HOMEWORK SET 1

CSCI5527 Deep Learning (Spring 2026)

Due 11:59 pm, Feb 21 2026

Instruction Your writeup, either typeset or scanned, should be a single PDF file. For problems requiring coding, organize all codes for each top-level problem (i.e., Problem 1, Problem 2, etc) into a separate Jupyter notebook file (i.e., `.ipynb` file). Your submission to Gradescope should include the single PDF and all notebook files—**please DO NOT zip them**! No late submission will be accepted. For each problem, you should acknowledge your collaborators—**including AI tools**, if any.

About the use of AI tools You are strongly encouraged to use AI tools—they are becoming our workspace friends. It takes a bit of practice to ask the right and effective questions/prompts to these tools; we highly recommend that you go through this popular free short course **ChatGPT Prompt Engineering for Developers** offered by https://learn.deeplearning.ai/ to get started.

If you use any AI tools for any of the problems, you should include screenshots of your prompting questions and their answers in your writeup. The answers provided by such AI tools often contain factual errors and reasoning gaps. **So, if you only submit an AI answer with such bugs for any problem, you will obtain a zero score for that problem**. You obtain the scores only when you explain the bugs and also correct them in your own writing. You can also choose not to use any of these AI tools, in which case we will grade based on the efforts you have made.

Notation Please refer to our supplementary notes on high-dimensional calculus

Important notes Please provide detailed steps with justification for all problems; jumping to the final results leads to a zero score. Also, **if we ask you to use certain facts/tools to obtain something, you have to use these facts/tools (perhaps plus others); otherwise, you get a zero score**, e.g., if you prove everything from scratch when we ask you to use an existing theorem.

Problem 1 (Neural networks can represent all Boolean functions; $5/15$) The standard perceptron is a single-layer, single-output neural network with the step function as the activation, i.e.,

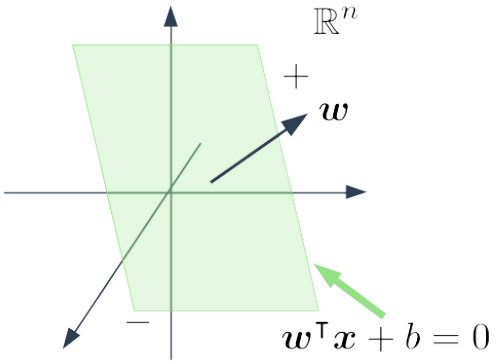
$$f(\boldsymbol{x}) = \text{step}(\boldsymbol{w}^\intercal \boldsymbol{x} + b),$$

where $\text{step}(z) = 1$ if $z \geq 0$ and 0 otherwise. Geometrically, f is a $\{0, 1\}$-valued function with the hyperplane $\{\boldsymbol{x} : \boldsymbol{w}^\intercal \boldsymbol{x} + b = 0\}$ as the separating boundary between the 0- and the 1-region; see Fig. 1 (left). Consider Boolean functions $\{0, 1\}^n \to \{0, 1\}$. We will work out how arbitrary Boolean functions can be represented by two-layer or deep neural networks.

(a) Consider $n = 1$ first. Show that the `NOT` function can be implemented using a single-input perceptron by setting the weight w and the bias b appropriately; **please write down the set of all possible pairs of** (w, b). $(0.5/15)$

(b) Now consider the case $n = 2$. Show that the two-input `AND`, `OR` functions can be implemented using a two-input perceptron; **please write down the set of all possible such** $(\boldsymbol{w}, b)$ **pairs**. Hint: the geometric view might help. For example, for the `AND` function, we are effectively trying to separate the point $(1, 1)$ from $(1, 0)$, $(0, 1)$ and $(0, 0)$. The hint applies to all subsequent subproblems of **Problem 1**. $(1/15)$

(c) Can we encode the `XOR` function (https://en.wikipedia.org/wiki/Exclusive_or) using a two-input perceptron? How if yes? Why if not? $(1/15)$



X_1	X_2	X_3	X_4	X_5	Y
0	0	1	1	0	1
0	1	0	1	1	1
0	1	1	0	0	1
1	0	0	0	1	1
1	0	1	1	1	1
1	1	0	0	1	1

Figure 1: (left) Geometric illustration of the perceptron. (right) An example truth table.

(d) For general $n \geq 2$, we consider general `AND` functions that take n inputs, where inputs are x_i's. A typical such function looks like $x_1 \cdot \overline{x_2} \cdot \overline{x_3} \cdot x_{n-1} \cdot x_n$. Show that all general n-input `AND` function can be implemented using an n-input perceptron. Similarly, show that all n-input general `OR` function can be implemented using an n-input perceptron. **Please write down the set of all possible such $(\boldsymbol{w}, b)$ pairs for both cases.** $(1/15)$

