Due: 21 Dec 2020

The code for this problem can be found at this Github link.

Suppose we want to approximate the function $g(x) = 1 - \cos(x)$ on the interval $[0, \pi/2]$ with the function $\text{ReLU}(ax - b)$ where a and b are to be determined. We take 6 training points $x_j = \pi j/10, j = 0, 1, 2, 3, 4, 5$ and set up the following loss function:

$$f(a, b) = \frac{1}{12} \sum_{j=0}^{5} [\text{ReLU}(ax_j - b) - g(x_j)]^2.$$

1. **Stationary points.** The set of stationary points of f (i.e., the set of points where $\nabla f = 0$) consists of the global minimizer and a flat region. Describe this set analytically using equalities and inequalities and show it in a figure. Provide an analytic formula for the global minimizer of f. What is the global minimum of f?

Solution. To begin, we compute the partial derivatives of f.

$$\frac{\partial f}{\partial a} = \frac{1}{6} \sum_{j=0}^{5} x_j [\text{ReLU}(ax_j - b) - g(x_j)] \text{ReLU}'(ax_j - b)$$

$$\frac{\partial f}{\partial b} = -\frac{1}{6} \sum_{j=0}^{5} [\text{ReLU}(ax_j - b) - g(x_j)] \text{ReLU}'(ax_j - b)$$

where the derivative of the ReLU is defined as

$$\text{ReLU}'(x) = \begin{cases} 1 & x > 0 \\ 0 & x < 0. \end{cases}$$

In order to guarantee that the partials are equal to 0, we look for values of a and b that make the terms in the interior equal to 0. First, we study the ReLU' factor and notice that

$$\text{ReLU}'(ax_j - b) = 0 \iff ax_j - b < 0$$

so $b > ax_j$ for all j in order to zero out every term in both sums. Since $x_0 = 0$, we know $b > 0$. When $a > 0$, we need $b > \max_j ax_j = \frac{\pi}{2} a$ and when $a < 0$, we need $b > \min_j ax_j = 0$ since the inequality flips when $a < 0$. Naturally, when $a = 0$ we simply need $b > 0$ (as $\text{ReLU}(-b) = 0$ for $b > 0$), which squares up with the above characterization. This flat region can be seen in Figure 1 below.

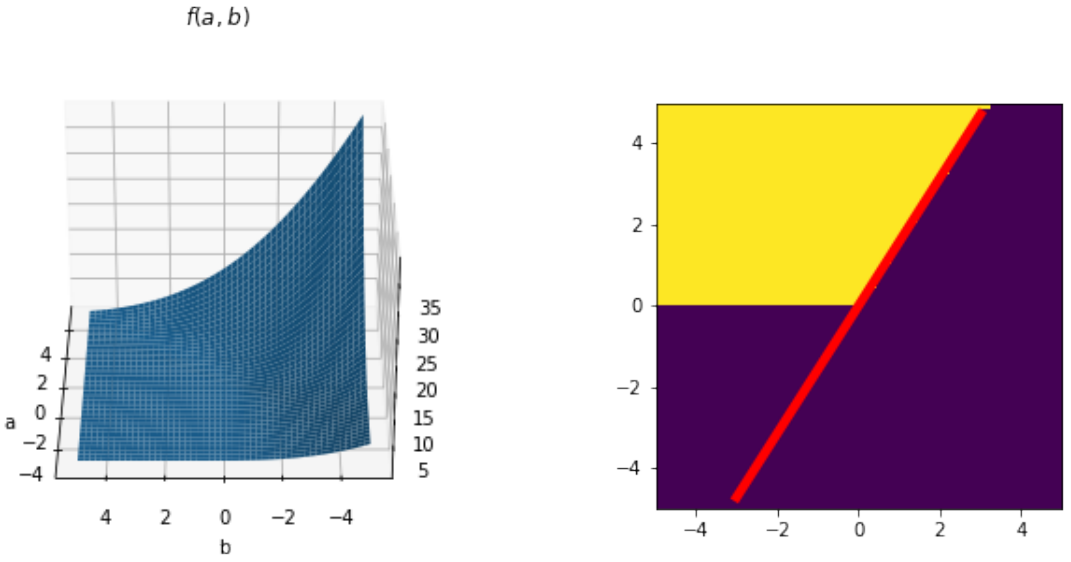

Figure 1: Left: surface of $f(a,b)$ near the origin. Right: flat region $\nabla f = 0$. The yellow region is the flat region, the purple region is the rest of the function, and the red line is the boundary $b = \frac{\pi}{2}a$.

