HW 1 PDF Report - Intro to ML (CS 135)

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

Figure 1 in Report

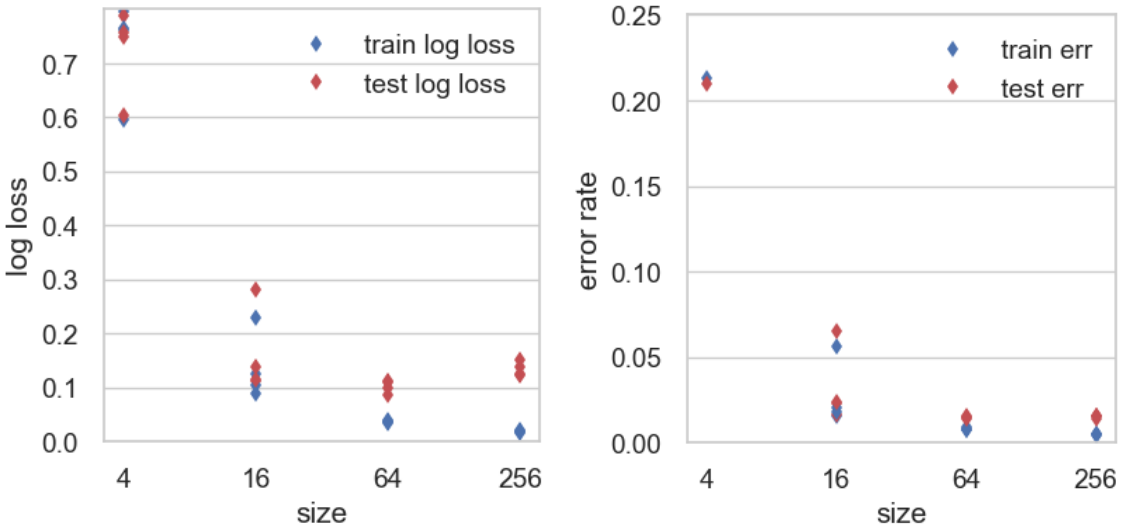

Figure 1. Performance of each model trained as a function of the size (log loss vs. error rate). Each dot represents the final result of one "run" of the optimizer and by the spread of dots in each graph, model sensitivity to random initialization and model size is apparent.

Short Answer 1a in Report

The hidden layer size I would recommend to achieve the best log loss on held out data is batch size 64. Size 64 has the smallest average log loss margin (around 0.07) between the train log loss and test log loss points and the test set achieved the lowest average log loss (around 0.10) among all batch sizes. These aspects show minimum overfitting / underfitting and great generalization. The main factor is the batch size, where too large of a layer size could lead to overfitting but too small of a layer size could lead to underfitting. Other batch sizes showed signs of overfitting (size=100), underfitting (size=4), or poorer overall performance (size=16).

Short Answer 1b in Report

Similarly, the hidden layer size I would recommend to achieve the best error rate on held out data is batch size 64. Size 64 has the smallest average error margin (around 0.01) between the train error rate and test error rate, and the lowest average test error rate (around 0.02). This shows minimum overfitting / underfitting and great generalization. Too large of a layer size could lead to overfitting but too small of a layer size could lead to underfitting. Other batch sizes show signs of underfitting (size=4), poorer overall performance (size=16), or a slightly bigger margin between the test and training error rate (size=256) based on this intuition.

Short Answer 1c in Report

The average final log loss on the training set was about 0.04. The average time to complete the maximum iteration on the training set was about 7.35 seconds. That said, none of the runs at layer size 64 converged.

Problem 2: MLPs with SGD: What batch size and step size?

Short Answer 2a in Report

The training objective for MLPs as a function of (batch_size, lr) pairs were convex for {(10000, 0.100), (10000, 0.300), (10000, 0.900), (10000, 2.700), (500, 0.100)} as the graphs showed smooth and consistent convergence in loss. Other graphs {(500, 0.300), (500, 0.900), (500, 2.700), (25, 0.100), (25, 0.300), (25, 0.900), (25, 2.700)} were not convex as they showed erratic fluctuations or plateau of loss.

Short Answer 2b in Report

Following 2a's format, I choose {(10000, 2.700), (500, 2.700), (25, 0.100)}. (10000, 2.700) produces good training loss values the fastest while also showing no sign of divergence. (500, 2.700) convergences faster than the rest than the other learning rate pair to loss level 0.1. Similarly, (25, 0.900) convergences faster than the rest than the other learning rate pair to loss level 0.1.

