COMP135 - Homework # 3

Camelia D. Brumar

October 31, 2020

Contents

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

1.1 Short Answer 1a

Based on your Figure 1, what hidden layer size would you recommend to achieve the best log loss on heldout data? Do you see signs of overfitting?

Answer: In order to achieve the best log loss, I would choose the $size = 64$. The reason is because for the case $size = 256$ I see signs of overfitting, i.e. the test log loss starts to increase while the training log loss continues to decrease as the size increases (in visual terms, the red diamond is higher than the red diamond for $size = 64$, while the blue diamond is lower than the blue diamond for the case $size = 64$).

1.2 Short answer 1b

Based on your Figure 1, what hidden layer size would you recommend to achieve the best error rate on heldout data? Do you see signs of overfitting?

Answer: In order to achieve the best error rate, I would choose the $size = 64$. The reason is because for the case $size = 256$ I see signs of overfitting, i.e. the test error rate starts to increase while the

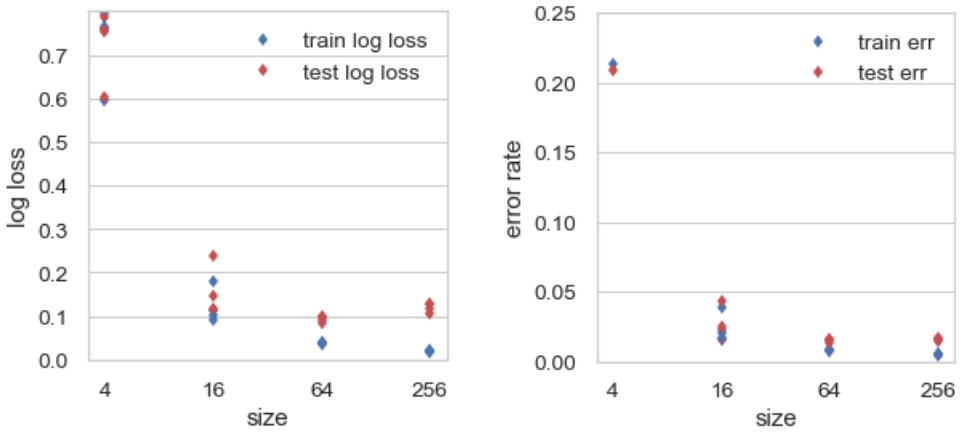

Figure 1: Comparing performance of each model I've trained as a function of the size.

training error rate continues to decrease as the size increases (in visual terms, the red diamond is higher than the red diamond for $size = 64$, while the blue diamond is lower than the blue diamond for the case $size = 64$).

1.3 Short Answer 1c

Consider a typical L-BFGS run with 64 hidden units. What final log loss on the training set does it reach (round to nearest 0.01)? About how many seconds does it take to converge or complete its maximum iteration (round to nearest 0.1 seconds)? Does it converge?

Answer: The final log loss on the training set reached is 0.04. It takes 7.8 seconds to complete its maximum iteration.

1.4 Short Answer 1d

You have fit an MLP with hidden_layer_sizes=[64] to this flower XOR dataset. How many total weight parameters are in each layer? How many total bias or intercept parameters in each layer? Show your work or justify your answer.

Answer: In order to obtain the number of weight parameter I will use the $coefs$ attribute of the MLP classifier. $coefs$ has two entries, the first one are the coefficients/weights for the first layer, and the second entry for the second layer.

- For the first layer, I check the size of $coefs_[0]$, and it returns 2, which means that there are two vectors. By checking the size of each vector, I obtain that each one has 64 coefficients.
- Now I check the size of the second layer by running the function len on $coefs_[1]$. This gives me 64, which means that there is one vector with 64 weight parameters.

In the case of the intercept, I do the same thing with the *intercept* attribute, and I get that there are a total of 64 intercepts in the first layer, and 1 weight parameter for the second layer.

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

2.1 Short Answer 2a

Based on Figure 2, is the training objective function for MLPs convex or not convex? What evidence in Figure 2 supports this answer?

Answer: It is NOT convex, and the evidence can be found in the plot of the model with lr = 0.300 and batch_size = 10000. In this plot I can see that different random states, i.e. with different initialization of the parameters, the optimization gives different losses when the algorithm converges. This means that the optimization gives us different local minima depending on the initial conditions, and thus, our log loss in not convex.

2.2 Short Answer 2b

For each batch size in Figure 2, which learning rate(s) do you recommend? You should prioritize learning rates that produce good training loss values quickly and consistently across at least 3 of the 4 runs, without showing severe divergence.

Answer:

- For batch size = 10000, I recommend **lr = 0.900**, because for lr = 0.100 it is too slow, for lr = 0.300 the curves are not consistent, and for lr = 2.700 one of the curves presents higher loss.
- For batch size = 500, I recommend **lr = 0.100**, because for the other learning rates the curves are not consistent.
- For batch size = 25, I recommend **lr = 0.100**, even though it takes a long time for the curves to reach stability. I didn't choose the other learning curves because for lr = 0.300 and 0.900 the curves are not consistent, and for lr = 2.700 it the optimization problem does not converge at all.

2.3 Short Answer 2c

Using the recommended learning rates you picked in 2b, report for each SGD batch size the time taken (in seconds, rounded to nearest whole number) to deliver a "good" training loss value of 0.1 (e.g the time when at least 3 out of 4 runs reach log loss of 0.1).

Answer:

1. The method only reached a loss of 0.175 consistently after 38 seconds.
2. The method only reached a loss of 0.1 consistently after 30 seconds.
3. The method only reached a loss of 0.1 consistently after 47 seconds.

