CS 229, Winter 2023
Problem Set #1

Due Wednesday, January 24 at 11:59 pm on Gradescope.

Notes: (1) These questions require thought, but do not require long answers. Please be as concise as possible.

(2) If you have a question about this homework, we encourage you to post your question on our Ed forum, at `https://edstem.org/us/courses/51342/discussion/`.

(3) If you missed the first lecture or are unfamiliar with the collaboration or honor code policy, please read the policy on the course website before starting work.

(4) For the coding problems, you may not use any libraries except those defined in the provided `environment.yml` file. In particular, ML-specific libraries such as scikit-learn are not permitted.

(5) The due date is Wednesday, January 24 at 11:59 pm. If you submit after Wednesday, January 24 at 11:59 pm, you will begin consuming your late days. The late day policy can be found in the course website: Course Logistics and FAQ.

All students must submit an electronic PDF version of the written question including plots generated from the codes. We highly recommend typesetting your solutions via LaTeX. All students must also submit a zip file of their source code to Gradescope, which should be created using the `make_zip.py` script. You should make sure to (1) restrict yourself to only using libraries included in the `environment.yml` file, and (2) make sure your code runs without errors. Your submission may be evaluated by the auto-grader using a private test set, or used for verifying the outputs reported in the writeup. Please make sure that your PDF file and zip file are submitted to the corresponding Gradescope assignments respectively. We reserve the right to not give any points to the written solutions if the associated code is not submitted.

Honor code: We strongly encourage students to form study groups. Students may discuss and work on homework problems in groups. However, each student must write down the solution independently, and without referring to written notes from the joint session. Each student must understand the solution well enough in order to reconstruct it by him/herself. It is an honor code violation to copy, refer to, or look at written or code solutions from a previous year, including but not limited to: official solutions from a previous year, solutions posted online, and solutions you or someone else may have written up in a previous year. Furthermore, it is an honor code violation to post your assignment solutions online, such as on a public git repo. We run plagiarism-detection software on your code against past solutions as well as student submissions from previous years. Please take the time to familiarize yourself with the Stanford Honor Code[1] and the Stanford Honor Code[2] as it pertains to CS courses.

[1] https://communitystandards.stanford.edu/policies-and-guidance/honor-code

[2] https://web.stanford.edu/class/archive/cs/cs106b/cs106b.1164/handouts/honor-code.pdf

1. **[30 points] Convergence and Learning Rate for Gradient Descent**

When running an optimization algorithm, finding a good learning rate may require some tuning. If the learning rate is too high, the gradient descent (GD) algorithm might not converge. If the learning rate is too low, GD may take too long to converge. In this question, we will investigate some theoretical guarantees for the convergence of GD on convex objective functions, and their implications on choosing the learning rate.

Recap and notation. Suppose we have a convex objective function $J(\theta)$. At each iteration, GD updates the iterate $\theta^{[t]}$ as follows:

$$\theta^{[t]} = \theta^{[t-1]} - \alpha \nabla J(\theta^{[t-1]})$$

where α is the learning rate and $\nabla J(\theta^{[t-1]})$ is the gradient of $J(\theta)$ evaluated at $\theta^{[t-1]}$.

(a) **[8 points] Quadratic Objective, Scalar Variable**

In this part, consider the simple objective function with one-dimensional variable θ:

$$J(\theta) = \frac{1}{2}\beta\theta^2$$

where $\theta \in \mathbb{R}$ is our parameter and $\beta\ > 0$ is a positive, constant scalar. We initialize GD at some initialization $\theta^{[0]} \neq 0$ and run GD with learning rate α.

i. **Derive** the range of α such that the iterate of GD converges in the sense that there exists a scalar $\theta^\dagger$ such that $\lim_{t\to\infty} |\theta^{[t]} - \theta^\dagger| = 0$. Provide the value of $\theta^\dagger$ when GD converges and show that it's equal to the global minimum $\theta^* = \arg\min_\theta J(\theta)$. The range of α can potentially depend on β.

ii. Given a desired accuracy $\epsilon > 0$, for all the α in the range that you derived, **derive** the minimum number of iterations T required to reach a point $\theta^{[T]}$ such that $|\theta^{[T]} - \theta^*| \leq \epsilon$. T can potentially depend on ϵ, $\theta^{[0]}$, α, and β. Investigate the behavior of T and whether T is increasing or decreasing for different values of α. Briefly discuss why choosing an inappropriate α can cause the algorithm to take longer to converge.

Hint: Express $\theta^{[t]}$ in terms of $\theta^{[0]}$, α, t, and β.

Hint: $\lim_{t\to\infty} c^t = 0, \quad \forall c \in \mathbb{R}\ s.t.\ |c| < 1$

Answer: For part (i), using hint 1, derive the following expression

$$\begin{aligned}
\nabla J(\theta^{[t-1]}) &= \frac{1}{2} \cdot 2 \cdot \beta\theta^{[t-1]} \\
&= \theta^{[t-1]}\beta \\
\theta^{[t]} &= \theta^{[t-1]} - \alpha\theta^{[t-1]}\beta \\
&= \theta^{[t-1]}(1-\alpha\beta) \\
&= \theta^{(t-2)}(1-\alpha\beta)^2 \\
&= \cdots \\
&= \theta^{[0]}(1-\alpha\beta)^t
\end{aligned}$$

We know that the optimal value θ^* is 0 (you can either get this by looking at the graph or by setting the derivative to 0 and solving for θ). Thus, in order for the algorithm to converge, we need $\lim_{t\to\infty} \theta^{[t]}\ = \theta^* = 0$. Using the second hint, we need the condition $|1-\alpha\beta| < 1$.

i. $1 - \alpha\beta < 1$ leads to $\alpha > 0$

ii. $\alpha\beta - 1 < 1$ leads to $\alpha < \frac{2}{\beta}$ (Note dividing by β does not change the sign since $\beta > 0$)

So the answer is $0 < \alpha < \frac{2}{\beta}$.

For part (ii), we first observe that when $\alpha = \frac{1}{\beta}$, we only need 1 iteration to achieve the optimal value $\theta^* = 0$. When $\alpha = \frac{2}{\beta}$, the sign of θ simply changes with each iteration and the algorithm does not converge. For all other $\alpha \neq \frac{1}{\beta}$ within our bound $0 < \alpha < \frac{2}{\beta}$, using the derivation from part (i), we can see that

$$
\begin{aligned}
|\theta^{[T]} - \theta^*| &\leq \epsilon \\
|\theta^{[0]}(1 - \alpha\beta)^T| &\leq \epsilon \\
|\theta^{[0]}||(1 - \alpha\beta)^T| &\leq \epsilon \\
|1 - \alpha\beta|^T &\leq \frac{\epsilon}{|\theta^{[0]}|} \\
T \ln|1 - \alpha\beta| &\leq \ln\epsilon - \ln|\theta^{[0]}| \\
T &\geq \frac{\ln\epsilon - \ln|\theta^{[0]}|}{\ln|1 - \alpha\beta|}
\end{aligned}
$$

When $0 < \alpha < \frac{1}{\beta}$, as α increases, the RHS decreases, meaning the minimum number of iterations decreases (the algorithm converges faster).

When $\frac{1}{\beta} < \alpha < \frac{2}{\beta}$, as α increases, the RHS increases, meaning the minimum number of iterations increases (the algorithm converges slower).

(b) **[4 points] Quadratic Objective, d-dimensional Variable**

In this part of the question, consider the objective function with d-dimensional variable θ:

$$J(\theta) = \frac{1}{2}\sum_{i=1}^{d} \beta_i \theta_i^2$$

where θ_i's are our parameters and $\beta_i \in \mathbb{R}$ $s.t.$ $\beta_i > 0$ are positive, constant scalars. Assume that we start from some $\theta^{[0]}$ where $\theta_i^{[0]} \neq 0$ for all i and run GD with learning rate α. Derive the range of learning rate α such that GD converges in the sense that there exists a vector $\theta^\dagger \in \mathbb{R}^d$ such that $\lim_{t\to\infty} \|\theta^{[t]} - \theta^\dagger\|_2 = 0$. Provide the value of $\theta^\dagger$ when GD converges. The range of α can potentially depend on β_i where $i \in \{1, \ldots, d\}$.

Answer: Similar to part (a), we quickly derive the update formula for each $\theta_i^{[t]}$ as follows

$$
\begin{aligned}
\theta_i^{[t]} &= \theta_i^{[t-1]} - \alpha\theta_i^{[t-1]}\beta_i \\
&= \theta_i^{[t-1]}(1 - \alpha\beta_i) \\
&= \theta_i^{[0]}(1 - \alpha\beta_i)^t
\end{aligned}
$$

It is also obvious that the optimal value $\theta_i^* = 0 \quad \forall i \in (1, d)$. So in order for $\theta_i^{[t]}$ to converge, $|1 - \alpha\beta_i| < 1 \; \forall i \in (1, d)$.

i. $1 - \alpha\beta_i < 1$ leads to $\alpha > 0$

ii. $\alpha\beta_i - 1 < 1$ leads to $\alpha < \frac{2}{\beta_{\max}}$ where $\beta_{\max} = \max_i \beta_i$

So the solution is $0 < \alpha < \frac{2}{\beta_{\max}}$

(c) [**4 points**] **Coding Question: Quadratic Multivariate Objective**

Let the objective function be

$$J(\theta) = \theta^\top A \theta$$

where $A \in \mathbb{R}^{2\times 2}$ is a 2×2 real, positive semi-definite matrix. Do the following exercises:

i. Implement the `update_theta` and `gradient_descend` function in `src/gd_convergence/experiment.py`. You can stop the GD algorithm when either of the following condition is satisfied:

   A. $|J(\theta^{[t]}) - J(\theta^{[t-1]})| < \epsilon$ where $\epsilon = 10^{-50}$ is given as the function parameter. This is when we assume the algorithm converged.

