CS 189 Introduction to Machine Learning
Fall 2017 HW3

This homework is due **Friday, September 15 at 12pm noon.**

1 Getting Started

You may typeset your homework in latex or submit neatly handwritten and scanned solutions. Please make sure to start each question on a new page, as grading (with Gradescope) is much easier that way! Deliverables:

1. Submit a PDF of your writeup to assignment on Gradescope, "HW[n] Write-Up"
2. Submit all code needed to reproduce your results, "HW[n] Code".
3. Submit your test set evaluation results, "HW[n] Test Set".

After you've submitted your homework, be sure to watch out for the self-grade form.

(a) Before you start your homework, write down your team. Who else did you work with on this homework? List names and email addresses. In case of course events, just describe the group. How did you work on this homework? Any comments about the homework?

(b) Please copy the following statement and sign next to it:

I certify that all solutions are entirely in my words and that I have not looked at another student's solutions. I have credited all external sources in this write up.

Throughout the homework, note that
$Z \in \mathbb{R}$ implies that Z is a scalar;
$Z \in \mathbb{R}^d$ is equivalent to saying that we have dimensions $\underbrace{Z}_{d \times 1}$;
$Z \in \mathbb{R}^{n \times d}$ is equivalent to saying that Z is a matrix with dimensions $\underbrace{Z}_{n \times d}$.
When we say that $Z, Z' \in \mathbb{R}$, we mean that $Z \in \mathbb{R}$ and $Z' \in \mathbb{R}$.

2 Probabilistic Model of Linear Regression

Both ordinary least squares and ridge regression have interpretations from a probabilistic standpoint. In particular, assuming a generative model for our data and a particular noise distribution, we will derive least squares and ridge regression as the maximum likelihood and maximum a-posteriori parameter estimates, respectively. This problem will walk you through a few steps to do that. (Along with some side digressions to make sure you get a better intuition for ML and MAP estimation.)

(a) Assume that X and Y are both one-dimensional random variables, i.e. $X, Y \in \mathbb{R}$. Assume an affine model between X and Y: $Y = Xw + b + Z$, where $w, b \in \mathbb{R}$, and $Z \sim N(0,1)$ is a standard normal (Gaussian) random variable. Assume w, b are not random. **What is the conditional distribution of Y given X?**

Solution: When we condition on the event $X = x$, the expression $Xw + b$ becomes $xw + b$, so

$$Y|(X = x) \sim xw + b + Z.$$

Now $xw + b + Z$ is a constant plus a standard normal, so

$$xw + b + Z \sim N(xw + b, 1).$$

The conditional density becomes:

$$p(Y|X = x) = \frac{1}{\sqrt{2\pi}} \exp\{-\frac{1}{2}(Y - (xw + b))^2\}. \tag{1}$$

(b) Given n points of training data $\{(X_1, Y_1), (X_2, Y_2), \ldots, (X_n, Y_n)\}$ generated in an iid fashion by the probabilistic setting in the previous part, **derive the maximum likelihood estimator for w, b from this training data.**

Solution: The log likelihood function is given by

$$\begin{aligned} \sum_{i=1}^{n} \log p(Y_i|X = X_i) &= \sum_{i=1}^{n} \log(\frac{1}{\sqrt{2\pi}} \exp\{-\frac{1}{2}(Y - (X_i w + b))^2) \\ &= -\frac{1}{2} \sum_{i=1}^{n} (Y_i - (X_i w + b))^2 + \text{Constant} \end{aligned} \tag{2}$$

Differentiating the log likelihood with respect to w and b, we obtain:

$$\frac{\partial}{\partial w}\sum_i \log p(Y_i|X=X_i) = \sum_{i=1}^n X_i(X_i w + b - Y_i)$$
$$\frac{\partial}{\partial b}\sum_i \log p(Y_i|X=X_i) = \sum_{i=1}^n (X_i w + b - Y_i)$$

Setting both of the partial derivatives to zero, we immediately get two equations with two unknowns. The terms that only have Y_i and $Y_i X_i$ are pulled to the other side of the equality:

$$w\sum_{i=1}^n X_i^2 + b\sum_{i=1}^n X_i = \sum_{i=1}^n X_i Y_i$$
$$w\sum_{i=1}^n X_i + bn = \sum_{i=1}^n Y_i$$

Then, these can be solved directly:

$$w = \frac{n\sum_{i=1}^n X_i Y_i - \sum_{i=1}^n X_i \sum_{i=1}^n Y_i}{n\sum_{i=1}^n X_i^2 - (\sum_{i=1}^n X_i)^2}$$
$$b = \frac{\sum_{i=1}^n Y_i \sum_{i=1}^n X_i^2 - \sum_{i=1}^n X_i \sum_{i=1}^n X_i Y_i}{n\sum_{i=1}^n X_i^2 - (\sum_{i=1}^n X_i)^2}.$$

(c) Now, we are going to change the probabilistic model slightly in two ways. There is no more b and Z has a different distribution. So assume the linear model between X and Y is given by $Y = Xw + Z$, where $Z \sim U[-0.5, 0.5]$ is a continuous random variable uniformly distributed between -0.5 and 0.5. Assume w is not random. **What is the conditional distribution of Y given X?**

Solution: As before, w is fixed, so we have

$$\begin{aligned} P(Y=y|X=x) &= P(z=y-Xw|X=x) \\ &= P(z=y-xw) \\ &= \begin{cases} 1 \text{ if } -0.5 < y-xw < 0.5 \\ 0 \text{ otherwise} \end{cases} \\ &= \begin{cases} 1 \text{ if } -0.5+xw < y < 0.5+xw \\ 0 \text{ otherwise} \end{cases} \end{aligned} \tag{3}$$

(d) Given n points of training data $\{(X_1,Y_1),(X_2,Y_2),\cdots,(X_n,Y_n)\}$ generated in an iid fashion in the setting of the previous part, **derive the maximum likelihood estimator of w.** (Note that this estimate may not be unique; you may report a set of values, or feel free to break ties in whatever fashion and report one value within this set.)

