CS 189 Introduction to Machine Learning
Fall 2017 HW6

This homework is due **Friday, October 6 at 10pm.**

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.

2 Step Size in Gradient Descent

By this point in the class, we know that gradient descent is a powerful tool for moving towards local minima of general functions. We also know that local minima of convex functions are global minima. In this problem, we will look at the convex function $f(x) = ||x-b||_2$. Note that we are using "just" the regular Euclidean ℓ_2 norm, *not* the norm squared! This problem illustrates the importance of understanding how gradient descent works and choosing step sizes strategically. In fact, there is a lot of active research in variations on gradient descent. We want to make sure the we get to some local minimum and we want to do it as quickly as possible.

You have been provided with a tool in `step_size.py` which will help you visualize the problems below.

(a) **Prove that $f(x) = \|x-b\|_2$ is a convex function of x.**

(b) We are minimizing $f(x) = \|x-b\|_2$, where $x \in \mathbb{R}^2$ and $b = [4.5, 6] \in \mathbb{R}^2$, with gradient descent. We use a constant step size of $t_i = 1$. That is,

$$x_{i+1} = x_i - t_i \nabla f(x_i) = x_i - \nabla f(x_i).$$

We start at $x_0 = [0,0]$. **Will gradient descent find the optimal solution? If so, how many steps will it take to get within 0.01 of the optimal solution? If not, why not?** Prove your answer. (Hint: use the tool to compute the first ten steps.) **What about general $b \neq \vec{0}$?**

(c) We are minimizing $f(x) = \|x-b\|_2$, where $x \in \mathbb{R}^2$ and $b = [4.5, 6] \in \mathbb{R}^2$, now with a decreasing step size of $t_i = (\frac{5}{6})^i$ at step i. That is,

$$x_{i+1} = x_i - t_i \nabla f(x_i) = x_i - (\frac{5}{6})^i \nabla f(x_i).$$

We start at $x_0 = [0,0]$. **Will gradient descent find the optimal solution? If so, how many steps will it take to get within 0.01 of the optimal solution? If not, why not?** Prove your answer. (Hint: examine $\|x_i\|_2$.) **What about general $b \neq \vec{0}$?**

(d) We are minimizing $f(x) = \|x-b\|_2$, where $x \in \mathbb{R}^2$ and $b = [4.5, 6] \in \mathbb{R}^2$, now with a decreasing step size of $t_i = \frac{1}{i+1}$ at step i. That is,

$$x_{i+1} = x_i - t_i \nabla f(x_i) = x_i - \frac{1}{i+1} \nabla f(x_i).$$

We start at $x_0 = [0,0]$. **Will gradient descent find the optimal solution? If so, how many steps will it take to get within 0.01 of the optimal solution? If not, why not?** Prove your answer. (Hint: examine $\|x_i\|_2$, consider what you know about the harmonic and alternating harmonic series.) **What about general $b \neq \vec{0}$?**

(e) Now, say we are minimizing $f(x) = \|Ax-b\|_2$. Use the code provided to test several values of A with the step sizes suggested above. Make plots to visualize what is happening. We suggest trying $A = [[10,0],[0,1]]$ and $A = [[15,8],[6,5]]$. **Will any of the step sizes above work for all choices of A and b?** You do not need to prove your answer, but you should briefly explain your reasoning.

3 Convergence Rate of Gradient Descent

In the previous problem, you examined $||Ax-b||_2$ (without the square). You showed that even though it is convex, getting gradient descent to converge requires some care. In this problem, you will examine $\frac{1}{2}||Ax-b||_2^2$ (with the square). You will show that now gradient descent converges quickly.

Consider the quadratic function $f(x) = \frac{1}{2}\|Ax-b\|_2^2$ such that A^TA is positive definite.

You may find Problem 3 on HW5 useful for various parts of this problem.

(a) First, consider the case $b = \vec{0}$, and think of each $x \in \mathbb{R}^d$ as a "state". Performing gradient descent moves us sequentially through the states, which is called a "state evolution" in the parlance of linear systems. **Write out the state evolution for iterations of gradient descent using step-size $\gamma > 0$.** Use x_0 to denote the initial condition of where you start gradient descent from.

(b) A state evolution is said to be stable if it does not blow up arbitrarily over time. **When is the state evolution of the iterations you calculated above stable when viewed as a dynamical system?**

(c) We want to bound the progress from steps of gradient descent in the general case. To do this, we first show a slightly more general bound, which relates how much the spacing between two points changes if they *both* take a gradient step. If this spacing shrinks, this is called a contraction. Define $\varphi(x) = x - \gamma\nabla f(x)$, for some constant step size $\gamma > 0$. **Show that for any $x, x' \in \mathbb{R}^n$,**

$$\|\varphi(x) - \varphi(x')\|_2 \leq \beta\|x - x'\|_2$$

where $\beta = \max\{|1-\gamma\lambda_{\max}(A^TA)|, |1-\gamma\lambda_{\min}(A^TA)|\}$. Note that $\lambda_{\min}(A^TA)$ denotes the smallest eigenvalue of the matrix A^TA; similarly, $\lambda_{\max}(A^TA)$ denotes the largest eigenvalue of the matrix A^TA.