   B. $J(\theta^{[t]}) > 10^{20}$. This is to prevent an infinite loop when the algorithm does not converge.

   To test your implementation, run `python src/gd_convergence/experiment.py`, which checks that your θ (approximately) converges to the optimal value.

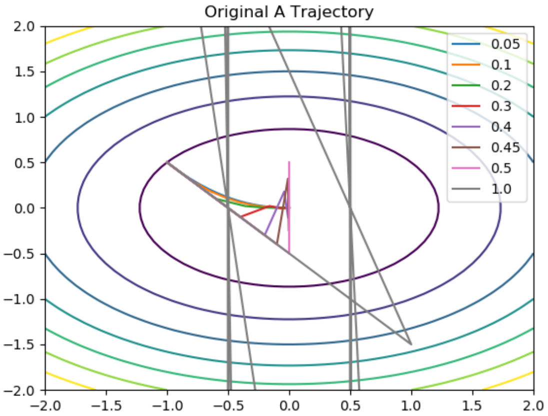
   Note that we have provided you a matrix $A = \begin{bmatrix} 1 & 0 \\ 0 & 2 \end{bmatrix}$ and $\theta^{[0]} = [-1, 0.5]$ at the beginning of the file for you to experiment with. Therefore, the objective function is a special case of the objective function in part (b) with dimension $d = 2$. Check if your theoretical derivation of the feasible range of the learning rate indeed matches empirical observations.

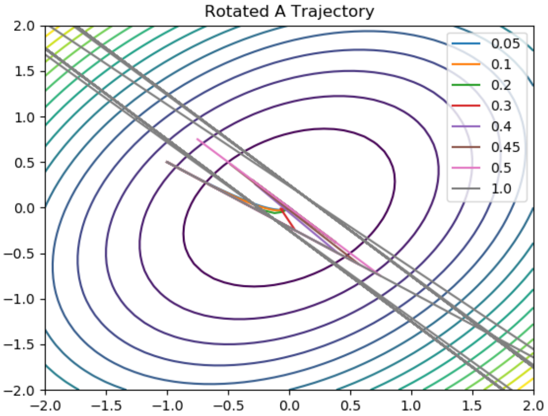
ii. Now, suppose we rotate the matrix A, what do we observe? Plot the trajectories of the GD algorithm using the following learning rates: 0.05, 0.1, 0.2, 0.3, 0.4, 0.45, 0.5, and 1. We have provided the rotation and plotting function for you. You need to simply run `python src/gd_convergence/plotting.py lr1 lr2 lr3 ...` with `lr*` replaced with desired learning rates. Include the output file `trajectories.png` and `trajectories_rotated.png`, as well as a brief discussion on the printed output of `plotting.py` in the write-up.

   Remark: If you find that a learning rate of 0.5 is resulting in non-terminating behavior, this may be due to numerical instability. In this case, feel free to use a learning rate of 0.55 in place of 0.5.

Note that setting the learning rate too high will cause the objective function to not converge. The convergence properties of these learning rates are also rotational invariant. We will show this formally in the next part of the problem.

Answer:





Printed output:

```
[diagonal A, learning rate 0.05] GD took 541 iterations and converged
[diagonal A, learning rate 0.1] GD took 258 iterations and converged
[diagonal A, learning rate 0.2] GD took 115 iterations and converged
[diagonal A, learning rate 0.3] GD took 65 iterations and converged
[diagonal A, learning rate 0.4] GD took 114 iterations and converged
[diagonal A, learning rate 0.45] GD took 257 iterations and converged
[diagonal A, learning rate 0.5] GD took 3 iterations and did not converge
[diagonal A, learning rate 1.0] GD took 23 iterations and did not converge
[rotated A, learning rate 0.05] GD took 531 iterations and converged
[rotated A, learning rate 0.1] GD took 254 iterations and converged
[rotated A, learning rate 0.2] GD took 113 iterations and converged
[rotated A, learning rate 0.3] GD took 64 iterations and converged
[rotated A, learning rate 0.4] GD took 116 iterations and converged
[rotated A, learning rate 0.45] GD took 260 iterations and converged
[rotated A, learning rate 0.5] GD took 3 iterations and did not converge
[rotated A, learning rate 1.0] GD took 22 iterations and did not converge
```

If the learning rate is set too high, GD does not converge. Within the range of learning rates such that GD converges, the convergence speed (i.e. number of iterations to converge) of GD varies, and we observe that increasing the learning rate first causes the number of iterations to converge to decrease but then causes the number of iterations to converge to increase. Note that it is not necessarily true that larger learning rates will make the algorithm converge faster (as shown in problem (a)).

For a specific choice of learning rate, the convergence and the convergence speed of GD are similar for the original A and the rotated A.

(d) **[12 points] Convergence of Gradient Descent for a General Convex Objective**

Now let's consider any convex, twice continuously differentiable objective function $J(\theta)$ that is bounded from below. Suppose that the largest eigenvalue of the Hessian matrix $H = \nabla^2 J(\theta)$ is less than or equal to $\beta_{\max}$ for all points θ.

Show that when running gradient descent, choosing a step size $0 < \alpha < \frac{1}{\beta_{\max}}$ guarantees that $J(\theta^{[t]})$ will converge to some finite value as t approaches infinity.[3] Specifically, you should derive an inequality for $J(\theta^{[t]})$ in terms of $J(\theta^{[t-1]})$, $\nabla J(\theta^{[t-1]})$, α, and $\beta_{\max}$, and show that the objective function is strictly decreasing during each iteration, i.e. $J(\theta^{[t]}) < J(\theta^{[t-1]})$.

Hint: You can assume that, by Taylor's theorem, the following statement is true:

$$J(y) = J(x) + \nabla J(x)^\top (y-x) + \frac{1}{2}(y-x)^\top \nabla^2 J(x + c(y-x))(y-x) \text{ for some } 0 \le c \le 1$$

Hint: The Hessian of a convex function is symmetric and positive semi-definite at any point θ, which means all of its eigenvalues are real and non-negative.

Hint: The Hessian matrix is symmetric. Consider using the spectral theorem introduced in problem 3 in homework 0.

Optional (no credit): Using the gradient descent inequality that you derived, show that the GD algorithm converges in the sense that $\lim_{t\to\infty} ||\nabla J(\theta^{[t]})||_2^2 = 0$.[4]

Remark: This question suggests that a smaller learning rate should be applied if we believe the curvature of the objective function is big.

Answer: First, using hint 3, we can see that

$$\begin{aligned} H &= U\Lambda U^\top \\ &\preceq U\beta_{\max} I U^\top \\ &= \beta_{\max} U U^\top \\ &= \beta_{\max} I \end{aligned}$$

By Taylor's theorem for $J(\theta)$ around $\theta^{[t-1]}$ and then plugging in $\theta = \theta^{[t]}$

$$\begin{aligned} J(\theta^{[t]}) &= J(\theta^{[t-1]}) + \nabla J(\theta^{[t-1]})^\top (\theta^{[t]} - \theta^{[t-1]}) + \frac{1}{2}(\theta^{[t]} - \theta^{[t-1]})^\top \nabla^2 f(\theta^{[t-1]} + c(\theta^{[t]} - \theta^{[t-1]}))(\theta^{[t]} - \theta^{[t-1]}) \\ &\le J(\theta^{[t-1]}) + \nabla J(\theta^{[t-1]})^\top (\theta^{[t]} - \theta^{[t-1]}) + \frac{1}{2}\beta_{\max} ||\theta^{[t]} - \theta^{[t-1]}||_2^2 \end{aligned}$$

[3] This definition of convergence is different from the definition in previous questions where we explicitly prove that the parameter θ converges to an optimal parameter θ^*. In this case, we do not know the optimal value θ^*, and it would be difficult to prove convergence using the previous definition.

[4] Note that $\lim_{t\to\infty} ||\nabla J(\theta^{[t]})||_2^2 = 0$ does not necessarily mean that there exists a vector $\hat{\theta}$ such that $\theta^{[t]} \to \hat{\theta}$. (Even an 1-dimensional counterexample exists.) However, for convex functions, when $\theta^{[t]}$ stays in a bounded set, $\lim_{t\to\infty} ||\nabla J(\theta^{[t]})||_2^2 = 0$ does imply that $J(\theta^{[t]}) \to \inf_\theta J(\theta)$ as $t \to \infty$. Therefore, $\lim_{t\to\infty} ||\nabla J(\theta^{[t]})||_2^2 = 0$ is typically considered a reasonable definition for an optimization algorithm's convergence.

