DD2424 – Assignment 2

Aloysius Chow

22/04/2023

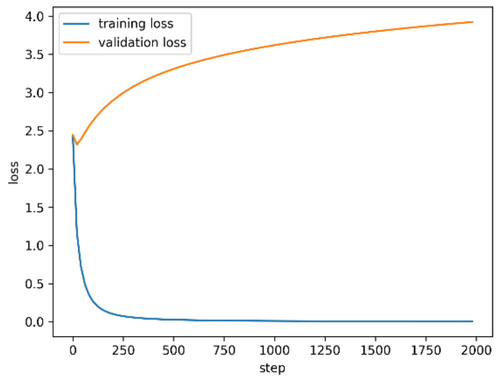
Analytical gradient computation and tests

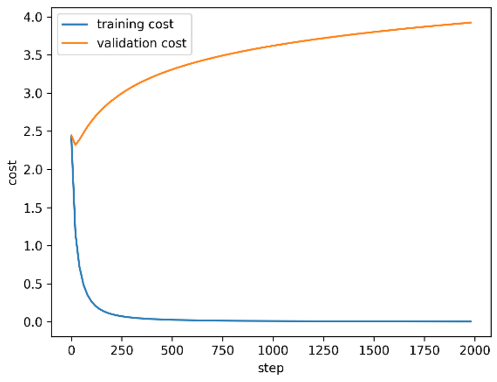
Testing between the analytical and numerical methods (both fast and slow) of computing gradients was performed to ensure accuracy, with varying number of entries and values of lambda. Relative error was calculated as per the instructions in Assignment 1, and the analytical gradient computation was able to calculate gradient to within 1×10^{-3} relative error for W1, W2, b1 and b2 respectively. Thus, we conclude that the function written to perform analytical gradient computation was correctly implemented, with a sufficiently small amount of error to proceed with training.

Overfitting test

Following this, a 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.





Cyclical learning rate

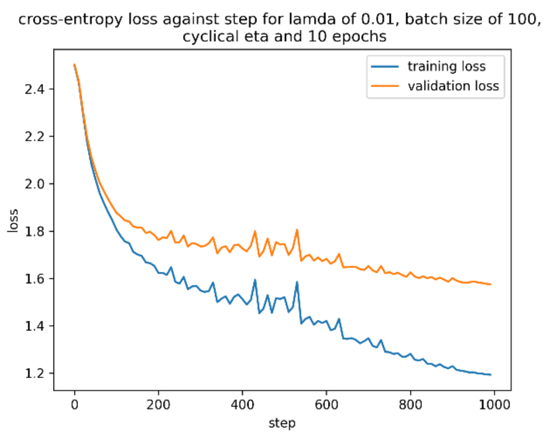
A function was written to perform mini batch gradient descent with a cylical learning rate, as described in the assignment instructions.

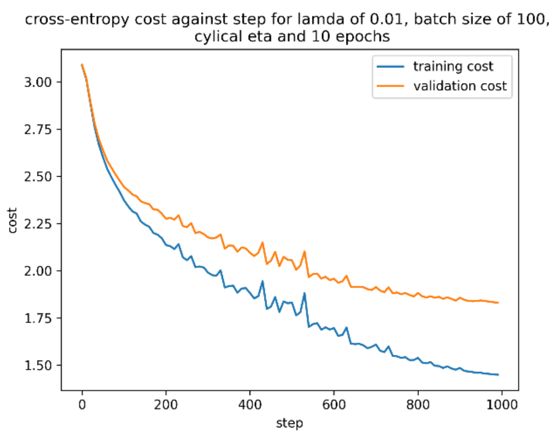
First, a model was trained following the parameters described for Figure 3 of:

eta_min = 1e-5, eta_max = 1e-1, lambda = 0.01, n_s = 500, n_batch = 100, n_epochs = 10

This resulted in one cycle of training.

The loss and cost graphs for the model are shown below (loss and cost were sampled more frequently than in the example plots, thus the curves shown are less smooth). We see a similar shape to the example curves, with spikes and decreases in loss and cost in accordance to the one learning rate cycle.



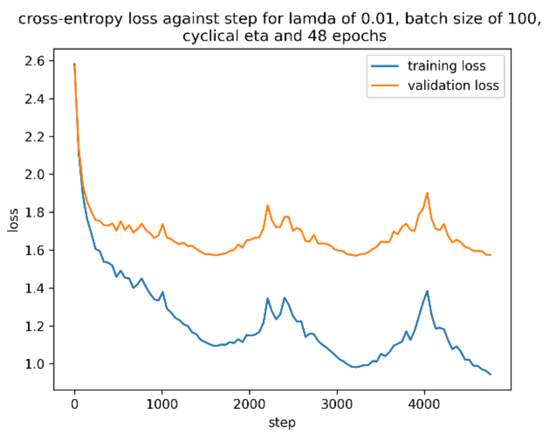


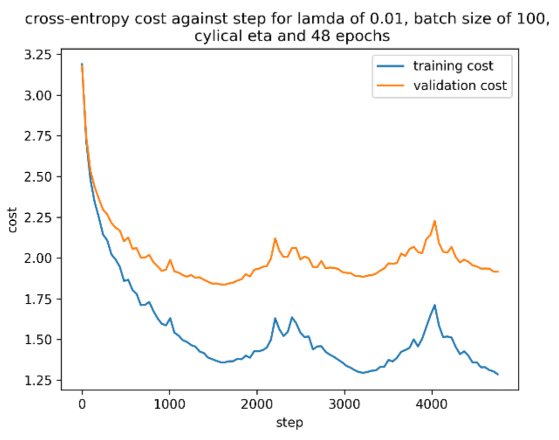
Next, a model was trained following the parameters described for Figure 4 of:

eta_min = 1e-5, eta_max = 1e-1, lambda = 0.01, n_s = 800, n_batch = 100, n_epochs = 48

This resulted in three cycles of training.