To find the minimizer, we need to consider other parts of the terms in the sum—finding where the ReLU$'$ factors are zero only gives us information about the flat region. We examine the $\text{ReLU}(ax_j - b) - g(x_j)$ factor and think about where it zeros out:

$$0 = \text{ReLU}\left(\frac{\pi j}{10}a - b\right) - 1 + \cos\left(\frac{\pi j}{10}\right) = \max\left(0, \frac{\pi j}{10}a - b\right) - 1 + \cos\left(\frac{\pi j}{10}\right)$$
$$\implies \max\left(0, \frac{\pi j}{10}a - b\right) = 1 - \cos\left(\frac{\pi j}{10}\right) \ \ \forall j = 0,1,2,3,4,5.$$

For $j = 0$, we get the equation $\max(0, -b) = 0$ which implies that $b > 0$. For all the other j, we can drop the max on the left-hand side because the right-hand side is greater than 0, so we know the left-hand side will need to similarly be positive. Plugging in the rest of the j gives the following system of equations:

$$\frac{\pi}{10}a - b = 1 - \cos\left(\frac{\pi}{10}\right)$$
$$\frac{\pi}{5}a - b = 1 - \cos\left(\frac{\pi}{5}\right)$$
$$\frac{3\pi}{10}a - b = 1 - \cos\left(\frac{3\pi}{10}\right)$$
$$\frac{2\pi}{5}a - b = 1 - \cos\left(\frac{2\pi}{5}\right)$$
$$\frac{\pi}{2}a - b = 1 - \cos\left(\frac{\pi}{2}\right)$$

which can be written as the matrix equation $Cx = d$ where

$$C = \begin{bmatrix} \pi/10 & -1 \\ \pi/5 & -1 \\ 3\pi/10 & -1 \\ 2\pi/5 & -1 \\ \pi/2 & -1 \end{bmatrix}, x = \begin{bmatrix} a \\ b \end{bmatrix}, \text{ and } d = \begin{bmatrix} 1 - \cos(\pi/10) \\ 1 - \cos(\pi/5) \\ 1 - \cos(3\pi/10) \\ 1 - \cos(2\pi/5) \\ 1 - \cos(\pi/2) \end{bmatrix}.$$

So we could say that the minimizer x^* is the solution of this system $x^* = C^{-1}d$. However, this system is overdetermined and is not guaranteed to have a nice, unique solution. This is because we are not taking into account the ReLU$'$ terms, which will turn on and off certain rows of the left-hand side based on whether or not the argument is negative, something which will be important in the minimizer. To incorporate this, we multiply both sides by the vector $y(a,b)$:

$$y(a,b) = \begin{bmatrix} \text{ReLU}'(\pi a/10 - b) \\ \text{ReLU}'(\pi a/5 - b) \\ \text{ReLU}'(3\pi a/10 - b) \\ \text{ReLU}'(2\pi a/5 - b) \\ \text{ReLU}'(\pi a/2 - b) \end{bmatrix}$$

which turns this into the nonlinear system $(y \odot C)x = y \odot d$. The minimizer will then be the solution $x^* = (y \odot C)^{-1}(y \odot d)$, which can be computed as $x^* = [0.86128902, 0.37349155]$ by solving this system (see code for details). Here, $\odot$ denotes the entry-wise (Hadamard product).

2. **Gradient descent.** Take $a = 1$ and $b = 0$ as the initial guess for gradient descent with constant stepsize. What is the minimal stepsize α^* such that the iterates end up in the flat region? Suppose we take $\alpha = 0.99\alpha^*$ and run gradient descent. Will the iterates approach the global minimizer? Either way, explain why. Propose a stepsize trying to make it as large as possible such that the iterates will necessarily converge to the global minimizer and give a rationale for your choice.

Solution. The gradient descent iteration is $x_{k+1} = x_k - \alpha \nabla f(x_k)$. Here, $x = [a, b]^\intercal \in \mathbb{R}^2$. See Figure 2 below for an illustration of the first step of gradient descent when run from $x_0 = [1, 0]$. The minimal step size α^* will be the step size that causes us to step over the blue line $b = \frac{\pi}{2}a$ and remain in the flat region for all future iterations. The slope of this orange line is

$$m = \frac{[\nabla f(1,0)]_2 - 0}{1 + [\nabla f(1,0)]_1 - 1} = \frac{[\nabla f(1,0)]_2}{[\nabla f(1,0)]_1}$$

where $[\cdot]_i$ indicates the ith component of that vector. Hence the orange line has equation $b = m(a-1)$ and we seek its intersection with the line $b = \frac{\pi}{2}a$ by solving the system:

$$\begin{cases} ma - b = m \\ \frac{\pi}{2}a - b = 0 \end{cases}$$

which gives the intersection point $(\hat{a}, \hat{b}) = (0.37958407, 0.59624926)$ and thus the distance from the intersection to the initial point is $d = \sqrt{(\hat{a}-1)^2 + \hat{b}^2} = 0.8604819$. Therefore, the minimal step size α^* such that we end up in the flat region is the value such that $\|\nabla f(1,0)\|\alpha = d$, implying that $\alpha^* = d/\|\nabla f(1,0)\| = 1.50996024102$.

