CSE 422 Hw3

Rohan Mukherjee

January 29, 2025

Throughout this homework, whenever I report the usual training loss (non-normalized), I will always report the average training loss, instead of the sum.

1. (a) The value of the objective (average training loss) for $\hat{\theta} = (X^T X)^{-1} X^T y$ is 0.245766421039127 while for the all zeros vector it is 16.300105308625568.

   (b) The test loss I got was 0.002693543440703437. This is indeed very small. It does not change very much for higher values of m, staying around the same thing. For extremely small values of m, like 1-10, there is a lot of variance, but it is still extremely small. This is probably because we sampled the data according to the rule $y = X\theta + \epsilon$ where ϵ is $N(0, 0.5I)$ random vector. Effectively, the data is just linear with some small noise. So the regression finds the parameter $\hat{\theta}$, which ends up being very close to the true parameter θ^*. This is why the test loss is so small.

   (c) I get the following table (with average training loss): As you can see, the more training data we have, the lower the test loss (and usually the training loss) ends up being. This means that we are learning the parameter θ^* with more accuracy. Clearly, the test loss correlates more with the l2 distance than the training loss. I think this is because the training loss can only get so good, since there is noise in the loss function for the training data, but the test loss does not have any noise associated with it. Since we are learning θ^* with more accuracy the more training data we have, the test loss decreases accordingly with the l2 norm.

Table 1: Model Performance Metrics

n	Distance (theta_hat - theta_star)	Training Loss	Test Loss
500	0.0944	0.2654	0.0094
1000	0.0779	0.2511	0.0062
1500	0.0517	0.2412	0.0027
2000	0.0347	0.2422	0.0012

(d) For fixed θ and θ^*, we know that, by i.i.d. of the training data,

$$\mathbb{E}\Big(\frac{1}{m}\sum(\langle\theta^*,\hat{x}_i\rangle - \langle\theta,\hat{x}_i\rangle)^2\Big) = \frac{1}{m}\sum\mathbb{E}\Big((\langle\theta^* - \theta,\hat{x}_i\rangle)^2\Big) = \mathbb{E}\Big(\langle\theta^* - \theta, x_1\rangle^2\Big)$$

Now recall that if $X = (X_1, \ldots, X_n)$ is a $N(0, I)$ random vector, then for $a \in R^n$, $\langle a, X\rangle = \sum a_i X_i$ is distributed according to $N(0, \|a\|^2)$. Then the expression above is precisely the variance of the random variable $\langle\theta^* - \theta, x_1\rangle$, which by this short calculation is just $\|\theta - \theta^*\|^2$.

(e) By chain rule, $\nabla_\theta L_i(\theta) = \nabla_\theta(\theta^T x_i - y_i)^2 = 2(\theta^T x_i - y_i)\nabla_\theta[\theta^T x_i - y_i]$. Finally using that $\nabla_x a^T x = a$, we get that this equals $2(\theta^T x_i - y_i)x_i$. Then the gradient of L_{train} is just the sum of these gradients, which is $2\sum x_i(x_i^T\theta - y_i)$. If X is the matrix with x_i^T as the ith row (and hence x_i is a column vector), then this just equals $2X^T(X\theta - y)$.

For regular gradient descent, I get the following graph:

