Gradient Descent Workshop CS 480

Write your answers below. You do not need to share your code for this workshop.

1. Show that for a feature vector $x = [x_1, x_2, x_3]$, a weight vector $w = [w_1, w_2, w_3]$ and a constant y, the least squares loss function $L(w) = (w \cdot x - y)^2$ has gradient $\nabla L(w) = 2(w \cdot x - y)x$. Clearly explain your answer with words and/or formulas.

High Bridge half-marathon revisited

We are going to revisit the problem of finding a least squares regression model, this time using gradient descent. Use the following code to load the data from the 2018 High Bridge half-marathon.

```
import numpy as np
import pandas as pd

df = pd.read_excel("https://people.hsc.edu/faculty-staff/blins/StatsExamples/halfmarathon.xlsx")
genders = list(df.Gender)
ages = list(df.Age)
minutes = list(df.Minutes)
y = np.array(minutes)
X = np.array([[1.0,ages[i],1 if genders[i] == 'F' else 0] for i in range(len(genders))])
```

2. Compute the gradient of the sum of squares loss function at $w = [0, 0, 0]$ for the data above. That is, use Python to calculate the gradient vector for each term corresponding to `X[i]` and `y[i]` using the formula from exercise 1, then add them up all up. What do you get?

3. The following code implements gradient descent for a sum of squared error loss function.

```
w = np.array([0,0,0])
eta = 10**(-6)
n = 100
for _ in range(n):
    gradient = sum(2*((X[i] @ w) - y[i])*X[i] for i in range(len(y)))
    w = w - eta*gradient
print(w)
```

Try adjusting the values of `eta` and `n`. Can you get the algorithm to converge? The least squares solution should be about $w = [84.25, 0.97, 21.0]$. What values of eta and n got you the closest to the correct value of w?

Stochastic gradient descent

As you can see, gradient descent can be very slow. Stochastic gradient descent is one strategy to speed it up. Instead of adding up the gradient for each term in the sum of squared error, you just pick one random term and calculate the gradient at that one feature vector and corresponding value of y.

4. Replace the line

```
gradient = sum(2*((X[i] @ w) - y[i])*X[i] for i in range(len(y)))
```

with a command that randomly selects one i and then computes the gradient at just that one feature vector `X[i]` and corresponding `y[i]` instead of summing over all i. You can use the function `random.randrange(len(y))` from the `random` library to select i. With this new algorithm, can you get values of w to converge? What `eta` and `n` did you use?