Can you see from the previous part why we are doing this?

(d) Now we give a bound for progress after k steps of gradient descent. Define

$$x^* = \arg\min_{x\in\mathbb{R}^d} f(x).$$

Show that

$$\|x_{k+1} - x^*\|_2 = \|\varphi(x_k) - \varphi(x^*)\|_2$$

and conclude that

$$\|x_{k+1} - x^*\|_2 \leq \beta^{k+1}\|x_0 - x^*\|_2.$$

(e) However, what we actually care about is progress in the objective value $f(x)$. That is, we want to show how quickly $f(x)$ is converging to $f(x^*)$. We can do this by relating $f(x) - f(x^*)$ to $\|x - x^*\|_2$; or even better, relating $f(x) - f(x^*)$ to $\|x_0 - x^*\|_2$, for some starting point x_0. First, **show that**

$$f(x) - f(x^*) = \frac{1}{2}\|A(x - x^*)\|_2^2.$$

(f) **Show that**

$$f(x_k) - f(x^*) \leq \frac{\alpha}{2}\|x_k - x^*\|_2^2,$$

for $\alpha = \lambda_{\max}(A^TA)$, and conclude that

$$f(x_k) - f(x^*) = \frac{\alpha}{2}\beta^{2(k+1)}\|x_0 - x^*\|_2^2.$$

(g) Finally, the convergence rate of is a function of β, so it's desirable for β to be as small as possible. **Pick γ such that β is as small as possible**, as a function of $\lambda_{\min}(A^TA), \lambda_{\max}(A^TA)$. Then, **write the resulting convergence rate as a function of $Q = \frac{\lambda_{\max}(A^TA)}{\lambda_{\min}(A^TA)}$**, the condition number of A^TA.

4 Sensors, Objects, and Localization

In this problem, we will be using gradient descent to solve the problem of figuring out where objects are given noisy distance measurements. (This is roughly how GPS works and students who have taken EE16A have seen a variation on this problem in lecture and lab.)

First, the setup. Let us say there are m sensors and n objects located in a $2d$ plane. The m sensors are located at the points $(a_1, b_1), \ldots, (a_m, b_m)$. The n objects are located at the points $(x_1, y_1), \ldots, (x_n, y_n)$. We have measurements for the distances between the sensors and the objects: D_{ij} is the measured distance from sensor i to object j. The distance measurement has noise in it. Specifically, we model

$$D_{ij} = ||(a_i, b_i) - (x_j, y_j)|| + Z_{ij},$$

where $Z_{ij} \sim N(0, 1)$. The noise is independent across different measuments.

Code has been provided for data generation to aid your explorations.

For this problem, all Python libraries are permitted.

(a) Consider the case where $m = 1$ and $n = 7$. That is, there are 7 sensors and 1 object. Suppose that we know the exact location of the 7 sensors but not the 1 object. We have 7 measurements of the distances from each sensor to the object $D_{i1} = d_i$ for $i = 1, \ldots, 7$. Because the underlying measurement noise is modeled as iid Gaussian, the interesting part of the log likelihood function is

$$L(x_1, y_1) = -\sum_{i=1}^{7} (\sqrt{(a_i - x_1)^2 + (b_i - y_1)^2} - d_i)^2, \tag{1}$$

ignoring the constant term. **Manually compute the symbolic gradient of the log likelihood function, with respect to x_1 and y_1.**

(b) The provided code generates

- $m = 7$ sensor locations (a_i, b_i) sampled from $N(0, \sigma_s^2 I)$
- $n = 1$ object locations (x_1, y_1) sampled from $N(\mu, \sigma_o^2 I)$
- $mn = 7$ distance measurements $D_{i1} = ||(a_i, b_i) - (x_1, y_1)|| + N(0, 1)$.

for $\mu = [0,0]^T$, $\sigma_s = 100$ and $\sigma_o = 100$. **Solve for the maximum likelihood estimator of (x_1, y_1) by gradient descent. Report the estimated (x_1, y_1) for the given sensor locations.** Try two approaches for initializing gradient descent: starting at $\vec{0}$ and starting at a random point. Describe how you chose your step size in a reasonable manner.

(c) (Local Mimima of Gradient Descent) In this part, we vary the location of the single object among different positions:

$$(x_1, y_1) \in \{(0,0), (100,100), (200,200), \ldots, (900,900)\}.$$

For each choice of (x_1, y_1), **generate the following data set** 10 **times**:

- Generate $m = 7$ sensor locations (a_i, b_i) from $N(0, \sigma_s^2 I)$ (Use the same σ_s from the previous part.)
- Generate $mn = 7$ distance measurements $D_{i1} = ||(a_i, b_i) - (x_1, y_1)|| + N(0, 1)$.