Now plug in the update formula $\theta^{[t]} = \theta^{[t-1]} - \alpha \nabla J(\theta^{[t-1]})$

$$\begin{aligned}
J(\theta^{[t]}) &\leq J(\theta^{[t-1]}) + \nabla J(\theta^{[t-1]})^\top (\theta^{[t]} - \theta^{[t-1]}) + \frac{1}{2}\beta_{\max}||\theta^{[t]} - \theta^{[t-1]}||_2^2 \\
&= J(\theta^{[t-1]}) + \nabla J(\theta^{[t-1]})^\top (\theta^{[t-1]} - \alpha\nabla J(\theta^{[t-1]}) - \theta^{[t-1]}) + \frac{1}{2}\beta_{\max}||\theta^{[t-1]} - \alpha\nabla J(\theta^{[t-1]}) - \theta^{[t-1]}||_2^2 \\
&= J(\theta^{[t-1]}) - \nabla J(\theta^{[t-1]})^\top (\alpha\nabla J(\theta^{[t-1]})) + \frac{1}{2}\beta_{\max}||\alpha\nabla J(\theta^{[t-1]})||_2^2 \\
&= J(\theta^{[t-1]}) - \alpha||\nabla J(\theta^{[t-1]})||_2^2 + \frac{1}{2}\beta_{\max}\alpha^2||\nabla J(\theta^{[t-1]})||_2^2 \\
&= J(\theta^{[t-1]}) - (1 - \frac{1}{2}\alpha\beta_{\max})\alpha||\nabla J(\theta^{[t-1]})||_2^2
\end{aligned}$$

When $\theta^{[t-1]}$ is not at an optimal solution, the gradient $\nabla J(\theta^{[t-1]}) \neq 0$, so $||\nabla J(\theta^{[t-1]})||_2^2 > 0$. Since $0 < \alpha < \frac{1}{\beta_{\max}}$, $(1 - \frac{1}{2}\alpha\beta_{\max}) > \frac{1}{2}$ and $\alpha > 0$.
Thus, $(1 - \frac{1}{2}\alpha\beta_{\max})\alpha||\nabla J(\theta^{[t-1]})||_2^2 > 0$, which means that $J(\theta^{[t]})$ is strictly decreasing.

Solution for optional part

By rearranging the inequality above, we have that

$$\begin{aligned}
0 \leq (1 - \frac{1}{2}\alpha\beta_{\max})\alpha||\nabla J(\theta^{[t-1]})||_2^2 &\leq J(\theta^{[t-1]}) - J(\theta^{[t]}) \\
0 \leq ||\nabla J(\theta^{[t-1]})||_2^2 &\leq \frac{J(\theta^{[t-1]}) - J(\theta^{[t]})}{(1 - \frac{1}{2}\alpha\beta_{\max})\alpha}
\end{aligned}$$

Now take the limit as $t \to \infty$. Since $\lim_{t\to\infty} \frac{J(\theta^{[t-1]}) - J(\theta^{[t]})}{(1-\frac{1}{2}\alpha\beta_{\max})\alpha} = 0$, this shows that $\lim_{t\to\infty} ||\nabla J(\theta^{[t-1]})||_2^2 = 0$.

(e) **[2 points] Learning Rate for Linear Regression**

Consider using GD on the LMS objective introduced in lecture:

$$J(\theta) = \frac{1}{2}||X\theta - y||_2^2$$

where $X \in \mathbb{R}^{n\times d}$ is the design matrix of our data, $\theta \in \mathbb{R}^d$ is our parameter, and $\vec{y} \in \mathbb{R}^n$ is the response variable.

Let $\beta_{\max}$ be the largest eigenvalue of $X^\top X$. Prove that for all $\alpha \in (0, \frac{1}{\beta_{\max}})$, GD with learning rate α satisfies that $J(\theta^{[t]})$ converges as $t \to \infty$.

Hint: You can invoke the statements in the part (d) (including the optional question in part (d)) to solve this part (even if you were not able to prove part (d).)

Remark: The conclusion here suggests that for linear regression, roughly speaking, you should use smaller learning rates if the scale of your data X is big or if the data points are correlated with each other, both of which will enable a large top eigenvalue of $XX^\top$.

Remark: Even though in many cases the LMS objective can be solved exactly by solving the normal equation as shown in lecture, it is still useful to be able to guarantee convergence when using the Gradient Descent algorithm in situations where it is difficult to solve for the optimal θ^* directly (such as having large amount of data making inverting X very expensive).

Answer: Referring to the lecture notes, we can see that

$$\nabla J(\theta) = X^\top X\theta - X^\top y$$

So it is easy to see that

$$\nabla^2 J(\theta) = X^\top X$$

Let $\beta_{\max}$ be the largest eigenvalue of $X^\top X$. By part (d), the range is $0 < \alpha < \frac{1}{\beta_{\max}}$

2. **[25 points] Locally weighted linear regression**

(a) [10 points] Consider a linear regression problem in which we want to "weight" different training examples differently. Specifically, suppose we want to minimize

$$J(\theta) = \frac{1}{2}\sum_{i=1}^{n} w^{(i)} \left(\theta^T x^{(i)} - y^{(i)}\right)^2 .$$

In class, we worked out what happens for the case where all the weights (the $w^{(i)}$'s) are the same. In this problem, we will generalize some of those ideas to the weighted setting. We will assume $w^{(i)} > 0$ for all i.

i. [2 points] Show that $J(\theta)$ can also be written

$$J(\theta) = (X\theta - y)^T W (X\theta - y)$$

for an appropriate matrix W, and where X and y are as defined in class. Clearly specify the value of each element of the matrix W.

ii. [4 points] If all the $w^{(i)}$'s equal 1, then we saw in class that the normal equation is

$$X^T X\theta = X^T y,$$

and that the value of θ that minimizes $J(\theta)$ is given by $(X^T X)^{-1} X^T y$. By finding the derivative $\nabla_\theta J(\theta)$ and setting that to zero, generalize the normal equation to this weighted setting, and give the new value of θ that minimizes $J(\theta)$ in closed form as a function of X, W and y.

iii. [4 points] Suppose we have a dataset $\{(x^{(i)}, y^{(i)}); i = 1\ldots, n\}$ of n independent examples, but we model the $y^{(i)}$'s as drawn from conditional distributions with different levels of variance $(\sigma^{(i)})^2$. Specifically, assume the model

$$p(y^{(i)}|x^{(i)};\theta) = \frac{1}{\sqrt{2\pi}\sigma^{(i)}} \exp\left(-\frac{(y^{(i)} - \theta^T x^{(i)})^2}{2(\sigma^{(i)})^2}\right)$$

That is, each $y^{(i)}$ is drawn from a Gaussian distribution with mean $\theta^T x^{(i)}$ and variance $(\sigma^{(i)})^2$ (where the $\sigma^{(i)}$'s are fixed, known, constants). Show that finding the maximum likelihood estimate of θ reduces to solving a weighted linear regression problem. State clearly what the $w^{(i)}$'s are in terms of the $\sigma^{(i)}$'s.

In other words, this suggests that if we have prior knowledge on the noise levels (the variance of the label $y^{(i)}$ conditioned on $x^{(i)}$) of all the examples, then we should use weighted least squares with weights depending on the variances.

Answer:

i. Let $W_{ii} = \frac{1}{2}w^{(i)}, W_{ij} = 0$ for $i \neq j$, let $\vec{z} = X\theta - \vec{y}$, i.e. $z_i = \theta^T x^{(i)} - y^{(i)}$. Then we have:

$$\begin{aligned}(X\theta - \vec{y})^T W (X\theta - \vec{y}) &= \vec{z}^T W \vec{z} \\ &= \frac{1}{2}\sum_{i=1}^{n} w^{(i)} z_i^2 \\ &= \frac{1}{2}\sum_{i=1}^{n} w^{(i)} (\theta^T x^{(i)} - y^{(i)})^2 \\ &= J(\theta)\end{aligned}$$

ii.

$$\nabla_\theta J(\theta) = \nabla_\theta(\theta^T X^T W X\theta + \vec{y}^T W\vec{y} - 2\vec{y}^T W X\theta) = 2(X^T W X\theta - X^T W\vec{y}),$$

so we have $\nabla_\theta J(\theta) = 0$ if and only if

$$X^T W X\theta = X^T W\vec{y}$$

These are the normal equations, from which we can get a closed form formula for θ.

$$\theta = (X^T W X)^{-1} X^T W\vec{y}$$

iii.

$$\begin{aligned}
\arg\max_\theta \prod_{i=1}^n p(y^{(i)}|x^{(i)};\theta) &= \arg\max_\theta \sum_{i=1}^n \log p(y^{(i)}|x^{(i)};\theta) \\
&= \arg\max_\theta \sum_{i=1}^n \left(\log\frac{1}{\sqrt{2\pi}\sigma^{(i)}} - \frac{(y^{(i)} - \theta^T x^{(i)})^2}{2(\sigma^{(i)})^2}\right) \\
&= \arg\max_\theta - \sum_{i=1}^n \frac{(y^{(i)} - \theta^T x^{(i)})^2}{2(\sigma^{(i)})^2} \\
&= \arg\min_\theta \frac{1}{2}\sum_{i=1}^n \frac{1}{(\sigma^{(i)})^2}(y^{(i)} - \theta^T x^{(i)})^2 \\
&= \arg\min_\theta \frac{1}{2}\sum_{i=1}^n w^{(i)}(y^{(i)} - \theta^T x^{(i)})^2
\end{aligned}$$

where in the last step, we substituted: $w^{(i)} = \frac{1}{(\sigma^{(i)})^2}$ to get the linear regression form.

(b) [10 points] **Coding problem.**

We will now consider the following dataset (the formatting matches that of Datasets 1-4, except $x^{(i)}$ is 1-dimensional):

`src/lwr/{train,valid,test}.csv`

In `src/lwr/lwr.py`, implement locally weighted linear regression using the normal equations you derived in Part (a) and using

$$w^{(i)} = \exp\left(-\frac{\|x^{(i)} - x\|_2^2}{2\tau^2}\right).$$

This is a fairly standard choice for weights, where the weight $w^{(i)}$ depends on the particular point x at which we're trying to evaluate y: if $|x^{(i)} - x|$ is small, then $w^{(i)}$ is close to 1; if $|x^{(i)} - x|$ is large, then $w^{(i)}$ is close to 0. Here, τ is the bandwidth parameter, and it controls how quickly the weight of a training example falls off with distance of its $x^{(i)}$ from the query point x.

