CS7643: Deep Learning
Fall 2019
Homework 1

Instructor: Dhruv Batra
TAs: Harsh Agrawal, Neha Jain, Ashwin Kalyan, Harish Kamath,
Anishi Mehta, Nirbhay Modhe, Michael Piseno, Viraj Prabhu, Sarah Wiegreffe
Discussions: `https://piazza.com/gatech/fall2019/cs48037643`

Due: Thursday, September 26, 11:55pm

Instructions

1. We will be using Gradescope to collect your assignments. Please read the following instructions for submitting to Gradescope carefully!
   - Each subproblem must be submitted on a separate page. When submitting to Gradescope, make sure to mark which page(s) corresponds to each problem/sub-problem. For instance, Q5 has 5 subproblems, and the solution to each must start on a new page. Similarly, Q8 has 8 subproblems, and the writeup for each should start on a new page.
   - For the coding problems (Q8), please use the provided `collect_submission.sh` script and upload `hw1.zip` to the HW1 Code assignment on Gradescope. While we will not be explicitly grading your code, you are still required to submit it. Please make sure you have saved the most recent version of your jupyter notebook before running this script. Further, append the writeup for each Q8 subproblem to your solution PDF.
   - 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.
2. LaTeX'd solutions are strongly encouraged (solution template available at cc.gatech.edu/classes/AY2020/cs7643_fall/assets/sol1.tex), but scanned handwritten copies are acceptable. Hard copies are **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 *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, in particular for minimizing a convex function $f(\mathbf{w})$. 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 $\mathbf{w}^{(t)}$:

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

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

$$f(\mathbf{w}) \geq \underbrace{f(\mathbf{w}^{(t)}) + \langle \mathbf{w} - \mathbf{w}^{(t)}, \nabla f(\mathbf{w}^{(t)}) \rangle}_{\text{affine lower bound to } f(\cdot)} \quad \forall \mathbf{w} \tag{3}$$

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 $\mathbf{w}$ close to $\mathbf{w}^{(t)}$. To solve both problems, we add a squared ℓ_2 *proximity term* to the approximation minimization:

$$\operatorname*{argmin}_{\mathbf{w}} \underbrace{f(\mathbf{w}^{(t)}) + \langle \mathbf{w} - \mathbf{w}^{(t)}, \nabla f(\mathbf{w}^{(t)}) \rangle}_{\text{affine lower bound to } f(\cdot)} + \underbrace{\frac{\lambda}{2}}_{\text{trade-off}} \underbrace{\left\| \mathbf{w} - \mathbf{w}^{(t)} \right\|^2}_{\text{proximity term}} \tag{4}$$

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 $\mathbf{w}^*$ 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) Let's prove a lemma that will initially seem devoid of the rest of the analysis but will come in handy in the next sub-question when we start combining things. Specifically, the analysis in this sub-question holds for any $\mathbf{w}^\star$, but in the next sub-question we will use it for $\mathbf{w}^\star$ that minimizes $f(\mathbf{w})$.

   Consider a sequence of vectors $\mathbf{v}_1, \mathbf{v}_2, ..., \mathbf{v}_T$, and an update equation of the form $\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} - \eta \mathbf{v}_t$ with $\mathbf{w}^{(1)} = 0$. Show that:

$$\sum_{t=1}^{T} \langle \mathbf{w}^{(t)} - \mathbf{w}^\star, \mathbf{v}_t \rangle \leq \frac{\|\mathbf{w}^\star\|^2}{2\eta} + \frac{\eta}{2} \sum_{t=1}^{T} \|\mathbf{v}_t\|^2 \tag{5}$$

3. (3 points) Now let's start putting things together and analyze the convergence rate of gradient descent *i.e.* how fast it converges to $\mathbf{w}^\star$.

   First, show that for $\bar{\mathbf{w}} = \frac{1}{T} \sum_{t=1}^{T} \mathbf{w}^{(t)}$

$$f(\bar{\mathbf{w}}) - f(\mathbf{w}^\star) \leq \frac{1}{T}\sum_{t=1}^{T}\langle \mathbf{w}^{(t)} - \mathbf{w}^\star, \nabla f(\mathbf{w}^{(t)})\rangle \tag{6}$$

Next, 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:

$$f(w) = \frac{1}{2}(w-2)^2 + \frac{1}{2}(w+1)^2 \tag{7}$$

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

$$f(x, y) = (x + y)(y + 1) \tag{8}$$

Let's define intermediate variables a and b such that

$$a = x + y \tag{9}$$
$$b = y + 1 \tag{10}$$
$$f = a \times b \tag{11}$$

A computation graph for the "forward pass" through f is shown in Fig. 1.

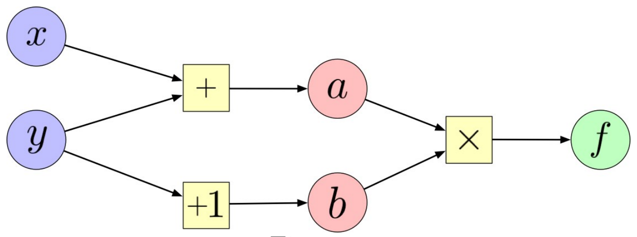

Figure 1

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:

$$f_1(w_1, w_2) = e^{e^{w_1}+e^{2w_2}} + \sin(e^{w_1} + e^{2w_2}) \tag{12}$$
$$f_2(w_1, w_2) = w_1 w_2 + \sigma(w_1) \tag{13}$$

(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?

3 Directed Acyclic Graphs (DAG)

One important property for feed-forward network that we have discussed in class is that it must be a directed acyclic graph (DAG). Recall that a *DAG is a directed graph that contains no directed cycles*. We will study some of its properties in this question.
Define $G = (V, E)$ in which V is the set of all nodes as $\{v_1, v_2, ..., v_i, ...v_n\}$ and E is the set of edges $E = \{e_{i,j} = (v_i, v_j) \mid v_i, v_j \in V\}$.
A *topological order of a directed graph* $G = (V, E)$ is an ordering of its nodes as $\{v_1, v_2, ..., v_i, ...v_n\}$ so that for every edge (v_i, v_j) we have $i < j$.
There are several lemmas can be inferred from the definition of DAG. One lemma is: if G is a DAG, then G has a node with no incoming edges.
Given the above lemma, prove the following two lemmas:

6. **[2 points]** If the graph G is a DAG, then G has a topological ordering.

7. **[2 points]** If the graph G has a topological order, then G is a DAG.

4 Implement and train a network on CIFAR-10

Setup Instructions: Before attempting this question, look at setup instructions here.

8. (Upto 29 points) Now, we will learn how to implement a softmax classifier, vanilla neural networks (or Multi-Layer Perceptrons), and ConvNets. You will begin by writing the forward and backward passes for different types of layers (including convolution and pooling), and then go on to train a shallow ConvNet on the CIFAR-10 dataset in Python. Next you will learn to use PyTorch, a popular open-source deep learning framework, and use it to replicate the experiments from before.

   Follow the instructions provided here.