For each data set, carry out gradient descents 100 **times to find a prediction for** (x_1, y_1). We are pretending we do not know (x_1, y_1) and are trying to predict it. For each gradient descent, take 1000 iterations with step-size 0.1 and a random initialization of (x, y) from $N(0, \sigma^2 I)$, where $\sigma = x_1 + 1$.

- **Draw the contour plot of the log likelihood function of a particular data set for $(x_1, y_1) = (0,0)$ and $(x_1, y_1) = (100,100)$.**
- For each of the ten data sets and each of the ten choices of (x_1, y_1), calculate the number of distinct points that gradient descent converges to. Then, for each of the ten choices of (x_1, y_1), calculate the average of the number of distinct points over the ten data sets. **Plot the average number of local minima against x_1.** For this problem, two local minima are considered identical if their distance is within 0.01.

  Hint: `np.unique` and `np.round` will help.
- For each of the ten data sets and each of the ten choices of (x_1, y_1), calculate the proportion of gradient descents which converge to what you believe to be a global minimum (that is, the minimum point in the set of local minima that you have found). Then, for each of the ten choices of (x_1, y_1), calculate the average of the proportion over the ten data sets. **Plot the average proportion against x_1.**

(d) Repeat the previous part, except explore what happens as you reduce the variance of the measurement noise. **Comment with appropriate plots justifying your comments.**

(e) Repeat the previous part again, except explore what happens as you increase the number of sensors. **Comment with appropriate plots justifying your commens.**

(f) Now, we are going to turn things around. Instead of assuming that we know where the sensors are, suppose that the sensor locations are unknown. But we get some training data for 100 object locations that are known. We want to use gradient descent to estimate the sensor locations, and then use these estimated sensor locations on new test data for objects.

Consider the case where $m = 7$ sensors and the training data consists of $n = 100$ object positions. We have 7 noisy measurements of the distances from each sensor to the object $D_{i1} = d_{ij}$ for $i = 1,\ldots,7; j = 1,2,\ldots,100$.

Use the provided code to generate

- $m = 7$ sensor locations (a_i, b_i) sampled from $N(0, \sigma^2 I)$
- $n = 100$ object locations (x_j, y_j) sampled from $N(\mu, \sigma^2 I)$ in two groups: (1) Training data with $\mu = \vec{0}$, (2) Interpolating Test data with $\mu = \vec{0}$, and (3) Extrapolating Test data with $\mu = [300, 300]^T$.
- $mn = 700$ distance measurements $D_{ij} = ||(a_i, b_i) - (x_j, y_j)|| + N(0,1)$ for each of the data sets.

Use the first dataset as the training data and the second two as two kinds of test data: points drawn similarly to the training data, and points drawn in different way.

Calculate the MLE for the sensor locations $(\hat{a}_i, \hat{b}_i)$ given the training object locations (x_j, x_j) and all the pairwise training distance measurements $(D_{ij} = d_{ij})$. (Use gradient descent with multiple random starts, picking the best estimates as your estimate.)

Use these estimated sensor locations as though they were true sensor locations to compute object locations for both sets of test data. (Use gradient descent with multiple random starts, picking the best estimate as your estimated position.) **Report the mean-squared error in object positions on both test data sets.**

5 Vegetables!

The goal of the problem is help the class build a dataset for image classification, which will be used later in the course to classify fruits and vegetables. Please pick ten of the following vegetables:

1. Spinach
2. Celery
3. Potato (not sweet potato)
4. Bell Peppers
5. Tomato
6. Cabbage

7. Radish
8. Broccoli
9. Cauliflower
10. Carrot
11. Eggplant
12. Garlic
13. Ginger

Take two pictures of each specific vegetable, for a total of 20 vegetable pictures, against any background such that the vegetable is centered in the image and the vegetable takes up approximately a third of the image in area; see below for examples. Save these pictures as .png files. **Do not** save the pictures as .jpg files, since these are lower quality and will interfere with the results of your future coding project. Place all the images in a folder titled *data*. Each image should be titled *[vegetable name]_[number].png* where [number]$\in \{0, 1\}$. Ex: broccoli_0.png, broccoli_1.png, carrot_0.png, etc ... (the ordering is irrelevant). Please also include a file titled *rich_labels.txt* which contain entries on new lines prefixed by the file name *[vegetable name]_[number]*, followed by a description of the image (maximum of eight words) with only a space delimiter in between. Ex: carrot_0 one purple dragon carrot on wood table. To turn in the folder compress the file to a .zip and upload it to Gradescope.

Figure 1: Example of five purple dragon carrots on green background. (This is a joke; we don't need pictures of purple carrots.)

Please keep in mind that data is an integral part of Machine Learning. A large part of research in this field relies heavily on the integrity of data in order to run algorithms. It is, therefore, vital that your data is in the proper format and is accompanied by the correct labeling not only for your grade on this section, but for the integrity of your data.

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.