Solution:

First consider the $X_i > 0$. The likelihood function is

$$\begin{aligned}\Pi_i p(Y_i|X_i) &= \Pi_{i=1}^n \mathbf{1}\{-0.5 + X_i w < Y_i < 0.5 + X_i w\} \\ &= \Pi_{i=1}^n \mathbf{1}\{(Y_i - 0.5)/X_i < w < (Y_i + 0.5)/X_i\} \\ &= \Pi_{i=1}^n \mathbf{1}\{\frac{Y_i}{X_i} - 0.5/|X_i| < w < \frac{Y_i}{X_i} + 0.5/|X_i|\}.\end{aligned}$$

Now consider the $X_i < 0$.

$$\begin{aligned}\Pi_i p(Y_i|X_i) &= \Pi_{i=1}^n \mathbf{1}\{-0.5 + X_i w < Y_i < 0.5 + X_i w\} \\ &= \Pi_{i=1}^n \mathbf{1}\{(Y_i + 0.5)/X_i < w < (Y_i - 0.5)/X_i\} \\ &= \Pi_{i=1}^n \mathbf{1}\{\frac{Y_i}{X_i} - 0.5/|X_i| < w < \frac{Y_i}{X_i} + 0.5/|X_i|\}.\end{aligned}$$

For $X_i = 0$, this does not constrain the w in any way. Putting it all together:

$$\Pi_i p(Y_i|X_i) = \Pi_{i=1, X_i \neq 0}^n \mathbf{1}\{\frac{Y_i}{X_i} - 0.5/|X_i| < w < \frac{Y_i}{X_i} + 0.5/|X_i|\}.$$

To maximize the log likelihood, all the indicators must have value one, which implies that all those regions intersect.

$$\max_{i, X_i \neq 0} (\frac{Y_i}{X_i} - 0.5/|X_i|) < w < \min_{i, X_i \neq 0} (\frac{Y_i}{X_i} + 0.5/|X_i|).$$

Checking for zero is a bit pedantic since this will never happen if the data is truly drawn from the specified model. However, actual code always has to deal with corner cases so we have included it. The case of negative values is important however since the problem did not say that the X were positive.

(e) Take the model $Y = Xw^* + b^* + Z$, where $Z \sim U[-0.5, 0.5]$ is a continuous random variable uniformly distributed between -0.5 and 0.5. **Use a computer to simulate n training samples $\{(X_1, Y_1), (X_2, Y_2), \cdots, (X_n, Y_n)\}$ and illustrate what the likelihood of the data looks like in the (w, b) model parameter space after $n = 5, 25, 125, 625$ training samples. Qualitatively describe what is happening as n gets large?**

(Note that you have considerable design freedom in this problem part. You get to choose how you draw the X_i as well as what true value (w^*, b^*) you want to illustrate. You have total freedom in using python libraries for this problem part. No restrictions.)

Solution: We generated samples assuming $Y = 3X + 2 + Z$. We assumed that the model parameters are between 0 and 4 (you can assume any range that includes tha true parameters.) and we calculated the likelihood by using the formula we obtained above for all possible combinations of parameters b and w in this interval. To do so, for a specific b and w we compute the likelihood $\Pi_{i=1}^n Pr(Y_i|X_i; w, b)$ which is either zero or 1 because the probability distribution is uniform. In the figure below, likelihood of the data can be seen in the model space (w, b). As we can see in the plots, increasing number of samples decreases the zone of plausible model parameters so we are able to select parameters with more certainty.

The code here uses a simple approach of just giving each training sample a veto into whether or not a particular set of parameters was possible or not. A more sophisticated approach could use a linear program to represent the votes of each sample point with two linear inequalities.

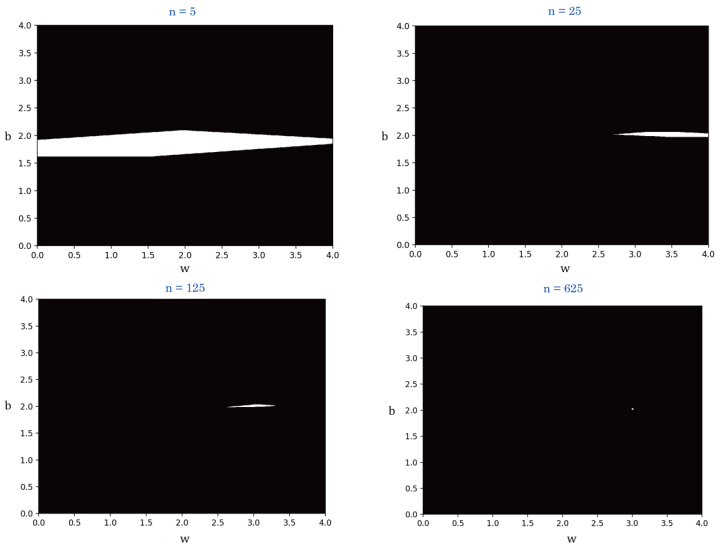

Figure 1: Heatplots of the likelihood of data in the model space (w, b) for different sample size

```
import numpy as np
import matplotlib.pyplot as plt

sample_size = [5,25,125,625]
for k in range(4):
    n = sample_size[k]
    X = np.random.normal(0,0.1,n)  # generating n X with normal dist and variance
    0.1
    Z = np.random.uniform(-0.5,0.5,n) # generating i.i.d random uniform noise
    Y = 3*X+2+Z # computing y from x and z assuming w=3 and b=2
    N = 1001
    B = np.linspace(0,4,N) # computing likelihood for b and w between 0 and 4 (1001
     values)
    W = np.linspace(0,4,N)
    likelihood = np.ones([N,N]) # likelihood as a function of w and b

    for i1 in range(N):
        w = W[i1]
        for i2 in range(N):
            b = B[i2]
            for i in range(n):
                # Yi has uniform distribution in [wXi+b-0.5,wXi+b+0.5]
                if abs(Y[i]-w*X[i]-b) > 0.5:
                    likelihood[i1][i2]=0
    plt.figure()
    plt.imshow(likelihood.transpose(), cmap='hot', aspect='auto',extent=[0,4,0,4])
    plt.show()
    print(n)
```

(f) (One-dimensional Ridge Regression) Now, let us return to the case of Gaussian noise. Given

n points of training data $\{(X_1,Y_1),(X_2,Y_2),\cdots,(X_n,Y_n)\}$ generated according to $Y_i = X_iW + Z_i$, where $Z_i \sim N(0,1)$ are iid standard normal random variables. Assume $W \sim N(0,\sigma^2)$ is also a standard normal and is independent of both the Z_i and the X_i. **Use Bayes' Theorem to derive the posterior distribution of W given the training data. What is the mean of the posterior distribution of W given the data?**