2.4 Short Answer 2d

Based on 2c, which batch size is fastest to deliver a good model? Can you explain why? What tradeoffs are at work?

Answer: The model with lr = 0.100 and batch_size = 500 is the fastest to deliver a good model, (why?) because there is consistency in the loss across different random states, and also the training error reaches 0.1 in less time that the other models. (Tradeoffs?) Maybe a competitor to this model is the one with lr = 0.900 and batch_size = 10000, which despite of being consistent across multiple random states, the training error is higher than for the previous model, and it is more expensive to train.

2.5 Short Answer 2e

Compare speed of the best SGD to what you observed for the best size 64 runs of LBFGS from Problem 1d. Which method is better for this problem? Why?

Answer: It appears that the MLP with 64 hidden units is better in terms of the training log loss (0.04) and also in terms of training time (7.8 seconds) than the model chosen in 2c (log loss of 0.1 in 30 seconds).

2.6 Short Answer 2f

Think in general about training neural networks on classification tasks like this one. List 2 reasons to prefer L-BFGS over SGD when optimizing your neural network, and 2 reasons to prefer SGD over L-BFGS.

Answer: Two reasons to choose L-BFGS over SGD are:

1. L-BFGS usually does not require extensive hyperparameter tuning.
2. L-BFGS can often get a better log loss than SGD in less iterations (thus in less time).

Two reasons to choose SGD over L-BFGS are:

1. SGD is computationally less expensive than L-BFGS.
2. L-BFGS requires more memory than SGD, because it needs to approximate the Hessian matrix and stores more search information.

3 Problem 3: Producing your own figure comparing batch size and learning rate

(Please see next page.)

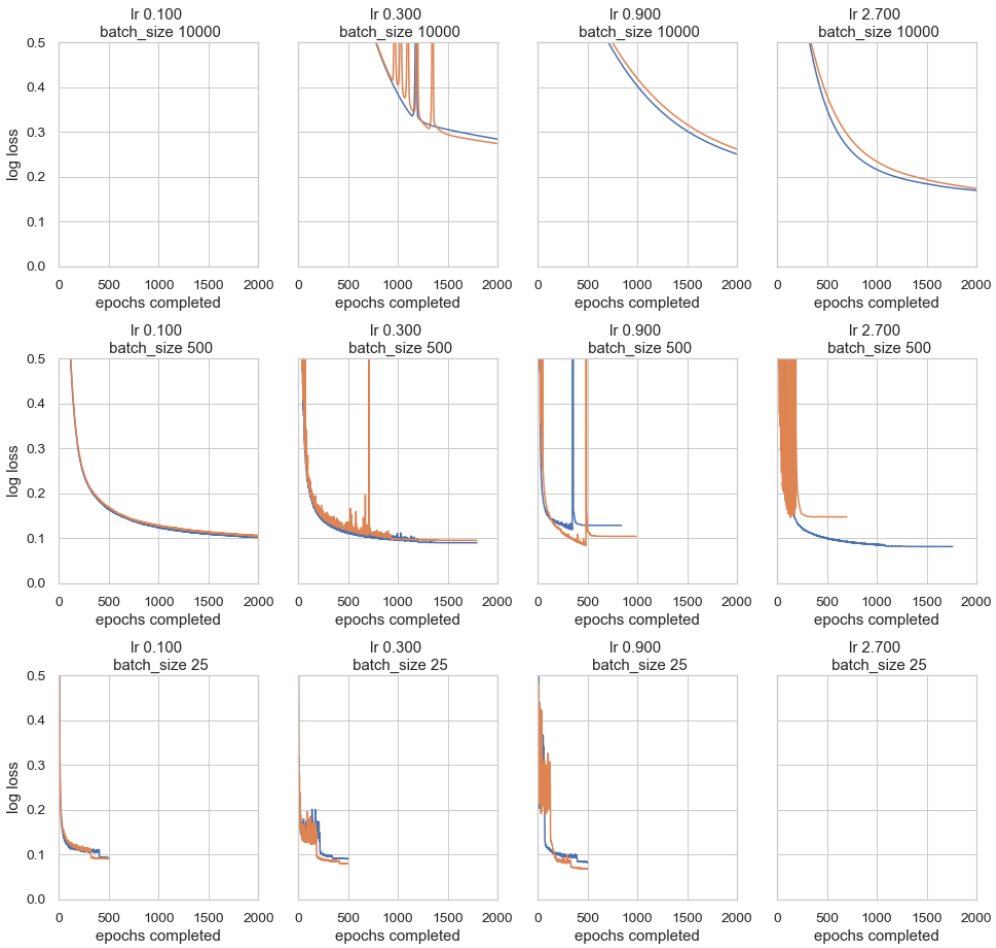

Figure 3: Training loss vs epochs completed when training MLPs with SGD on the flower xor dataset. Some significant changes to the Figure 2 on the homework 3 website are: 1) that in this case we considered completed epochs instead of elapsed time in seconds for the x axis, and this is because if we plot with elapsed time the plots are not too informational, and the curves gather on the left side of the plot; 2) now the model with lr = 0.100 and batch_size = 10000 does not converge; 3) the loss for batch_size = 10000 and learning rates 0.900 and 2.700 converge somewhere later than epoch 2000, while in Figure 2 we could see where it was converging (at approx. 40 seconds). In conclusion, in contrast with the answer give in 2d, the fastest model to deliver a good model is the one with batch_size = 25 and lr = 0.100, but it still does not compete in terms of loss with the LBFGS model from problem 1d.