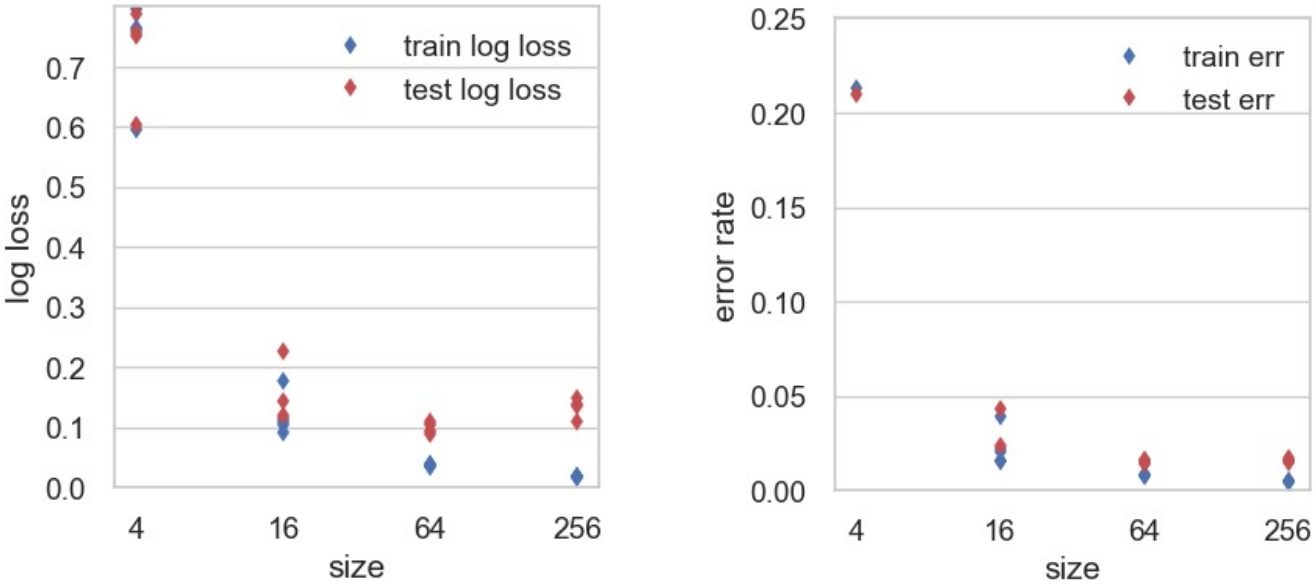
Problem 1: MLPs with L-BFGS: What model size is effective?

Figure 1 in Report:



Short Answer 1a in Report:

Based on Figure 1, I would recommend an MLP with hidden layer of 64 neurons in order to achieve best log loss on heldout data. I would make this choice because the MLP with 64 neurons produces the least test log loss results with about 0.1 on average. The MLP improves with increasing size until the size of 64, but then overfits at the next size of 256. This is evident from the fact that although the train log loss decreases, the test log loss increases due to increasing model complexity.

Short Answer 1b in Report:

Based on Figure 1, I would again recommend an MLP with hidden layer of 64 neurons in order to achieve best log loss on heldout data, but I don't think there is as much difference between the performance of different sizes as for the log loss. I would make this choice because the MLP with 64 neurons produces the least error rate results again with less than 0.02 on average. I don't see a significant overfitting in this graph. The MLP improves with increasing size until the size of 64, and then remains the same for train error rate but very slightly increases for the test error rate for the size of 256. That slight increase in the test error rate does not signify a substantial overfitting, but we can for sure anticipate that for sizes even more than 256, the MLP model would overfit.

Short Answer 1c in Report:

The average log loss on the training set is 0.04, and it takes about 13.5 seconds to complete its maximum iteration. It does not converge in none of the runs.

