DD2424 – Assignment 3

Aloysius Chow

04/05/2023

Analytical gradient computation and tests

Testing between the analytical and the slow numerical method of computing gradients was performed to ensure accuracy. Relative error was calculated, and the analytical gradient computation was able to calculate gradient with the meanrelative errors shown below for the first 100 samples of the training set with lambda set to 0, for the three layer network with batch normalisation:

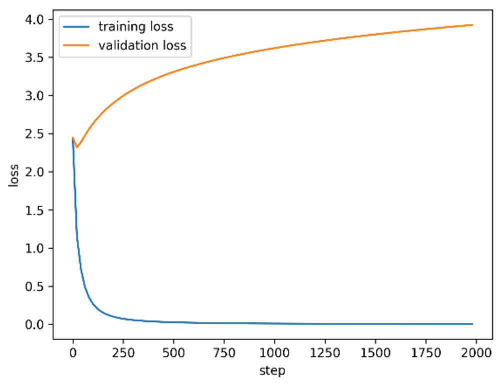
Parameters	Mean relative error
W1	0.005414455629139292
B1	1.080359863117337e-05
gamma1	6.515186950002155e-09
beta1	1.2223807096654742e-09
W2	1.2750894725570875e-08
B2	1.4901625225320459e-06
gamma2	3.9453241462885226e-10
beta2	3.086684973476819e-10
W3	1.1450432427165202e-08
B3	1.129795398785572e-09

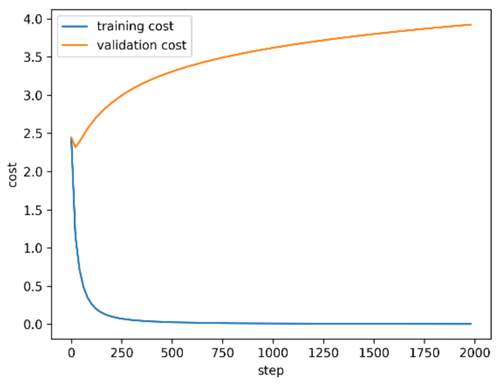
While the errors are larger for the earlier layers due to the amplification of small errors through the backward passes, the function written to perform analytical gradient computation seems to be correctly implemented, with a sufficiently small amount of error to proceed with training.

Overfitting test with multiple layers

Following this, a two layer model was trained to overfit on the training set to ensure that gradient calculations were being performed correctly, and that training loss could be reduced (in other words, to show that gradient descent was being performed correctly).

The loss and cost graphs for this overfitted model are shown below, and we can verify that the model was able to train correctly and overfit on the training data.





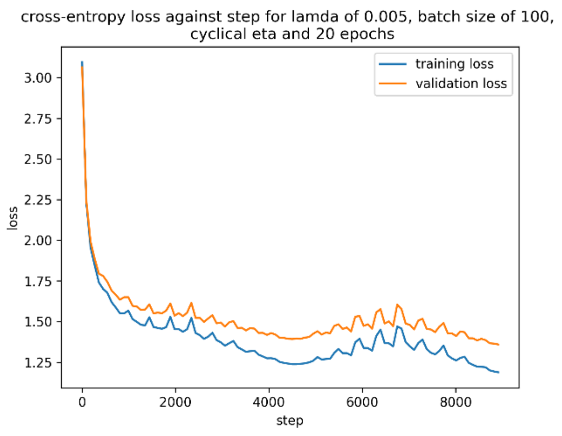
Tests with three layers

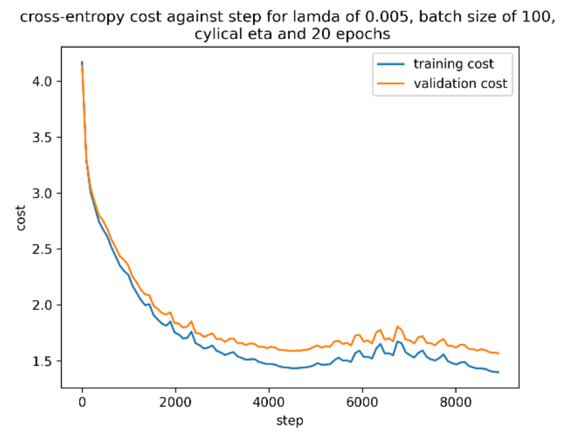
First, a model with three layers was trained without batch normalisation, with 50,50 and 10 nodes at each layer respectively.