If we take $\alpha = 0.99\alpha^*$ and run gradient descent, the iterates will approach the global minimizer but slowly and never quite get there. They do not make it out of the neighborhood of the minimizer, but rather alternate between sides of this neighborhood as the gradient very slowly decays. All of my numerical tests using $\alpha = 0.99\alpha^*$ resulted in the method taking around 10^6 iterations to converge and even then it never exactly hit the minimizer since it relied on the gradient being small enough to carry it there, which didn't happen.

As for the largest possible step size, it must be small enough that the iterates will reach the minimizer at the very end of the iteration. I ran experiments for a number of values of α and examined if they hit the flat region or not. If the gradient descent iteration hit 1000 iterations (far more than any of the convergent experiments), then I stopped it early and recorded that value of α as hitting the flat region. Figure 3 shows this plot, where the y axis is 1 if the cutoff was reached and 0 otherwise. In this sweep, the largest value of α that does not hit the cutoff is $\alpha = 1.30476051596$, so it is my answer for the largest possible step size where the iterates still converge to the global minimizer.

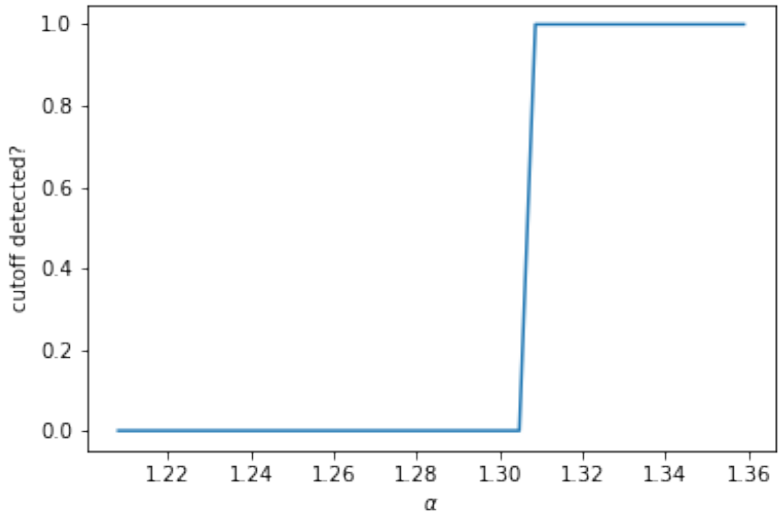

Figure 3: Stepsizes and if they hit the flat region for $\alpha = p\alpha^*$ for 40 values of p between 0.8 and 0.9.

