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) Let $x, b \in \mathbb{R}^d$. **Prove that $f(x) = \|x-b\|_2$ is a convex function of x.**

Solution: Recall that a function is convex if

$$f(\lambda x_1 + (1-\lambda)x_2) \leq \lambda f(x_1) + (1-\lambda) f(x_2)$$

where $0 \leq \lambda \leq 1$. In this case, the triangle inequality gives us:

$$\begin{aligned} \|\lambda x_1 + (1-\lambda)x_2 - b\|_2 &= \|\lambda(x_1 - b) + (1-\lambda)(x_2 - b)\|_2 \\ &\leq \|\lambda(x_1 - b)\|_2 + \|(1-\lambda)(x_2 - b)\|_2 \\ &= \lambda\|x_1 - b\|_2 + (1-\lambda)\|x_2 - b\|_2. \end{aligned}$$

(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 $b \neq \vec{0}$?**

Solution: Using the tool provided (or computing gradients by hand), we get

$$x_6 = x_8 = [4.2, 5.6]$$

and

$$x_7 = x_9 = [4.8, 6.4].$$

Examine the formula

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

and notice that x_{i+1} only depends on x_i, regardless of what step we are on (this is only the case because we are using a constant step size). It means that if $x_i = x_j$, then $x_{i+1} = x_{j+1}$.

Thus, this pattern will repeat indefinitely and we do not reach the optimal solution. It is also helpful to notice that

$$\|x_6\|, \|x_7\| \in \mathbb{Z}.$$

In general, notice that

$$-\nabla f(x_i) = \frac{b - x_i}{\|x_i - b\|}$$

will always have unit length. In fact, we can prove by induction that x_k is always an integer multiple of the unit vector $\frac{b}{\|b\|}$. Thus,

$$\|x_k\| \in \mathbb{Z}.$$

If we are lucky and $\|b\|$ happens to be an integer (or very close to integer), then we will reach or get near the optimal solution. Otherwise, our step size is too course to hit it.

In fact, for the function $\|x - b\|$, a constant step size will rarely work. We are too prone to "jumping over" our solution. A decreasing step size is necessary.

(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** $b \neq \vec{0}$**?**

Solution: Notice that we can express x_{i+1} as the sum of all the steps taken, so we can express its norm as

$$\|x_{i+1}\| = \|x_0 - \sum_{j=0}^{i} (\frac{5}{6})^j \nabla f(x_i)\| \leq \|x_0\| + \sum_{j=0}^{i} \|(\frac{5}{6})^j \nabla f(x_i)\|$$

But all of $\|\nabla f(x_i)\|$ are exactly 1, so we can write this as

$$\|x_{i+1}\| \leq \|x_0\| + \sum_{j=0}^{i} (\frac{5}{6})^j \|\nabla f(x_i)\| = 0 + \sum_{j=0}^{i} (\frac{5}{6})^j \leq 6.$$

In words, we can never "travel" a distance of more than 6 from $\vec{0}$, so we will never get to our optimal solution b if $\|b\| > 6$. So, while we need a decreasing step size, we also have to be sure it does not decrease too quickly.

(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 $b \neq \vec{0}$?**

Solution: The key realization here is that all our gradient steps have $\|\nabla f(x_i)\| = 1$ and move along the same vector towards b (to be precise, we could prove by induction that our x_i and $\nabla f(x_i)$ will always be in the span of b). We can write

$$\|x_{i+1}\| = \|x_0 - \sum_{j=0}^{i} (\frac{1}{j+1})\nabla f(x_i)\| = \sum_{j=0}^{i} \|(\frac{1}{j+1})\nabla f(x_i)\| = \sum_{j=1}^{i} \frac{1}{j+1} \approx \ln i.$$

This sum diverges, so we will reach the optimal solution b eventually, but it could take quite a while. For $b = [4.5, 6]$ it takes about 1000 steps. In generally, the number of steps it takes will be exponential in $\|b\|$, since we have

$$\ln i \approx \|b\| \implies i \approx e^{\|b\|}.$$

We may not hit b exactly. Once we exceed b, we start oscillating around it. Let i_0 be the first step where we "cross over" b. After that, we know we are always moving towards b (whether we are stepping backwards or forwards), so after step i (assuming $i > i_0$), we can be at most $\frac{1}{i+1}$ from the optimal solution (if we are more than $\frac{1}{i+1}$ away, that means our previous step went in the wrong direction).