The model was trained for 20 epochs with:

- lambda = 0.005
- n_batch = 100
- Cyclical learning rate of eta_min = 1e-5, eta max = 1e-1 and n_s = 5 * 45,000 / n_batch.

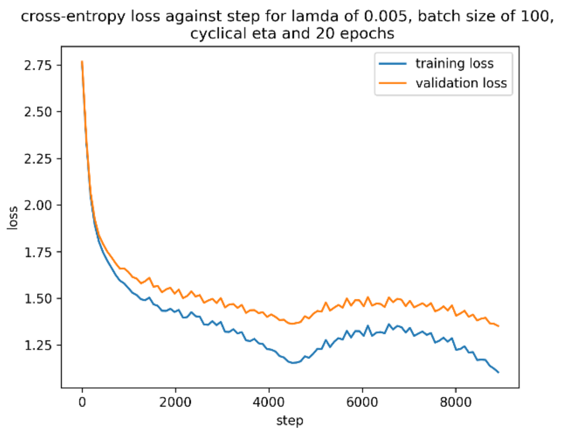
This resulted in two cycles of the cyclical learning rate function, and the model achieved an accuracy of 53.13%.

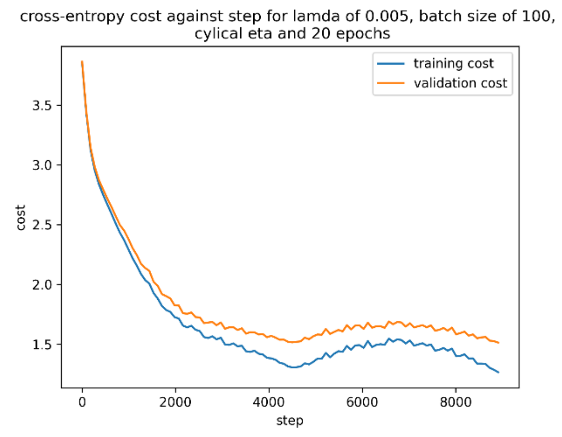




A new model was trained with the same parameters, except with batch normalisation enabled. The model managed to achieve an improved accuracy of 53.74%.

We can see that the loss and cost curves are smoothed out, suggesting that batch normalisation could indeed be aiding in the stability of training even with models that do not have many layers.





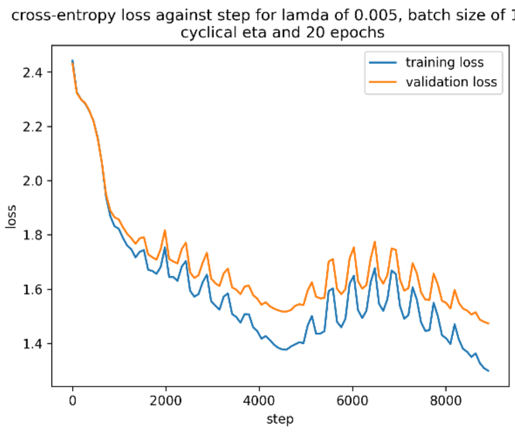
Tests with nine layers

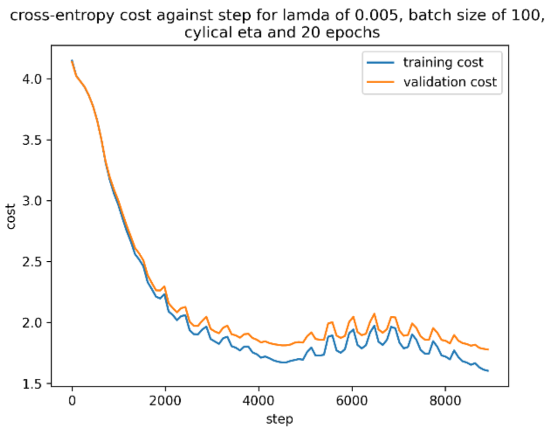
Again, a model with nine layers was trained without batch normalisation, with 50,30,20,20,10,10,10 and 10 nodes at each layer respectively.

The model was trained for 20 epochs with:

- lambda = 0.005
- n_batch = 100
- Cyclical learning rate of eta_min = 1e-5, eta max = 1e-1 and n_s = 5 * 45,000 / n_batch.