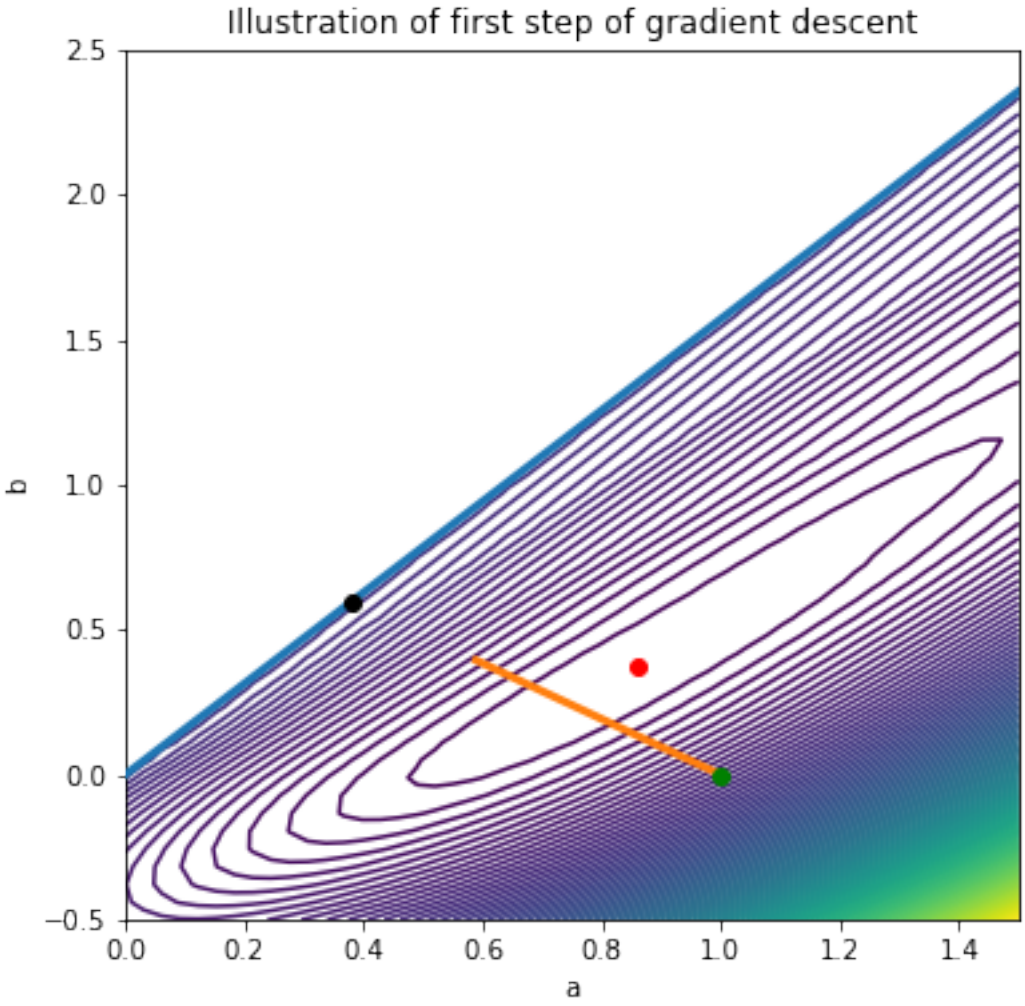

Figure 2: Illustration of the first step of gradient descent. Green dot is the initial point $x_0 = [1, 0]$, red dot is the global minimizer x^*, the black dot is the intersection between the gradient line and the blue line, the blue line is the line $b = \frac{\pi}{2}a$, the orange line is the vector $\nabla f(1, 0)$, and the contours are the contours of the function $f(a, b)$.

3. **Stochastic gradient descent.** As above, take $a = 1$ and $b = 0$ as the initial guess. Use a simple stochastic gradient descent with a single training point chosen randomly for approximating the gradient of f at each step. Find a strategy for stepsize reduction such that the stochastic gradient descent will converge to the global minimizer.

Solution. See `prob1.ipynb` for my stochastic gradient descent code. I tested the following schedules:

- Reciprocal: $\alpha_k = L/k$
- Time-based: $\alpha_k = \frac{\alpha_{k-1}}{1+rk}$ where r controls the speed of the decay
- Exponential $\alpha_k = Le^{-rk}$ where r controls the speed of the decay

In all of these, L is a constant initial learning rate . These all satisfy $\sum_{k=0}^{\infty} \alpha_k = \infty$, $\sum_{k=0}^{\infty} \alpha_k^2 < \infty$. I also experimented with a constant learning rate fixed at L, which does not satisfy such constraints on the reduction schedule. In addition, all of the schedules have the ability to be stepped with parameter M. This replaces each k with $\lfloor k/M \rfloor$ when computing α_k, adding a step function to delay the decay. I recorded the gradient norms over the iterations for

all the schedules and computed the error between the solution found by the optimization and the global minimizer. I extensively tested different hyperparameters to manually tune them and ended up fairly happy with the results. Finally, I wrote a version of this neural network using PyTorch to validate my answers.

My code for this can be found near the end of `prob1.ipynb` in the cell labeled "My training routine". To experiment with these hyperparameters, change the parameters in the `SGD` call and then run the cell below it to view the gradient norms over the iteration. I tested learning rates from 0.1 to 2, r values from 0.1 to 0.95, and M from 1 (no drop) to 100. Increasing M generally led to better performance, allowing the algorithm to fully explore the virtually flat region around the minimizer at each scale before shrinking the step size.

One set of hyperparameters that seemed to work pretty well for all the schedules was `maxiter` $= 500$, $L = 1.6$, $r = 0.1$, $M = 40$ (Figure 3). Adding the drop ($M > 1$) to the schedules smoothed out the decay and allowed the iterations to stabilize at each step size level before proceeding. This combination is the worst for the time-based schedule, which gets close to the minimizer in the first coordinate but does not get near it in the second coordinate, and the best for the exponential schedule, which converges to the minimizer. Therefore, my strategy for stepsize reduction and hyperparameter combination is this combination along with the exponential reduction strategy utilizing drops.