Run #	tr log loss	1000 iterations
1	0.04	13.3
2	0.039	13.6
3	0.035	13.6
4	0.037	13.4
average	0.04	13.5

Figure 2: SGD training loss vs. elapsed wallclock time, varying batch size (rows) and learning rate (columns)

Short Answer 2a in Report

Based on Figure 2, the training objective for MLPs as a function of all learnable weight parameters is not convex. This is evident from the fact that the stochastic gradient descent (SGD) training losses level off at different log losses across the graphs, indicating the existence of multiple local minima within the training objective curve and therefore the graph is not convex. Also, specifically in the graph for learning rate of 0.300 and a batch size of 10000, you can observe all the random seed curves level off at different log loss levels.

Short Answer 2b in Report

(1) For batch size 10000, I recommend the learning rate of 2700. At this learning rate, the log loss converges faster than all other learning rates (at about 25 sec) , has no excessive oscillations or diverging (with 4 smooth curves), is consistent across all 3 in 4 random seeds (only one seed converges at a slightly higher loss values), and the loss curves level f at the lowest log value.

(2) (2) For batch size 500, I recommend the learning rate of 2.700. At this learning rate, the log loss levels off under 0.1, and the convergences faster than all except for the learning rate 0.900, which levels off at a higher log loss though, and is consistent across all 3 random seeds in 4.

(3) (3) For batch size 25, I recommend the learning rate of 0.900. At this learning rate, the log loss, levels off under 0.1, converges around the same amount of time as other, but doesn't diverge on any seed. Initially, there are oscillations, but as you progress, the curves become smoother.

Short Answer 2c in Report

(1) For batch size 10000 and learning rate 2.700, the method only reached a loss of 0.15 consistently after 25 seconds.

(2) For batch size 500 and learning rate 2.700, the method took about 10 seconds to converge at log loss 0.1.

(3) For batch size 25 and learning rate 0.900, the method took about 20 seconds to converge at log loss 0.1.

Short Answer 2d in Report

The batch size of 500 delivers a good model the fastest, and this a pattern observed for all learning rates across all graphs. Compared to a batch size of 10,000, a batch size of 500 results in significantly lower log loss values as it converges. Similarly, in contrast to a batch size of 25, the batch size of 500 converges much faster while reaching a similar log loss level as the batch size of 10,000. In terms of trade-offs , very large batch size is computationally expensive because calculating the gradient for all data points takes time, whereas, very small batch size introduce noise in the gradient estimates which can cause the loss to jump around and make it harder to converge.

Short Answer 2e in Report

L-BFG with hidden layer size of 64 has about 0.04 log loss, and SGD with a batch size of 500 and a learning rate of 0.300 has about 0.1 log loss, which means L-BFG outperforms SGD in terms of model generalization. Also, L-BFGS has an average runtime of 13.5 seconds, whereas SGD averages at 20 seconds, which indicates L-BFG is faster than SGD by about 6.5 seconds.