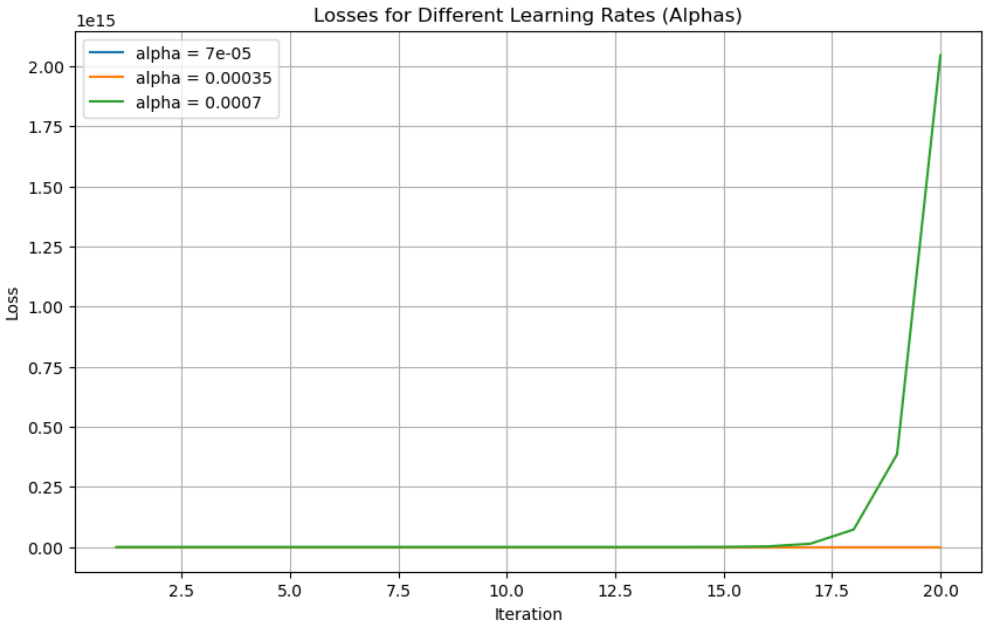

Figure 1: Gradient Descent

I get the following table for the final (average) training loss:

Table 2: Alpha vs Final Training Loss

Alpha	Final Training Loss
0.00007	2.553572e-01
0.00035	2.553122e-01
0.00070	2.044280e+15

As you can see, the best training loss is achieved by the second one, $\alpha = 0.00035$. The first one is also really good. The last one is absolutely terrible. I think this is an example of it got to the min, as you can see in the graph, then overstepped the min and then the gradient started exploding, and it runs quickly away from the local minimum it had found. I also believe for learning rates that were too small, like something $1e-10 \ll 7e-5$ that worked well, it just would take too long to converge to the minimum. 0.00035 feels like a good middle ground.

(f) Pseudocode for SGD is as follows:

i. Initialize θ to 0

ii. For $t = 1, 2, \ldots, T$:

A. Sample i uniformly at random from $\{1, \ldots, n\}$

B. $\theta = \theta - \alpha \nabla L_i(\theta)$

iii. Return θ

(g) The graph I get for SGD for $T = 20,000$ iterations is as follows:

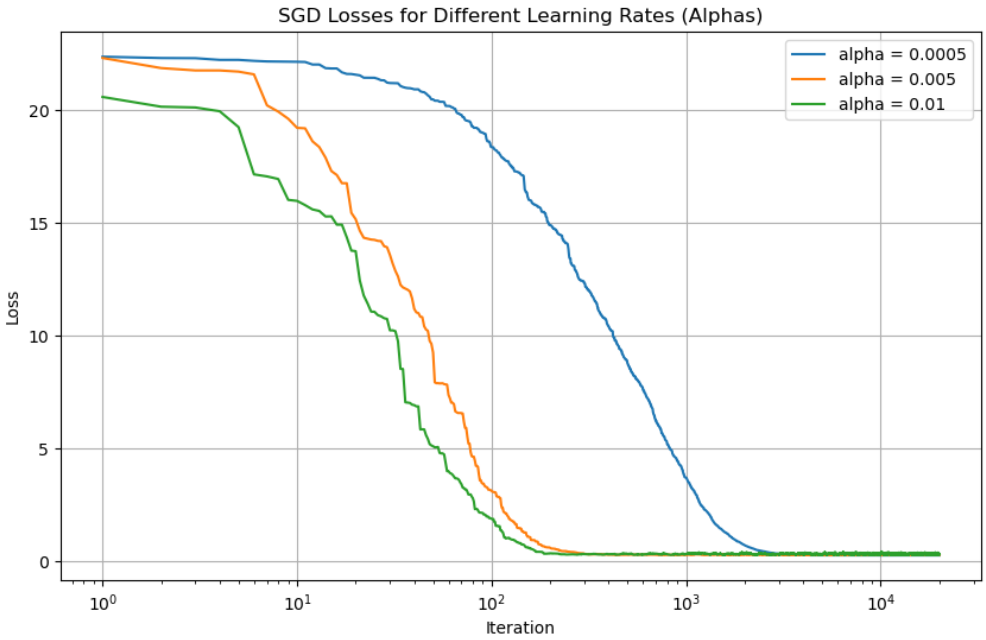

Figure 2: SGD

The final objective values are like this:

Table 3: Final Objective Values for Different Alpha

Alpha	Final Objective Value
0.0005	0.254497
0.005	0.262516
0.01	0.338374

Pictorially, they all look around the same, and the final output values reflects that. However, as expected, the larger the α the larger the final objective value. This is because the larger α introduces more variance into the SGD process, leading to the very small fluctuations in the objective value you can see in the image. From a previous part, $\nabla_\theta L_i(\theta) = 2(\theta^T x_i - y_i)x_i$. So, the function uses precisely one data point per iteration. Since we run for $T = 10,000$ iterations, and there are 2,000 training samples, we use each smaple 5 times on average. This is much better than the 20 times we used them in the gradient descent part. It also gets the same final output value, which is interesting.