Solution: By Bayes' Theorem, we have

$$\begin{aligned}
P(w|X_1,Y_1,\cdots,X_n,Y_n) &\propto \Pi_{i=1}^n P(Y_i|X_i,w)\cdot P(w)\\
&\propto \Pi_{i=1}^n \exp\{-0.5(Y_i-wX_i)^2\}\exp\{-0.5\frac{w^2}{\sigma^2}\}\\
&\propto \Pi_{i=1}^n \exp\{-0.5(Y_i^2-2Y_iwX_i+w^2X_i^2)\}\exp\{-0.5\frac{w^2}{\sigma^2}\}\\
&\propto \exp\{-0.5(\sum_{i=1}^n Y_i^2-2\sum_{i=1}^n Y_iwX_i+w^2\sum_{i=1}^n X_i^2)\}\exp\{-0.5\frac{w^2}{\sigma^2}\}\\
&\propto \exp\{-0.5[(\sum_{i=1}^n X_i^2+\frac{1}{\sigma^2})w^2-(2\sum_{i=1}^n X_iY_i)w]+\text{Constant}\}\\
&\propto \exp\{-0.5[(\sum_{i=1}^n X_i^2+\frac{1}{\sigma^2})(w^2-\frac{2\sum_{i=1}^n X_iY_i}{\sum_{i=1}^n X_i^2+\frac{1}{\sigma^2}}w)]+\text{Constant}\}\\
&\propto \exp\{-0.5[(\sum_{i=1}^n X_i^2+\frac{1}{\sigma^2})(w-\frac{\sum_{i=1}^n X_iY_i}{\sum_{i=1}^n X_i^2+\frac{1}{\sigma^2}})^2]+\text{Constant}\}
\end{aligned}$$

To get to the last line, we used a trick of completing the square. Consider the term we would need to add to

$$(w^2-\frac{2\sum_{i=1}^n X_iY_i}{\sum_{i=1}^n X_i^2+\frac{1}{\sigma^2}}w)$$

to get

$$(w-\frac{\sum_{i=1}^n X_iY_i}{\sum_{i=1}^n X_i^2+\frac{1}{\sigma^2}})^2.$$

The term we would add does not depend on w, only the sample points. From the perspective of the distribution on w, this is just a constant, so we folded it into the constant term.

The final step is to notice that this posterior of w given $X_1,Y_1,\cdots,X_n,Y_n$ is a normal distribution. To see the connection, ignore the constant term in the posterior

$$P(w|X_1,Y_1,\cdots,X_n,Y_n) \propto \exp\{-0.5(\sum_{i=1}^n X_i^2+\frac{1}{\sigma^2})(w-\frac{\sum_{i=1}^n X_iY_i}{\sum_{i=1}^n X_i^2+\frac{1}{\sigma^2}})^2\}$$

and recall the formula for a normal distribution

$$N(\mu,\sigma) \propto \exp\{-\frac{1}{2\sigma^2}(x-\mu)^2\}.$$

We see that the mean of the posterior distribution is

$$\frac{\sum_{i=1}^n X_iY_i}{\sum_{i=1}^n X_i^2+\frac{1}{\sigma^2}}. \tag{4}$$

Since the posterior is Gaussian, the mean equals the mode, and so this is the MLE.

(g) Consider n training data points $\{(X_1,Y_1),(X_2,Y_2),\cdots,(X_n,Y_n)\}$ generated according to $Y_i = w^T X_i + Z_i$ where $Y_i \in \mathbb{R}, w, X_i \in \mathbb{R}^d$ with w fixed, and $Z_i \sim N(0,1)$ iid standard normal random variables. **Derive why the maximum likelihood estimator for w is the solution to a least squares problem.**

Solution: Since the logarithm is a monotically increasing function, we can just look at the log likelihood function.

$$\sum_i \log p(Y_i|X_i) = -\frac{1}{2}\sum_{i=1}^{n}(Y_i - (X_i^T w))^2 + \text{Constant}$$
$$= -\frac{1}{2}\|Xw - Y\|_2^2 + \text{Constant},$$

with $X \in \mathbb{R}^{n\times d}$ whose i^th row is X_i^T and $Y \in \mathbb{R}^n$ whose i^th entry is Y_i.

At this point, we are done. We have shown that maximizing the maximum-likelihood objective is the same as maximizing a negative squared error objective, which is the same as minimizing the squared error objective, the same as ordinary least squares.

However, if we wanted to, we could just keep solving this.

Differentiate the log likelihood with respect to w and setting the gradient to zero, we get

$$\nabla_w \sum_i \log p(Y_i|X_i) = X^T(Xw - Y).$$

Setting the gradient to zero, we get

$$w = (X^T X)^{-1} X^T Y.$$

which is the same solution as the least squares problem.

(h) (Multi-dimensional ridge regression) Consider the setup of the previous part: $Y_i = W^T X_i + Z_i$, where $Y_i \in \mathbb{R}, W, X_i \in \mathbb{R}^d$, and $Z_i \sim N(0,1)$ iid standard normal random variables. Now, however, we have a prior for the vector W (now a random variable): $W_j \sim N(0,\sigma^2)$ are iid for $j = 1,2,\ldots,d$. **Derive the posterior distribution of W given all the X_i, Y_i pairs. What is the mean of the posterior distribution of the vector W?**

Solution: $Y|X,w \sim N(X^T w, 1)$. Concretely, we have

$$p(Y|X) \propto \exp\{-\frac{1}{2}(Y - (X^T w + b))^2\}. \tag{5}$$

By Bayes' Theorem, we have

$$\begin{aligned}
P(w|X_1,Y_1,\cdots,X_n,Y_n) &\propto \Pi_{i=1}^n P(Y_i|X_i,w)\Pi_{j=1}^d P(w_j) \\
&\propto \Pi_{i=1}^n \exp\{-0.5(Y_i - X_i^T w)^2\}\exp\{-\frac{\|w\|_2^2}{2\sigma^2}\} \\
&\propto \Pi_{i=1}^n \exp\{-0.5(Y_i^2 - 2Y_i X_i^T w + (X_i^T w)^2)\}\exp\{-\frac{\|w\|_2^2}{2\sigma^2}\} \\
&\propto \exp\{-0.5(\sum_{i=1}^n Y_i^2 - 2\sum_{i=1}^n Y_i X_i^T w + \sum_{i=1}^n (X_i^T w)^2)\}\exp\{-\frac{\|w\|_2^2}{2\sigma^2}\} \\
&\propto \exp\{-0.5(\sum_{i=1}^n Y_i^2 - 2\sum_{i=1}^n Y_i X_i^T w + \sum_{i=1}^n w^T X_i X_i^T w)\}\exp\{-\frac{\|w\|_2^2}{2\sigma^2}\} \\
&\propto \exp\{-0.5(\sum_{i=1}^n Y_i^2 - 2\sum_{i=1}^n Y_i X_i^T w + w^T \sum_{i=1}^n X_i X_i^T w)\}\exp\{-\frac{\|w\|_2^2}{2\sigma^2}\} \\
&\propto \exp\{-0.5(\sum_{i=1}^n Y_i^2 - 2\sum_{i=1}^n (X_i Y_i)^T w + w^T \sum_{i=1}^n X_i X_i^T w)\}\exp\{-\frac{\|w\|_2^2}{2\sigma^2}\} \\
&\propto \exp\{-\frac{1}{2}[w^T(X^T X + \frac{1}{\sigma^2} I_d)w - 2(X^T Y)^T w]\},
\end{aligned} \tag{6}$$

