Deep Learning DD2424 - Assignment 3 Bonus Points

Anton Stagge

April 2019

1 Leaky relu

As a first 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 the best 3 layered model from assignment 3 reached a test accuracy of 50.37 percent.

Then I trained a model with leaky relu with the hyper-parameters 3 layers with 50 and 50 hidden nodes in the first two layers respectively, $lambda = 0.005$, $eta\_min = 10^{-5}$, $eta\_max = 10^{-1}$, 2 cycles of training and $n\_s = 5 * 45000/n\_batch$.

The leaky relu model instantly outperformed every other batch normalization model I've trained before with a test accuracy of 50.98 percent. The training and validation costs can be seen in Figure 1.

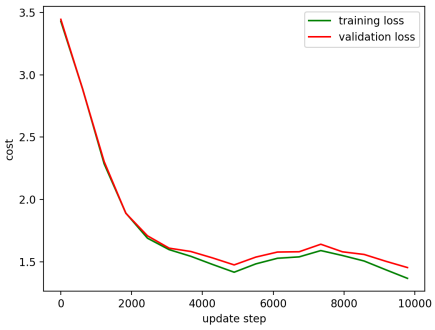

Figure 1: The graph of the training and validation cost for 2 cycles of training for the 3 layer BN model with leaky relu.

2 Random jitter to images

I then tried augmenting the training data by applying a jitter to it on the fly during training. This was simply done by the line

```
X_batch += np.random.normal(0, 0.1, size=X_batch.shape)
```

This, however, seemed to have little to no effect. In fact the test accuracy was lowered to 50.11 percent. Thus, i tried lowering the sigma value of the jitter to 0.01 instead of 0.1. But the effects were the same. Figure 2 show the evolution of the cost function while training with jitter with sigma 0.1.

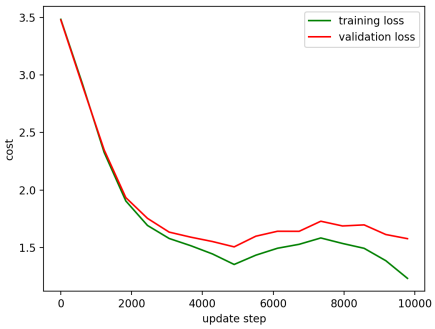

Figure 2: The graph of the training and validation cost for 2 cycles of training for the 3 layer BN model with leaky relu and jitter with sigma 0.1.

3 BN after relu

This seemed like an easy thing to do, but it was harder than I thought. You really had to think about which variables you needed to save in the forward pass, because you needed them in the backward pass. Finally I managed to do it, I verified by checking against the numerically calculated gradients and got result: Layer 0
gradient W0 relative error: 7.595472674116509e-06
gradient b0 relative error: 5.992503948208 49e-06
gradient gamma0 relative error: 4.7516699341198765e-06
gradient beta0 relative error: 3.157105324694648e-06

Layer 1
gradient W1 relative error: 1.1505422849041325e-05
gradient b1 relative error: 1.0780378411015063e-05
gradient gamma1 relative error: 4.365113532219046e-06
gradient beta1 relative error: 4.90697116424749e-06

Layer 2
gradient W2 relative error: 5.906929495852288e-06
gradient b2 relative error: 8.05481973163494e-06

Then I trained the models for 2 cycles with the same hyper-parameters as before and got a test accuracy of 52.57 percent. All the effort was worth it. This is the largest increase in accuracy so far! The evolution of the cost function for the training and validation data can be seen in Figure 3.

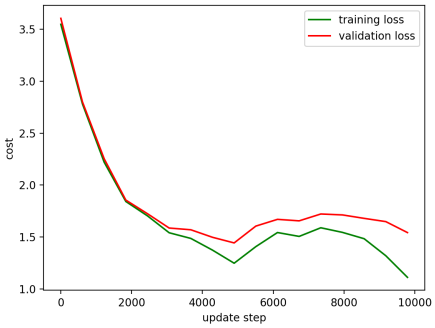

Figure 3: The graph of the training and validation cost for 2 cycles of training for the 3 layer BN model with leaky relu and jitter with sigma 0.1. and BN before the leaky relu.

4 Better search for good lambda

Since a lot of changes have been made to the model now, I thought it would be a good idea to do a more thorough search for the best lambda parameter. I automated the process by first searching for 20 random values between 10^{-1} and 10^{-6}, picked the best one, then searched for 10 random value between $10^{-(best-1)}$ and $10^{-(best+1)}$ picked the best one and searched for 10 values between $10^{-((best-1)+0.99)}$ and $10^{-(best+0.01)}$. I trained every model om all data but 5000 for validation, for two cycles but with $n\_s = 2 * 45000/n\_batch$. The script took 20 min to run and the results were that $lambda = 0.003149$ had the best validation accuracy. Training the model with this new lambda for 2 cycles with $n\_s = 5 * 45000/n\_batch$ again gave a test accuracy of 52.43 percent. Not that much improvement, which is a pattern that I've noticed. Perhaps this is because batch normalization acts like some form of regularization itself, and therefore lambda does not have too much of an influence. Figure 4 shows the evolution of the cost function for this final model.

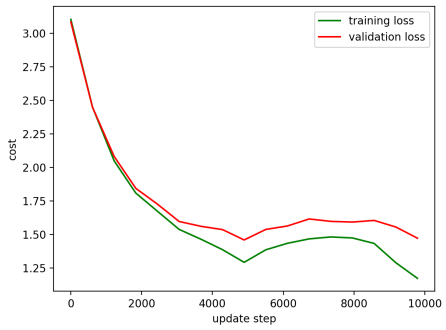

Figure 4: The graph of the training and validation cost for 2 cycles of training for the 3 layer BN model with leaky relu and jitter with sigma 0.1, BN before the leaky relu and $lambda = 0.003149$ as per the coarse to fine search