Train your model on the `train` split using $\tau = 0.5$, then run your model on the `valid` split and report the mean squared error (MSE). Finally plot your model's predictions on the validation set (plot the training set with blue 'x' markers and the predictions on the validation set with a red 'o' markers). Does the model seem to be under- or overfitting?

Answer: Validation MSE: 0.3305. Underfitting.

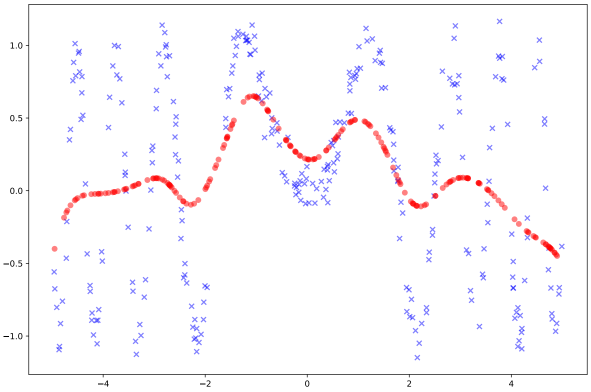


Figure 1: Training set and first validation example with predicted regression line using $\tau = 0.5$.

(c) [5 points] **Coding problem.**

We will now tune the hyperparameter τ. In `src/lwr/tau.py`, find the MSE value of your model on the validation set for each of the values of τ specified in the code. For each τ, plot your model's predictions on the validation set in the format described in part (b). Report the value of τ which achieves the lowest MSE on the `valid` split, and finally report the MSE on the `test` split using this τ-value.

Answer: $\tau^* = 0.05$
$\mathsf{MSE}_{\texttt{valid}} = 0.0124001$
$\mathsf{MSE}_{\texttt{test}} = 0.0169901.$

Validation prediction plots:

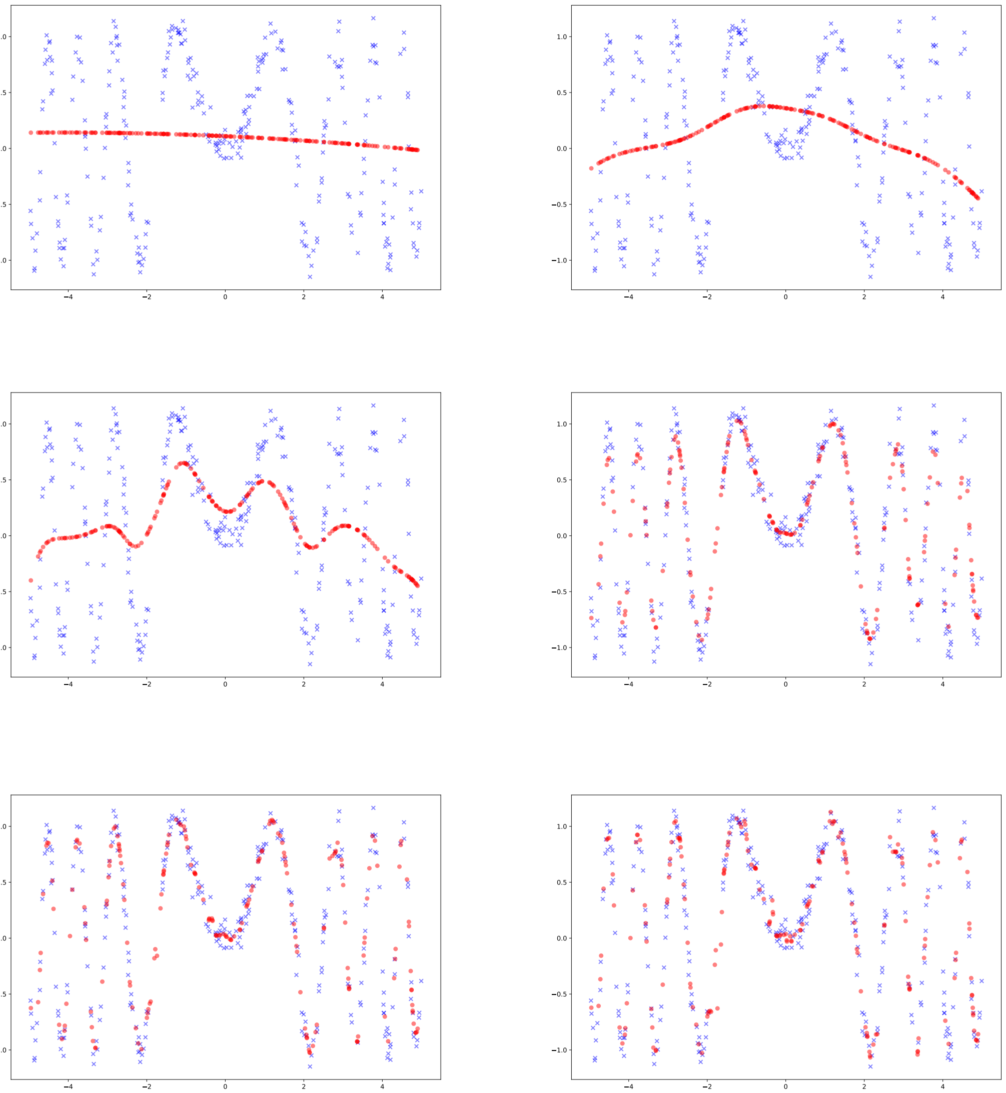


Figure 2: LWR predictions on the validation set for $\tau = 10, 1, 0.5, 0.1, 0.05, 0.03$ (from left-to-right, then top-to-bottom). Notice the progression from underfitting to overfitting as we decrease τ.

3. **[25 points] Linear regression: linear in what?**

In the first two lectures, you have seen how to fit a linear function of the data for the regression problem. In this question, we will see how linear regression can be used to fit non-linear functions of the data using feature maps. We will also explore some of its limitations, for which future lectures will discuss fixes.

(a) [5 points] **Learning degree-3 polynomials of the input**

Suppose we have a dataset $\{(x^{(i)}, y^{(i)})\}_{i=1}^n$ where $x^{(i)}, y^{(i)} \in \mathbb{R}$. We would like to fit a third degree polynomial $h_\theta(x) = \theta_3 x^3 + \theta_2 x^2 + \theta_1 x^1 + \theta_0$ to the dataset. The key observation here is that the function $h_\theta(x)$ is still linear in the unknown parameter θ, even though it's not linear in the input x. This allows us to convert the problem into a linear regression problem as follows.

Let $\phi : \mathbb{R} \to \mathbb{R}^4$ be a function that transforms the original input x to a 4-dimensional vector defined as

$$\phi(x) = \begin{bmatrix} 1 \\ x \\ x^2 \\ x^3 \end{bmatrix} \in \mathbb{R}^4 \tag{1}$$

Let $\hat{x} \in \mathbb{R}^4$ be a shorthand for $\phi(x)$, and let $\hat{x}^{(i)} \triangleq \phi(x^{(i)})$ be the transformed input in the training dataset. We construct a new dataset $\{(\phi(x^{(i)}), y^{(i)})\}_{i=1}^n = \{(\hat{x}^{(i)}, y^{(i)})\}_{i=1}^n$ by replacing the original inputs $x^{(i)}$'s by $\hat{x}^{(i)}$'s. We see that fitting $h_\theta(x) = \theta_3 x^3 + \theta_2 x^2 + \theta_1 x^1 + \theta_0$ to the old dataset is equivalent to fitting a linear function $h_\theta(\hat{x}) = \theta_3 \hat{x}_3 + \theta_2 \hat{x}_2 + \theta_1 \hat{x}_1 + \theta_0$ to the new dataset because

$$h_\theta(x) = \theta_3 x^3 + \theta_2 x^2 + \theta_1 x^1 + \theta_0 = \theta_3 \phi(x)_3 + \theta_2 \phi(x)_2 + \theta_1 \phi(x)_1 + \theta_0 = \theta^T \hat{x} \tag{2}$$

In other words, we can use linear regression on the new dataset to find parameters $\theta_0, \ldots, \theta_3$. Please write down 1) the objective function $J(\theta)$ of the linear regression problem on the new dataset $\{(\hat{x}^{(i)}, y^{(i)})\}_{i=1}^n$ and 2) the update rule of the batch gradient descent algorithm for linear regression on the dataset $\{(\hat{x}^{(i)}, y^{(i)})\}_{i=1}^n$.

Terminology: In machine learning, ϕ is often called the feature map which maps the original input x to a new set of variables. To distinguish between these two sets of variables, we will call x the input **attributes**, and call $\phi(x)$ the **features**. (Unfortunately, different authors use different terms to describe these two things. In this course, we will do our best to follow the above convention consistently.)

Answer:

$$J(\theta) = \frac{1}{2} \sum_{i=1}^{n} (\theta^T \hat{x}^{(i)} - y^{(i)})^2$$

Differentiating this objective, we get the update rule:

$$\begin{aligned} \nabla_\theta J(\theta) &= \sum_{i=1}^{n} (\theta^T \hat{x}^{(i)} - y^{(i)})(\hat{x}^{(i)}) \\ &\propto \lambda \sum_{i=1}^{n} (\theta^T \hat{x}^{(i)} - y^{(i)})(\hat{x}^{(i)}) \end{aligned}$$

Where λ is the learning rate. The update rule is:

$$\theta := \theta - \lambda \sum_{i=1}^{n} (\theta^T \hat{x}^{(i)} - y^{(i)})(\hat{x}^{(i)})$$

(b) [5 points] **Coding question: degree-3 polynomial regression**

For this sub-question question, we will use the dataset provided in the following files:

`src/featuremaps/{train,test}.csv`

Each file contains two columns: x and y. In the terminology described in the introduction, x is the attribute (in this case one dimensional) and y is the output label.