with $X \in \mathbb{R}^{n\times d}$ whose i^{th} row is X_i^T and $Y \in \mathbb{R}^n$ whose i^{th} entry is Y_i. The last line uses the fact that

$$X^T X = \sum_{i=1}^n X_i X_i^T$$

and

$$X^T Y = \sum_{i=1}^n X_i Y_i.$$

Now, as we did earlier, we complete the square, except in the matrix case. To make the notation easier, let's define the matrix M as

$$M = (X^T X + \frac{1}{\sigma^2} I_d).$$

Then, noticing that M is symmetric, we see that:

$$\begin{aligned}
P(w|X_1,Y_1,\cdots,X_n,Y_n) &\propto \exp\{-\frac{1}{2}[w^T M w - 2(X^T Y)^T w]\} \\
&\propto \exp\{-\frac{1}{2}[(w - M^{-1}(X^T Y))^T M (w - M^{-1}(X^T Y))] + \text{Constant}\}
\end{aligned} \tag{7}$$

How did we know this was how we should complete the square? The w part on the sides is clear as is the central M since we knew we wanted the $w^T M w$ term. Now, because we knew that we wanted to get an $(X^T Y)^T w$ term as the linear term in the square, we had to cancel the M in the middle with an M^{-1}. Of course, as argued earlier, the constant here absorbs the compensation for the term that we added while completing the square.

Therefore, the posterior of w given $X_1, Y_1, \cdots, X_n, Y_n$ is a multivariate Gaussian. The mean of this multivariate Gaussian is

$$(X^T X + \frac{1}{\sigma^2} I_d)^{-1} X^T Y. \tag{8}$$

(i) Consider $d = 2$ and the setting of the previous part. **Use a computer to simulate and illustrate what the a-posteriori probability looks like for the W model parameter space after $n = 5, 25, 125$ training samples for different values of σ^2.**

(Note that you have considerable design freedom in this problem part. You have total freedom in using python libraries for this problem part. No restrictions. Either contour plots or 3d plots of the a-posteriori probability are fine. You don't have to label the contours with values if the story is clear from the plots visually.)

Solution: First, we assumed a variance for the random variable W and we generate d numbers from $N(0, \sigma^2)$. Then, we generated the samples using these parameters and computed a value proportional to the probability using Equation 6 from above. We could have normalized to get the exact posterior probability; our results would have been equivalent as far the plots would go.

In the figures below, the a-posteriori probability of the model parameters given data can be seen in the model space $W = (W_1, W_2)$. As we can see in the plots, increasing the number of samples results in shrinking the zone of likely parameters.

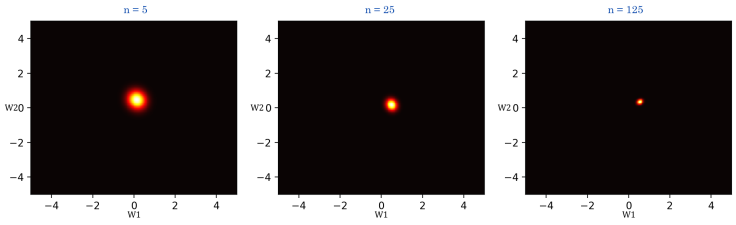

Figure 2: Heatplots of the a-posteriori probability of the model given the training data, plotted in the model space (W_1, W_2) for different sample sizes with $\sigma = 0.5$

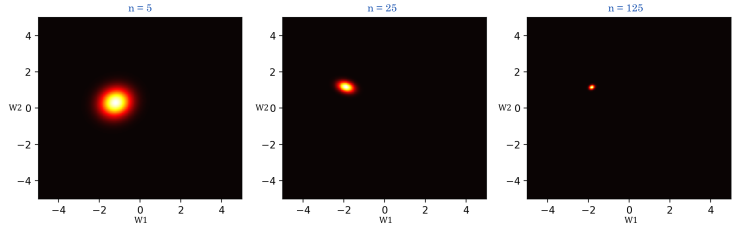

Figure 3: Heatplots of the a-posteriori probability of the model given the training data, plotted in the model space (W_1, W_2) for different sample sizes with $\sigma = 1.5$

```
import numpy as np
import matplotlib.pyplot as plt
Samples=[5,25,125]
Sigma = [0.5**2,1.5**2] #plotting for two different variance
w1_s = [0.4,1.3] # true W1 drawn from the gaussian with s.d 0.5 and 1.5
    respectively
```

```
w2_s = [0.5,-1.7] # true W2 drawn from the gaussian with s.d 0.5 and 1.5
    respectively
for s in range(2):
    sigma=Sigma[s]
    A=w1_s[s]
    B=w2_s[s]
    plt.figure()
    for count in range(3):
        n = Samples[count]
        # X = (X1,X2) Y = AX1+AX2+Z
        X1 = np.random.normal(0,1, n) # generating random samples for X1 from
    normal dist
        X2 = np.random.normal(0,1, n)
        Z = np.random.normal(0,1, n)
        Y = A*X1 +B*X2 +Z
        N = 201
        W = np.linspace(-5,5,N)
        prob = np.ones([N,N])
        for i1 in range(N):
            w1 = W[i1] #taking different values for w1 from -5 to 5 to compute
    something proportional to the posteriori probability
            for i2 in range(N):
                w2 = W[i2]
                L=1
                for i in range(n): # this part can vectorized as well
                    L = L*np.exp(-0.5*(Y[i]-X1[i]*w1-X2[i]*w2)**2)
                L = L*np.exp(-0.5*(w1**2+w2**2)/sigma)
                prob[i1][i2]=L
        plt.subplot(1,3,count+1)
        plt.imshow(np.flipud(prob), cmap='hot', aspect='auto',extent=[-5,5,-5,5])
        plt.show()
```

3 Simple Bias-Variance Tradeoff

Consider a random variable X, which has unknown mean μ and unknown variance σ^2. Given n iid realizations of training samples $X_1 = x_1, X_2 = x_2, \ldots, X_n = x_n$ from the random variable, we wish to estimate the mean of X. We will call our estimate the random variable $\hat{X}$ with mean $\hat{\mu}$. There are a few ways we can estimate μ given the realizations of the n samples:

1. Average the n samples: $\frac{x_1+x_2+\ldots+x_n}{n}$.
2. Average the n samples and one sample of 0: $\frac{x_1+x_2+\ldots+x_n}{n+1}$.
3. Average the n samples and n_0 samples of 0: $\frac{x_1+x_2+\ldots+x_n}{n+n_0}$.
4. Ignore the samples: just return 0.