(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.

Solution: The first two step sizes did not work for $A = I$, so they will not work in general. The last step sizes *does* always work, though it may take a very long time to converge.

You are not expected to have a proof, but the key idea is to notice that even though we are not always moving directly towards b, we can decompose our gradient steps into two parts: the projection along the axis towards b and the projection along the orthogonal axis. In this perspective, we can repeat the arguments of the previous part along each axis.

Another perspective is to notice that the x_i are still ultimately moving towards b, but in this case along a curved arc instead of a line. Still, the arc has finite length, so we will eventually reach b. Once we are close to b, successive step sizes will oscillate around b.

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.

For a matrix $A \in \mathbb{R}^{n\times d}$ and a vector $b \in \mathbb{R}^n$, 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.

Solution: The general formula for the gradient update is

$$x_{n+1} = x_n - \gamma\nabla f(x_n)$$

which in this case is

$$x_{n+1} = x_n - \gamma(A^TAx_n - A^Tb) = (I-\gamma A^TA)x_n - \gamma A^Tb.$$

Since we assumed $b = \vec{0}$, this works out to

$$x_{n+1} = (I-\gamma A^TA)x_n,$$

so by induction we can show

$$x_{n+1} = (I-\gamma A^TA)^{n+1}x_0.$$

(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?**

Solution: An important fact in linear algebra is that if we take a matrix M with eigenvalues λ_i to the power of k, then the resulting matrix M^k has eigenvalues λ_i^k with the same eigenvectors. The proof of this fact (inductively) is

$$M^kv = M^{k-1}Mv = M^{k-1}\lambda v = \lambda M^{k-1}v = \lambda\lambda^{k-1}v = \lambda^k v.$$

In order for the state evolution to be stable, we need *all* the eigenvalues of $(I-\gamma A^TA)$ to be less than or equal to 1 in absolute value (that is, they lie in the unit circle), otherwise $(I-\gamma A^TA)^k$ can have huge eigenvalues (and thus, when applied to the corresponding eigenvector, produces a huge vector).

More formally, this connects to an identity we have previously reviewed, which is

$$\|(I-\gamma A^TA)^kv\|_2 \le |\lambda_{\max}(I-\gamma A^TA)^k|\|v\|_2 = (|\lambda_{\max}(I-\gamma A^TA)|)^k\|v\|_2.$$

So if the largest eigenvalue of $(I-\gamma A^TA)$ (in absolute value) is bounded by 1, then the system is stable.

(c) We want to bound the progress from steps of gradient descent in the general case, when b is arbitrary. 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** $x, x' \in \mathbb{R}^d$,

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

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

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

Solution: We start off by expanding the left side of the equation we are interested in:

$$\begin{aligned}\|\varphi(x) - \varphi(x')\|_2 &= \|(x - \gamma A^T(Ax - b)) - (x' - \gamma A^T(Ax' - b))\|_2 \\ &= \|(x - x') - \gamma A^T A(x - x')\|_2 \\ &= \|(I - \gamma A^T A)(x - x')\|_2\end{aligned}$$

Let $B = I - \gamma A^T A$. We can now square both sides to obtain

$$\|\varphi(x) - \varphi(x')\|_2^2 \leq (x - x')^\top B^\top B(x - x').$$

Recall the definition of the Rayleigh quotient (you computed a generalization of this in Problem 3 of HW5), by which we know that for a symmetric matrix X, we have

$$R(v) = \frac{v^\top X v}{v^\top v} \leq R(v_1),$$

when v_1 is the maximal eigenvector of the matrix. In particular, we have $R(v_1) = \lambda_1(X)$, the maximum eigenvalue of the matrix X.

