CS7643: Deep Learning
Spring 2020
Problem Set 1

Instructor: Zsolt Kira
TAs: Yihao Chen, Sameer Dharur, Rahul Duggal, Patrick Grady, Harish Kamath
Yinquan Lu, Anishi Mehta, Manas Sahni, Jiachen Yang, Zhuoran Yu
Discussions: `https://piazza.com/gatech/spring2020/cs4803dl7643a/home`

Due: Tuesday, February 11, 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 PS1 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:

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

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?
Answer: Take the derivative of w:

$$f'(w) = \nabla f(\mathbf{w}^{(t)} + \lambda(\mathbf{w} - \mathbf{w}^{(t)}) = 0 \tag{5}$$

$$\mathbf{w}^* = \mathbf{w}^{(t)} - \frac{1}{\lambda} \nabla f(\mathbf{w}^{(t)}) \tag{6}$$

$$\eta = \frac{1}{\lambda} \tag{7}$$

Gradient decent can minimize the approximation function at each step. The value of λ means the penalty of proximity term. When λ increases, the learning rate will decrease, which means the step along GD can be small, and vice versa.

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:

$$\sum_{t=1}^{T} \langle \mathbf{w}^{(t)} - \mathbf{w}^\star, \mathbf{v}_t \rangle \leq \frac{\|\mathbf{w}^\star\|^2}{2\eta} + \frac{\eta}{2} \sum_{t=1}^{T} \|\mathbf{v}_t\|^2 \tag{8}$$

Answer:

$$\begin{aligned}
\langle \mathbf{w}^{(t)} - \mathbf{w}^\star, \mathbf{v}_t \rangle &= \frac{1}{\eta} \langle \mathbf{w}^{(t)} - \mathbf{w}^\star, \eta \mathbf{v}_t \rangle & (9) \\
&= \left\| \mathbf{w}^{(t)} - \mathbf{w}^\star \right\|^2 + \|\eta \mathbf{v}_t\|^2 + \left\| \mathbf{w}^{(t)} - \mathbf{w}^\star - \eta \mathbf{v}_t \right\|^2 & (10) \\
&= \frac{1}{4\eta} (\left\| \mathbf{w}^{(t)} - \mathbf{w}^\star + \eta \mathbf{v}_t \right\|^2 - \left\| \mathbf{w}^{(t)} - \mathbf{w}^\star - \eta \mathbf{v}_t \right\|^2) & (11) \\
&= \frac{1}{2\eta} (\left\| \mathbf{w}^{(t)} - \mathbf{w}^\star \right\|^2 - \left\| \mathbf{w}^{(t+1)} - \mathbf{w}^\star \right\|^2) + \frac{\eta}{2} \|\mathbf{v}_t\|^2 & (12)
\end{aligned}$$

$$\sum_{t=1}^{T} \langle \mathbf{w}^{(t)} - \mathbf{w}^\star, \mathbf{v}_t \rangle = \sum_{t=1}^{T} \frac{1}{2\eta} (\left\| \mathbf{w}^{(t)} - \mathbf{w}^\star \right\|^2 - \left\| \mathbf{w}^{(t+1)} - \mathbf{w}^\star \right\|^2) + \frac{\eta}{2} \sum_{t=1}^{T} \|\mathbf{v}_t\|^2 \tag{13}$$

we can find:

$$\sum_{t=1}^{T} \frac{1}{2\eta} (\left\| \mathbf{w}^{(t)} - \mathbf{w}^\star \right\|^2 - \left\| \mathbf{w}^{(t+1)} - \mathbf{w}^\star \right\|^2) = \frac{1}{2\eta} (\|\mathbf{w}^\star\|^2 - \left\| \mathbf{w}^{(t+1)} - \mathbf{w}^\star \right\|^2) \leq \frac{\|\mathbf{w}^\star\|^2}{2\eta} \tag{14}$$

Thus we can prove:

$$\sum_{t=1}^{T} \langle \mathbf{w}^{(t)} - \mathbf{w}^\star, \mathbf{v}_t \rangle \leq \frac{\|\mathbf{w}^\star\|^2}{2\eta} + \frac{\eta}{2} \sum_{t=1}^{T} \|\mathbf{v}_t\|^2 \tag{15}$$

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 $\bar{\mathbf{w}} = \frac{1}{T} \sum_{t=1}^{T} \mathbf{w}^{(t)}$

   $$f(\bar{\mathbf{w}}) - f(\mathbf{w}^\star) \leq \frac{1}{T} \sum_{t=1}^{T} \langle \mathbf{w}^{(t)} - \mathbf{w}^\star, \nabla f(\mathbf{w}^{(t)}) \rangle \tag{16}$$

   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 $f(\bar{\mathbf{w}}) - f(\mathbf{w}^\star) \propto \frac{1}{\sqrt{T}}$.