In the parts of this question, we will measure the *bias* and *variance* of each of our estimators. The *bias* is defined as

$$E[\hat{X} - \mu]$$

and the *variance* is defined as

$$\text{Var}[\hat{X}].$$

(a) **What is the bias of each of the four estimators above?**

Solution: Using the linearity of expectation, we write $E[\hat{X} - X]$ as $E[\hat{X}] - E[X] = E[\hat{X}] - \mu$, so we have the following biases:

(a) $E[\hat{X}] = E[\frac{X_1+X_2+\ldots+X_n}{n}] = \frac{n\mu}{n} \implies \text{bias} = 0$

(b) $E[\hat{X}] = E[\frac{X_1+X_2+\ldots+X_n}{n+1}] = \frac{n\mu}{n+1} \implies \text{bias} = -\frac{1}{n+1}\mu$

(c) $E[\hat{X}] = E[\frac{X_1+X_2+\ldots+X_n}{n+n_0}] = \frac{n\mu}{n+n_0} \implies \text{bias} = -\frac{n_0}{n+n_0}\mu$

(d) $E[\hat{X}] = 0 \implies \text{bias} = -\mu$

(b) **What is the variance of each of the four estimators above?**

Solution: The two key identities to remember are $\text{Var}[A+B] = \text{Var}[A] + \text{Var}[B]$ (when A and B are independent) and $\text{Var}[kA] = k^2\text{Var}[A]$, where A and B are random variables and k is a constant.

(a) $\text{Var}[\hat{X}] = \text{Var}[\frac{X_1+X_2+\ldots+X_n}{n}] = \frac{1}{n^2}\text{Var}[X_1+X_2+\ldots+X_n] = \frac{1}{n^2}(n\sigma^2) = \frac{\sigma^2}{n}$

(b) $\text{Var}[\hat{X}] = \text{Var}[\frac{X_1+X_2+\ldots+X_n}{n+1}] = \frac{1}{(n+1)^2}\text{Var}[X_1+X_2+\ldots+X_n] = \frac{1}{(n+1)^2}(n\sigma^2) = \frac{n}{(n+1)^2}\sigma^2$

(c) $\text{Var}[\hat{X}] = \text{Var}[\frac{X_1+X_2+\ldots+X_n}{n+n_0}] = \frac{1}{(n+n_0)^2}\text{Var}[X_1+X_2+\ldots+X_n] = \frac{1}{(n+n_0)^2}(n\sigma^2) = \frac{n}{(n+n_0)^2}\sigma^2$

(d) $\text{Var}[\hat{X}] = 0$

(c) Hopefully you see that there is a trade-off. As we add more copies of 0 to our estimator, the bias increases (in absolute value) and the variance decreases. It is a common mistake to assume that an unbiased estimator is always "best." Let's explore this a bit further. Define the *expected total error* as the sum of the variance and the square of the bias. **Compute the expected total error for each of the estimators above.**

Solution: Adding the previous two answers:

(a) $\frac{\sigma^2}{n}$

(b) $\frac{1}{(n+1)^2}(\mu^2 + n\sigma^2)$

(c) $\frac{1}{(n+n_0)^2}(n_0^2\mu^2 + n\sigma^2)$

(d) μ^2

(d) **Argue that all four estimators are really just special cases of the third estimator (with the hyperparameter n_0).**

Solution: The derivation for the third estimator works for *any* value of n_0. The first estimator is just the third estimator with n_0 set to 0:

$$\frac{x_1+x_2+\ldots+x_n}{n+n_0} = \frac{x_1+x_2+\ldots+x_n}{n+0} + \frac{x_1+x_2+\ldots+x_n}{n}.$$

The second estimator is just the third estimator with n_0 set to 1:

$$\frac{x_1+x_2+\ldots+x_n}{n+n_0} = \frac{x_1+x_2+\ldots+x_n}{n+1}.$$

The last estimator is the limiting behavior as n_0 goes to ∞. In other words, we can get arbitrarily close to the fourth estimator by setting n_0 very large:

$$\lim_{n_0\to\infty} \frac{x_1+x_2+\ldots+x_n}{n+n_0} = 0.$$

(e) Say that $n_0 = \alpha n$. **Find the setting for α that would minimize the expected total error, assuming you secretly knew μ and σ.** Your answer will depend on σ, μ, and n.

Solution: First, we write our expression for the total error in terms of α:

$$\frac{1}{(n+n_0)^2}(n_0^2\mu^2+n\sigma^2)$$

$$\frac{1}{(n+\alpha n)^2}((\alpha n)^2\mu^2+n\sigma^2)$$

$$\frac{1}{(1+\alpha)^2}\frac{1}{n^2}(\alpha^2 n^2\mu^2+n\sigma^2)$$

$$\frac{1}{(1+\alpha)^2}(\alpha^2\mu^2+\frac{\sigma^2}{n})$$

$$\frac{\alpha^2}{(1+\alpha)^2}\mu^2+\frac{1}{(1+\alpha)^2}\frac{\sigma^2}{n}$$

Now take the derivative with respect to α and set it equal to 0:

$$\frac{2\alpha}{(1+\alpha)^3}\mu^2-\frac{2}{(1+\alpha)^3}\frac{\sigma^2}{n}=0.$$

$$\frac{2\alpha}{(1+\alpha)^3}\mu^2=\frac{2}{(1+\alpha)^3}\frac{\sigma^2}{n}$$

$$2\alpha\mu^2=2\frac{\sigma^2}{n}$$

$$\alpha=\frac{\sigma^2}{n\mu^2}.$$

(f) For this part, let's assume that we had some reason to believe that μ *should be small* (close to 0) and σ *should be large*. In this case, **what happens to the expression in the previous part?**
Solution: The value of α can be quite large, since the solution has a small value of μ in the denominator. In mathematical terms, we could write the limit as μ goes to 0:

$$\lim_{\mu\to 0}\frac{\sigma^2}{n\mu^2}=\infty.$$

(g) In the previous part, we assumed there was reason to believe that μ *should be small*. Now let's assume that we have reason to believe that μ is not necessarily small, but *should be close to some fixed value* μ_0. **In terms of X and μ_0, how can we define a new random variable X' such that X' is expected to have a small mean?**

Solution: Shift the random variable by X by the constant guess μ_0 to get the random variable $X' = X - \mu_0$. Let's calculate the mean and variance of this new random variable:

$$E[X'] = E[X - \mu_0] = E[X] - \mu_0 = \mu - \mu_0 \approx 0.$$

The last line ($\mu - \mu_0 \approx 0$) comes from the assumption that μ is close to μ_0.