Substituting this in our result, we see that

$$\|\varphi(x) - \varphi(x')\|_2^2 \leq \lambda_1(B^\top B)\|x - x'\|_2^2,$$

and consequently, we have

$$\|\varphi(x) - \varphi(x')\|_2 \leq \sqrt{\lambda_1(B^\top B)}\|x - x'\|_2.$$

We must now compute the maximum eigenvalue of $B^\top B$. We know that the eigenvalues of $B^\top B$ are the squared eigenvalues of B. In particular, that means that the maximum eigenvalue of $B^\top B$ is either the square of the maximum eigenvalue of B (when that is large and positive), or the minimum eigenvalue of B (when that is large and negative). Since the maximum and minimum eigenvalues of $I - \gamma A^\top A$ are given by $1 - \gamma \lambda_{\min}(A^\top A)$, and $1 - \gamma \lambda_{\max}(A^\top A)$, we have

$$\lambda_1(B^\top B) = \max((1 - \gamma \lambda_{\min}(A^\top A))^2, (1 - \gamma \lambda_{\max}(A^\top A))^2).$$

Why are the maximum and minimum eigenvalues as stated? This is because if $\lambda_i(A^TA)$ is an eigenvalue of A^TA, then $1-\gamma\lambda_i(A^TA)$ is an eigenvalue of $(I-\gamma A^TA)$ (write out the definition of an eigenvalue to see this).

Taking the square root and combining the pieces, we have

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

(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

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

Solution: Note that $\varphi(x^*) = x^*$ at optimality and $\varphi(x_k) = x_k - \gamma\nabla f(x_k) = x_{k+1}$. Then,

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

Applying the previous part, we have

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

Applying the same bound k times, we have

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

(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**

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

Solution: Note that at $\nabla f(x^*) = A^T(Ax^*-b) = 0$ (the gradient evaluated at x^*), implying that $A^TAx^* = A^Tb$. From here we do a bit of algebra to reach our answer, canceling terms and then substituting for $A^T b$ and rearranging:

$$\begin{aligned}
f(x) - f(x^*) &= \frac{1}{2}\|Ax-b\|_2^2 - \frac{1}{2}\|Ax^*-b\|_2^2 \\
&= \frac{1}{2}\left((x^T A^T Ax - 2b^T Ax + b^T b) - (x^{T*}A^T Ax^* - 2b^T Ax^* + b^T b)\right) \\
&= \frac{1}{2}\left((x^T A^T Ax - 2b^T Ax) - (x^{T*}A^T Ax^* - 2b^T Ax^*)\right) \\
&= \frac{1}{2}\left((x^T A^T Ax - 2x^{T*}A^T Ax) - (x^{T*}A^T Ax^* - 2x^{T*}A^T Ax^*)\right) \\
&= \frac{1}{2}\left(x^T A^T Ax - 2x^{T*}A^T Ax + x^{T*}A^T Ax^*\right) \\
&= \frac{1}{2}\|Ax - Ax^*\|_2^2 \\
&= \frac{1}{2}\|A(x-x^*)\|_2^2
\end{aligned}$$

(f) **Show that**

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

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

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

Solution: Starting from the previous part, we have

$$\begin{aligned}
f(x_k) - f(x^*) &= \frac{1}{2}\|A(x_k - x^*)\|_2^2 \\
&= (x_k - x^*)^T A^T A(x_k - x^*) \\
&\le \frac{1}{2}\lambda_{\max}(A^T A)\|x_k - x^*\|_2^2 \\
&= \frac{\alpha}{2}\|x_k - x^*\|_2^2
\end{aligned}$$

Combining this with part ((e)), we get the desired geometric convergence rate for least squares (quadratic functions)

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

(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 $\lambda_{\min}(A^T A), \lambda_{\max}(A^T A)$.

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.

