Deep Learning Lab Course – Report for assignment 1

Finding a good neural network

First of all, I noticed that different initial weights can heavily influence the training result (e.g. training error varying between 44 and 66 percent for a simple network with a single layer of size 100). Therefore I decided to keep weights fixed (by seeding the random generator) and use standard gradient descent throughout the experimentation phase to make the results more deterministic. Data for the results gathered during experimentation can be found in the `results.csv` file.

I started with a network with just one intermediate layer (apart from the input and output layer) and gradually adjusted the number of units to measure the effect of layer size on the learning procedure. As rectified linear units are said to give good results in many cases, I decided to stick to ReLU as activation function for the first few experiments. The result is visualized in the following diagrams:

Figure 1: Training loss for increasing layer size

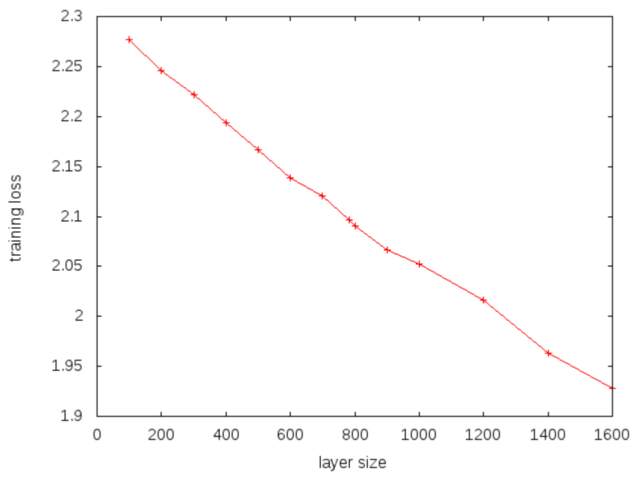
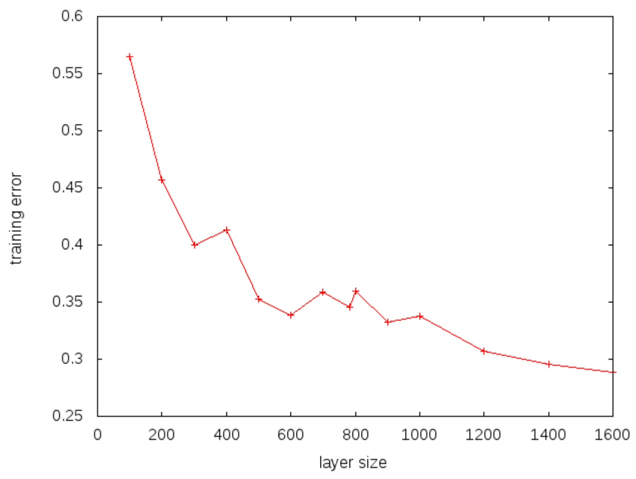


Figure 2: Training error for increasing layer size



One can see that adding more units to a single intermediate layer linearly leads to lower training loss, and usually decreases training error as well (with decreasing slope). Next, I tried to decrease the layer size a bit and add more layers instead.

Adding a second layer (still with ReLU) made the optimization procedure completely stagnate and training loss decrease only by around 0.0002 in each iteration, with the validation and test error usually staying constant (at a high value, around 89 percent). Sometimes the error decreased in the first iterations but then increased again and converged back to the usual high value of around 89 percent. I tried a few combinations including further layers with different sizes, but the result was always about the same.

So I went back to a single intermediate layer, fixed it at 600 units (which I chose as tradeoff between low classification error and acceptable runtime for experimentation) and tried the other activation functions (sigmoid and tanh). Using sigmoid led to a similar result as adding more layers, i.e. the classification error stagnated at around 89 percent as the training loss decreased only by a rather small amount. tanh on the other hand lead to a very good result: compared to the same network using a ReLU, both the training error and validation error decreased by approximately 4 percent. Using a layer with only 100 units, the errors were even up to 15 percent smaller than for the equivalent ReLU network.

Therefore I switched to tanh as activation function, and again tried to add more layers with different sizes. Unlike the experiments with ReLU, adding a second layer didn't make the optimization procedure stagnate (but adding a third layer did). However the results were still worse compared to a single-layer network.

Lastly, I tried to speed up convergence. I increased the learning rate in steps of 0.1 (starting from 0.1). When choosing a value larger than 0.6, the loss started diverging rather quickly, so I decided to stay below that. I also switched to stochastic gradient descent (since standard gradient descent started taking over 10 minutes without even going below 10 percent classification error).

Since I am running out of time, I did not try adjusting the standard deviation for the random weight initialization. I also did not try experimenting with the parameters for stochastic gradient (batch size and learning rate decay) or implement any kind of regularization.

Final neural network and results

My final neural network contains a single intermediate layer with 600 units and tanh as the activation function, optimized using stochastic gradient descent with a learning rate of 0.4 (no decay) and a batch size of 64, trained until the error on the validation set is below 3 percent (or 30 iterations at most, each iteration involving multiple parameter updates).

Running `exercise_1.py` will train my chosen network on the MNIST data, evaluate it on the test set, and print the results. On my machine it took my neural network a little more than 3 minutes (5 SGD iterations) to reach a validation error less than 3 percent (specifically 2.7 percent). The error on the test set was similar (2.75 percent). Allowing the network to do more iterations would have probably further decreased the error, as the loss and training/validation errors did not show any signs of having converged and were still making large improvements in the final iterations.