(h) My intuition is that if the gradient is large, this means that the minimum is very far away. So we should probably move even more than the small learning rate we

have put. On the other hand, if the gradient is small, and we had the same constant learning rate, we would overstep the minimum, and the loss would increase. So I propose that we use a step size proportional to the size of $\nabla L_i(\theta)$, such as its absolute max coordinate (I would suspect that this is faster to compute than the norm of the gradient).

I have tried the following, where α_0 is a hyperparameter and d is the length of the input samples:

i. $\alpha_t = \frac{\alpha_0}{d} \max_j |\nabla L_i(\theta)|$

ii. $\alpha_t = \frac{\alpha_0}{d} \|\nabla L_i(\theta)\|$

iii. $\alpha_t = \frac{\alpha_0}{t}$

Here are the plots, in order:

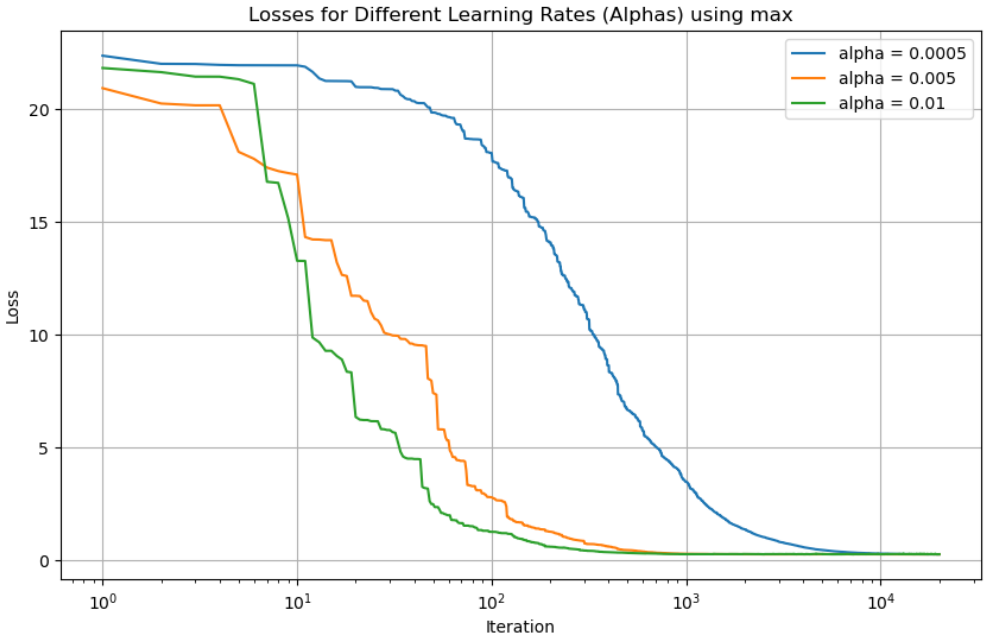

Figure 3: $\alpha_t = \frac{\alpha_0}{d} \max_j |\nabla L_i(\theta)|$

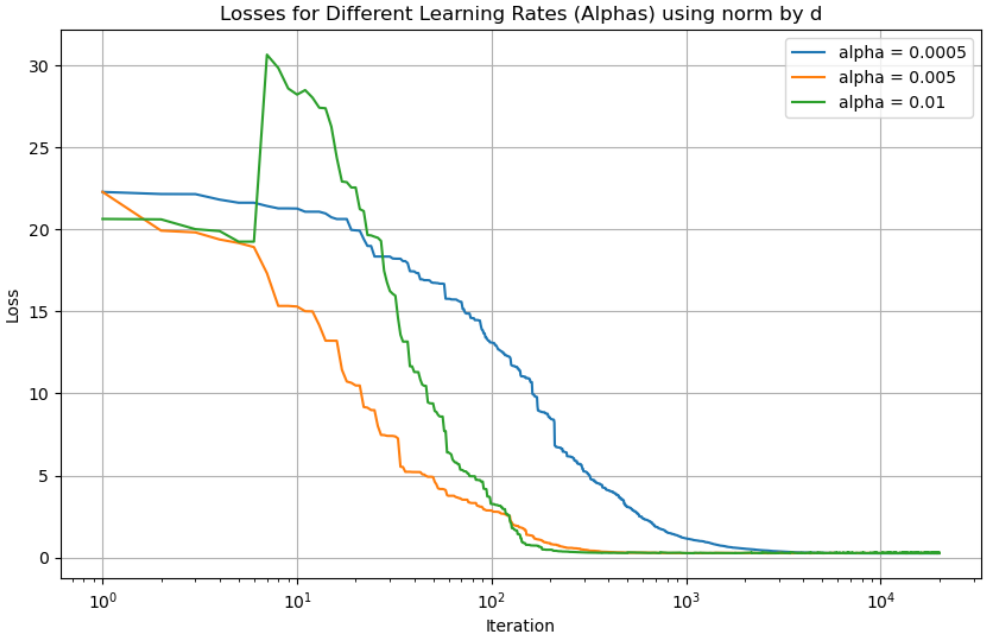

Figure 4: $\alpha = \frac{\alpha_0}{d}||\nabla L_i(\theta)||$

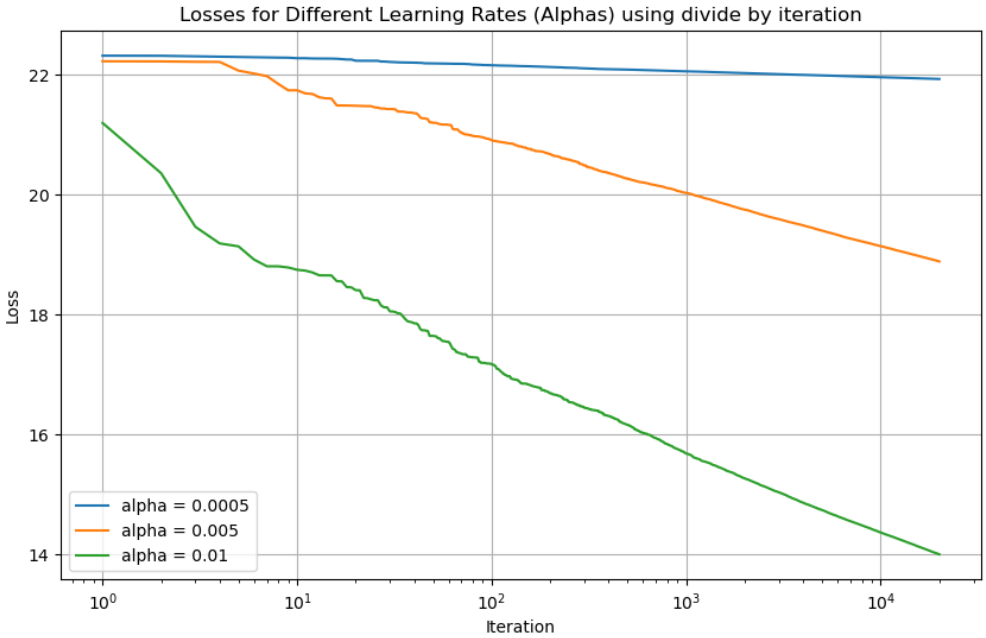

Figure 5: $\alpha_t = \frac{\alpha_0}{t}$

The tables are:

α	Final Loss
0.0005	0.2554149788012268
0.005	0.25913216028792124
0.01	0.27304750826263313

Table 4: Final loss values using max.

α	Final Loss
0.0005	0.2547512193059818
0.005	0.26364601860013565
0.01	0.28366985519259025

Table 5: Final loss values using norm / d.

α	Final Loss
0.0005	22.04319880187047
0.005	17.55662684111231
0.01	15.383248856596401

Table 6: Final loss values using divide by iteration.

The best one is the first option I used as you can see, which is $\alpha_t = t\frac{\alpha_0}{d} \max_j |\nabla L_i(\theta)|$. The last one is really bad as it does not follow the general pipeline I first proposed and instead i just proportional to the iteration, but this doesn't look at all to see how close we are to a local min like the other two methods try to do. I suspect that the first method, using only the max coordinate instead of the full norm of the gradient, has smaller variance since it uses a smaller value. As you can also see for the max alpha, it seems to converge at around the same rate as the first one, being almost done at iteration 100, but it is much more stable, as the 0.01 learning rate has a lot less variance then in the normal SGD.