Using the formulation of the previous sub-question, implement linear regression with **normal equations** using the feature map of degree-3 polynomials. Use the starter code provided in `src/featuremaps/featuremap.py` to implement the algorithm.

Create a scatter plot of the training data, and plot the learnt hypothesis as a smooth curve over it. Submit the plot in the writeup as the solution for this problem.

Remark: Suppose $\widehat{X}$ is the design matrix of the transformed dataset. You may sometimes encounter a non-invertible matrix $\widehat{X}^T\widehat{X}$. For a numerically stable code implementation, always use `np.linalg.solve` to obtain the parameters directly, rather than explicitly calculating the inverse and then multiplying it with $\widehat{X}^T y$.

Answer:

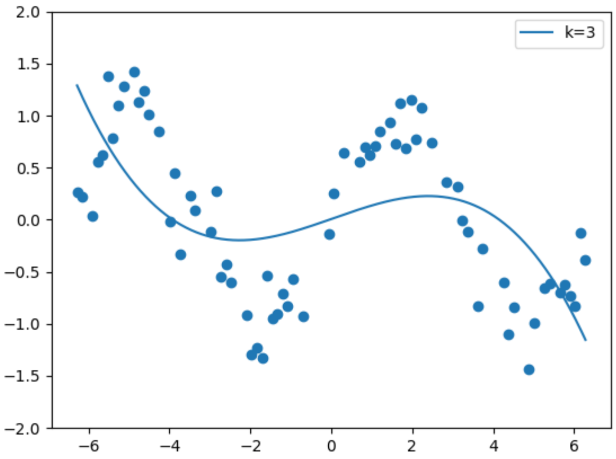


Figure 3: Polynomial regression with degree 3

(c) [5 points] **Coding question: degree-k polynomial regression**

Now we extend the idea above to degree-k polynomials by considering $\phi : \mathbb{R} \to \mathbb{R}^{k+1}$ to be

$$\phi(x) = \begin{bmatrix} 1 \\ x \\ x^2 \\ \vdots \\ x^k \end{bmatrix} \in \mathbb{R}^{k+1} \tag{3}$$

Follow the same procedure as the previous sub-question, and implement the algorithm with $k = 3, 5, 10, 20$. Create a similar plot as in the previous sub-question, and include the hypothesis curves for each value of k with a different color. Include a legend in the plot to indicate which color is for which value of k.

Submit the plot in the writeup as the solution for this sub-problem. Observe how the fitting of the training dataset changes as k increases. Briefly comment on your observations in the plot. **Answer:**

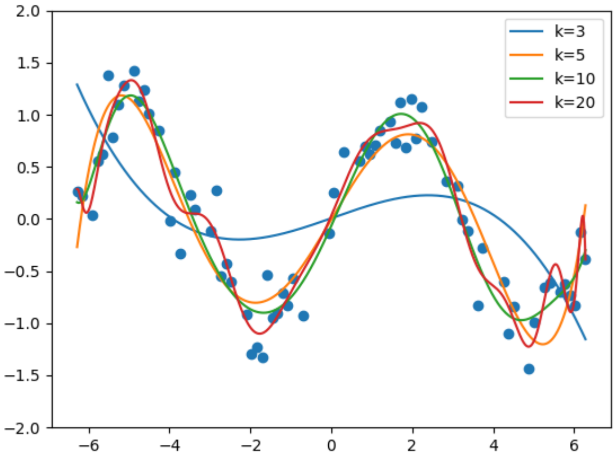


Figure 4: Polynomial regression with kernel sizes 3,5,10 and 20

Generally higher degree polynomials tend to fit the data better, though very high degree polynomials can be numerically unstable.

(d) [5 points] **Coding question: other feature maps**

You may have observed that it requires a relatively high degree k to fit the given training data, and this is because the dataset cannot be explained (i.e., approximated) very well by low-degree polynomials. By visualizing the data, you may have realized that y can be approximated well by a sine wave. In fact, we generated the data by sampling from $y = \sin(x) + \xi$, where ξ is noise with Gaussian distribution. Please update the feature map ϕ to include a sine transformation as follows:

$$\phi(x) = \begin{bmatrix} 1 \\ x \\ x^2 \\ \vdots \\ x^k \\ \sin(x) \end{bmatrix} \in \mathbb{R}^{k+2} \tag{4}$$

With the updated feature map, train different models for values of $k = 0, 1, 2, 3, 5, 10, 20$, and plot the resulting hypothesis curves over the data as before.

Submit the plot as a solution to this sub-problem. Compare the fitted models with the previous sub-question, and briefly comment about noticeable differences in the fit with this feature map. **Answer:**

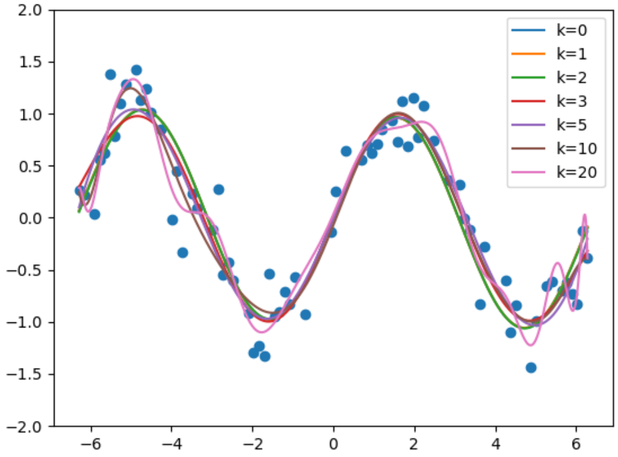


Figure 5: Polynomial regression with other features with kernel sizes 0,1,2,3,5,10 and 20

In the presence of the sin(x) feature, the models seem to fit the data better / more robustly. However the numerical instability that comes with high degree polynomials remain.

(e) [5 points] **Overfitting with expressive models and small data**
For the rest of the problem, we will consider a small dataset (a random subset of the dataset you have been using so far) with much fewer examples, provided in the following file:

`src/featuremaps/small.csv`

We will be exploring what happens when the number of features start becoming bigger than the number of examples in the training set. Run your algorithm on this small dataset using the following feature map

$$\phi(x) = \begin{bmatrix} 1 \\ x \\ x^2 \\ \vdots \\ x^k \end{bmatrix} \in \mathbb{R}^{k+1} \tag{5}$$

with $k = 1, 2, 5, 10, 20$.
Create a plot of the various hypothesis curves (just like previous sub-questions). Observe how the fitting of the training dataset changes as k increases. Submit the plot in the writeup and comment on what you observe.

Answer:

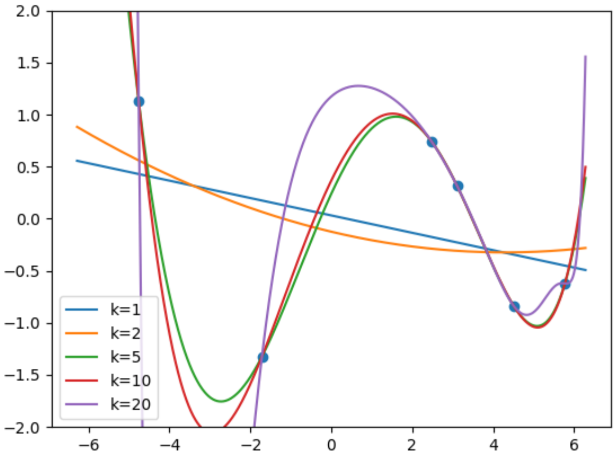


Figure 6: Polynomial regression with kernel sizes 1,2,5,10 and 20 on small dataset

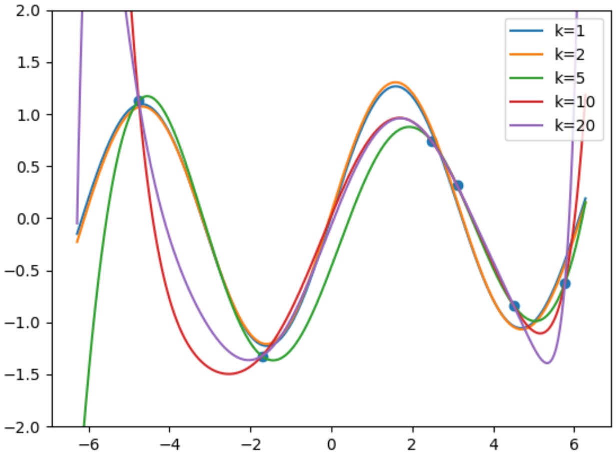


Figure 7: Regression with other features with kernel sizes 1,2,5,10 and 20 on small dataset

We see that when the dataset is small, higher degree polynomials tend to pass through all the points, but qualitatively seem like a poor fit. Numerical instability with high degree polynomials remain a problem even with small data, with or without sin(x).

4. **[35 points] Implicit Regularization**

Recall that in the overparameterized regime (where the number of parameters is larger than the number of samples), typically there are infinitely many solutions that can fit the training dataset perfectly, and many of them cannot generalize well (that is, they have large test errors). However, in many cases, the particular optimizer we use (e.g., GD, stochastic GD with particular learning rates (see part (g) for more details), batch sizes, noise, etc.) tends to find solutions that generalize well. This phenomenon is called implicit regularization effect (also known as algorithmic regularization or implicit bias).

In this problem, we will look at the implicit regularization effect on two toy examples in the overparameterized regime: linear regression and a quadratically parameterized model. For linear regression, we will show that gradient descent with zero initialization will always find the minimum norm solution (instead of an arbitrary solution that fits the training data), and in practice, the minimum norm solution tends to generalize well. For a quadratically parameterized model, we will show that initialization and batch size also affect generalization.

