DD2424 Assignment 2 Report

Arturs Kurzemnieks (artursk@kth.se)

April 2019

This assignment was implemented in Python using a Jupyter notebook. The code attached with the assignment is a compilation of the code used in the notebook.

1 Gradient computations

The gradient computation for multi-layer setup was modified accordingly to the course materials. Implementation was seemingly successful, giving good results. For checking if the analytical gradients are correct, I reimplemented the numerical gradient method provided (finite difference method) and compared the results using element-wise absolute differences for biases and weights of both layers, computed as

```
// Example for comparing gradients of b for one layer
np.abs(grad_b[0].T - ngrad_b[1]) / np.maximum(1e-5, np.abs(grad_b[0].T)
    + np.abs(ngrad_b[1]))
```

These gradients were computed using a reduced number of **20 input dimensions**. For biases, the differences didn't exceed order of **1e-6**, for weights didn't exceed order of **1e-5**

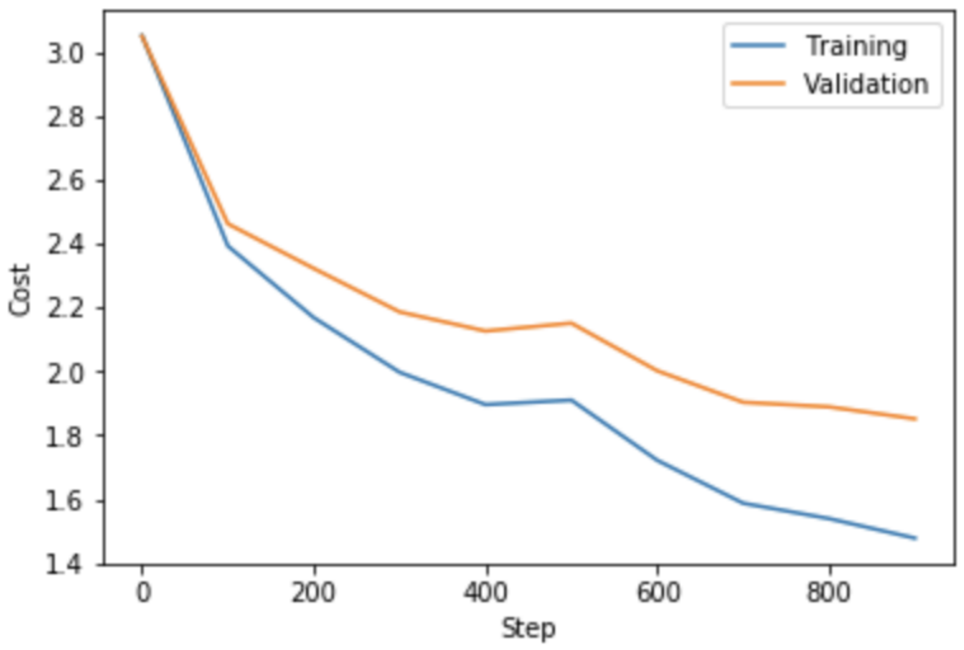
```
\\ Example output for b_2:
2.24521925e-06 2.26164420e-06 2.23953034e-06 2.24687523e-06
2.24781743e-06 2.25052211e-06 2.47856810e-07 2.25262913e-06
2.25260408e-06 2.25075809e-06
```

The computed analytical gradients are practically usable and good enough for stable training to achieve results as described in the assignment.

2 Cyclical learning

The cyclical learning rates were implemented succesfully. The following training cost curves were produced for the two basic scenarios:

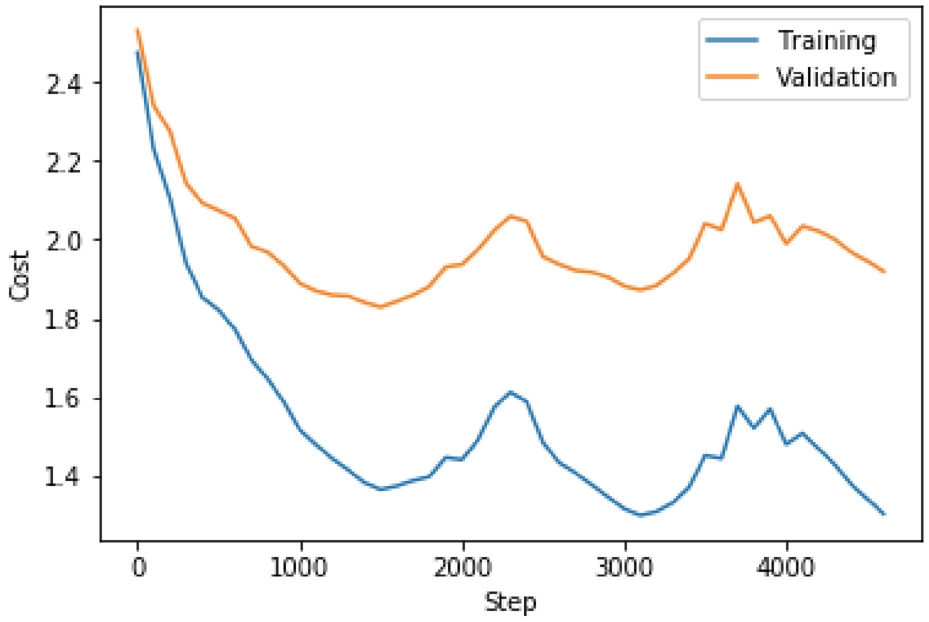
2.1 Cost plot, $n_s = 500$, 1 cycle



```
Test cost:  1.8086210412224772
Test accuracy:  0.4591
```

