Gradient Descent Mini-Quiz

TA: Raguvir Kunani **DS100 Spring 2020**

The questions are meant to increase in difficulty, with a challenge question at the end. The challenge question is at least as hard, if not harder, than a hard exam question.

1. **What is the purpose of gradient descent (i.e. what goal does it accomplish for us that is relevant to the modeling process)?**

> **Solution:** Any time we model data, we need to specify a loss function that expresses how "off" a model's predictions are from the true values. The best model, then, is the model that minimizes the loss function, since that model will give the predictions that are the least "off". Gradient descent is an algorithm that can find the minimum of a (convex) function, so we use gradient descent to find the model that minimizes the loss function.

2. **Recall the gradient descent update rule:**

$$\theta^{(t+1)} \leftarrow \theta^{(t)} - \rho \left(\nabla L(\theta) \big|_{\theta=\theta^{(t)}} \right)$$

   (a) **Could gradient descent still find a minimum if the minus sign was changed to a plus sign (no other changes)?**

> **Solution:** No. The gradient of the loss function indicates the direction of steepest *ascent* (i.e. the direction in which the loss function increases the fastest). Since we are trying to *minimize* the loss function, we want to take steps in the opposite direction, the direction of steepest *descent*. Thus, we must multiply the gradient term by -1 in the update rule.

   (b) **Could gradient descent still find a minimum if the ρ term was removed (no other changes)?**

> **Solution:** Yes. The ρ term is called the stepsize or learning rate. Since it multiplies the gradient term, we can use ρ to control how big our steps are towards the minimum of the loss function. But gradient descent still takes steps towards the minimum without the ρ term, so it *could* still find a minimum.

   (c) **When does gradient descent stop?** *Hint:* See when $\theta^{(t+1)} = \theta^{(t)}$ in the update rule.

> **Solution:** Gradient descent stops when the update rule doesn't actually update θ. If there is no update to θ, then $\theta^{(t+1)} = \theta^{(t)}$. When $\nabla L(\theta)\big|_{\theta=\theta^{(t)}} = 0$, then $\theta^{(t+1)} = \theta^{(t)} - \rho \times 0 = \theta^{(t)} - 0 = \theta^{(t)}$. This means gradient descent stops when the gradient term is 0. This makes sense because if the gradient is 0, then we must be at the minimum of the loss function!

3. **Are there any functions for which gradient descent is not guaranteed to find the global minimum? If so, give an example and explain.**

Solution: Yes. One example is $f(x) = x^4 + 3x^3$. Gradient descent is only guaranteed to find the minimum of convex functions. To see why, note that any local minimum of a convex function is also a global minimum. Since gradient descent stops when it reaches a local minimum (where the gradient is 0), it is possible that if a function has multiple local minima, gradient descent can stop at any of them. But in a convex function any local minimum is a global minimum, so gradient descent is guaranteed to find the global minimum.

4. **Which values of ρ are guaranteed to produce incorrect results for gradient descent?** Assume the initial guess $\theta^{(0)}$ is not the minimum. Select all that apply.

   A. 1

   B. $-\frac{1}{2}$

   C. 0

   D. 100

   E. -1

> **Solution:** Any negative value of ρ will cause gradient descent to go in the opposite direction of what we want. $\rho = 0$ prevents the update rule from updating θ.

5. **We often replace the ρ term in the gradient descent update rule with a $\rho(t)$ term. This allows each iteration of gradient descent to have its own value of ρ. Give an example of one appropriate selection of $\rho(t)$ and explain why your choice is appropriate.**

> **Solution:** Choose $\rho(t) = \frac{1}{t}$. This choice results in $\rho(t)$ decreasing each iteration. As a result, such choices of $\rho(t)$ are called *decaying* learning rates. This choice is appropriate because each step of gradient descent gets closer to the minimum, so after many iterations we should be taking smaller and smaller steps so as to not overshoot the minimum. Any decreasing function of t is an appropriate choice for $\rho(t)$.

6. **Challenge Question**

   Recall the gradient descent update rule:

$$\theta^{(t+1)} \leftarrow \theta^{(t)} - \rho \left(\nabla L(\theta) \big|_{\theta=\theta^{(t)}} \right)$$

   For this question, **assume that $L(\theta)$ is MSE (mean squared error) and the model is $f_\theta(x)$.**

   (a) **Rewrite the gradient descent update rule replacing the gradient with a summation of terms, one for each of n data points. The final form should look like: $\theta^{(t+1)} \leftarrow \theta^{(t)} - \rho \frac{1}{n} \sum_{i=1}^{n} g(x_i, y_i, \theta^{(t)})$, where you define the g function.**

> **Solution:**
>
> $$\begin{aligned} \nabla L(\theta) \big|_{\theta=\theta^{(t)}} &= \nabla \left[\frac{1}{n} \sum_{i=1}^{n} (y_i - f_{\theta^{(t)}}(x_i))^2 \right] \\ &= \frac{1}{n} \sum_{i=1}^{n} \nabla \left[(y_i - f_{\theta^{(t)}}(x_i))^2 \right] \\ &= \frac{1}{n} \sum_{i=1}^{n} 2(y_i - f_{\theta^{(t)}}(x_i)) \times -\nabla_\theta \left[f_{\theta^{(t)}}(x_i) \right] \end{aligned}$$
>
> where $f_{\theta^{(t)}}(x)$ is the model being considered at time t. The notation ∇_θ is to emphasize we are taking the gradient with respect to θ.

