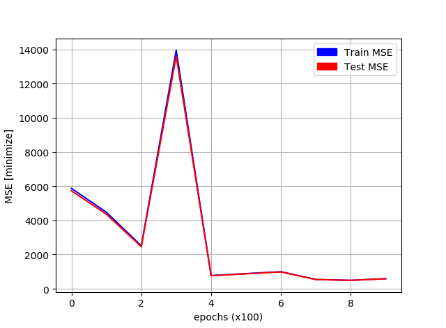
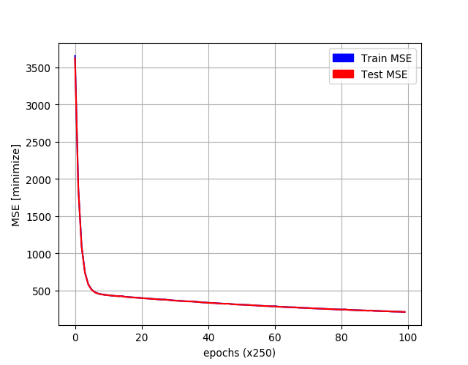
Linear regression and MLP models

# Submitting: Arad Zekler : 305600579, Ido Kahana : 316606680

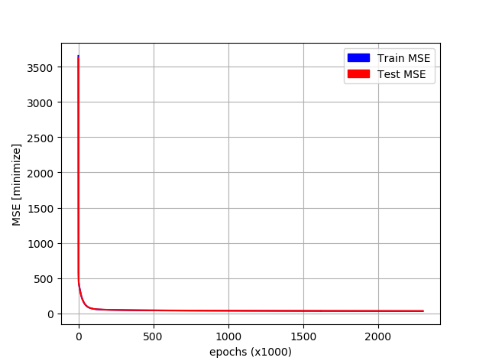


The first issue we faced is how to represent the data itself. We decided on features (such as city, school, profession) and converted them into numeric values. For the training itself we split the data and shuffled it, there was too much data to learn in a single step (at least for our machines) so we used mini-batch training. It is important to note that the machine that the linear regression model ran on used tensor-gpu to speed things up.

Initially the learning rate was too high (figure 1) (0.00018 learning rate) and it caused the model to have infinitely big loss. we had too much data to cause big overfitting to the model and with over 50k records, about 1k features we tried bad learning rate that give us not stable graph (0.00018) (figure 1)

Figure

In the next experiment (figure 2) , and 25k trains in the linear model we reached a training loss of 200 with a learning rate of 0.0001.

After 27950 Runnings (figure 3) with the same configuration we started to experience a very small overfitting and noticed that the train loss was overtaking the test loss by a very small margin. It continued that way until in reached a loss of about 33.52 train and 35.46 test after 39125 seconds of runtime and 2.3Million runs with batch size of 1000, and print each 1000, it took 300k runs to get down from 34.5 to 33.5 so it is probably the limit ded end with simple linear regression.

Figure

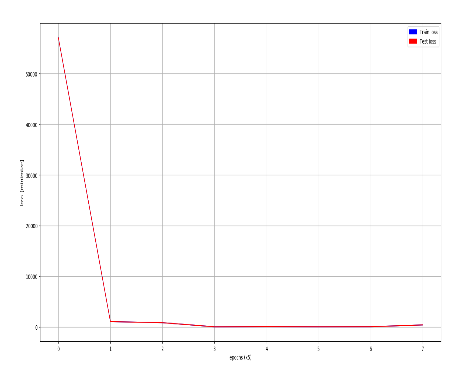
For the neural-net model we used a simple MLP, in the first experiment (figure 4) we ran the model with input neuron, two hidden layers and an output layer. Loss is calculated with absolute value instead of squared to ‘ignore’ edge-points in the data (such as a test with very small test-takers). I used Adam as optimizer and an initial learning rate of 0.05. **Starting Loss (Epoch: 0005)**: Train: 57065.85547, Test: 57033.65625, **End Loss: (Epoch: 0040)**: Train loss: 398.05978, Test loss: 397.83340

Figure 4

Figure

We tried (figure 5) lowering the model to just one hidden layer composed of 7 neurons (one per feature), and as the results show, dumbing down the network doesn’t necessarily always help: **Starting Loss (Epoch: 0005):** Train: 14603.95410, Test: 14595.69629, **End Loss: (Epoch: 0040):** Train loss: 98252.92218, Test loss: 97851.91423.

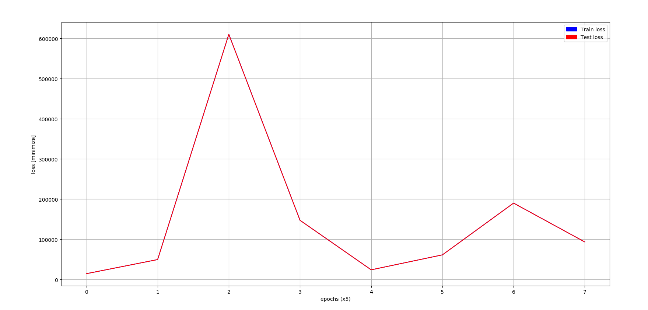
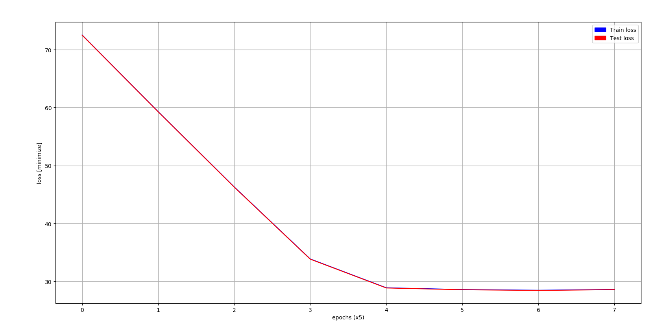


Figure 5

In the next experiment (figure 6) we tried the opposite: two hidden layers composed of 7 neurons each with the 2 layers contain ReLu and the first contains dropout. The results now were more promising: **Starting Loss (Epoch: 0005):** Train: 77.73695, Test: 77.72823, **End Loss: (Epoch: 0040):** Train loss: 28.58452, Test loss: 28.54260.

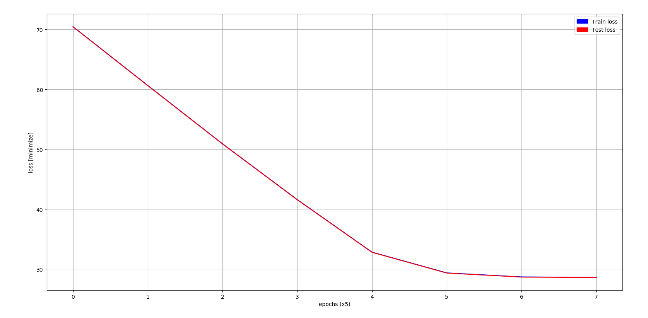
We tried to improve it further (figure 7), with first hidden layer with ReLu+ dropout and the second layer with double the neurons (14) and ReLu+dropout.

Figure 6

**Starting Loss (Epoch: 0005):** Train: 70.47420, Test: 70.43286, **End Loss: (Epoch: 0040):** Train loss: 28.62569, Test loss: 28.58469.

Surprisingly the model started with smaller loss but again stopped after achieving about 28 loss, making this model and the model in figure 6 the ones with the lowest loss.

Figure 7