Solution: We would like to solve

$$\min \max\{|1-\gamma\lambda_{\max}(A^TA)|, |1-\gamma\lambda_{\min}(A^TA)|\}$$

Recall that $0 < \lambda_{\min}(A^TA) \leq \lambda_{\max}(A^TA)$ and that $\gamma > 0$. Then, the optimal solution is when the two values are equal in absolute value but opposite in value.

$$-(1-\gamma\lambda_{\max}(A^TA)) = 1-\gamma\lambda_{\min}(A^TA)$$

This gives $\gamma = \frac{2}{\lambda_{\max}(A^TA)+\lambda_{\min}(A^TA)}$.

Then, the convergence result from the previous part can we written as

$$\begin{aligned} f(x_k) - f(x^*) &= \frac{\alpha}{2}\left(\frac{\lambda_{\max}(A^TA)-\lambda_{\min}(A^TA)}{\lambda_{\max}(A^TA)+\lambda_{\min}(A^TA)}\right)^{2k}\|x_0-x^*\|_2^2 \\ &= \frac{\alpha}{2}\left(\frac{Q-1}{Q+1}\right)^{2k}\|x_0-x^*\|_2^2 \end{aligned}$$

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

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

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

Solution: We identify x, y with x_1, y_1 respectively for notational simplicity. We have

$$D_{i1} \sim N(\sqrt{(a_i - x)^2 + (b_i - y)^2}, 1). \tag{2}$$

The log likelihood function is

$$\begin{aligned} L(x,y) &= \sum_{i=1}^{7} \log P(D_{i1} = d_i) \\ &= -\frac{1}{2}\sum_{i=1}^{7}(\sqrt{(a_i - x)^2 + (b_i - y)^2} - d_i)^2 + \text{Constant} \end{aligned}$$

For notational simplicity, define $\phi_i(x,y) = \sqrt{(a_i - x)^2 + (b_i - y)^2}$. Then the gradient of $\phi_i(x,y)$ with respect to x, y is

$$\nabla \phi_i(x,y) = (\frac{x - a_i}{\phi_i(x,y)}, \frac{y - b_i}{\phi_i(x,y)})^T.$$

The gradient of $L(x,y)$ is

$$\begin{aligned} \nabla L(x,y) &= \nabla[-\frac{1}{2}\sum_{i=1}^{7}(\phi_i(x,y) - d_i)^2] \\ &= -\sum_{i=1}^{7}(\phi_i(x,y) - d_i)\nabla\phi_i(x,y) \\ &= -\sum_{i=1}^{7}\frac{\phi_i(x,y) - d_i}{\phi_i(x,y)}(x - a_i, y - b_i)^T, \end{aligned}$$

where the second line follows by the chain rule and the third line follows from plugging in the gradient of ϕ_i directly.

(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 on the negative log-likelihood. 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.

Solution: Please refer to the solution code. We should use a step size that is not too large to make sure that the algorithm converges and also avoid choosing a step size that is too small so that the algorithm converges faster.

```

##############################################################################
#########   Results  #########################################################
##############################################################################
The real object location is
[[ 44.38632327  33.36743274]]
The estimated object location with zero initialization is
[[ 43.07188426  32.71217807]]
The estimated object location with random initialization is
[[ 43.07188426  32.71217807]]
```

(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**:

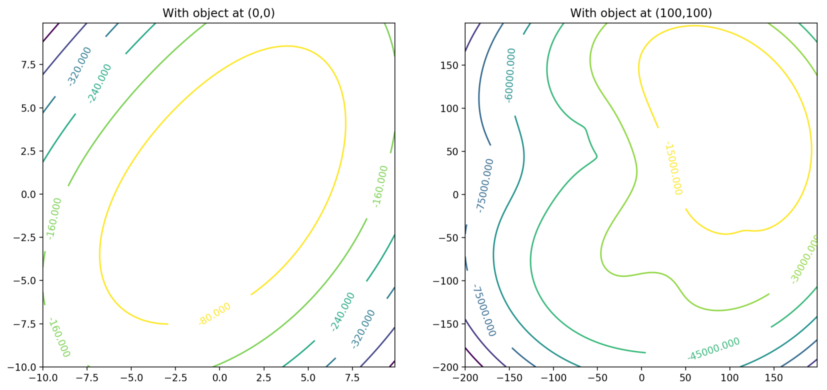
- Generate $m = 7$ sensor locations (a_i, b_i) from $N(0, \sigma_s^2 I)$ (Use the same σ_s from the previous part.)
- Generate $mn = 7$ distance measurements $D_{i1} = ||(a_i, b_i) - (x_1, y_1)|| + N(0, 1)$.

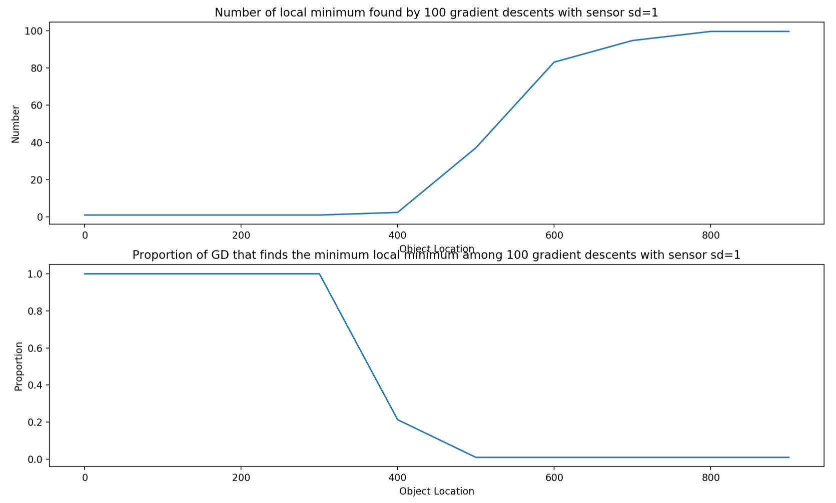
For each data set, carry out gradient descents 100 **times to find a prediction for** (x_1, y_1). We are pretending we do not know (x_1, y_1) and are trying to predict it. For each gradient descent, take 1000 iterations with step-size 0.1 and a random initialization of (x, y) from $N(0, \sigma^2 I)$,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

Solution: Please refer to the solution code and figures **??** and **??**. You will notice that as we get to the larger values for the true location of the object, gradient descent converges to more and more local minima. This is happening for two reasons. First, we are creating more local minima because the distance of the object to the sensors is slowly dominating the distance between the sensors themselves. Image the extreme case where sensors are within a radius of 1 while the object is about 1×10^8 away. Then the distances of the object to each of the sensor are almost the same. And the estimated location of the object can be anywhere around the circle of radius 1×10^8. In other words, all of these places would be local minima. Second, gradient descent is starting further away from the true optimal, which increases the chances that gradient descent get diverted to a local minima along the way.





(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.**

Solution: For the sake of this solution, it is important to distinguish between the process of generating the D_{i1} and the optimization process for a fixed set of D_{i1}.

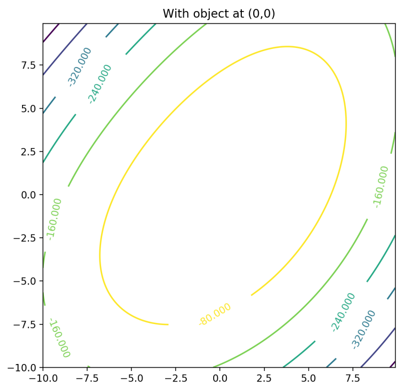
First, think about how incorrect local minima might arise from the data generation process. The first way that local minima are created is by noise in the neighborhood of the global minimum. This creates local minima that are near the true object location, but their distance from the true object location will grow with the noise σ. The second way local minima are created is by bad geometry, meaning that there is not a unique global minimum (because the sensors are not located in a way that can disambiguate certain points). Decreasing noise helps with the first case by making the objective value at the true object location deeper, while it does not help in the second case. To be more concrete, imagine the extreme case where the noise σ is very small (practically 0). The true object location is a global minimum. The only other minima are points which are almost exactly D_{i1} from each of the i; the geometric ambiguities.