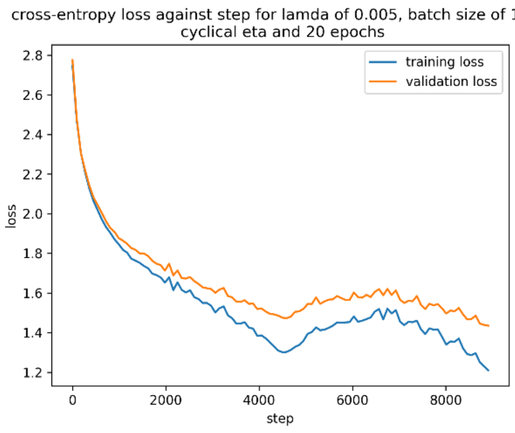
This resulted in two cycles of the cyclical learning rate function, and the model achieved an accuracy of 48.01%. As suggested in the assignment, it seems that training is very unstable and that the depth of the network is causing struggles in training.

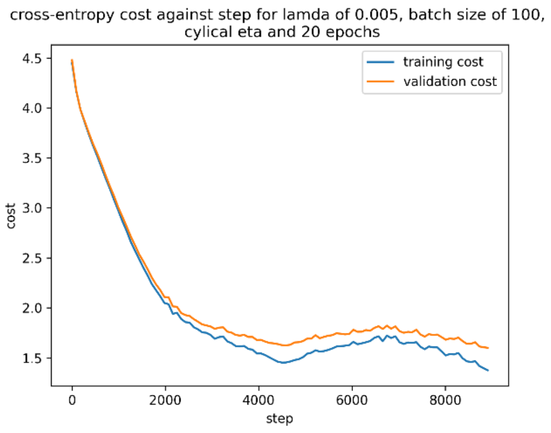




A new model was trained with the same parameters, except with batch normalisation enabled. The model managed to achieve an improved accuracy of 50.69%.

The great instabilities in the loss and cost curves are smoothed out even more so than with the three-layer model with batch normalisation, and it demonstrates the effect of batch normalisation more clearly in deeper models with a greater number of layers.





Hyper-parameter search for lambda

A search for the best value of lambda was carried out, to optimise the three-layer network with batch normalisation.

All models were trained for 20 epochs with:

- n_batch = 100
- Cyclical learning rate of eta_min = 1e-5, eta max = 1e-1 and n_s = 5 * 45,000 / n_batch.

First, a coarse search was performed for lambda, using $\lambda = 1 \times 10^{i}$ for 10 values of i, uniformly distributed from i = -5 to i = -1.

The results are shown below, with the values of i =-2.33333, -1.888889 and -3.22222 achieving the best accuracy.

i	average validation accuracy
-5	0.495
-4.55556	0.495
-4.11111	0.4938
-3.66667	0.5026
-3.22222	0.503
-2.77778	0.5066
-2.33333	0.5242
-1.88889	0.5154
-1.44444	0.5034
-1	0.4786

From the best 3 values, the maximum and minimum are taken as the new range of i for a fine search.

The fine search was performed for lambda, using $\lambda = 1 \times 10^{i}$ for 20 values of i, uniformly distributed from i = -1.888889 to i = -3.22222.

The results are shown below, with the values of i = -2.076023, -2.637427 and -2.169591 achieving the best accuracy.

i	average validation accuracy
-1.88889	0.5128
-1.93567	0.5168
-1.98246	0.5142
-2.02924	0.5126
-2.07602	0.5238
-2.12281	0.5142
-2.16959	0.5212
-2.21637	0.5112
-2.26316	0.5142
-2.30994	0.5176
-2.35673	0.5206
-2.40351	0.5188
-2.45029	0.5148
-2.49708	0.5082
-2.54386	0.5136
-2.59064	0.5128
-2.63743	0.5236
-2.68421	0.5122
-2.73099	0.5186
-2.77778	0.5104

From here, we selected the value of i = -2.076023, which performed the best with an accuracy of 52.38%.

