Exercise 1 – Report

Antigen Discovery for SARS-CoV-2 (“Corona”) Virus Vaccine

# Data Handling

Each data item is a 9-lengthed sequence of characters, and all of the characters are in a vocabulary of 20 characters. We decided to map each sequence to 9\*20 vector – concatenation of nine (one for character) one-hot encoding vectors, each one is of size 20. This seems like a natural choice because we do not have prior assumptions about the data, we want to keep the order and we do not know if there is any connection between some of the characters.

# Models Architectures

We will try multi-layered perceptron networks with different parameters. The parameters are numbers of layers, number of neurons in each hidden layer and activation function. In all cases the first layer will get 9\*20-sized input vector and the last layer will output 1-sized output vector after going through sigmoid activation. In inference, we round the output, so effectively the threshold is 0.5.

# First Training

For the first training, we tried feed forward network with one hidden layer with 128 neurons with relu activation. We trained the network for 5 epochs. We used batch size of one for simplicity.

I used binary cross entropy loss and learning rate of 0.001.

Results on the test set:

Accuracy: 0.917

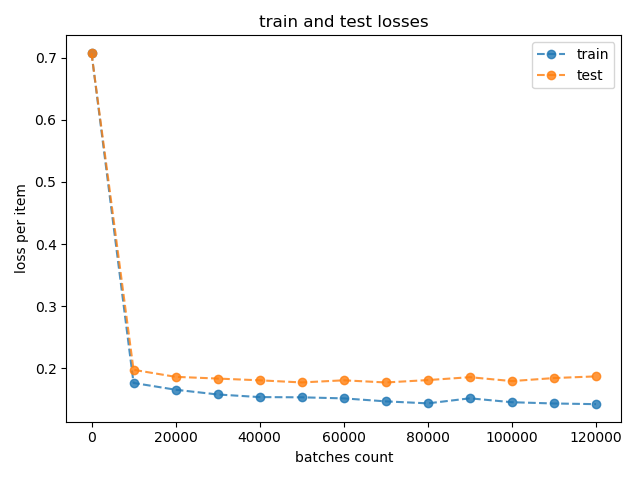
Recall: 0.968

Precision: 0.941

f1: 0.954

We can see that the results are fine – all of the metrics I checked are above 90%. Especially it is good to see that both the recall and the precision are pretty high even though the data set is imbalanced.

Those are the learning curves:



According to those results, I decided that comparing the different architectures with 2 epochs for each (about 50,000 batches) would be enough.

## Architectures Comparison

I started by trying different numbers of hidden layers and neurons in each layer. The neurons numbers series I chose are geometric for effective searching.

I tried:

1. 0 hidden layers
2. 1 hidden layer of sizes: 8, 16, 32, 64, 128, 256
3. 2 hidden layers from same size each: 8, 16, 32, 64, 128, 256

For now, I follow the instructions and look at the accuracy (and not recall or precision).

We looked at the learning curves as sanity check and they were just fine, similar to first part the curve I showed before, just as expected.

The results are:

|  |  |  |
| --- | --- | --- |
| **# hidden layers** | **# neurons in each hidden layer** | **accuracy** |
| 0 | - | 90.9% |
| 1 | 8 | 91.4% |
| 1 | 16 | 92.4% |
| 1 | 32 | 92.4% |
| 1 | 64 | 92.6% |
| 1 | 128 | 91.9% |
| 1 | 256 | 91.4% |
| 2 | 8 | 91.7% |
| 2 | 16 | 92.7% |
| 2 | 32 | 91.6% |
| 2 | 64 | 92.7% |
| 2 | 128 | 92.5% |
| 2 | 256 | 93.6% |

We can see that there are differences, but they are not dramatic – the accuracies from 90.9% to 93.6%.

The worst result (90.9%) is with the small network – without hidden layers, and the best one (93.6%) is with the biggest network – two hidden layers of 256 neurons each.

For each positive number of hidden layers I plotted the results as function of the number of neurons in each layer.

For one layer the best results are with medium number of neurons, and for two layers the best results are with many neurons. Anyway, the trends are not very clear, and maybe much of what we see here is only statistic noise.