Instructions

1. We will be using Gradescope to collect your assignments. Please read the following instructions for submitting to Gradescope carefully!
   - Each problem/sub-problem should be on one or more pages. Anyone who violates this policy will receive an additional penalty on the grade.
   - When submitting to Gradescope, make sure to mark which page corresponds to each problem/sub-problem.
   - Note: This is a large class and Gradescope’s assignment segmentation features are essential. Failure to follow these instructions may result in parts of your assignment not being graded. We will not entertain regrading requests for failure to follow instructions. Please read https://stats200.stanford.edu/gradescope_tips.pdf for additional information on submitting to Gradescope.
   - This portion (PS1) counts 8% of your total grade

2. \LaTeX'd solutions are strongly encouraged (solution template available at cc.gatech.edu/classes/AY2019/cs7643_fall/assets/sol1.tex), but scanned handwritten copies are acceptable. Hard copies are \textbf{not} accepted.

3. We generally encourage you to collaborate with other students.
   You may talk to a friend, discuss the questions and potential directions for solving them. However, you need to write your own solutions and code separately, and \textit{not} as a group activity. Please list the students you collaborated with.

1 \textbf{Gradient Descent}

1. (3 points) We often use iterative optimization algorithms such as Gradient Descent to find $\mathbf{w}$ that minimizes a loss function $f(\mathbf{w})$. Recall that in gradient descent, we start with an initial
value of $w$ (say $w^{(1)}$) and iteratively take a step in the direction of the negative of the gradient of the objective function $i.e.$

$$w^{(t+1)} = w^{(t)} - \eta \nabla f(w^{(t)})$$

for learning rate $\eta > 0$.

In this question, we will develop a slightly deeper understanding of this update rule, in particular for minimizing a convex function $f(w)$. Note: this analysis will not directly carry over to training neural networks since loss functions for training neural networks are typically not convex, but this will (a) develop intuition and (b) provide a starting point for research in non-convex optimization (which is beyond the scope of this class).

Recall the first-order Taylor approximation of $f$ at $w^{(t)}$:

$$f(w) \approx f(w^{(t)}) + \langle w - w^{(t)}, \nabla f(w^{(t)}) \rangle$$

When $f$ is convex, this approximation forms a lower bound of $f$, $i.e.$

$$f(w) \geq f(w^{(t)}) + \langle w - w^{(t)}, \nabla f(w^{(t)}) \rangle \quad \forall w$$

affine lower bound to $f(\cdot)$

(3)

Since this approximation is a ‘simpler’ function than $f(\cdot)$, we could consider minimizing the approximation instead of $f(\cdot)$. Two immediate problems: (1) the approximation is affine (thus unbounded from below) and (2) the approximation is faithful for $w$ close to $w^{(t)}$. To solve both problems, we add a squared $\ell_2$ proximity term to the approximation minimization:

$$\argmin_w f(w^{(t)}) + \langle w - w^{(t)}, \nabla f(w^{(t)}) \rangle + \frac{\lambda}{2} \|w - w^{(t)}\|^2$$

affine lower bound to $f(\cdot)$

trade-off proximity term

(4)

Notice that the optimization problem above is an unconstrained quadratic programming problem, meaning that it can be solved in closed form (hint: gradients).

What is the solution $w^*$ of the above optimization? What does that tell you about the gradient descent update rule? What is the relationship between $\lambda$ and $\eta$?

2. (2 points) Let’s assume we have a training set $X$. It has only one single instance $x_0$, repeated 150 times. In 120 of the 150 cases, the single output value is 1; in the other 30, it is 0. What will a backpropagation network predict for this example, assuming it has been trained and reaches a global optimal? Please verify your answer. (Hint: to find the global optimal, differentiate the error function and set it to zero.)

3. (2 points) Consider a objective function comprised of $N = 2$ terms:

$$f(w) = \frac{1}{2}(w - 2)^2 + \frac{1}{2}(w + 1)^2$$

Now consider using SGD (with a batch-size $B = 1$) to minimize this objective. Specifically, in each iteration, we will pick one of the two terms (uniformly at random), and take a step in the direction of the negative gradient, with a constant step-size of $\eta$. You can assume $\eta$ is small enough that every update does result in improvement (aka descent) on the sampled term.

Is SGD guaranteed to decrease the overall loss function in every iteration? If yes, provide a proof. If no, provide a counter-example.
2 Automatic Differentiation

4. (4 points) In practice, writing the closed-form expression of the derivative of a loss function $f$ w.r.t. the parameters of a deep neural network is hard (and mostly unnecessary) as $f$ becomes complex. Instead, we define computation graphs and use the automatic differentiation algorithms (typically backpropagation) to compute gradients using the chain rule. For example, consider the expression

$$f(x, y) = (x + y)(y + 1)$$  \hspace{1cm} (6)

Let’s define intermediate variables $a$ and $b$ such that

$$a = x + y$$  \hspace{1cm} (7)
$$b = y + 1$$  \hspace{1cm} (8)
$$f = a \times b$$  \hspace{1cm} (9)

A computation graph for the “forward pass” through $f$ is shown in Fig. 1.

![Figure 1](image)

We can then work backwards and compute the derivative of $f$ w.r.t. each intermediate variable ($\frac{\partial f}{\partial a}$, $\frac{\partial f}{\partial b}$) and chain them together to get $\frac{\partial f}{\partial x}$ and $\frac{\partial f}{\partial y}$.

Let $\sigma(\cdot)$ denote the standard sigmoid function. Now, for the following vector function:

$$f_1(w_1, w_2) = e^{w_1 + w_2} + \sin(e^{w_1} + e^{w_2})$$  \hspace{1cm} (10)
$$f_2(w_1, w_2) = w_1 w_2 + \sigma(w_1)$$  \hspace{1cm} (11)

(a) Draw the computation graph. Compute the value of $f$ at $\bar{w} = (2, 1)$.

(b) At this $\bar{w}$, compute the Jacobian $\frac{\partial \bar{f}}{\partial \bar{w}}$ using numerical differentiation (using $\Delta w = 0.01$).

(c) At this $\bar{w}$, compute the Jacobian using forward mode auto-differentiation.

(d) At this $\bar{w}$, compute the Jacobian using backward mode auto-differentiation.

(e) Don’t you love that software exists to do this for us?