We can also calculate the variance of X':

$$\text{Var}[X'] = \text{Var}[X - \mu_0] = \text{Var}[X] - \text{Var}\mu_0 = \text{Var}[X] - 0 = \text{Var}[X].$$

This is a useful step to understand the relation between X and X', but not necessary for a full solution to the question asked.

(h) Draw a connection between α in this problem and the regularization parameter λ in the ridge-regression version of least-squares. **What does this problem suggest about choosing a regularization coefficient and handling our data-sets so that regularization is most effective**? This is an open-ended question, so do not get too hung up on it.

Solution: The key lesson is another reminder that regularization reduces variance, at the cost of increasing bias, by forcing solutions towards zero. But the bias-variance trade-off is not always the same. If we first center our data around some prior, so that the model is supposed to be close to zero anyways, then we can use larger values of α or λ and reduce variance considerably for a small cost in bias. It may also be instructive to realize that regularization can be thought of as adding "fake" training data which is uniformly zero.

4 Estimation in linear regression

In linear regression, we estimate a vector $y \in \mathbb{R}^n$ by using the columns of a feature matrix $A \in \mathbb{R}^{n \times d}$. Assume that the number of training samples $n \geq d$ and that A has full column rank. Let us now show how well we can estimate the underlying model as the number of training samples grows.

Assume that the true underlying model for our noisy training observations is given by $Y = \underbrace{Ax^*}_{y^*} + W$, with $W \in \mathbb{R}^n$ having iid $W_j \sim \mathcal{N}(0, \sigma^2)$ representing the random noise in the observation Y. Here, the $x^* \in \mathbb{R}^d$ is something arbitrary and not random. After obtaining $\hat{X} = \arg\min_x \|Y - Ax\|_2^2$, we would like to bound the error $\|A\hat{X} - y^*\|_2^2$, which is the prediction error incurred based on the specific n training samples we obtained.

Initially, it might seem that getting a good estimate of this prediction error is hopeless since the training data's A matrix is involved in some complicated way in the estimate. The point of this problem is to show you that this is not the case and we can actually understand this.

(a) Using the standard closed form solution to the ordinary least squares problem, **show that**

$$\|A\hat{X} - y^*\|_2^2 = \|A(A^\top A)^{-1}A^\top W\|_2^2.$$

Solution: Note that the solution to the least squares problem is given by

$$\begin{aligned}\hat{X} &= (A^\top A)^{-1}A^\top y \\ &= (A^\top A)^{-1}A^\top Ax^* + (A^\top A)^{-1}A^\top w \\ &= x^* + (A^\top A)^{-1}A^\top w.\end{aligned}$$

Hence, we have

$$\|A\hat{X} - Ax^*\|_2^2 = \|Ax^* - Ax^* + A(A^\top A)^{-1}A^\top w\|_2^2,$$

which completes the proof.

(b) Given the *reduced* singular value decomposition $A = U\Sigma V^\top$ (with $U \in \mathbb{R}^{n\times d}$, $\Sigma \in \mathbb{R}^{d\times d}$, and $V \in \mathbb{R}^{d\times d}$. The reduced SVD just ignores all the singular vectors that correspond to directions that are in the nullspace of $A^\top$ since A is a tall skinny matrix.), **show that we have**

$$\|A\hat{X} - y^*\|_2^2 = \|U^\top W\|_2^2.$$

(Hint: Recall that the standard Euclidean ℓ_2 norm is unitarily invariant.)

Solution: Given the SVD of A, notice that $(A^\top A)^{-1} = V\Sigma^{-2}V^\top$, and that $A^\top = V\Sigma U^\top$. Using part (a), we have

$$\begin{aligned}\|A\hat{X} - y^*\|_2^2 &= \|A(A^\top A)^{-1}A^\top w\|_2^2 \\ &= \|U\Sigma V^\top V\Sigma^{-2}V^\top V\Sigma U^\top w\|_2^2.\end{aligned}$$

For a matrix X with orthonormal columns, we have $X^\top X = I$, and so performing a string of cancellations yields

$$\begin{aligned}\|A\hat{X} - y^*\|_2^2 &= \|UU^\top w\|_2^2 \\ &= \|U^\top w\|_2^2.\end{aligned}$$

Here the last step uses unitary invariance of the ℓ_2-norm (see discussion 1 for proof): for a unitary matrix U, we have $\|Uz\|_2^2 = \|z\|_2^2$.

(c) If W_0 is a vector with i.i.d standard Gaussians, recall that $a^\top W_0 \sim \mathcal{N}(0, \|a\|_2^2)$. **Use this fact to show that**

$$\frac{1}{n}\mathbb{E}\left[\|A\hat{X} - y^*\|_2^2\right] = \sigma^2\frac{d}{n}.$$

Notice that this kind of scaling is precisely how the average squared error for simple averaging scales when we have multiple iid samples of a random variable and we want to estimate its

mean. This is why we often think about ordinary least squares as just being a kind of generalized averaging. However, the dimension of the parameters being estimated also matters.

Solution:

Notice that the ith component of the vector $U^\top w$ is given by $u_i^\top w \sim \mathcal{N}(0, \sigma^2 \|u_i\|_2^2) = \mathcal{N}(0, \sigma^2)$, since each vector u_i has unit norm.

Hence,

$$\begin{aligned}\mathbb{E}\|U^\top w\|_2^2 &= \mathbb{E}\left[\sum_{i=1}^{d} (u_i^\top w)^2\right] \\ &= \sum_{i=1}^{d} \mathbb{E}[(u_i^\top w)^2] \\ &= \sigma^2 d.\end{aligned}$$

This completes the proof.

(d) Let us now try to answer another basic question. Let us say we have a true linear model, but we try to estimate it using a polynomial of degree D. Clearly, this is overkill, but what do we lose?

Given scalar samples $\{x_i, Y_i\}_{i=1}^n$, let us say that the underlying model is a noisy linear model, i.e. $Y_i = mx_i + c + W_i$. Let us, however, perform polynomial regression: we form the matrix A by using $D+1$ polynomial features (including the constant) of the *distinct* sampling points $\{x_i\}_{i=1}^n$. **How many samples n do we need to ensure that our average squared error is bounded by ε?** Your answer should be expressed as a function of D, σ^2, and ε.

Conclude that as we increase model complexity, we require a proportionally larger number of samples for accurate prediction.