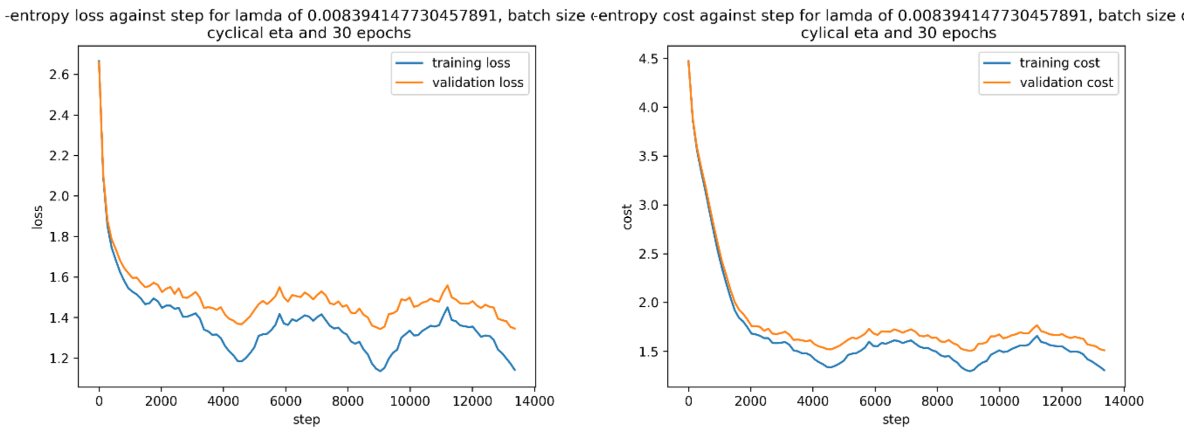
Final model

Finally, a model (with batch normalisation) was trained with the final value of lambda, $1 \times 10^{-2.076023}$.

The model was trained for 30 epochs with:

- lambda = $1 \times 10^{-2.076023}$.
- n_batch = 100
- Cyclical learning rate of eta_min = 1e-5, eta max = 1e-1 and n_s = 5 * 45,000 / n_batch.

This resulted in 3 cycles of training, and the loss and cost curves are shown below:



After training, the model achieved an accuracy of 53.06% on the test set. Interestingly, this is lower than the test accuracy achieved with lambda = 0.005, and suggests that it might be beneficial to perform the parameter search with even more values or to perform another round of fine search, given more time.

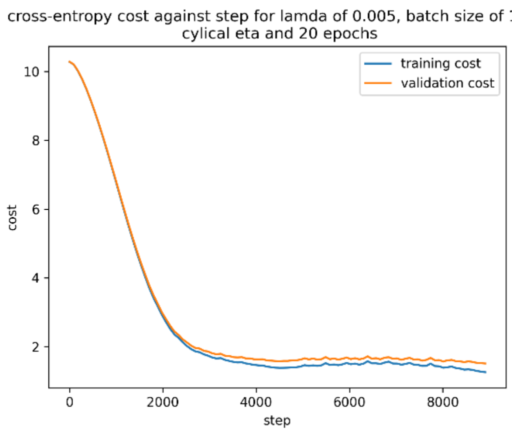
Sensitivity to initialization

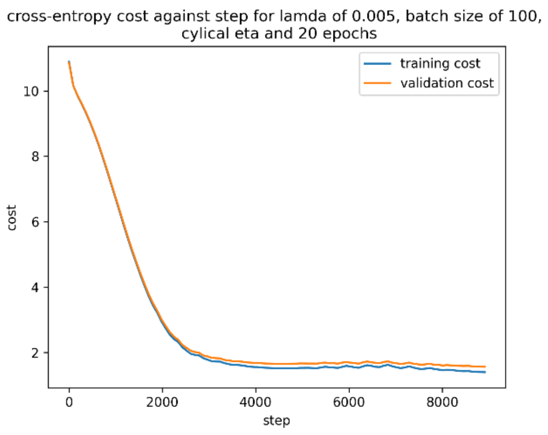
Next, experiments were run with initialisation of weights based on given sigmas, rather than using the He method.

These models were trained for 20 epochs with:

- lambda = 0.005
- n_batch = 100
- Cyclical learning rate of eta_min = 1e-5, eta max = 1e-1 and n_s = 5 * 45,000 / n_batch.

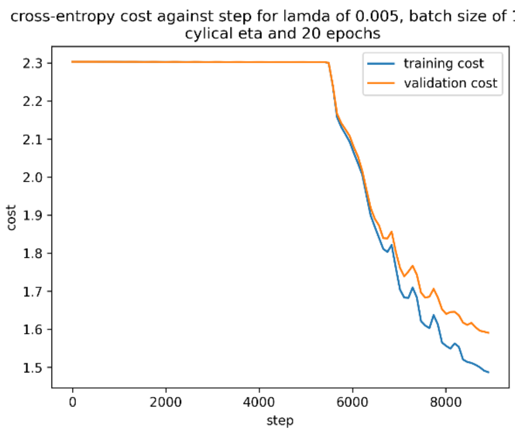
Here are the cost curves for sigma = 1e-1, without and with batch normalisation respectively.

