Assignment #10

Due: 6:00pm Friday 9 December 2022

Upload at: `https://www.gradescope.com/courses/438130/assignments/2475394`

Assignments in COS 302 should be done individually. See the course syllabus for the collaboration policy.

Remember to append your Colab PDF as explained in the first homework, with all outputs visible.
When you print to PDF it may be helpful to scale at 95% or so to get everything on the page.

Problem 1 (35pts)

In gradient descent, we attempt to minimize some function $f(\boldsymbol{x})$ by starting with some initial $\boldsymbol{x}_0$ and then iteratively modifying the parameters $\boldsymbol{x} \in \mathbb{R}^n$ according to the following formula:

$$\boldsymbol{x}_{t+1} \leftarrow \boldsymbol{x}_t - \lambda(\nabla_{\boldsymbol{x}} f(\boldsymbol{x}_t))^T$$

for some small $\lambda \geq 0$ known as the *learning rate* or *step size*. This procedure modifies $\boldsymbol{x}$ so as to move in a direction proportional to the negative gradient. This basic procedure is the basis for many optimization algorithms across science and engineering.

Consider the simple function $f(\boldsymbol{x}) = \boldsymbol{x}^T A\boldsymbol{x}$ for a (constant) symmetric positive definite matrix $A \in \mathbb{R}^{n\times n}$.

(A) Implement a function `f(A, x)` that takes as input an $n \times n$ numpy array `A` and a 1D array `x` of length n and returns the output for $f(\boldsymbol{x})$ above. Test it on the values

$$\boldsymbol{x} = \begin{bmatrix} -1 \\ 2 \end{bmatrix} \qquad\qquad \boldsymbol{A} = \begin{bmatrix} 3 & -1 \\ -1 & 2 \end{bmatrix}.$$

(B) Implement a function `grad_f(A, x)` that takes the same two arguments as above but returns $\nabla_{\boldsymbol{x}} f(\boldsymbol{x})$ evaluated at `x`. Test it on the values $\boldsymbol{x}$ and $\boldsymbol{A}$ above.

(C) Now implement a third and final function `grad_descent(A, x0, lr, num_iters)` that takes the additional arguments `lr`, representing the learning rate λ above, and `num_iters` indicating the total number of iterations of gradient descent to perform. The function should loop and print the values of $\boldsymbol{x}_t$ and $f(\boldsymbol{x}_t)$ at each iteration of gradient descent.

(D) Use your function to perform gradient descent on f with

$$\boldsymbol{x}_0 = \begin{bmatrix} 10 \\ 10 \end{bmatrix} \qquad\qquad \boldsymbol{A} = \begin{bmatrix} 1 & 0 \\ 0 & 4 \end{bmatrix}.$$

Run gradient descent for 50 iterations with learning rates of 1, 0.25, 0.1, and 0.01. What do you notice? Does $\boldsymbol{x}_t$ always converge to the same value? Does our gradient descent algorithm work every time?

Problem 2 (35pts)

Debugging gradients is one of the important skills for machine learning. One powerful debugging tool is to use *finite differences* to estimate the gradient of a function numerically. The idea of finite differences is simple: estimate the rise versus run of the function using a small step size. This is kind of like when you take the limit in the definition of the derivative, except you just don't take the limit quite to zero. If $f : \mathbb{R}^d \to \mathbb{R}$ and $\epsilon > 0$, then

$$\frac{\partial}{\partial x_i} f(\boldsymbol{x}) \approx \frac{1}{2\epsilon}(f(x_1, \ldots, x_i + \epsilon, \ldots, x_d) - f(x_1, \ldots, x_i - \epsilon, \ldots, x_d)\,.$$

This is a "two-sided" finite differences estimate because it is moving up and down relative to $\boldsymbol{x}$. In a typical application, you might code this up using $\epsilon \approx 10^{-4}$ or so. To estimate a gradient, you would loop from $i = 1 \ldots d$ and evaluate all the partials.

(A) Implement a function `finite_diff(func, x, epsilon)` that takes a function `func` as an argument (which itself is of the form `func(x)` taking a vector and returning a scalar) and estimates the gradient via finite differences as above. This will involve a loop over the entries of `x` that modifies the entries and evaluates the function twice. The function `finite_diff` should return a vector that is the same size as `x`.

(B) Consider the function $f(\boldsymbol{x}) = (\boldsymbol{c} - \boldsymbol{A}\boldsymbol{x})^{\mathsf{T}}(\boldsymbol{c} - \boldsymbol{A}\boldsymbol{x})$ for $\boldsymbol{x} \in \mathbb{R}^5$ where

$$\boldsymbol{c} = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix} \qquad \boldsymbol{A} = \begin{bmatrix} 1 & 2 & 3 & 4 & 5 \\ 6 & 7 & 8 & 9 & 10 \\ 11 & 12 & 13 & 14 & 15 \end{bmatrix}$$

Implement this function and its gradient in Python as a function of `x`. The gradient function will require that you use the differentiation identities as in Sec 5.5 of the book, just like you did in HW9; just now you'll also write it in code.

(C) Choose some non-zero vector for `x` and compare the result from your implementation to what you get from the finite differences estimate. They should be similar but not exactly the same. (If you get weird values that are all integers, make sure you're using floating point numbers in your numpy arrays.)

(D) Now try automatic differentiation. Rather than `import numpy as np` do something like this:

```
import autograd.numpy as np
from autograd import grad
```

Then get a third estimate of the gradient by calling `grad` on your function from part (B) and evaluating that at `x`.

Problem 3 (28pts) (A) Use `autograd` as in the previous problem to write a general-purpose function that takes an update step for Newton's method of minimization. The signature for this function should look like `newton_step(func, x)`. You'll need to use `grad` to compute first and second derivatives within the function.

(B) Consider the function $f(x) = \sin(x)/x$, which is sometimes called the "sinc" function. Try minimizing this function starting from several different points: $x_0 = 3.0$, $x_0 = -4.0$, and $x_0 = 0.5$. You'll need to write a loop that iterates your `newton_step` function. You should not need to take many steps in the loop (not more than ten). Explain what happens for the different initializations.

Problem 4 (2pts)
Approximately how many hours did this assignment take you to complete?

My notebook URL: `https://colab.research.google.com/XXXXXXXXXXXXXXXXXXXXXXX`

Changelog

- 1 December 2022 – Initial version.