At this point, you are encouraged to take a fresh look at the discussion worksheet which is, in effect, a continuation of this problem to understand the ramifications of the bias/variance tradeoff in the context of ordinary least-squares regression.

Solution: For a polynomial regression model, the matrix A has full column rank when the points are distinct (refer to HW2), and so we can apply the previous part with $d = D+1$. Hence, the error is bounded as

$$\frac{1}{n}\mathbb{E}\left[\|A\hat{X} - y^*\|_2^2\right] = \sigma^2 \frac{D+1}{n},$$

and it suffices to have $n > \frac{\sigma^2(D+1)}{\varepsilon}$ to ensure that this is upper bounded by ε.

Clearly, the number of samples required for error ε increases linearly with the degree D and with the variance σ^2.

(e) Try the above problem out by yourself, by setting $m = 1$, $c = 1$, and sample n points $\{x_i\}_{i=1}^n$ uniformly from the interval $[-1, 1]$. Generate $Y_i = mx_i + c + W_i$ with W_i representing standard Gaussian noise. **Fit a D degree polynomial to this data and show how the average error**

$\frac{1}{n}\|A\hat{X} - y^*\|_2^2$ **scales as a function of both** D **and** n**.** You may show separate plots for the two scalings. It may also be helpful to average over multiple realizations of the noise (or to plot point clouds) so that you obtain smooth curves.

(For this part, any and all python libraries are allowed.)

Solution: In the figures below, we plotted the error vs degree of the polynomial and log of error vs log of the sample size for better understanding. As we can see the error increases with D and decreases with n.

```
import numpy as np
import matplotlib.pyplot as plt

error = np.zeros((19,40)) # defining error as a function of d (degree of the
    polynomial) and n (sample size)
log_n = np.arange(80,160,2)
X = np.random.uniform(-1,1,80) #generating X from unifrom dist with sample size 80
W = np.random.normal(0,1,80) # generating i.i.d noise with sample size 80
Y_s = X+1 # this is Y*
Y = X+W+1 # Y = 1.X+W+1
for N in range(0,40): # number of steps
    n = N+80 # sample size at each step
    for d in range(1,20): #degree of polynomial from 1 to 19
        F = np.polyfit(X,Y,d) #fitting degree d polynomial
        E = 0 #initializing error for this fit with degree=d and sample size = n
        for i in range(n):
            E = E + (np.polyval(F,X[i])-Y_s[i])**2 # adding error of each sample to
     total error
        error[d-1][N]=E/n # normalize the error
    X_add = np.random.uniform(-1,1,1) # generating a new sample for the next step
    to increase the sample size by 1
    W_add = np.random.normal(0,1,1) # generating noise for the new sample for the
    next step to increase the sample size by 1
    X = np.append(X,X_add) # adding the new sample to the sample array
    W = np.append(W,W_add) # adding the new noise value to the noise array
    Y_s = X+1 # computing Y*
    Y = X+W+1 # computing Y

plt.figure()
plt.subplot(121)
plt.plot(np.arange(1,20),error[:,20])
plt.xlabel('degree of polynomial')
plt.ylabel('error')
plt.subplot(122)
plt.loglog(np.arange(80,120),error[5,:])
plt.xlabel('log of number of samples')
plt.ylabel('log of error')
```

5 Robotic Learning of Controls from Demonstrations and Images

Huey, a home robot, is learning to retrieve objects from a cupboard, as shown in Fig. 5. The goal is to push obstacle objects out of the way to expose a goal object. Huey's robot trainer, Anne, provides demonstrations via tele-operation.

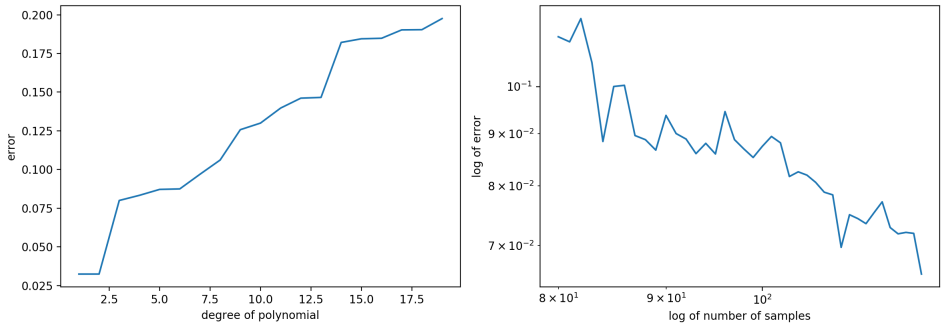

Figure 4: Average error vs D and n

During a demonstration, Huey records the RGB images of the scene for each timestep, $x_0, x_1, ..., x_T$, where $x_i \in \mathbb{R}^{30x30x3}$ and the controls for his mobile base, $u_0, u_1, \ldots, u_T$, where $u_i \in \mathbb{R}^3$. The controls correspond to making small changes in the 3D pose (i.e. translation and rotation) of his body. Examples of the data are shown in the figure.

Under an assumption (sometimes called the Markovian assumption) that all that matters for the current control is the current image, Huey can try to learn a linear *policy* π (where $\pi \in \mathbb{R}^{2700x3}$) which linearly maps image states to controls (i.e. $\pi^\top x = u$). We will now explore how Huey can recover this policy using linear regression. Note please use **numpy** and **numpy.linalg** to complete this assignment.

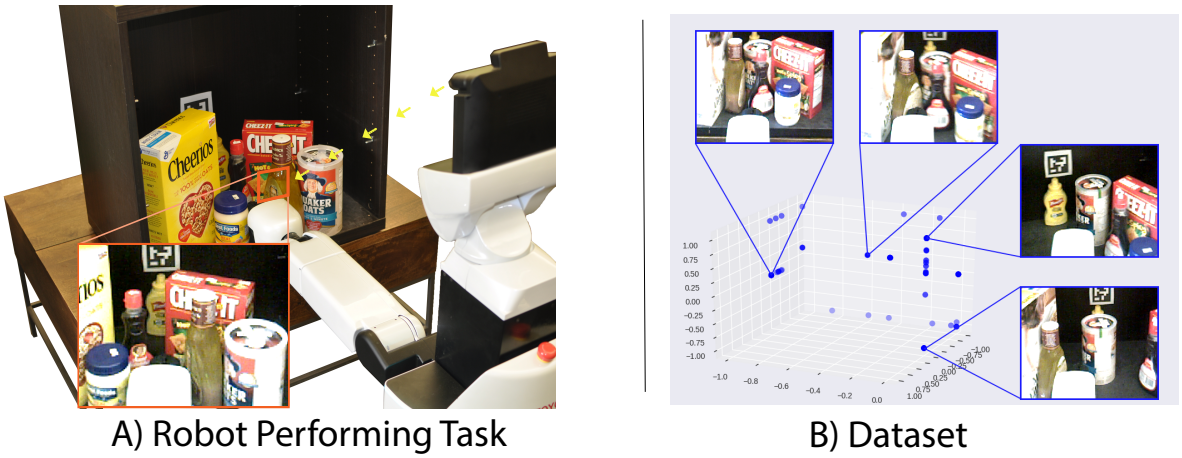

Figure 5: A) Huey trying to retrieve a mustard bottle. An example RGB image of the workspace taken from his head mounted camera is shown in the orange box. The angle of the view gives Huey and eye-in-hand perspective of the cupboard he is reaching into. B) A scatter plot of the 3D control vectors, or u labels. Notice that each coordinate of the label lies within the range of $[-1, 1]$ for the change in position. Example images, or states x, are shown for some of the corresponding control points. The correspondence is indicated by the blue lines.