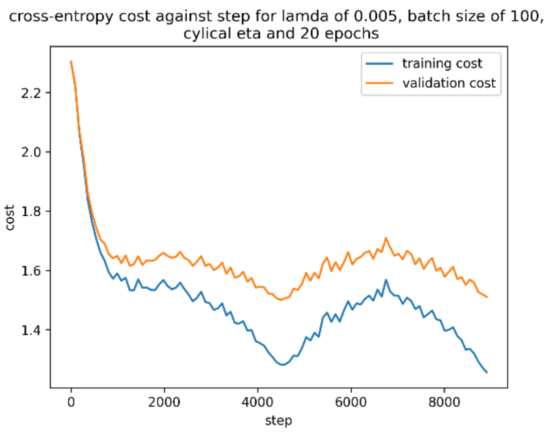




Batch normalisation did not seem to make a significant difference for sigma = 1e-1, and it seems that this value of sigma allows the model to perform and train well without the need for batch normalisation.

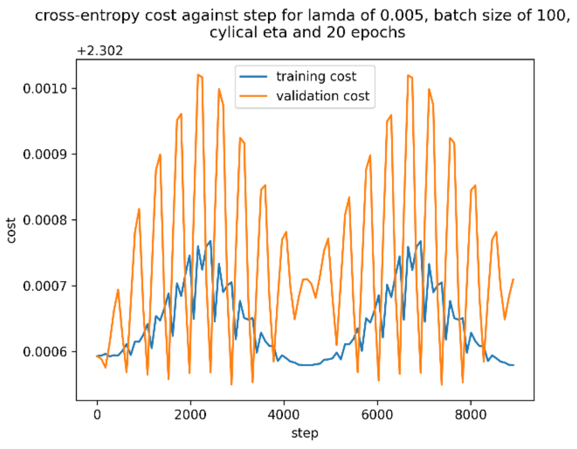
Here are the cost curves for sigma = 1e-3, without and with batch normalisation respectively.

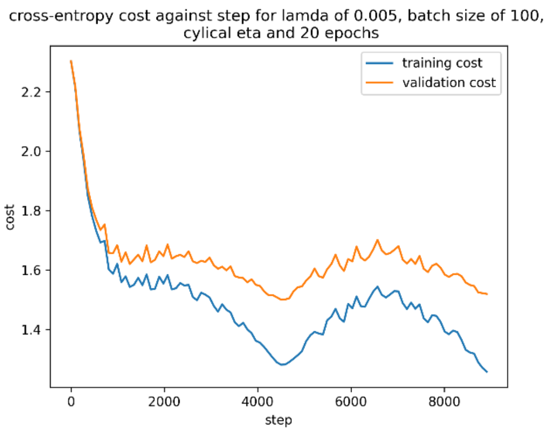




This time, we can see that the model struggles to make any progress without batch normalisation, with virtually no change in cost for the first 5000 steps. Batch normalisation helps to overcome this, and the model achieves a better test accuracy of 53.62% compared to 49.00% without batch normalisation.

Lastly, here are the cost curves for sigma = 1e-4, without and with batch normalisation respectively.





The differences are most dramatic for sigma = 1e-4, and the model is very unstable without batch normalisation. It seem as if the model never manages to make any progress beyond a random guess (hence the final accuracy of 10%), and the model is unusable. With batch-normalisation enabled, the model is able to train successfully and reduce its loss and achieves a final test accuracy of 52.88%.

Below are the test accuracies for all the above values of sigma, with and without batch normalisation.

sigma	batch-normalisation?	test accuracy
1e-1	No	52.66%
	Yes	52.58%
1e-3	No	49.00%
	Yes	53.62%
1e-4	No	10.00%
	Yes	52.88%

Conclusion

Introducing batch normalisation was a difficult task, as it required significant updates to the function for the back pass and gradient calculations. This was especially so for the function for the batch norm back pass.

However, it was interesting to see the performance benefits that batch normalisation could yield, by increasing the stability of training.