Deep Learning Lab Course – Report for assignment 2

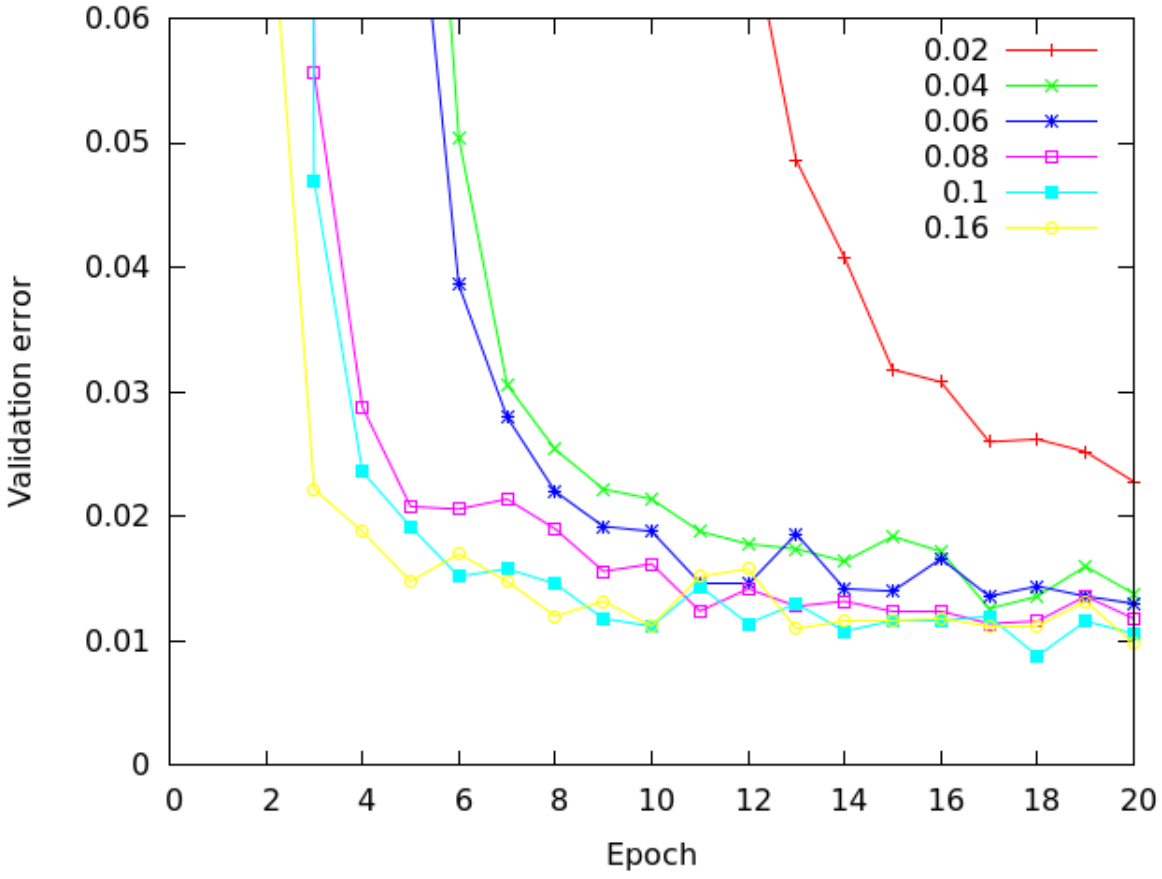
1 Implementing a CNN in Tensorflow

My implementation for a neural network class supporting convolutional and pooling layers can be found in `neural_network.py`. The specification for a particular network must be stored in a JSON file, and can be given to the program as a command line argument. The specification for the convolutional network used for this assignment can be found in `convolutional_nn.json`.

2 Changing the Learning Rate

The measurements show that with any learning rate below 0.01, the optimization procedure stagnates and validation error does not decrease within the first 20 iterations (loss decreases only minimally). Therefore I decided to try different values for the learning rate than the ones given on the exercise sheet, in particular $\{0.01, 0.02, 0.04, 0.06, 0.08, 0.1, 0.12, 0.14, 0.16\}$. The learning curves for most of these values are visualized in figure 1.
One can see from the curves for the values between 0.04 and 0.16 that the learning becomes a bit slower when decreasing the learning rate, therefore a larger learning rate (≥ 0.1) would probably be the better choice. However, the differences decrease over time and the curves converge to similar validation errors in the final iterations. Only when going below a learning rate of 0.02, the learning starts to become abruptly slower. I am not sure why this happens, and I would have expected a rather smooth decrease in performance, as can be seen for the larger learning rates.

Figure 1: Learning curves for different learning rates



3 Runtime

For the runtime measurement, I let the convolutional network run on the GPU for 20 epochs. Due to a lack of memory, 64 was the largest number of filters I tested. I did not measure the runtimes for the convolutional network on the CPU for a full run with 20 epochs, as each epoch takes painfully long (more than half a minute for just 8 filters per convolutional layer), and therefore just measured the runtime for 2 epochs and multiplied it by 10. The scatter plots for the runtimes on the GPU and CPU are shown in figures 2 and 3 respectively.

Figure 2: Runtimes on a GPU

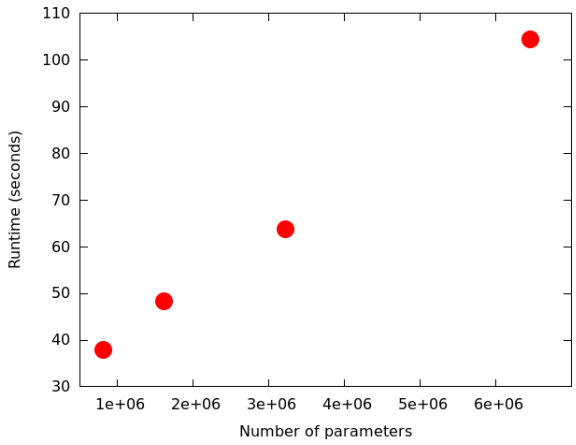
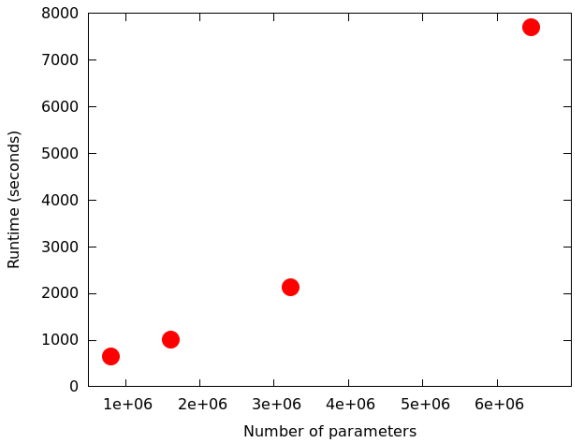


Figure 3: Runtimes on a CPU



One can observe that the training time on the GPU grows linearly with the number of parameters (which in turn grows linearly with the number of filters per layer). On the CPU, the plot appears to be growing rather exponentially, although this may also be influenced by other factors: e.g. too few measurements, memory problems or approximation errors (since I measured only 2 iterations and multiplied by 10). Anyhow, the runtime difference between GPU and CPU is significant: training times on the GPU are at least 20× smaller.