During this one cycle nothing specific happens, the training performs better initially, then as the learning rate is increased, it plateaus.

2.2 Cost plot, $n_s = 800$, 3 cycles



```
Test cost:  1.8752230784352144
Test accuracy:  0.4719
```

3 cycles of learning were done here. It can be seen that in the middle of 2nd and 3rd cycle (around 2400 steps and 4000) where learning rate is at its highest, the cost also increases as the learning rate gets too high.

3 Lambda - coarse search

For coarse search of lambda I used $l\_min = -7, l\_max = -0.5$, with 40 samples in this interval and 2 cycles of training for each.

Best results:

```
Lambda: 1.9433438313018646e-07 (l: -6.711450353889899),
 test cost: 1.3919308043169243,
 test accuracy: 0.517
---
Lambda: 0.000814024070237211 (l: -3.0893627530756094),
 test cost: 1.4493419714745006,
 test accuracy: 0.5151
---
```

```
Lambda: 0.006088551747151061 (l: -2.2154859985033317),
test cost: 1.533140238496957,
test accuracy: 0.5135
```

Although there was a high result with $l = -6.7$, I chose to explore $-3.2 - - - 2.2$ interval, as it seemed to have more stable results.

4 Lambda - fine search

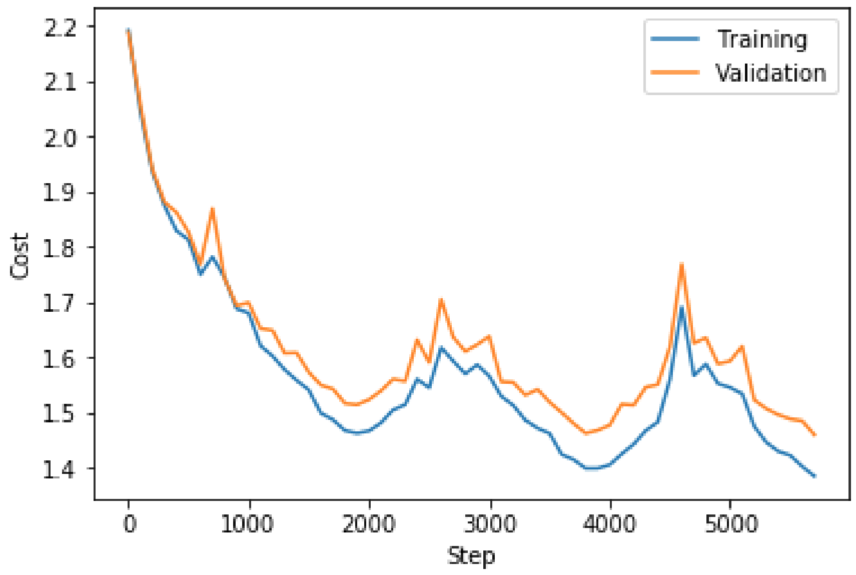
For the fine search I used $l\_min = -3.2, l\_max = -2.2$, with 40 samples in this interval and 2 cycles of training for each.

Best results:

```
Lambda: 0.0007259585887334052 (l: -3.13908815229924),
test cost: 1.4462508747087506,
test accuracy: 0.5177
---
Lambda: 0.0012831058967807164 (l: -2.8917374991280846),
test cost: 1.4684733861665245,
test accuracy: 0.5166
---
Lambda: 0.004728919109833769 5 (l: -2.325238114712273),
test cost: 1.5180556285495759,
test accuracy: 0.517
```

5 Best result

I tried further training for all three of the best lambdas, i.e., with all but 1000 samples for training with 3 cycles. Best result was achieved with **l = -2.325238114712273, lambda = 0.0047289191098337695**



```
Test cost:  1.5058642198595416
Test accuracy:  0.5204
```