2. (a) We were told to ignore the written part for this question in an email. I get, over 10 trials, an average train loss of 1.1304786543239968e-11 and an average test loss of 1.2260962605261125. We can see an insane amount of overfitting here.

   (b) Here is the plot of the choice of λ vs training and test loss:

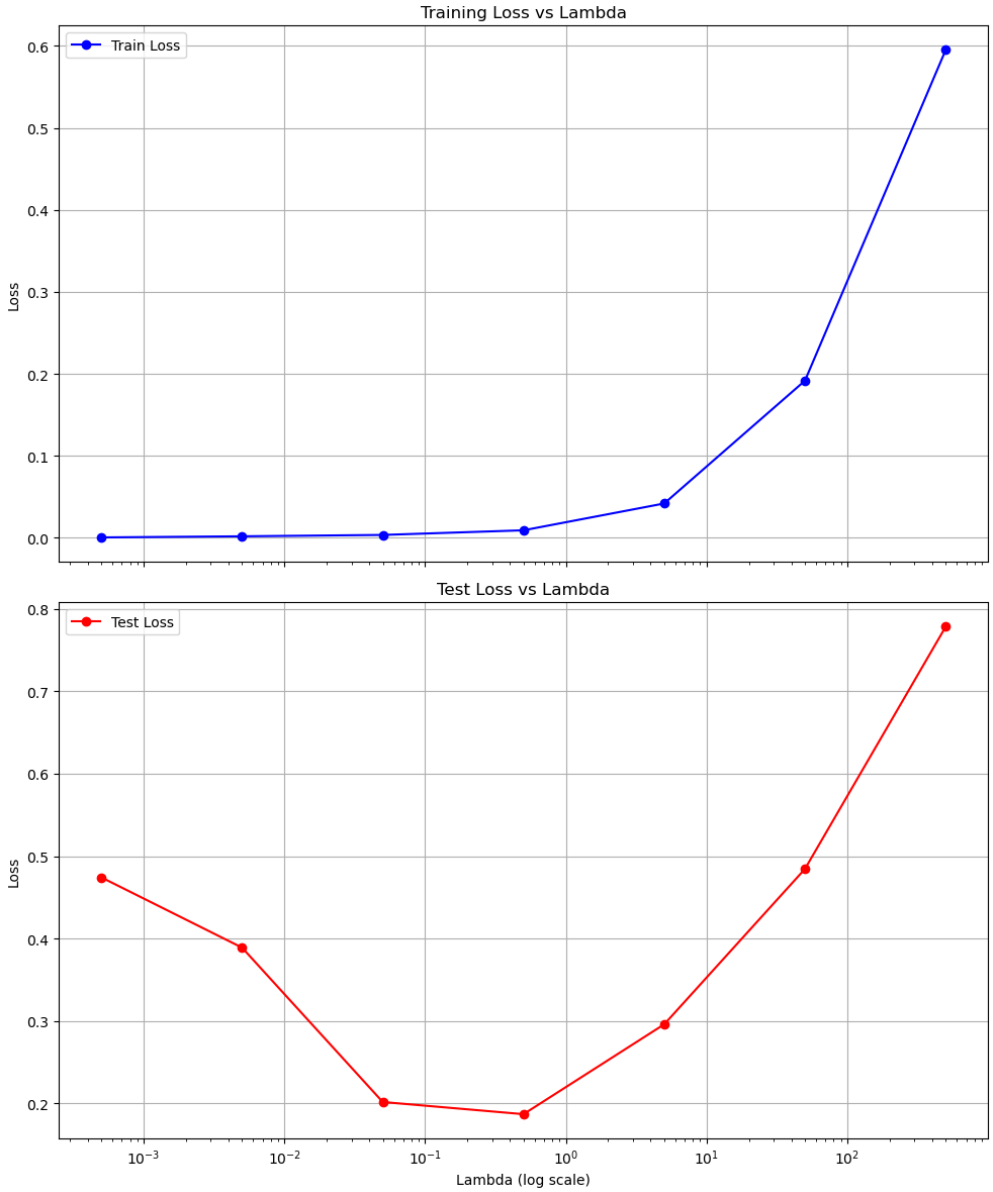

Figure 6: Ridge Regression

We can see a lot of overfitting for the smaller lambdas, like $\lambda < 10^{-1}$. In the picture, the training loss keeps going down but the test loss starts going up. However, compared to part (a) where the test loss was like 11 orders of magnitude smaller than the training loss, this is much better! We don't get anywhere near that kind of overfitting, which is great. It seems like the best λ is slightly lower than 1.