Short Answer 2c in Report

The first (10000, 2.700) only reached a loss of around 0.15 consistently after around 25 seconds. The second (500, 2.700) reached a loss of around 0.10 consistently after around 10 seconds. The third (25, 0.900) reached a loss of around 0.10 consistently after around 20 seconds.

Short Answer 2d in Report

The batch size of 500 is fastest to deliver a good model. The tradeoff is that smaller batch sizes allow for faster updates to the model's parameters while larger batch sizes take more time. This is part of what leads a model to converge quicker to a lower loss value. However, small batches, such as 25, also tend to be non convex due to the noisy updates from small batches, while larger batch sizes tend to be more stable and convex. This allows for "medium" sized batches like 500 to be an optimal option.

Short Answer 2e in Report

The best SGD run (500, 2.700) reaches a loss around 1.00 consistently after around 10 seconds. However, the average run of L-BFGS (at layer_sizes=64) reaches a loss of around 0.04 after around 7 seconds. Thus, the L-BFGS has higher performance in both speed and loss reduction.

Short Answer 2f in Report

Two reasons to prefer L-BFGS over SGD when optimizing a neural network are that L-BFGS tends to converge faster on smaller data sets by using the first and second

gradient, and L-BFGS provides a deterministic convergence to a local minimum (making its results more consistent). Two reasons to prefer SGD when optimizing a neural network are that SGD can better handle larger data sets with its mini-batch size processing, and can visit more local minima with its randomly determined (batch) sampling for gradient calculations.

Problem 3: Make your own figure comparing batch size and step size

Figure 3 in Report

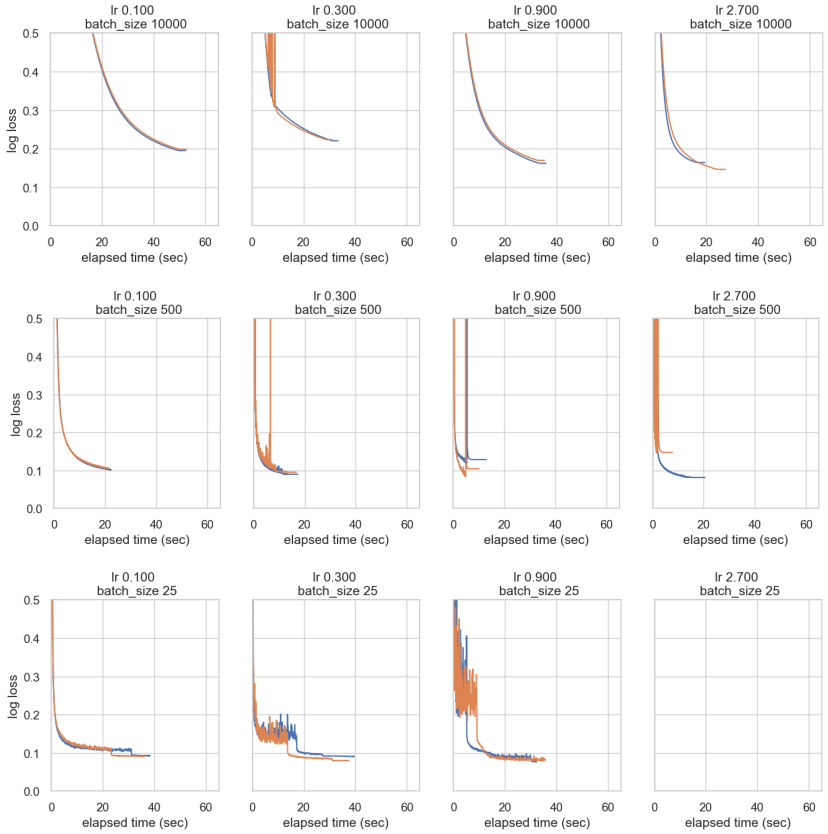

Figure 3. Training loss versus elapse wall clock time (seconds) when training MLPs with SGD on the flower xor dataset. Performance across different batch sizes (rows) and learning rates (columns) are shown, where each panel shows 2 runs of SGD with different random seeds, run to convergence. Between Figure 2 and Figure 3, there are minor discrepancies, but major conclusions from Problem 2 hold.