Then we can rewrite the gradient descent update equation as:

$$\theta^{(t+1)} \leftarrow \theta^{(t)} - \rho \frac{1}{n} \sum_{i=1}^{n} 2(y_i - f_{\theta^{(t)}}(x_i)) \times -\nabla_\theta \left[f_{\theta^{(t)}}(x_i) \right]$$

$$\leftarrow \theta^{(t)} - \rho \frac{1}{n} \sum_{i=1}^{n} g(x_i, y_i, \theta^{(t)})$$

where $g(x_i, y_i, \theta^{(t)}) = 2(y_i - f_{\theta^{(t)}}(x_i)) \times -\nabla_\theta \left[f_{\theta^{(t)}}(x_i) \right]$

(b) **Gradient descent requires the computation of _____ n _____ gradients during each iteration.**
Hint: Use the result from the previous problem.

(c) (**Hard**) Since our datasets are large, we want to avoid computing so many gradients each iteration. Let's see if we can take advantage of our old friend, random sampling. **Show that the average gradient of the loss function evaluated at B randomly chosen data points is an unbiased estimator of the average gradient of the loss function evaluated at all n data points.** Let $l(x_i, y_i, \theta)$ denote the loss of the model $f_\theta(x)$ on a single data point (x_i, y_i) and assume $B < n$. ***Hint:*** Z **is an unbiased estimator of** Y **if** $E[Z] = E[Y]$**.**

Solution: Let L_n denote the loss on all n data points and L_B denote the loss on B randomly chosen points. With the given notation:

$$L_n = \sum_{i=1}^{n} l(x_i, y_i, \theta)$$

$$L_B = \sum_{i=1}^{B} l(x_i, y_i, \theta)$$

Then the average gradient on all n data points is $\frac{1}{n}\nabla_\theta L_n$ and the average gradient on B points is $\frac{1}{B}\nabla_\theta L_B$.

If $E\left[\frac{1}{B}\nabla_\theta L_B\right] = E\left[\frac{1}{n}\nabla_\theta L_n\right]$, then $\frac{1}{B}\nabla_\theta L_B$ is an unbiased estimator of $\frac{1}{n}\nabla_\theta L_n$. Note that the expectation is over the data points (x_i, y_i).

$$\begin{aligned} E\left[\frac{1}{n}\nabla_\theta L_n\right] &= \frac{1}{n}\nabla_\theta L_n \\ &= \overline{\nabla_\theta} \end{aligned}$$

where $\overline{\nabla_\theta}$ is the average gradient of the loss on all n data points.

$$\begin{aligned} E\left[\frac{1}{B}\nabla_\theta L_B\right] &= \frac{1}{B}E\left[\nabla_\theta L_B\right] \\ &= \frac{1}{B}E\left[\nabla_\theta \sum_{i=1}^{B} l(x_i, y_i, \theta)\right] \\ &= \frac{1}{B}E\left[\sum_{i=1}^{B} \nabla_\theta l(x_i, y_i, \theta)\right] \\ &= \frac{1}{B}\sum_{i=1}^{B} E\left[\nabla_\theta l(x_i, y_i, \theta)\right] \\ &= \frac{1}{B}\sum_{i=1}^{B} \overline{\nabla_\theta} \\ &= \frac{1}{B} \times B\overline{\nabla_\theta} \\ &= \overline{\nabla_\theta} \end{aligned}$$

The difficult part of this question is seeing that $E\left[\nabla_\theta l(x_i, y_i, \theta)\right]$ is just $\overline{\nabla_\theta}$ because the expected gradient of the loss at any random point is just the average gradient, since all points are equally likely to be chosen.

(d) This is great news! We can compute B gradients at each iteration and still find the minimum. **Update the gradient descent update rule you wrote in part (a) to reflect the change that we are only computing B gradients at each iteration.**

> **Solution:** The average gradient is now an average over a random subset of B points, rather than the average over all n points.
>
> $$\theta^{(t+1)} \leftarrow \theta^{(t)} - \rho \frac{1}{B} \sum_{i=1}^{B} g(x_i, y_i, \theta^{(t)})$$

(e) **For a fixed B, how does the performance of the updated gradient descent (which computes only B gradients per iteration) relate to the original gradient descent? Select all that apply.**

A. On average, finds a minimum in less iterations than original gradient descent
B. On average, finds a minimum in more iterations than original gradient descent
C. Will find the same minimum as original gradient descent
D. On average, finds a minimum in less time than original gradient descent
E. On average, finds a minimum in more time than original gradient

> **Solution:** Our updated gradient descent essentially takes random steps that are *on average* in the direction of the steepest descent, but not always in that direction. Thus, it will take more steps (iterations) to reach a minimum. But each step it takes requires less computation, so it takes less time to reach a minimum. If a function has 2 minima, then our updated gradient descent could randomly converge to a different minima than the original gradient descent.

(f) **How does increasing B affect the performance of the updated version of gradient descent? Select all that apply.**

A. Finds a minimum in less iterations
B. Finds a minimum in less time
C. Finds a minimum in more iterations
D. Finds a minimum in more time

> **Solution:** Increasing B means computing more gradients at each iteration. This makes our steps, on average, go closer to the direction of steepest descent, so we will take fewer steps (iterations) to reach a minimum. Each step, though, will require the computation of more gradients, so it will take a longer time to reach a minimum.

(g) **The updated gradient descent is also known as __stochastic__ gradient descent.**