   Answer:

$$f(\bar{\mathbf{w}}) - f(\mathbf{w}^\star) = f(\frac{1}{T}\sum_{t=1}^{T}\mathbf{w}^{(t)}) - f(\mathbf{w}^\star) \tag{17}$$

$$\leq \frac{1}{T}f(\sum_{t=1}^{T}\mathbf{w}^{(t)}) - f(\mathbf{w}^\star) \tag{18}$$

Because of the convexity in question 1, we have:

$$f(\sum_{t=1}^{T}\mathbf{w}^{(t)}) - f(\mathbf{w}^\star) \leq \langle \mathbf{w}^{(t)} - \mathbf{w}^\star, \nabla f(\mathbf{w}^{(t)})\rangle \tag{19}$$

Thus we can prove:

$$f(\bar{\mathbf{w}}) - f(\mathbf{w}^\star) \leq \frac{1}{T}\sum_{t=1}^{T}\langle \mathbf{w}^{(t)} - \mathbf{w}^\star, \nabla f(\mathbf{w}^{(t)})\rangle \tag{20}$$

Using the conclusion from part2, we can get:

$$f(\bar{\mathbf{w}}) - f(\mathbf{w}^\star) \leq \frac{1}{T}(\frac{\|\mathbf{w}^\star\|^2}{2\eta} + \frac{\eta}{2}\sum_{t=1}^{T}\left\|\nabla f(\mathbf{w}^{(t)}\right\|^2) \tag{21}$$

$$\leq \frac{1}{T}(\frac{B^2}{2\eta} + \frac{\eta}{2}Tp) \tag{22}$$

$$= \frac{B^2}{2\eta T} + \frac{\eta p}{2} \tag{23}$$

$$= \frac{1}{\sqrt{T}}\frac{BP+P}{2} \tag{24}$$

Thus the convergence rate of gradient descent is $\mathcal{O}(1/\sqrt{T})$

4. (2 points) Consider an objective function $f(w) := f_1(w) + f_2(w)$ comprised of $N = 2$ terms:

$$f_1(w) = -\ln\left(1 - \frac{1}{1+\exp(-w)}\right) \quad \text{and} \quad f_2(w) = -\ln\left(\frac{1}{1+\exp(-w)}\right) \tag{25}$$

Now consider using SGD (with a batch-size $B = 1$) to minimize $f(w)$. 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.

Answer: No, SGD does not guarantee to decrease the objective function at every iteration. For example, let $w^{(0)} = 0$, $w^{(1)} = w^{(0)} - \eta(1 - \frac{1}{(2\mathbf{e}^{-\mathbf{w}_0}(1+)\mathbf{e}^{-\mathbf{w}_0})^3}) = -\frac{\eta}{2}$. Since η is positive, f(w(1)) $>$ f(w(0))

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

$$f(x, y) = (x + y)(y + 1) \tag{26}$$

Let's define intermediate variables a and b such that

$$a = x + y \tag{27}$$

$$b = y + 1 \tag{28}$$

$$f = a \times b \tag{29}$$

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

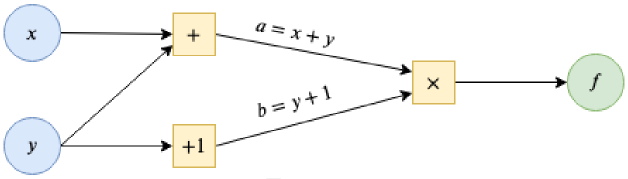

Figure 1

We can then work backwards and compute the derivative of f w.r.t. each intermediate variable ($\frac{\partial f}{\partial a}$ and $\frac{\partial f}{\partial b}$) 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:

