1XC3 Final Project Readme Updated

Greg Forster

August 2023

Table 1: epochs = 100000, train split = 0.003, **Learning Rate (LR) is variable**

Test	Cost Train	Cost Validation	Train Accuracy[%]	Validation Accuracy[%]
0.0005	0.446292	0.477919	58.33%	55.41%
0.001	0.151618	0.338970	88.89%	74.53%
0.005	0.036326	0.388433	97.92%	76.87%
0.01	0.004250	0.436481	99.31%	75.13%

Table 2: Learning rate = 0.005, train split = 0.003, **epochs is variable**

Test	Cost Train	Cost Validation	Train Accuracy[%]	Validation Accuracy[%]
100	0.477319	0.506892	58.33%	49.98%
1000	0.459160	0.487961	58.33%	55.70%
10000	0.440711	0.480684	63.89%	57.23%
100000	0.016085	0.455258	98.61%	73.68%

Learning Rate:

We can see pretty quickly see signs of overfitting when the learning rate is variable (not necessarily the fault of the learning rate; it just uncovers the faults in our model quickly). Our training accuracy approaches 100%, which means that our learning rate is accurate for our training set, but it unfortunately doesn't translate well to the rest of the dataset.

Epochs:

If we have a correct learning rate, we need enough epochs to see our model converge! If our learning rate is too big, sometimes it'll be better if we have less epochs, because of overshooting. In our example, it wasn't until we had 20000 epochs was it relatively successful. It was at 78% validation accuracy at around 20000 epochs and then slowly dropped until 100000 epochs.. indicating overshooting.

Table 3: epochs = 100000, Learning rate = 0.005, **train split (TS) is variable**

Test	Cost Train	Cost Validation	Train Accuracy[%]	Validation Accuracy[%]
0.0003	0.000041	0.896091	100.00%	53.25%
0.003	0.019103	0.454344	98.61%	73.22%
0.005	0.005196	0.411600	99.31%	76.90%
0.01	0.043178	0.307131	97.51%	81.67%
0.1	NA	NA	NA	NA

Train split:
0.1 ends up with running out of space on the stack. I also answered this in the evaluation question in main.c

Overfitting:
I used 20 neurons for the first hidden layer and 10 neurons for the second hidden layer. My learning rate was 0.003 and my epochs were 30000, as it began to overshoot after that. My training split was 0.015. With this I achieved

I struggled with getting the validation accuracy any higher. If I increased my training set then I wasn't able to get the model to converge to a decent accuracy...

epoch 10000: Train Cost 0.478174, Accuracy: 56.17% Validation Cost 0.477423, Accuracy: 57.01%

epoch 20000: Train Cost 0.220230, Accuracy: 84.60% Validation Cost 0.237002, Accuracy: 83.16%

epoch 30000: Train Cost 0.160906, Accuracy: 88.35% Validation Cost 0.205068, Accuracy: 85.59%

Python versus C

Tensorflow and other frameworks will always be much easier than using C! You no longer have to worry about memory management and a lot of the tools are abstracted away. There are some people even working on making ML frameworks faster and more efficient, such as tinygrad! tinygrad

C also has it's advantages, especially for neural networks. The usual bottleneck to creating performant models is speed! That's why ML engineers have begun using GPUs over the last couple of decades because they are designed to run computations in parralel. Specifically for C, it is incredibly performant and the compiler is able to optimize it more than other lanaguages.