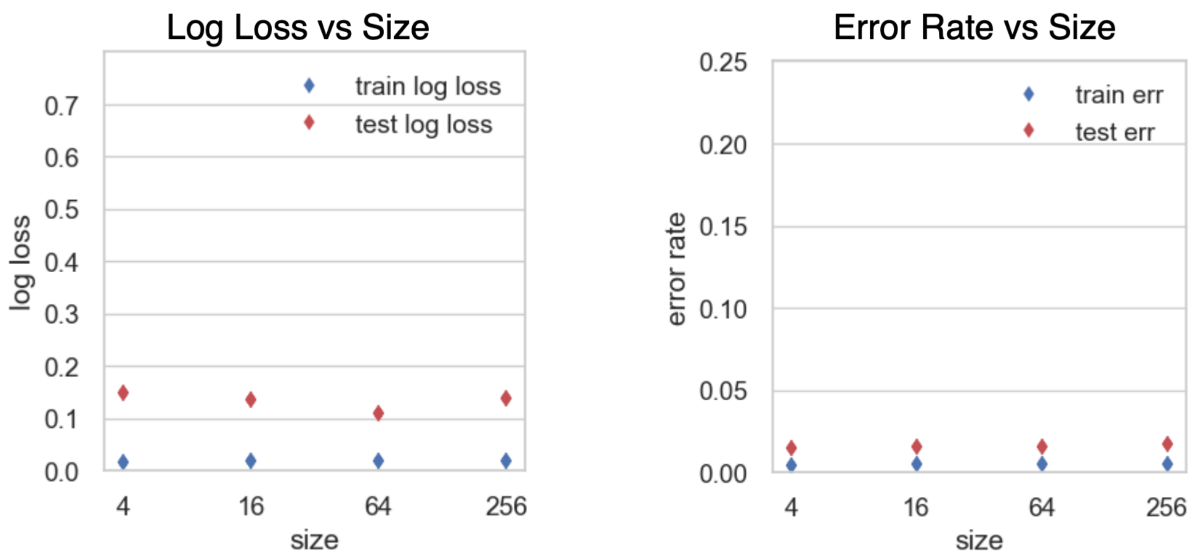
CS 135 HW 3 - Neural Networks and Gradient Descent

Liam Strand

Problem 1

Figure 1



Short Answer 1a

I would recommend a hidden layer size of 64 to minimize log loss on heldout data. That size produces the lowest log loss compared to other sizes. I see signs of overfitting with a hidden layer size of 256 because the performance on heldout data decreases, while the performance on training data remains very good.

Short Answer 1b

The error rate on heldout data remained low across all test cases. The lowest was with a hidden layer size of 4. That selection has the added benefit of training much faster than larger sizes. I see some sign of overfitting with a hidden layer size of 256 because the gap between training and testing performance is a bit wider than with smaller models.

Short Answer 1c

A typical run with 64 hidden units had a final log loss of 0.02 and took 12.1 seconds to run. Runs with 64 hidden units did not converge.

Problem 2

Short Answer 2a

Based on Figure 2, the training objective for MLPs as a function of all learnable weight parameters is not convex. We can see, especially with a batch size of 10000, that models with different initializations will converge at different values. This is most obvious with a learning rate of 0.3, where the log_loss of the different initializations is clearly very different, having converged at different local minima.

Short Answer 2b

- For a batch size of 10000, I would recommend the highest learning rate of 2.7, as all runs converged quickly.
- For a batch size of 500, I would recommend a learning rate of 0.3, which allowed runs to converge quickly, while rarely showing signs of divergence.
- For a batch size of 25, I would be forced to recommend a learning rate of 0.1, which is the only learning rate that does not show signs of divergence. A learning rate of 0.3 would be expected to be faster, but it just produces more thrashing so it takes just as long as a learning rate of 0.1.

Short Answer 2c

- With a batch size of 10000 and a learning rate of 2.7, the method only reached a loss of 0.15 consistently after 25 seconds.
- With a batch size of 500 and a learning rate of 0.3, the method delivered a good loss after 20 seconds.
- With a batch size of 25 and a learning rate of 0.1, the method delivered a good loss after 45 seconds.

Short Answer 2d

A batch size of 500 is the fastest to deliver a good model. That batch size seems to be a happy medium between very large batches that require a very expensive gradient estimation, and very small batches that estimate the gradient quickly, but not very accurately.

Short Answer 2e

The final model quality is dramatically better with L-BFGS. L-BFGS also ran faster, but I suspect that is due to me having a pretty fast laptop, since the whole point of SGD is that it is a faster than L-BFGS.

