CS7643: Deep Learning
Fall 2017
Problem Set 1

Instructor: Dhruv Batra
TAs: Michael Cogswell, Abhishek Das, Zhaoyang Lv
Discussions: http://piazza.com/gatech/fall2017/cs7643

Due: Friday, Sep 8, 11:55pm

Instructions

1. Please upload your answer sheet on Canvas with the following format: FirstName_LastName_PSx.pdf.
   LaTeX'd solutions are preferred (solution template available at cc.gatech.edu/classes/AY2018/cs7643_fall/assets/sol0.tex), but scanned handwritten copies are acceptable. Hard copies are not accepted.

2. 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, your discussions should not involve one person doing all the thinking and simply telling others the solution. We expect you to be mature enough to make that judgement call yourself. Finally, you need to write your own solutions separately, and *not* as a group activity. Please list the students you collaborated with.

1 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 $\mathbf{w}$ (say $\mathbf{w}^{(1)}$) and iteratively take a step in the direction of the negative of the gradient of the objective function *i.e.*

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

   for learning rate $\eta > 0$.

   In this question, we will develop a slightly deeper understanding of this update rule. Recall the first-order Taylor approximation of f at $\mathbf{w}^{(t)}$:

$$f(\mathbf{w}) \approx f(\mathbf{w}^{(t)}) + \langle \mathbf{w} - \mathbf{w}^{(t)}, \nabla f(\mathbf{w}^{(t)}) \rangle \tag{2}$$

   When f is convex, this approximation forms a lower bound of f. 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 $\mathbf{w}$ close to $\mathbf{w}^{(t)}$. To solve both problems, we add a squared ℓ_2 *proximity term* to the approximation minimization:

$$\operatorname*{argmin}_{\mathbf{w}} \underbrace{f(\mathbf{w}^{(t)}) + \langle \mathbf{w} - \mathbf{w}^{(t)}, \nabla f(\mathbf{w}^{(t)}) \rangle}_{\text{affine lower bound to } f(\cdot)} + \underbrace{\frac{\lambda}{2}}_{\text{trade-off}} \underbrace{\left\| \mathbf{w} - \mathbf{w}^{(t)} \right\|^2}_{\text{proximity term}} \tag{3}$$

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

What is the solution $\mathbf{w}^*$ of the above optimization? What does that tell you about the gradient descent update rule? What is the relationship between λ and η?

2. (3 points) Show that for a sequence of vectors $\mathbf{v}_1, \mathbf{v}_2, ..., \mathbf{v}_T$ and $\mathbf{w}^\star$ that minimizes $f(\mathbf{w})$, an update equation of the form $\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} - \eta \mathbf{v}_t$ with $\mathbf{w}^{(1)} = 0$ satisfies

$$\sum_{t=1}^{T} \langle \mathbf{w}^{(t)} - \mathbf{w}^\star, \mathbf{v}_t \rangle \leq \frac{\|\mathbf{w}^\star\|^2}{2\eta} + \frac{\eta}{2} \sum_{t=1}^{T} \|\mathbf{v}_t\|^2 \tag{4}$$

3. (3 points) Let's now analyze the convergence rate of gradient descent *i.e.* how fast it converges to $\mathbf{w}^\star$. Show that for $\bar{\mathbf{w}} = \frac{1}{T} \sum_{t=1}^{T} \mathbf{w}^{(t)}$

$$f(\bar{\mathbf{w}}) - f(\mathbf{w}^\star) \leq \frac{1}{T} \sum_{t=1}^{T} \langle \mathbf{w}^{(t)} - \mathbf{w}^\star, \nabla f(\mathbf{w}^{(t)}) \rangle \tag{5}$$

Further, use the result from part 2, with upper bounds B and ρ for $\|\mathbf{w}^\star\|$ and $\left\|\nabla f(\mathbf{w}^{(t)})\right\|$ respectively and show that for fixed $\eta = \sqrt{\frac{B^2}{\rho^2 T}}$, the convergence rate of gradient descent is $\mathcal{O}(1/\sqrt{T})$ *i.e.* the upper bound for $f(\bar{\mathbf{w}}) - f(\mathbf{w}^\star) \propto \frac{1}{\sqrt{T}}$.

4. (2 points) Consider a objective function comprised of $N = 2$ terms:

$$f(w) = \frac{1}{2}(w-2)^2 + \frac{1}{2}(w+1)^2 \tag{6}$$

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 η. You can assume η 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

5. (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

$$f(x, y) = (x + y)(y + 1) \tag{7}$$

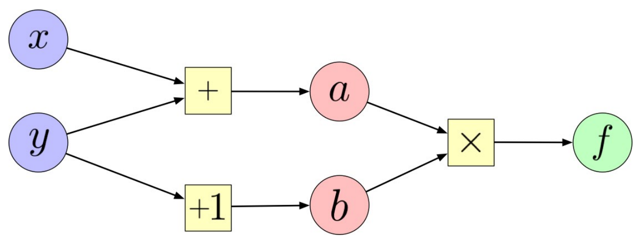
Let's define intermediate variables a and b such that

$$a = x + y \tag{8}$$

$$b = y + 1 \tag{9}$$

$$f = a \times b \tag{10}$$

A computation graph for the "forward pass" through f looks like the following



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:

$$f_1(w_1, w_2) = e^{e^{w_1} + e^{2w_2}} + \sin(e^{w_1} + e^{2w_2}) \tag{11}$$

$$f_2(w_1, w_2) = w_1 w_2 + \sigma(w_1) \tag{12}$$

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

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

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

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

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