(a) [3 points] Suppose we have a dataset $\{(x^{(i)}, y^{(i)}); i = 1, \cdots, n\}$ where $x^{(i)} \in \mathbb{R}^d$ and $y^{(i)} \in \mathbb{R}$ for all $1 \le i \le n$. We assume the dataset is generated by a linear model without noise. That is, there is a vector $\beta^\star \in \mathbb{R}^d$ such that $y^{(i)} = (\beta^\star)^\top x^{(i)}$ for all $1 \le i \le n$. Let $X \in \mathbb{R}^{n\times d}$ be the matrix representing the inputs (i.e., the i-th row of X corresponds to $x^{(i)}$)) and $\vec{y} \in \mathbb{R}^n$ the vector representing the labels (i.e., the i-th row of $\vec{y}$ corresponds to $y^{(i)}$)):

$$X = \begin{bmatrix} - & x^{(1)} & - \\ - & x^{(2)} & - \\ \vdots & \vdots & \vdots \\ - & x^{(n)} & - \end{bmatrix}, \qquad \vec{y} = \begin{bmatrix} y^{(1)} \\ y^{(2)} \\ \vdots \\ y^{(n)} \end{bmatrix}.$$

Then in matrix form, we can write $\vec{y} = X\beta^\star$. We assume that the number of examples is less than the number of parameters (that is, $n < d$).

We use the least-squares cost function to train a linear model:

$$J(\beta) = \frac{1}{2n}\|X\beta - \vec{y}\|_2^2. \tag{6}$$

In this sub-question, we characterize the family of global minimizers to Eq. (6). We assume that $XX^\top \in \mathbb{R}^{n\times n}$ is an invertible matrix. **Prove that** β achieves zero cost in Eq. (6) if and only if

$$\beta = X^\top (XX^\top)^{-1}\vec{y} + \zeta \tag{7}$$

for some ζ in the subspace orthogonal to all the data (that is, for some ζ such that $\zeta^\top x^{(i)} = 0, \forall 1 \le i \le n$.)

Note that this implies that there is an infinite number of β's such that Eq. (6) is minimized. We also note that $X^\top(XX^\top)^{-1}$ is the pseudo-inverse of X, but you don't necessarily need this fact for the proof.

Answer: We prove the statement in two steps. In the first step, we find one solution that achieves zero cost: $\rho = X^\top(XX^\top)^{-1}\vec{y}$ (note that $X^\top(XX^\top)^{-1}$ is the Moore–Penrose pseudo inverse of the matrix X). Then in the second step, we prove that adding ζ doesn't change the prediction on $x^{(1)}, \cdots, x^{(n)}$ if and only if ζ is orthogonal to all the data.

For the first step, let $\rho = X^\top(XX^\top)^{-1}\vec{y}$. Since $XX^\top$ is invertible, we have

$$X\rho = XX^\top(XX^\top)^{-1}\vec{y} = \vec{y}. \tag{8}$$

Now we consider the second step. For any β, we define $\zeta = \beta - \rho$. Note that $J(\beta) = 0$ is equivalent to $X\beta = \vec{y}$. Rewriting β in terms of ζ, we get

$$\vec{y} = X\beta = XX^\top (XX^\top)^{-1}\vec{y} + X\zeta = \vec{y} + X\zeta. \tag{9}$$

As a result, $J(\beta) = 0 \Leftrightarrow X\zeta = 0 \Leftrightarrow \zeta^\top x^{(i)} = 0, \forall 1 \le i \le n$.

(b) [3 points] We still work with the setup of part (a). Among the infinitely many optimal solutions of Eq. (6), we consider the *minimum norm* solution. Let $\rho = X^\top (XX^\top)^{-1}\vec{y}$. In the setting of (a), **prove that** for any β such that $J(\beta) = 0$, $\|\rho\|_2 \le \|\beta\|_2$. In other words, ρ is the minimum norm solution.

Hint: As a intermediate step, you can prove that for any β in the form of Eq. (7),

$$\|\beta\|_2^2 = \|\rho\|_2^2 + \|\zeta\|_2^2.$$

Answer: We first prove that $\|\beta\|_2^2 = \|\rho\|_2^2 + \|\zeta\|_2^2$. By sub-question (a), we can write $\beta = \rho + \zeta$ for some $X\zeta = 0$. As a result,

$$\begin{align}
\|\beta\|_2^2 &= (\rho + \zeta)^\top (\rho + \zeta) \tag{10}\\
&= \|\rho\|_2^2 + 2\zeta^\top \rho + \|\zeta\|_2^2 \tag{11}\\
&= \|\rho\|_2^2 + \|\zeta\|_2^2 + 2\zeta^\top X^\top (XX^\top)^{-1}\vec{y} \tag{12}\\
&= \|\rho\|_2^2 + \|\zeta\|_2^2 + 2(X\zeta)^\top (XX^\top)^{-1}\vec{y} \tag{13}\\
&= \|\rho\|_2^2 + \|\zeta\|_2^2, \tag{14}
\end{align}$$

where the last equation comes from the fact $X\zeta = 0$. Since $\|\zeta\|_2 \ge 0$, we get $\|\beta\|_2^2 \ge \|\rho\|_2^2$.

(c) [5 points] **Coding question: minimum norm solution generalizes well**

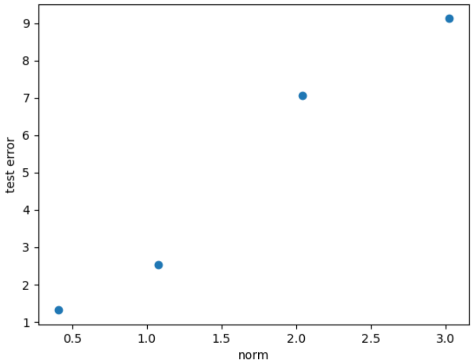
For this sub-question, we still work with the setup of parts (a) and (b). We use the following datasets:

`src/implicitreg/ir1_train.csv,ir1_test.csv`

Each file contains $d+1$ columns. The first d columns in the i-th row represents $x^{(i)}$, and the last column represents $y^{(i)}$. In this sub-question, we use $d = 200$ and $n = 40$.

Using the formula in sub-question (b), **compute** the minimum norm solution using the training dataset. Then, **generate** three other different solutions with zero costs and different norms using the formula in sub-question (a). The starter code is in `src/implicitreg/linear.py`. **Plot** the test error of these solutions (including the minimum norm solution) in a scatter plot. Use the norm of the solutions as x-axis, and the test error as y-axis. For your convenience, the plotting function is provided as the method `generate_plot` in the starter code. Your plot is expected to demonstrate that the minimum norm solution generalizes well.

Answer: Note: there is lots of flexibility in this plot. A plot is acceptable as long as it demonstrates that the minimum norm solution generalizes well.



In this case, the minimum norm solution generalizes well.

(d) [5 points] For this sub-question, we work with the setup of part (a) and (b). In this sub-question, you will prove that the gradient descent algorithm with *zero initialization* always converges to the minimum norm solution. Let $\beta^{(t)}$ be the parameters found by the GD algorithm at time step t. Recall that at step t, the gradient descent algorithm update the parameters in the following way

$$\beta^{(t)} = \beta^{(t-1)} - \eta \nabla J(\beta^{(t-1)}) = \beta^{(t-1)} - \frac{\eta}{n} X^\top (X\beta^{(t-1)} - \vec{y}). \tag{15}$$

As in sub-question (a), we also assume $XX^\top$ is an invertible matrix. **Prove** that if the GD algorithm with zero initialization converges to a solution $\hat{\beta}$ satisfying $J(\hat{\beta}) = 0$, then $\hat{\beta} = X^\top (XX^\top)^{-1}\vec{y} = \rho$, that is, $\hat{\beta}$ is the minimum norm solution.

Hint: As a first step, you can prove by induction that if we start with zero initialization, $\beta^{(t)}$ will always be a linear combination of $\{x^{(1)}, x^{(2)}, \cdots, x^{(n)}\}$ for any $t \geq 0$. Then, for any $t \geq 0$, you can write $\beta^{(t)} = X^\top v^{(t)}$ for some $v^{(t)} \in \mathbb{R}^n$. As a second step, you can prove that if $\hat{\beta} = X^\top v^{(t)}$ for some $v^{(t)}$ and $J(\hat{\beta}) = 0$, then we have $\hat{\beta} = \rho$.

You don't necessarily have to follow the steps in this hint. But if you use the hint, you need to prove the statements in the hint.

Answer: We first prove by induction that if $\beta^{(0)} = 0$, for any $t \geq 1$ there exists $v^{(t)} \in \mathbb{R}^n$ such that $\beta^{(t)} = X^\top v^{(t)}$ (equivalently, this is saying $\beta^{(t)}$ can be written as a linear combination of $\{x^{(1)}, \cdots, x^{(n)}\}$).

For the base case, we set $v^{(0)} = 0$. Now suppose $\beta^{(t)} = X^\top v^{(t)}$ for some $t \geq 0$, we show that $\beta^{(t+1)} = X^\top v^{(t+1)}$ for some $v^{(t+1)} \in \mathbb{R}^n$. By the update rule we have

$$\begin{aligned}\beta^{(t+1)} &= \beta^{(t)} - \frac{\eta}{n} X^\top (X\beta^{(t)} - \vec{y}) \quad (16)\\ &= X^\top v^{(t)} - \frac{\eta}{n} X^\top (X\beta^{(t)} - \vec{y}). \quad (17)\end{aligned}$$

As a result, we can set $v^{(t+1)} = v^{(t)} - \frac{\eta}{n}(X\beta^{(t)} - \vec{y})$.