Short Answer 2f in Report

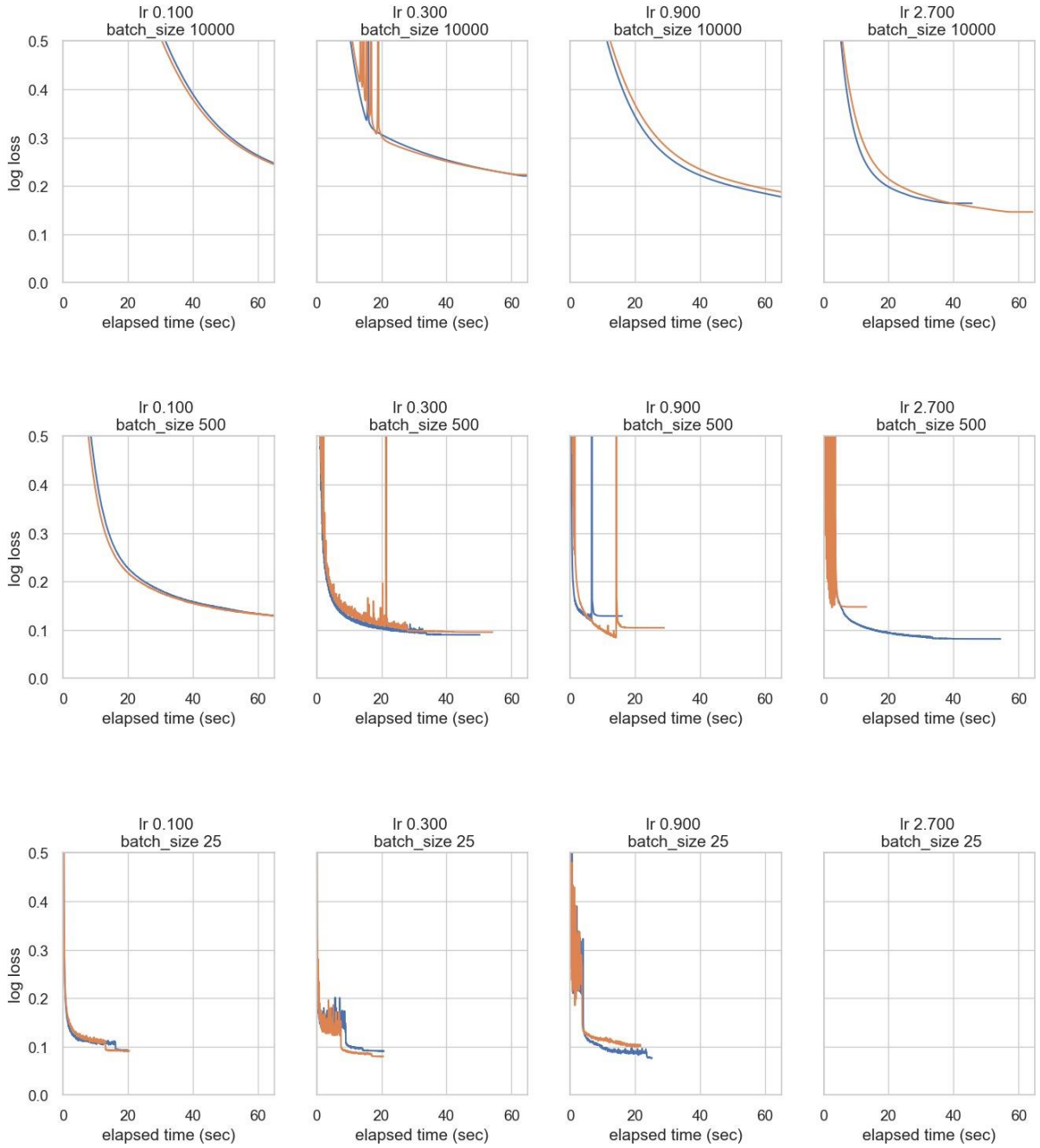
Reasons to prefer L-BFGS over SGD:

1. Faster Convergence for Small Datasets: L-BFGS often converges faster on small to medium-sized datasets because it utilizes both the first and second gradient information related to the convexity when finding the gradient.
2. Fewer Hyperparameters to Tune: L-BFGS requires fewer hyperparameters to be tuned compared to SGD which involves tuning learning rates, batch sizes, and other hyperparameters to achieve convergence.

Reasons to prefer SGD over L-BFGS:

1. Faster Convergence to Large Datasets: SGD can converge faster and is more adaptable to large datasets due to its mini-batch processing technique, whereas L-BFGS can become memory-intensive and impractical when dealing with large datasets because calculating gradients for all datapoints.
2. Escaping Local Minima and Easier to Implement: SGD is preferable for non-convex functions as it can escape local minima by using different batch samples for gradient calculations. In contrast, L-BFGS can get stuck in local minima if the initial point is far from the global minimum. Moreover, implementing SGD is straightforward, involving a simple algorithm. In contrast, L-BFGS requires complex calculations like computing first and second gradients, which are not as straightforward.

Figure 3 in Report



Caption: *In general, there are no significant differences between the graphs you provided and your figure; however, certain conclusions have slightly changed due to using a smaller number of random seeds. For a batch size of 10,000, the best model appears to have a learning rate of 2.700. On the other hand, for a batch size of 500, a learning rate of 2.700 does not seem to produce the best curves because there are two curves on the contrary to the provided figure, with one converging at a much higher log loss. Therefore, the optimal choice appears to be a learning rate of 0.300. Finally, for a batch size of 25, a learning rate of 0.900 seems to converge more slowly compared to a learning rate of 0.300, which differs from what is observed in the provided figure.*