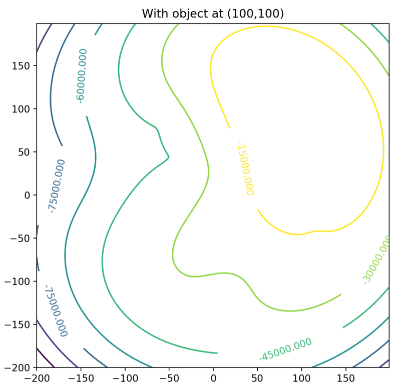
The argument above explains how having smaller noise in data generation can create an optimization landscape with fewer local minima in the neighborhood of the global minimum. The argument below will show that once the optimization landscape is fixed, the choice of σ only scales our gradient descent algorithm.

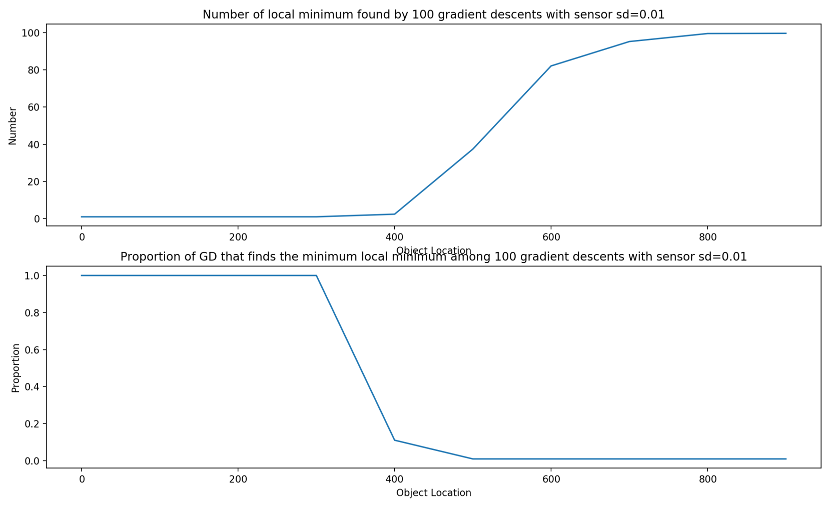
Consider the optimization process once the D_{i1} are fixed. Reducing the variance of measurement to $\sigma^2 < 1$ will scale the gradient at every point by $\frac{1}{\sigma^2}$ (check this by re-writing the log likelihood function). We can counteract this by making our step size significantly smaller. The result is that these two changes cancel each other out and we end up taking the same set of steps as before if we start at the same initialization.

Overall, we do not see significant changes in our results in practice. If you did see an effect, hopefully GD improved with smaller noise σ. If you saw erratic behavior when you changed σ^2, it is possible that you did not change the step size and thus your gradient was huge and your GD diverged.

See figures **??** and **??**.