Now we prove that if $\beta^{(t)} = X^\top v^{(t)}$ for some v and $J(\beta^{(t)}) = 0$, we get $\beta^{(t)} = \rho$. To this end, note that $J(\beta^{(t)}) = 0$ implies that $X\beta^{(t)} = \vec{y}$. Rewrite the equation using $v(t)$ we get

$$XX^\top v^{(t)} = \vec{y}. \tag{18}$$

Since $XX^\top$ is invertible, we get $v^{(t)} = (XX^\top)^{-1}\vec{y}$, which means that $\beta^{(t)} = X^\top v^{(t)} = X^\top(XX^\top)^{-1}\vec{y} = \rho$.

Combining steps (1) and (2), we prove that gradient descent with zero initialization converges to minimum norm solution (if it converges) in this case.

(e) [3 points] In the following sub-questions, we consider a slightly more complicated model called quadratically parameterized model. A quadratically parameterized model has two sets of parameters $\theta, \phi \in \mathbb{R}^d$. Given a d-dimensional input $x \in \mathbb{R}^d$, the output of the model is

$$f_{\theta,\phi}(x) = \sum_{k=1}^{d} \theta_k^2 x_k - \sum_{k=1}^{d} \phi_k^2 x_k. \tag{19}$$

Note that $f_{\theta,\phi}(x)$ is linear in its input x, but non-linear in its parameters θ, ϕ. Thus, if the goal was to learn the function, one should simply just re-parameterize it with a linear model and use linear regression. However, here we insist on using the parameterization above in Eq. (19) in order to study the implicit regularization effect in models that are nonlinear in the parameters.

Notations: To simplify the equations, we define the following notations. For a vector $v \in \mathbb{R}^d$, let $v^{\odot 2}$ be its element-wise square (that is, $v^{\odot 2}$ is the vector $[v_1^2, v_2^2, \cdots, v_d^2] \in \mathbb{R}^d$.) For two vectors $v, w \in \mathbb{R}^d$, let $v \odot w$ be their element-wise product (that is, $v \odot w$ is the vector $[v_1 w_1, v_2 w_2, \cdots, v_d w_d] \in \mathbb{R}^d$.) Then our model can be written as

$$f_{\theta,\phi}(x) = x^\top(\theta^{\odot 2} - \phi^{\odot 2}). \tag{20}$$

Suppose we have a dataset $\{(x^{(i)}, y^{(i)}); i = 1, \cdots, n\}$ where $x^{(i)} \in \mathbb{R}^d$ and $y^{(i)} \in \mathbb{R}$ for all $1 \le i \le n$, and

$$y^{(i)} = (x^{(i)})^\top((\theta^\star)^{\odot 2} - (\phi^\star)^{\odot 2})$$

for some $\theta^\star, \phi^\star \in \mathbb{R}^d$. Similarly, we use $X \in \mathbb{R}^{n\times d}$ and $\vec{y} \in \mathbb{R}^n$ to denote the matrix/vector representing the inputs/labels respectively:

$$X = \begin{bmatrix} - & x^{(1)} & - \\ - & x^{(2)} & - \\ \vdots & \vdots & \vdots \\ - & x^{(n)} & - \end{bmatrix}, \qquad \vec{y} = \begin{bmatrix} y^{(1)} \\ y^{(2)} \\ \vdots \\ y^{(n)} \end{bmatrix}.$$

Let $J(\theta, \phi) = \frac{1}{4n}\sum_{i=1}^{n}(f_{\theta,\phi}(x^{(i)}) - y^{(i)})^2$ be the cost function.

First, when $n < d$ and $XX^\top$ is invertible, **prove** that there exists infinitely many optimal solutions with zero cost.

Hint: Find a mapping between the parameter β in linear model and the parameter θ, ϕ in quadratically parameterized model. Then use the conclusion in sub-question (a).

Answer: First we prove that the label $y^{(i)}$ is linear in $x^{(i)}$. Indeed, define $\beta^\star = (\theta^\star)^{\odot 2} - (\phi^\star)^{\odot 2}$ and we will get $y^{(i)} = (x^{(i)})^\top \beta^\star$ for all $1 \le i \le n$.

By sub-question (a), there exists infinitely many solutions β with zero cost. For any $\beta \in \mathbb{R}^d$, we define $\theta(\beta), \phi(\beta) \in \mathbb{R}^d$ in the following way,

$$\theta(\beta)_k = \begin{cases} 0, & \text{if } \beta_k \le 0, \\ \sqrt{\beta_k}, & \text{if } \beta_k > 0. \end{cases} \tag{21}$$

$$\phi(\beta)_k = \begin{cases} 0, & \text{if } \beta_k > 0, \\ \sqrt{-\beta_k}, & \text{if } \beta_k \le 0. \end{cases} \tag{22}$$

We can verify that $f_{\theta(\beta),\phi(\beta)}(x) = x^\top \beta$ for all $x \in \mathbb{R}^d$. Therefore, $(\theta(\beta), \phi(\beta))$ is also a solution with zero cost.

Note that for any $\beta_1 \neq \beta_2$, we have $[\theta(\beta_1), \phi(\beta_1)] \neq [\theta(\beta_2), \phi(\beta_2)]$. As a result, there exists infinitely many solutions to the quadratically parameterized model.

(f) [10 points] **Coding question: implicit regularization of initialization**

We still work with the setup in part (e). For this sub-question, we use the following datasets:

`src/implicitreg/ir2_train.csv,ir2_test.csv`

Each file contains $d+1$ columns. The first d columns in the i-th row represents $x^{(i)}$, and the last column represents $y^{(i)}$. In this sub-question, we use $d = 200$ and $n = 40$.

First of all, the gradient of the loss has the following form:

$$\nabla_\theta J(\theta, \phi) = \frac{1}{n}\sum_{i=1}^{n}((x^{(i)})^\top(\theta^{\odot 2} - \phi^{\odot 2}) - y^{(i)})(\theta \odot x^{(i)}), \tag{23}$$

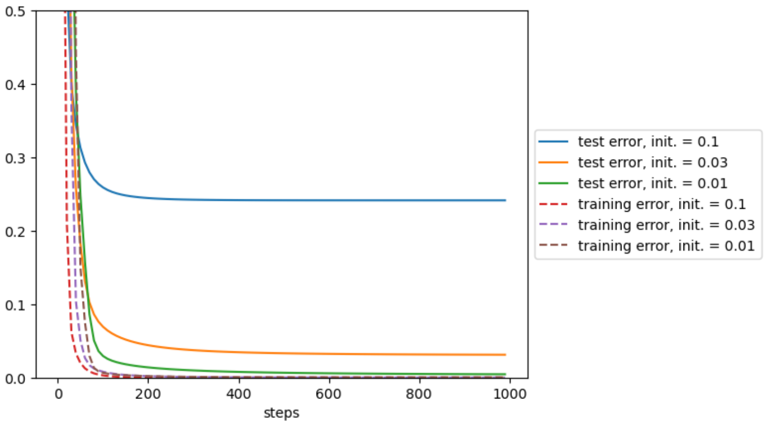
$$\nabla_\phi J(\theta, \phi) = -\frac{1}{n}\sum_{i=1}^{n}((x^{(i)})^\top(\theta^{\odot 2} - \phi^{\odot 2}) - y^{(i)})(\phi \odot x^{(i)}). \tag{24}$$

You don't need to prove these two equations. They can be verified directly using the chain rule.

Using the formula above, run gradient descent with initialization $\theta = \alpha\mathbf{1}, \phi = \alpha\mathbf{1}$ with $\alpha \in \{0.1, 0.03, 0.01\}$ (where $\mathbf{1} = [1, 1, \cdots, 1] \in \mathbb{R}^d$ is the all-1's vector) and learning rate 0.08. We provide the starter code in `src/implicitreg/qp.py`. **Plot** the curve of training error and test error with different α. Use the number of gradient steps as x-axis, and training/test error as y-axis. Include your plot in the writeup and **answer** the following two questions based on your plot: which models can fit the training set? Which initialization achieves the best test error?

Remark: Your plot is expected to demonstrate that the initialization plays an important role in the generalization performance—different initialization can lead to different global minimizers with different generalization performance. In other words, the initialization has an implicit regularization effect.

Answer:



All the models can fit the training set, and $\alpha = 0.01$ achieves the best test error.

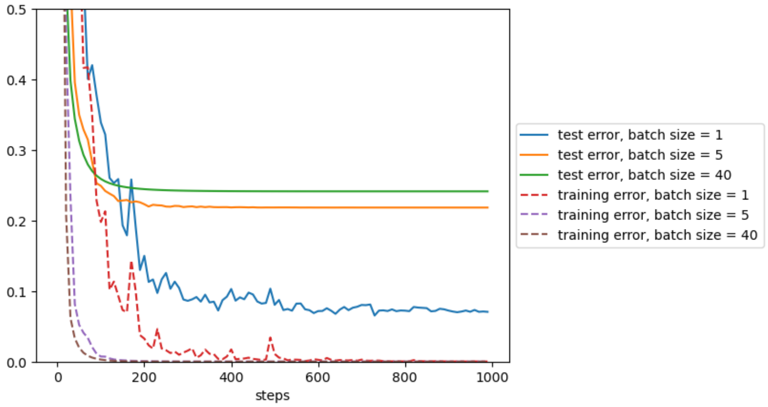
(g) [6 points] **Coding question: implicit regularization of batch size**

We still work with the setup in part (e). For this sub-question, we use the same dataset and starter code as in sub-question (f). We will show that introducing noise in the training process also induces implicit regularization. In particular, we will observe how noise introduced by *stochastic* gradient descent (SGD) helps generalization. The gradient descent algorithm we have covered so far in lecture is also known as *batch* gradient descent, in which we look at the entire training set before taking a single update step. In SGD, we only look at a subset of the training examples before taking an update step. (Note that "true" SGD looks at a single training example before taking an update step. In this problem, we will also be considering the generalization of SGD – mini-batch gradient descent – to generate more empirical evidence for your final observation!) SGD will be covered in more detail later in the course, but this is all you need to know to solve this problem.

