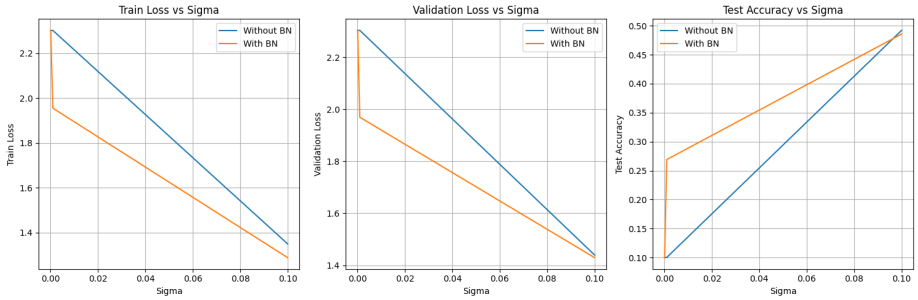

Figure 7: Sensitivity to initialization

Here again we gain 1.74% of accuracy with the lambda tuning which is really good.

Sensitivity to initialization

I tested the sensitivity to initialization for the 3-layer network with and without batch normalization by initializing the weights with gaussian distribution with mean 0 and standard deviation of 1e-1, 1e-3, 1e-4. I used `lambda=.005` for the 3-layer network.

We can see the all the networks are really sensitive to the initialization. Moreover, it is really interesting to see that the network with batch normalization is less sensitive to the initialization.