(c) Running SGD on the orginal loss function yields:

Alpha	Average Train Loss	Average Test Loss
5×10^{-5}	0.0119	0.2009
5×10^{-4}	0.0047	0.1816
5×10^{-3}	∞	∞

Table 7: Effect of Different Learning Rates on Training and Testing Loss

This solution absolutely blows the first one out of the water. Without even doing any explicit regularization, it beats the best ridge solution. Wow.

(d) Here is the plot of the train loss vs converging α, test loss vs converging α and l2 norm vs converging α:

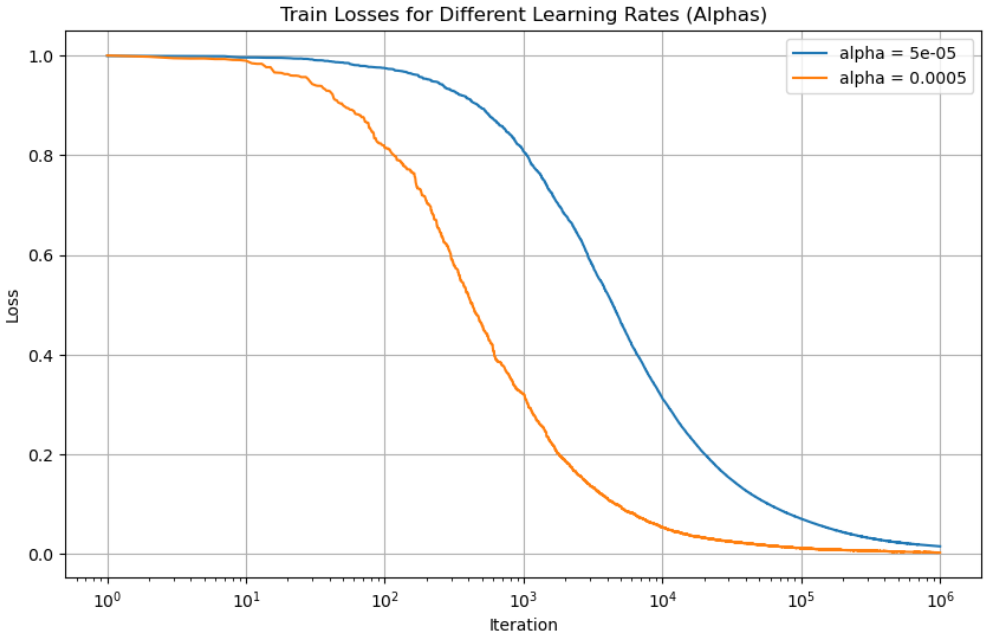

Figure 7: Converging Alpha

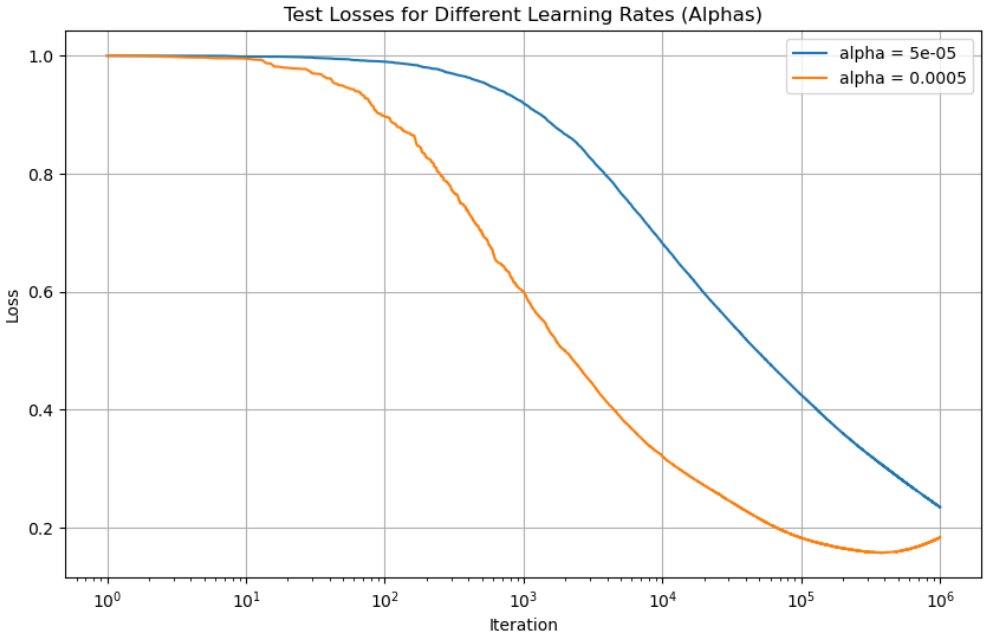

Figure 8: Converging Alpha

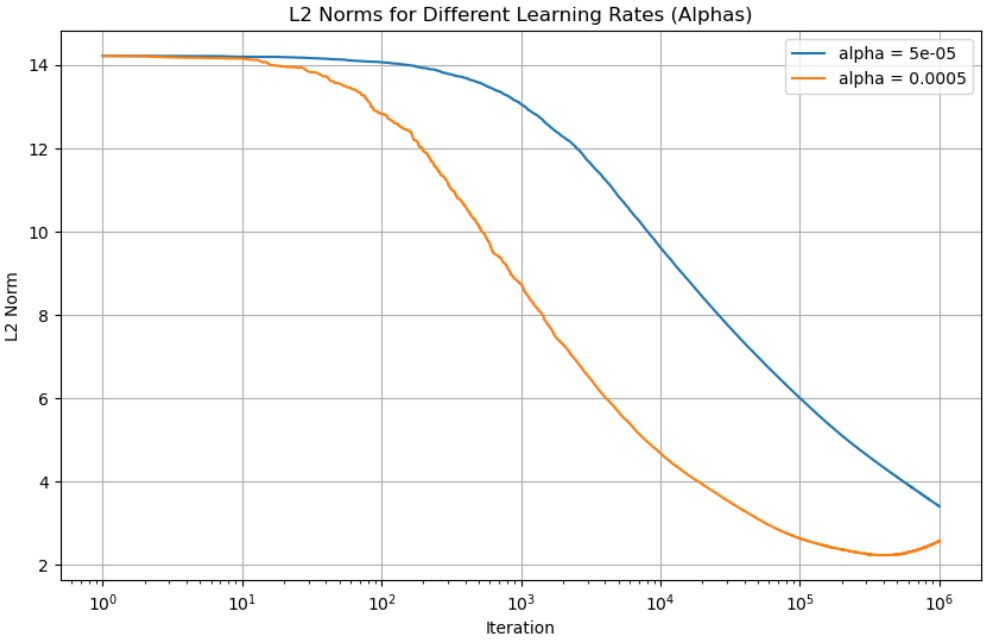

Figure 9: Converging Alpha

SGD seems to generalize extremely well with different step sizes. Although they converge at different rates, as you can see, since it doesn't seem to overfit until maybe at the very end, and seems to be constantly converging towards a minimum. It

seems that the amount that a specific α generalizes is positively correlated with the ℓ^2 norm of the distance between $\hat{\theta}$ and θ^*, which makes sense. If we had a perfect $\hat{\theta} = \theta^*$ then the test loss would be 0. So test loss is very related to the distance.

3. When we have $d = 300$ dimensions and $n = 200$ training points, we saw above that equal amounts of training data and dimensions works really well with SGD, and the full gradient optimizaiton problem does not. So I will use a form of SGD, as it seems from above to generalize better. I also don't want to have to worry about learning rates, so I will use the adaptive learning rate that I found above works well, $\alpha_t = \frac{\alpha_0}{d} \max_j |\nabla L_i(\theta)|$. I ran a hyperparameter search over $\alpha \in \{0.00005, 0.0005, 0.005\}$ and $\lambda \in \{0.0005, 0.005, 0.05, 0.5, 5\}$. I learned that $T = 10,000$ iterations wasn't enough to converge, so I ended up using $T = 100,000$ iterations. After searching through the entire (α, λ) hyperparameter space, the best parameters I got were $\alpha = 0.0005$ and $\lambda = 0.05$ with final average normalized train loss of 0.07226665890624548 and the best normalized test loss I got was 0.592696195064302. This wasn't very good but the data is very noisy and we don't have a lot of data, so in the grand scheme of things it is probably okay.