The constant step size schedule (no step size reduction at all) surprisingly worked very well in the widest variety of hyperparameter combinations. It routinely had the least error and would find the minimizer in regimes where none of the other reduction strategies would. This flexibility was highly unexpected when I started out on this problem—I think it comes as a result of the stochasticity that the constant step size gets the iteration to the global minimum even for values of α that do not work for the batch gradient descent.

Lastly, the PyTorch implementation does not always find the minimum, although I did not implement any step size reduction strategies for it. It does require a significantly smaller learning rate than my homemade version, so there are definitely implementation differences in what that library uses.

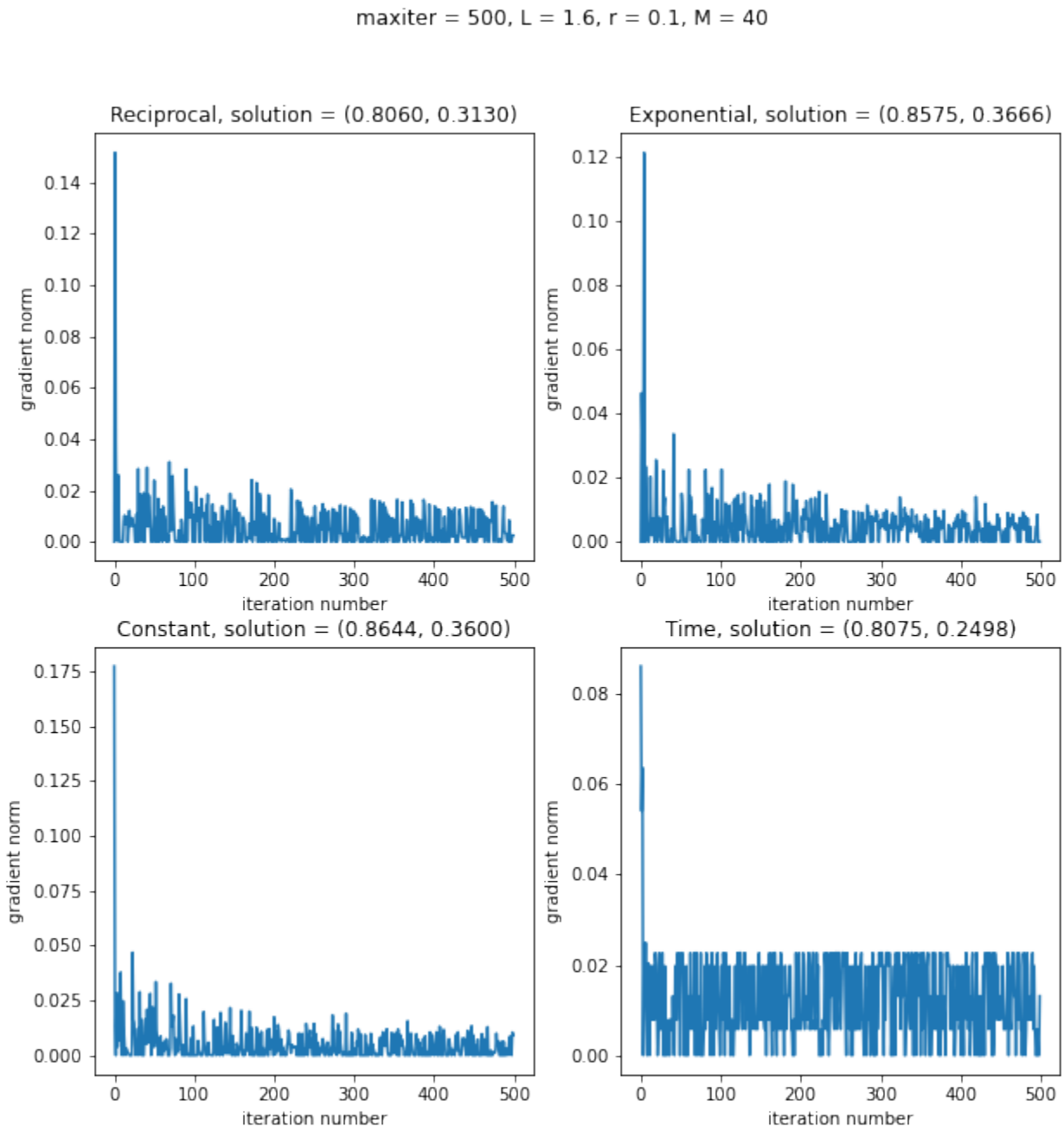

Figure 3: Sample gradient norms for the given hyperparameter combination.