CS 4644/7643: Deep Learning
Summer 2025
Problem Set 2

Instructor: Zsolt Kira
TAs: Mili Das, Yipu Chen, Kausar Patherya
Discussions: https://piazza.com/class/mafsg3dobtu42c

Due: June 22, 2025, 11:59pm

Instructions

1. We will be using Gradescope to collect your assignments. Please read the following instructions for submitting to Gradescope carefully!

   - Please upload separate pdfs for **HW2 Theory** and **HW2 Code PDF** sections on Gradescope. The instructions for the latter are included in the coding section of the assignment. **When submitting to Gradescope, please make sure to mark the page(s) corresponding to each problem/sub-problem**.
   - For the **HW2 Code Part 1 and Part 2** components on Gradescope, please use the `collect_submission.ipynb` notebook provided and upload the resulting zip files to each Gradescope section respectively. Please make sure you have saved the most recent version of your code.
   - 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 in the zip file in HW2 under the Assignments tab on Canvas), 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 Collaborators [0.5 points]

Please list your collaborators and assign this list to the corresponding section of the outline on Gradescope. If you don't have any collaborators, please write 'None' and assign it to the corresponding section of the Gradescope submission regardless.

2 Activation Function

1. **[2 points]**

   In neural networks, activation functions introduce non-linearities, enabling the network to approximate complex functions. One of the desirable properties for an activation function is to be *zero-centered*. Being zero-centered helps in achieving faster convergence during training, as the weights can adjust in both positive and negative directions more efficiently. Additionally, zero-centered functions can help mitigate the vanishing gradient problem, ensuring that gradients during backpropagation do not diminish too quickly.

   Definition: A function $g(x)$ is said to be zero-centered if, for every value x in its domain where $g(x)$ is positive, there exists an equivalent negative value $-x$ such that:

   $$g(-x) = -g(x).$$

   In other words, the function is symmetric about the origin, producing positive outputs for positive inputs and negative outputs for negative inputs.

   Consider the following activation function, defined for all real x:

   $$g(x) \;=\; \frac{x}{1+|x|}.$$

   (a) Zero-Centered Property. Show that $g(x)$ is zero-centered by proving $g(-x) = -g(x)$ for all x. (Hint: carefully handle the absolute value.)

   (b) Derivative and Gradient Behavior

   i. Compute the derivative $g'(x)$ for $x \neq 0$.

   ii. Evaluate $g'(0)$ (Hint: the function is continuous).

   iii. Based on your results, show whether $g(x)$ might cause vanishing or exploding gradients for large values of $|x|$ (Hint: show the limits).

3 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:

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

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?

2. **[Extra Credit - 2 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. **[Extra Credit - 2 points]**

   Now let's start putting things together and analyze the convergence rate of gradient descent *i.e.* how fast it converges to $\mathbf{w}^\star$. For this question, assume that f is convex and ρ-Lipschitz. A function that is ρ-Lipschitz is one where the norm of its gradient is bounded by ρ, i.e. $\|\nabla f(\mathbf{w})\| \leq \rho$.

   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 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

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

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 is shown in Fig. 1.

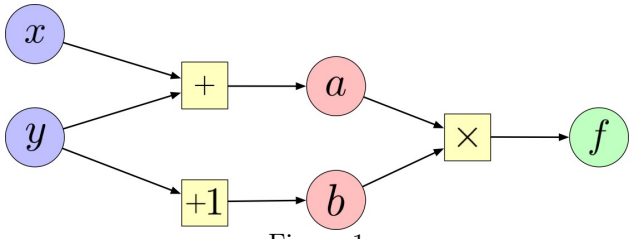

Figure 1

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:

$$f_1(w_1, w_2) = \cos(w_1)\cos(w_2) + \sigma(w_2) \tag{11}$$
$$f_2(w_1, w_2) = ln(w_1 + w_2) + w_1^2 w_2 \tag{12}$$

(a) Draw the computation graph (you might find draw.io useful for this). Compute the value of f at $\mathbf{w} = (1, 2)$.

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

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

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

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

5 Convolutions

5. **[5 points]**

We'll start to introduce the properties of convolutions here that serve as a foundation for many computer vision applications in deep learning. In class, we discussed convolutions. In this question, we will develop formal intuition around a slight modification of that idea – circular convolutions.

First, let's define a circular convolution of two n-dimensional vectors $\mathbf{x}$ and $\mathbf{w}$:

$$(\mathbf{x} * \mathbf{w})_i = \sum_{k=0}^{n-1} x_k w_{(i-k) \bmod n} \tag{13}$$

We can write the above equation as a matrix-vector multiplication.

Given an n-dimensional vector $\mathbf{a} = (a_0, ..., a_{n-1})$, we define the associated matrix $C_\mathbf{a}$ whose first column is made up of these numbers, and each subsequent column is obtained by a circular shift of the previous column.

$$C_\mathbf{a} = \begin{bmatrix} a_0 & a_{n-1} & a_{n-2} & \dots & a_1 \\ a_1 & a_0 & a_{n-1} & & a_2 \\ a_2 & a_1 & a_0 & & a_3 \\ \vdots & & \ddots & \ddots & \vdots \\ a_{n-1} & a_{n-2} & a_{n-3} & \dots & a_0 \end{bmatrix}$$

Such matrices are called *circulants*. Any convolution $\mathbf{x} * \mathbf{w}$ can be equivalently represented as a multiplication by the circulant matrix $C_\mathbf{a}\mathbf{x}$.

Note that a circulant matrix is a kind of Toeplitz matrix with the additional property that $a_i = a_{i+n}$. Next, let's introduce a special type of circulant called a shift matrix. A shift matrix is a circulant matrix where only one dimension of the vector $\mathbf{a}$ can be set to 1, *i.e.*, $\mathbf{a} = (0, 1, ..., 0)$. Let S be the circular right-shift operator, defined by the following action on vectors:

$$S\mathbf{x} = \begin{bmatrix} 0 & & & 1 \\ 1 & & & \\ & \ddots & \ddots & \\ & & 1 & 0 \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_{n-1} \end{bmatrix} = \begin{bmatrix} x_{n-1} \\ x_0 \\ \vdots \\ x_{n-2} \end{bmatrix}$$

Notice that after applying the shift-matrix, all the element of $\mathbf{x}$ have been shifted by 1.

(a) Prove that *any* circulant matrix is commutative with a shift matrix. Note that this directly implies that convolutions are commutative with shift operators.

This leads to very important property called translation or shift equivariance. A function is shift equivariant if $f(S\mathbf{x}) = Sf(\mathbf{x})$. Convolution's commutativity with shift implies that it does not matter whether we first shift a vector and then convolve it, or first convolve and then shift – the result will be the same. Notice that you just proved that circular convolutions are shift equivariant.

(b) Now prove that the a (circular) convolution is the *only* linear operation with shift equivariance. (Hint: how do you prove a bidirectional implication?)

(c) (Open-ended question) What does this tell you about designing deep learning architectures for processing spatial or spatio-temporal data like images and videos?

6 SGD

6. **[3 points]**

Consider an objective function comprised of $N = 2$ terms:

$$f(w) = \frac{1}{2}(w-2)^2 + \frac{1}{2}(w+1)^2 \tag{14}$$

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.

7 Implement and train a network on CIFAR-10

7. **[18 points + 2 Extra Credits]**

In homework 1, you learned how to implement a softmax classifier and vanilla neural networks. Now, you will learn how to implement ConvNets. You will begin by writing the forward and backward passes for 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.