Deep Learning DD2424 - Assignment 2 Bonus Points

Anton Stagge

April 2019

1 Optimizing model

1.1 Increasing number of hidden nodes

For reference I trained a model with 50 nodes on the training data from the first batch only (#10000) for 3 cycles and with lambda set to 4.1×10^{-5} as in the previous model. It had test accuracy of 44.69 percent. I then trained a model with 80 hidden nodes, with the same hyper-parameters it got a test accuracy of 46 percent. As you can notice in Figure 1, the validation loss remains almost the same, were as the training cost gets substantially lower, to reduce this over-fitting I raise lambda to 8×10^{-3} and got a test accuracy of 46.95 percent and reduced the over-fitting substantially. I also tries increasing the number of hidden nodes to 100 but this took substantially longed to train and had about the same accuracy as the model with 80 hidden nodes.

1.2 Ensamble learning

I saved the weights and biases from every model where *eta* was *eta_min*, during my training of 3 cycles, and thus ended up with 3 models in the end. I the used there three models to classify the test data, and used majority vote for the final classification. This was quite easy to do, but in the end i did not see any improvement in performance, quite the opposite. In most cases the ensamble model performed slightly worse than the single model in the end. For example: with 3 cycles, 80 hidden nodes and lambda set to 4.1×10^{-3}; the test accuracy for the single model was 46.46 percent, whereas the test accuracy for the ensamble model was 46.31. This was surprising to me, as I was sure this would boost the performance of the model. My theory as to why it did not is that the first model in the ensamble model, meaning the model produced after one cycle, might be too bad and therefore collude the result. When training with more cycles, perhaps the ensamble method will show its contribution. To test this theory I trained a model with the same parameters for 8 whole cycles and then got a test accuracy for the single model of 45.41 percent and for the ensamble model i got a test accuracy of 46.33 percent. Although the ensamble

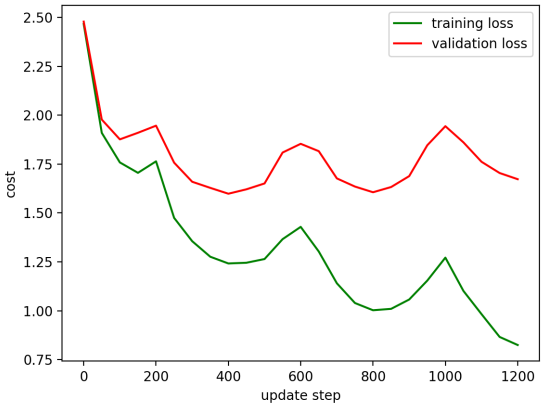

Figure 1: The graph of the training and validation cost computed 9 times per cycle. This is for the model with 80 hidden nodes and lambda 4.1×10^{-5}. A lot of overfitting.

model now had higher accuracy, this is not enough evidence to conclude that my theory is correct.

1.3 Leaky relu

As a third and last optimization I wanted to try how swapping the relu activation for a leaky relu would effect the accuracy. I implementer leaky relu simply as np.maximum(0.02*s, s), and when computing the gradient i simply had to change some values of 0 to 0.02 instead. I tested my implementation by checking the gradients vs numerically calculated gradients and the relative error was tiny, that way I knew my leaky relu implementation was good.

For reference I trained a model with normal relu activation and 80 hidden nodes for 3 cycles. The reference model had an ensamble test accuracy of 46.29 percent.

Then I trained a model with leaky relu and the same hyper-parameters and that model had an ensamble test accuracy of 46.37 percent. Not that much difference. Perhaps leaky relu is better suited in deeper networks.

1.4 All optimizations on more data

I finally trained a model on all the available training data, except 1000 for validation, over 5 cycles and got the best model so far with a test accuracy: 54.06 percent and an ensamble test accuracy of 53.45 percent. The cost/loss function for training and evaluation data is shown in Figure 2 and the training and evaluation accuracies are shown in Figure 3.

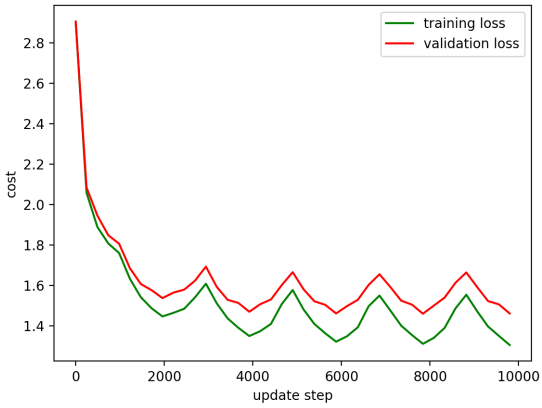

Figure 2: The graph of the training and validation cost computed 9 times per cycle. This is for the best model trained after all optimizations.

2 Smith, eta min and max

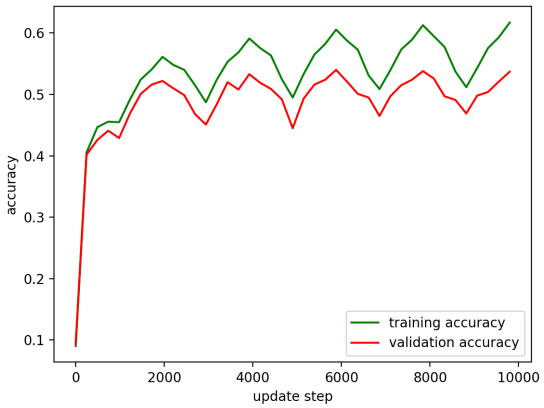

Figure 3: The graph of the training and validation cost computed 9 times per cycle. This is for the best model trained after all optimizations