Short Answer 2f

L-BFGS is better than SGD

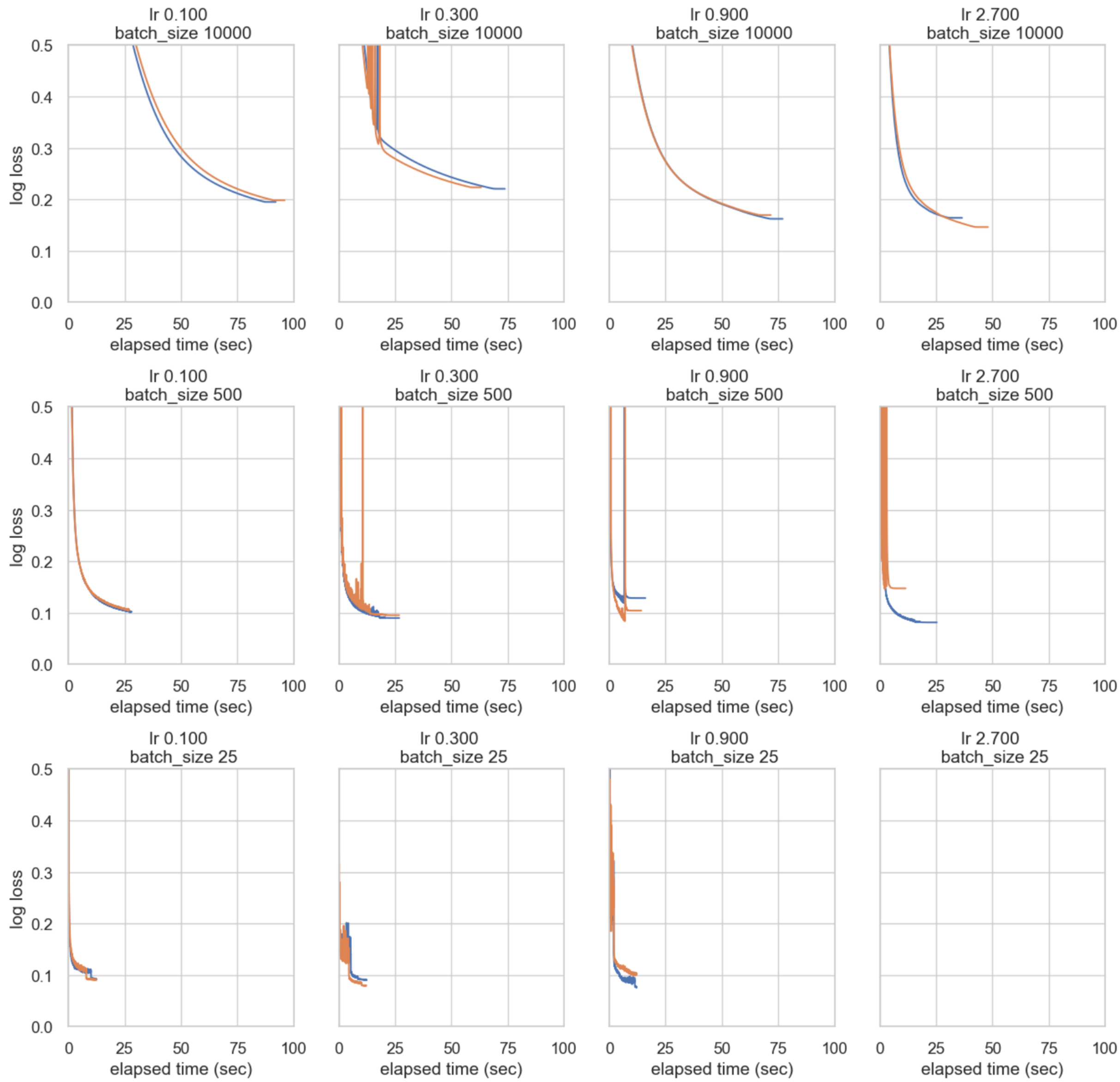
- The final models are higher quality than SGD, in general
- Uses second order gradient information to select the best step direction

SGD is better than L-BFGS

- Generally fits faster than L-BFGS
- Only uses first-order information, and a subset at that.

Problem 3

Figure 3



One obvious difference is that my computer seems to run much faster on runs with a small batch size, and much slower on runs with a large batch size. Furthermore, there is less thrashing and divergence in general, though the run with a large learning rate and small batch size did not do well! The overall conclusions from Figure 2 still hold: 500 is the best batch size for consistency and performance.