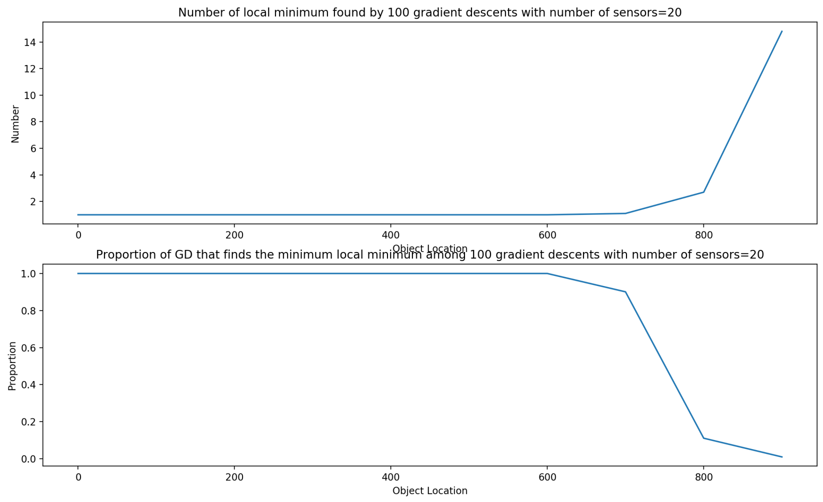


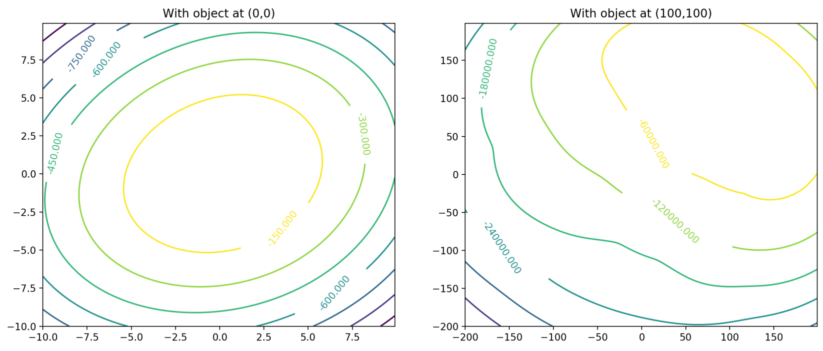
(e) Repeat the previous part again, except explore what happens as you increase the number of sensors. **Comment with appropriate plots justifying your comments.**

Solution: Intuitively, more sensors lead to more measurements from different locations. This can help reduce the negative effects of measurement errors and locate the object.

Let's be a little more precise and explore what happens to local minima. When we increase the number of sensors, the log likelihood will change to include more information about the location of the object. Imagine that we examine the global minimum and a local minimum before and after adding a new sensor. When we add the new sensor, the local minimum will be pulled upwards more than the global minimum. That is because the global minimum already roughly "agrees" with the data from the new sensor (that is, adds a small penalty to the likelihood) while the local minimum might "disagree" (that is, adds a strong penalty to the likelihood). Ultimately, this addition of sensors will eliminate some of the local minima and so we expect gradient descent to work better. See figures **??** and **??**.

Consider a concrete example where more sensors can help determine the location of the object better geometrically. Imagine the case where we only have two sensors at $(-1,0)$ and $(1,0)$ and each report a distance of $\sim \sqrt{2}$ from the object. Then we cannot decide whether an object is around $(0,1)$ or around $(0,-1)$. A new sensor, preferably one which is far from the x-axis, will likely resolve this issue.





(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.**

Solution: See the solution code. This problem is closely tied to the idea of parameter estimation, where we estimate the parameters of a model (in this case the sensors) and then use those parameters to predict more outputs of the model in an iterative fashion.

To solve this problem, we need to make a few key observations. First, the likelihood is symmetric. If we want to get the location of 7 unknown sensors from 50 known objects, it is the same as getting the location of 7 unknown objects from 50 known sensors. For that reason, learning the location of the 7 sensors from the 50 training objects is a problem we have already solved. Second, we notice that when the sensors are in a fixed location, finding the location of 50 objects can be broken down into finding the location of each of the 50 objects one at a time. More formally, we could say the likelihood function is separable.

The MSE reported are composed of two parts. First, there is the error that results from the difference between the estimated parameters and the true sensor locations. Second, there is the error that results from the noise of distance measurements. The first part can decrease when more training data is added. The second part cannot.

Since we trained the model with object locations around the origin, testing on the objects with similar location results in a smaller error (case 1) than the objects that are located further away.

The MSE for Case 1 is 1323.6

The MSE for Case 2 is 9247.6

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.