$$f_1(w_1, w_2) = e^{e^{w_1} + e^{2w_2}} + \sigma(e^{w_1} + e^{2w_2}) \tag{30}$$

$$f_2(w_1, w_2) = w_1 w_2 + \max(w_1, w_2) \tag{31}$$

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

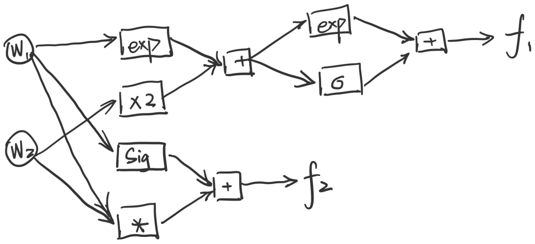

Figure 2

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

$$\begin{bmatrix} \frac{\partial f_1}{\partial w_1} & \frac{\partial f_1}{\partial w_2} \\ \frac{\partial f_2}{\partial w_1} & \frac{\partial f_2}{\partial w_2} \end{bmatrix} = \begin{bmatrix} \frac{f_1(w_1+0.01,w_2)-f_1(w_1,w_2)}{0.01} & \frac{f_2(w_1,w_2+0.01)-f_2(w_1,w_2)}{0.01} \\ \frac{f_2(w_1+0.01,w_2)-f_1(w_1,w_2)}{0.01} & \frac{f_2(w_1,w_2+0.01)-f_2(w_1,w_2)}{0.01} \end{bmatrix} = \begin{bmatrix} 48.19 & 4.76 \\ 0 & 1 \end{bmatrix}$$

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

$$\begin{aligned}
y_1 &= e^{w_1} && (32) \\
y_2 &= e^{w_1} + e^{2w_2} && (33) \\
y_3 &= e^{y_2} && (34) \\
y_4 &= \sigma(y_3) && (35) \\
\frac{\partial f_1}{\partial w_1} &= \frac{\partial y_5}{\partial w_1} && (36) \\
&= \frac{\partial y_3}{\partial w_1} + \frac{\partial y_4}{\partial w_1} && (37) \\
&= 47.28739 && (38)
\end{aligned}$$

Similarly,

$$\begin{aligned}
\frac{\partial f_1}{\partial w_2} &= 4.710 && (39) \\
\frac{\partial f_2}{\partial w_1} &= 1 && (40) \\
\frac{\partial f_2}{\partial w_2} &= 0 && (41)
\end{aligned}$$

$$\begin{bmatrix} 47.28739 & 4.71 \\ 0 & 1 \end{bmatrix}$$

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

$$\begin{aligned}
\frac{\partial f_1}{\partial w_1} &= \vec{a_1}\vec{b_1} && (42) \\
&= \vec{a_1}\frac{exp(a)}{1+exp(a)^2} + \vec{d_2}exp(c_2) && (43) \\
&= 47.28739 && (44)
\end{aligned}$$

Similarly,

$$\begin{bmatrix} 47.28739 & 4.71 \\ 0 & 1 \end{bmatrix}$$

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

3 Paper Review

The first of our paper reviews for this course comes from a much acclaimed spotlight presentation at NeurIPS 2019 on the topic 'Weight Agnostic Neural Networks' by Adam Gaier and David Ha from Google Brain.
The paper presents a very interesting proposition that, through a series of experiments, re-examines some fundamental notions about neural networks - in particular, the comparative importance of architectures and weights in a network's predictive performance.
The paper can be viewed here. The authors have also written a blog post with intuitive visualizations to help understand its key concepts better.
Guidelines: Please restrict your reviews to no more than 350 words. The evaluation rubric for this section is as follows :

6. (2 points) What is the main contribution of this paper? Briefly summarize its key insights, strengths and weaknesses.
   Contributions:
   In this paper, author starts using deemphasizing weights to search neural network. They aim to search for weight agnostic neural networks, architectures with strong inductive biases that can already perform various tasks with random weights.
   First they created an initial population having minimal neural network topologies. Each rollout has shared weight values and can be used to evaluate the network.Then they can rank these networks. They repeat evaluation and eventually they can create a new population.
   They use 3 models to test, which is CartPoleSwingUp, BipedalWalker-v2 and CarRacing-v0. Researchers found the results were surprisingly good, as the WANN models with the best-performing shared weight values reached an upright pole position on the CartPoleSwingUp task after only after a few swings. Experiment results also proved that WANNs are no match for convolutional neural networks, which was an expected outcome.

7. (2 points) What is your personal takeaway from this paper? This could be expressed either in terms of relating the approaches adopted in this paper to your traditional understanding of learning parameterized models, or potential future directions of research in the area which the authors haven't addressed, or anything else that struck you as being noteworthy.
   Personal takeaway:

   This is a brand new method for searching neural network without using gradient descent. It's not like traditional neural network and it may lessen the computational resources. As with the age-old ânature versus nurtureâ debate, AI researchers want to know whether architecture or weights play the main role in the performance of neural networks. This paper definitely provide a promising start. For me, I think it is interesting but still need to bring it to actual practice to show its performance of the untrained neural network.

4 Implement and train a network on CIFAR-10

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