(a) Load the n training examples from x_train.p and compose the matrix X, where $X \in \mathbb{R}^{nx2700}$. Note, you will need to flatten the images to reduce them to a single vector. The flattened image

vector will be denoted by $\bar{x}$ (where $\bar{x} \in \mathbb{R}^{2700x1}$). Next, load the n examples from y_train.p and compose the matrix U, where $U \in \mathbb{R}^{nx3}$. Try to perform ordinary least squares to solve:

$$\min_{\pi} \|X\pi - U\|_2^F$$

to learn the *policy* $\pi \in \mathbb{R}^{2700\times 3}$. **Report what happens as you attempt to do this and explain why.**

Solution: The matrix is singular and not invertible. The reason is there isn't enough data to cover the high dimensional image space, so many solutions exist to solve the problem. Specifically, we only train the policy on 91 images, however a single RGB image is 30x30x3 in dimensionaility. So, with that many parameters, we can basically fit any arbitrary set of controls. (This is called the temptation to memorize in machine learning.) We can use ridge regression though to help condition the optimization to favor solutions with a small ℓ_2 norm.

(b) Now try to perform ridge regression:

$$\min_{\pi} ||X\pi - U||_2^2 + \lambda ||\pi||_2^2$$

on the dataset for regularization values $\lambda = \{0.1, 1.0, 10, 100, 1000\}$. Measure the average squared Euclidean distance for the accuracy of the policy on the training data:

$$\frac{1}{n} \sum_{i=0}^{n-1} ||\bar{x}_i^T \pi - u_i||_2^2$$

Report the training error results for each value of λ.

Solution:

The learned policy should match the training data with very low error. For all values of λ, in the specified range, you should see the error is below 10^{-8}, which is a very good fit for the datset. It should be noted that the ability to fit training data perfectly does not neccesarily mean the robot will perform well on unseen data. Even with the ridge penalties set to these values, the policy has apparently just memorized the controls.

Why was it able to do this? Remember, all the controls are small numbers between -1 and 1 while the training image data has large numbers. Consequently, small numbers for the parameters will allow these controls to be recovered. Furthermore, there are so many parameters that the work of memorizing the controls can be distributed across them. This further shrinks the parameters required to do this memorization. The ridge penalties are simply incapable of discouraging this memorization.

(c) Next, we are going to try standardizing the states. For each pixel value in each data point, x, perform the following operation:

$$x = \frac{x}{255} * 2 - 1.$$

Since we know the maximum pixel value is 255, this rescales the data to be between $[-1,1]$. **Repeat the previous part and report the average squared training error for each value of λ.**

Solution:

The answers for the fitting error for $\lambda = \{0.1, 1.0, 10, 100, 1000\}$ are $\{0.000, 0.000, 0.0016, 0.035, 0.25\}$. With standardization applied, we see as the regularization term is decreased the training loss is lowered. This can be interpreted as the variance of the model is increased as the possible function class is expanded.

We can also see that indeed having the smaller values in the images is forcing the system to use bigger values for the parameters if it wants to memorize the controls. The regularization penalty is now able to begin to discourage this.

(d) Evaluate both *policies* (i.e. with and without standardization on the new validation data x_test.p and y_test.p for the different values of λ. **Report the average squared Euclidean loss** and **qualitatively explain how changing the values of λ affects the performance in terms of bias and variance**.

Solution:

The answer is for $\lambda = \{0.1, 1.0, 10, 100, 1000\}$ is $\{0.87, 0.86, 0.83, 0.72, 0.73\}$ for with standardization and $\{0.77, 0.77, 0.77, 0.77, 0.77\}$ for with out.

The results with standardization illustrate that because the state space is so high dimensional the policy has trouble generalizing, thus adding bias (i.e. increasing λ can help generalization). However, increasing it too much can lead to worst performance.

We emperically see that without standardization the test error is higher, in the next section we will examine how we can characterize this.

It should be noted that the ability of Huey to generalize is still quite inadequate for a real robot policy. Later, in the course we will explore how to get significantly lower error on a larger dataset using convolutional neural networks.

(e) To better understand how standardizing improved the loss function, we are going to evaluate the *condition number k* of the optimization, which is defined as

$$k = \frac{\sigma^2_{\max}(X^T X + \lambda I)}{\sigma^2_{\min}(X^T X + \lambda I)}$$

or the ratio of the maximum eigenvalue to the minimum eigenvalue of the relevant matrix. For the regularization value of $\lambda = 100$, **report the condition number with the standardization technique applied and without**.

Solution: The condition number without standardization is $k = 2.78x10^{15}$ and with standardization is $k = 197781$. By standardizing our data, we are able to significantly reduce the ratio of the eigenvalues, which makes our optimization less sensitive to noise in the data when performing matrix inversion.

6 Your Own Question

Write your own question, and provide a thorough solution.

Writing your own problems is a very important way to really learn material. The famous "Bloom's Taxonomy" that lists the levels of learning is: Remember, Understand, Apply, Analyze, Evaluate, and Create. Using what you know to create is the top-level. We rarely ask you any HW questions about the lowest level of straight-up remembering, expecting you to be able to do that yourself. (e.g. make yourself flashcards) But we don't want the same to be true about the highest level.

As a practical matter, having some practice at trying to create problems helps you study for exams much better than simply counting on solving existing practice problems. This is because thinking about how to create an interesting problem forces you to really look at the material from the perspective of those who are going to create the exams.

Besides, this is fun. If you want to make a boring problem, go ahead. That is your prerogative. But it is more fun to really engage with the material, discover something interesting, and then come up with a problem that walks others down a journey that lets them share your discovery. You don't have to achieve this every week. But unless you try every week, it probably won't happen ever.