Implement the SGD algorithm, and **plot** the training and test errors with mini-batch sizes $\{1, 5, 40\}$, learning rate 0.08, and initialization $\alpha = 0.1$. Similarly, use the number of gradient steps as x-axis, and training/test error as y-axis. For simplicity, the code for selecting a batch of examples is already provided in the starter code. **Compare** the results with those in sub-question (g) with the same initialization. Does SGD find a better solution?

Your plot is expected to show that the stochasticity in the training process is also an important factor in the generalization performance — in our setting, SGD finds a solution that generalizes better. In fact, a conjecture is that stochasticity in the optimization process (such as the noise introduced by a small batch size) helps the optimizer to find a solution that generalizes better. This conjecture can be proved in some simplified cases, such as the quadratically parameterized model in this sub-question (adapted from the paper HaoChen et al., 2020), and can be observed empirically in many other cases.

Answer:



Comparing with the same initialization using GD, SGD has a smaller test error using batch size 1.

5. **[12 points] Double Descent on Linear Models**

In this question, you will empirically observe the sample-wise double descent phenomenon. While this problem is self-contained, additional information on this phenomenon (as well as the related model-wise double descent phenomenon) is available in section 8.2 of the lecture notes.

In the sample-wise double descent phenomenon, the test losses of some learning algorithms or estimators do not monotonically decrease as we have more training examples, but instead have a curve with two U-shaped parts. The double descent phenomenon can be observed even for simple linear models. In this question, we consider the following setup. Let $\{(x^{(i)}, y^{(i)})\}_{i=1}^n$ be the training dataset. Let $X \in \mathbb{R}^{n\times d}$ be the matrix representing the inputs (i.e., the i-th row of X corresponds to $x^{(i)}$)), and $\vec{y} \in \mathbb{R}^n$ the vector representing the labels (i.e., the i-th row of $\vec{y}$ corresponds to $y^{(i)}$)):

$$X = \begin{bmatrix} - & x^{(1)} & - \\ - & x^{(2)} & - \\ \vdots & \vdots & \vdots \\ - & x^{(n)} & - \end{bmatrix}, \qquad \vec{y} = \begin{bmatrix} y^{(1)} \\ y^{(2)} \\ \vdots \\ y^{(n)} \end{bmatrix}.$$

Similarly, we use $X_v \in \mathbb{R}^{m\times d}, \vec{y}_v \in \mathbb{R}^m$ to represent the test dataset, where m is the size of the test dataset. We assume that the data are generated with $d = 500$.

In this question, we consider *regularized* linear regression. For a regularization level $\lambda \geq 0$, define the regularized cost function

$$J_\lambda(\beta) = \frac{1}{2}\|X\beta - \vec{y}\|_2^2 + \frac{\lambda}{2}\|\beta\|_2^2,$$

and its minimizer $\hat{\beta}_\lambda = \arg\min_{\beta\in\mathbb{R}^d} J_\lambda(\beta)$.

(a) [2 points] In this sub-question, we derive the closed-form solution of $\hat{\beta}_\lambda$. **Prove** that when $\lambda > 0$,

$$\hat{\beta}_\lambda = (X^\top X + \lambda I_{d\times d})^{-1} X^\top \vec{y} \tag{25}$$

(recall that $I_{d\times d} \in \mathbb{R}^{d\times d}$ is the identity matrix.)

Note: $\lambda = 0$ is a special case here. When $\lambda = 0$, $(X^\top X + \lambda I_{d\times d})$ could be singular. Therefore, there might be more than one solutions that minimize $J_0(\beta)$. In this case, we define $\hat{\beta}_0$ in the following way:

$$\hat{\beta}_0 = (X^\top X)^+ X^\top \vec{y}. \tag{26}$$

where $(X^\top X)^+$ denotes the Moore-Penrose pseudo-inverse of $X^\top X$. You don't need to prove the case when $\lambda = 0$, but this definition is useful in the following sub-questions.

Answer: For a fixed λ, take the derivative of J_λ w.r.t. β we get

$$\nabla_\beta J_\lambda(\beta) = X^\top(X\beta - \vec{y}) + \lambda\beta = (X^\top X + \lambda I_{d\times d})\beta - X^\top \vec{y}. \tag{27}$$

Solve the equation $\nabla_\beta J_\lambda(\beta) = 0$ we get, $\hat{\beta}_\lambda = (X^\top X + \lambda I_{d\times d})^{-1} X^\top \vec{y}$.

(b) [5 points] **Coding question: the double descent phenomenon for unregularized models**

In this sub-question, you will empirically observe the double descent phenomenon.You are given 13 training datasets of sample sizes $n = 200, 250, \ldots, 750$, and 800, and a test dataset, located at

- `src/doubledescent/train200.csv`, `train250.csv`, etc.
- `src/doubledescent/test.csv`

For each training dataset $(X, \vec{y})$, compute the corresponding $\hat{\beta}_0$, and evaluate the mean squared error (MSE) of $\hat{\beta}_0$ on the test dataset. The MSE for your estimators $\hat{\beta}$ on a test dataset $(X_v, \vec{y}_v)$ of size m is defined as:

$$\text{MSE}(\hat{\beta}) = \frac{1}{2m}\|X_v\hat{\beta} - \vec{y}_v\|_2^2.$$

Complete the `regression` method of `src/doubledescent/doubledescent.py` which takes in a training file and a test file, and computes $\hat{\beta}_0$. You can use `numpy.linalg.pinv` to compute the pseudo-inverse.

In your writeup, include a line plot of the test losses. The x-axis is the size of the training dataset (from 200 to 800); the y-axis is the MSE on the test dataset. You should observe that the test error increases and then decreases as we increase the sample size.

Note: When $n \approx d$, the test MSE could be very large. For better visualization, it is okay if the test MSE goes out of scope in the plot for some points.

Answer:

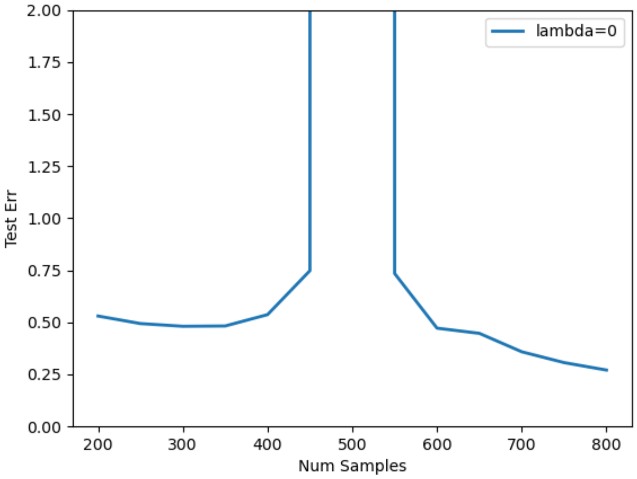


Figure 8: Num of samples vs. test error.

(c) [5 points] **Coding question: double descent phenomenon and the effect of regularization.**

In this sub-question, we will show that regularization mitigates the double descent phenomenon for linear regression. We will use the same datasets as specified in sub-question (b). Now consider using various regularization strengths. For $\lambda \in \{0, 1, 5, 10, 50, 250, 500, 1000\}$, you will compute the minimizer of $J_\lambda(\beta)$.

Complete the `ridge_regression` method of `src/doubledescent/doubledescent.py` which takes in a training file and a test file, computes the $\hat{\beta}_\lambda$ that minimizes the training objective under different regularization strengths, and returns a list of test errors (one for each choice of λ).

In your writeup, include a plot of the test losses of these models. The x-axis is the size of the training dataset (from 200 to 800); the y-axis is the MSE on the test dataset. Draw one line for each choice of λ connecting the test errors across different training dataset sizes. Therefore, the plot should contain 8×13 points and 8 lines connecting them.

You should observe that for some small λ's, the test error may increase and then decrease as we increase the sample size. However, double descent does not occur for a relatively large λ.

Remark: If you want to learn more about the double descent phenomenon and the effect of regularization, you can start with this paper Nakkiran, et al. 2020.

Answer:

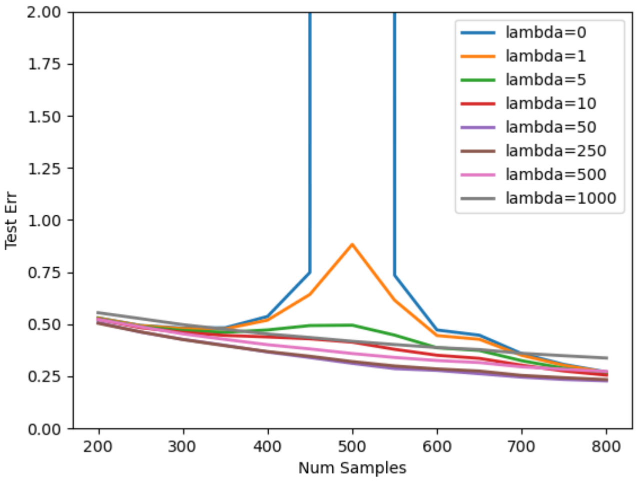


Figure 9: Num of samples vs. test error for different regularization strengths