The loss and cost graphs for the model are shown below (loss and cost were sampled more frequently than in the example plots, thus the curves shown are less smooth). We see a similar shape to the example curves, with spikes and decreases in loss and cost in accordance to the three learning rate cycles.





Compared to the cyclical graph of the learning rate, as shown below in a test plot to ensure the cylical learning rate function was working correctly, we can observe that loss tends to spike in the increasing portion of the learning rate cycle, and that the loss decreases in the decreasing portion of the learning rate cycle.

Hyper-parameter search for lambda

From here on, data from all 5 batches is used, with 5000 samples randomly split for validation using Scikit-learn's train_test_split function.

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, -2.77778 and -3.22222 achieving the best accuracy.

i	average validation accuracy
-5	0.4996
-4.55556	0.492
-4.11111	0.4998
-3.66667	0.498
-3.22222	0.5022
-2.77778	0.5072
-2.33333	0.5116
-1.88889	0.5022
-1.44444	0.4546
-1	0.376

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

The 1st fine search was performed for lambda, using $\lambda = 1 \times 10^{i}$ for 10 values of i, uniformly distributed from i = -2.33333 to i = -3.22222.

The results are shown below, with the values of i = -2.33333, -2.62963 and -2.53086 achieving the best accuracy.

i	average validation accuracy
-2.33333	0.5184
-2.4321	0.5146
-2.53086	0.5158
-2.62963	0.5166
-2.7284	0.5088
-2.82716	0.505
-2.92593	0.5044
-3.02469	0.503
-3.12346	0.496
-3.22222	0.4984

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

Finally, the 2nd fine search was performed for lambda, using $\lambda = 1 \times 10^{i}$ for 20 values of i, randomly sampled with a uniform distribution between the values of -2.62963 and -2.33333.

i	average validation accuracy
-2.61905	0.521
-2.6019	0.5182
-2.59992	0.5174
-2.59831	0.5172
-2.57039	0.5146
-2.55739	0.5122
-2.50443	0.5168

-2.4881	0.5188
-2.47945	0.5132
-2.43978	0.5116
-2.4354	0.5186
-2.41775	0.5226
-2.40755	0.512
-2.40511	0.5094
-2.39455	0.5226
-2.38241	0.5174
-2.373	0.5216
-2.3649	0.5236
-2.35029	0.519
-2.33403	0.5218

From here, we selected the value of i = -2.3649, which performed the best with an accuracy of 52.36%

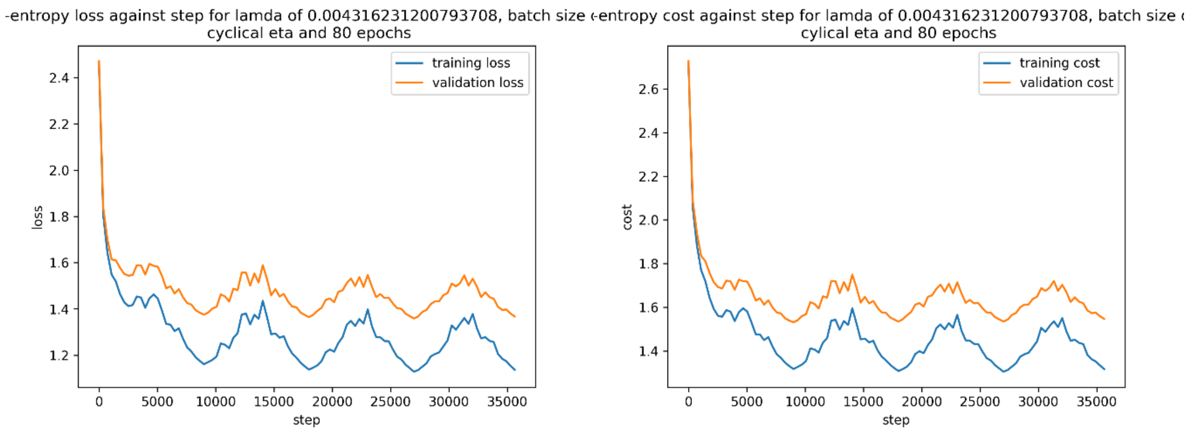
Final model

Finally, a model was trained with the final value of lambda, $1 \times 10^{-2.3649}$.

This model was trained with:

eta_min = 1e-5, eta_max = 1e-1, lambda = 0, n_s = 800, n_batch = 100, n_epochs = 80

This resulted in 4 cycles of training, and the loss and cost curves are shown below:



After training, the model achieved an accuracy of 52.77% on the test set, which is the second best performance that I have managed so far on this test set from Assignment 1 till now.

As a learning point, I initially performed hyperparameter tuning by selecting based on the best test accuracy by mistake, which resulted in a higher accuracy of 53.22%. Re-reading the assignment instructions, I realised that we were only supposed to select our hyperparameters based on the validation accuracy, and I realised this was important because performing hyperparameter tuning with the test set would result in a form of data leakage and not be representative of the true performance of the model, without prior knowledge of the test set.

Conclusion

The results show the importance of having an appropriate learning rate, as one that is too large could lead to failure to find the local minima and result in an inability to reduce loss.

At the same time, it also shows the role of regularisation in ensuring that the model we train is able to pick up on the features we consider important while avoiding overfitting.

Lastly, this assignment also served as an introduction for me on using Numpy to perform matrix calculations for me, and it forced me to better visualise what each matrix means when performing evaluation, forward passes or backward passes.