(e) Any Boolean function is fully specified by a list of all variable combinations that are evaluated to 1. Such a list is often tabulated, and the resulting table is called the truth table. For example, in the truth table of Fig. 1 (right), the Boolean function represented reads (i.e., "1" position we put the variable itself, and "0" position we put the vaiable negated, and each summand below corresponds to a row of the table)

$$\overline{x_1}\,\overline{x_2}x_3x_4\overline{x_5} + \overline{x_1}x_2\overline{x_3}x_4x_5 + \overline{x_1}x_2x_3\overline{x_4}\,\overline{x_5} + x_1\overline{x_2}\,\overline{x_3}\,\overline{x_4}\,x_5 + x_1\overline{x_2}x_3x_4x_5 + x_1x_2\overline{x_3}\,\overline{x_4}x_5,$$

where product $\cdot$ (which is omitted) means `AND` and summation $+$ means `OR`. In Boolean logic, this is called the disjunctive normal form (`https://en.wikipedia.org/wiki/Disjunctive_normal_form`). All Boolean functions can be represented in the disjunctive normal form. Based on these, show that we can construct a two-layer neural network that can represent any n-input Boolean functions. $(1/15)$

(f) How many neurons do we need for the hidden layer? Answering with big-O notation is fine. $(0.5/15)$

Problem 2 (Universal approximation property of absolute-value networks; $2/15$) Recall how we argued that two-layer neural networks with the sigmoid activation function (i.e., $\sigma(z) = \frac{1}{1+e^{-z}}$) can approximate any functions that map $\mathbb{R}$ to $\mathbb{R}$. We constructed the step function and then the bump function, and finally, we summed up the bumps to form the approximation. We also briefly discussed in class how to use two ReLU functions ($h(z) = \max(0, z)$) to approximate a step function.

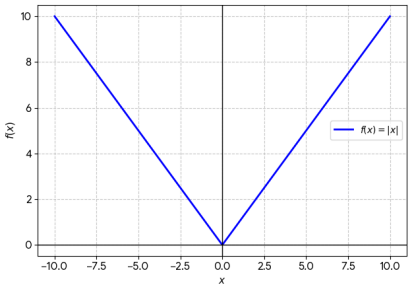


Figure 2: Absolute-value function

(a) Obviously, we can choose the activation σ directly as a step function or even as a bump function. This not only makes the approximation much more accurate, but also simplifies the neural network we need. Is this good for computation? And why? $(1/15)$

(b) Sketch the main steps of using absolute-value networks, i.e., all activations are the absolute-value function, for approximating an arbitrary function that maps from $\mathbb{R}$ to $\mathbb{R}$, i.e., absolute-value function $\rightarrow$ step function $\rightarrow$ bump function $\rightarrow$ approximating the original function. $(1/15)$

Problem 3 (Optimality conditions; $4/15$)

(a) Derive the gradient and Hessian of the quadratic function $h(\boldsymbol{x}) = \boldsymbol{x}^\intercal \boldsymbol{A}\boldsymbol{x} + \boldsymbol{b}^\intercal \boldsymbol{x}$ and remember to include the detailed steps. Here, $\boldsymbol{A}$ is square but **not necessarily symmetric**. $(1/15)$

(b) We talk about the algebraic and geometric definitions of convex functions in class. But it is often a tedious process to tell convex functions using the definition. To simplify the job, we rely on additional properties and characterizations. A twice-differentiable function $f(\boldsymbol{x})$ is convex if and only if its Hessian is positive semidefinite, i.e., $\nabla^2 f \succeq \boldsymbol{0}$ for all $\boldsymbol{x}$. Apply this to $h(\boldsymbol{x})$ in (a) and state the condition for $h(\boldsymbol{x})$ being convex. When it's convex, is there a unique *minimizer* for $h(\boldsymbol{x})$ or not, and why? $(1/15)$

(c) We talked of the first- and second-order optimality conditions for $\min_{\boldsymbol{x}} \ f(\boldsymbol{x})$ for a generic differentiable function f. What are the first- and second-order optimality conditions for $\max_{\boldsymbol{x}} \ f(\boldsymbol{x})$, i.e., conditions for locating local maximizers? And why? $(1/15)$

(d) Consider constrained optimization problems of the form

$$\min_{\boldsymbol{x}} \ f(\boldsymbol{x}) \quad \text{s.t. } g(\boldsymbol{x}) = \boldsymbol{0},$$

where $g(\boldsymbol{x})$ is a vector-to-vector function and conveniently collects together all the single scalar constraints. Introduce a Lagrangian multipler vector $\boldsymbol{\lambda}$ and form the Lagrangian function

$$\mathcal{L}(\boldsymbol{x}, \boldsymbol{\lambda}) = f(\boldsymbol{x}) + \langle \boldsymbol{\lambda}, g(\boldsymbol{x}) \rangle .$$

The first-order optimality condition says if $\boldsymbol{x}^*$ is an optimal solution and if $\boldsymbol{J}_g(\boldsymbol{x}^*)$ has full row rank, then there exists a $\boldsymbol{\lambda}^*$ so that $\nabla_{\boldsymbol{x}} \mathcal{L}(\boldsymbol{x}^*, \boldsymbol{\lambda}^*) = \boldsymbol{0}$. Use the optimality condition to prove that the global minimize of

$$\min_{\boldsymbol{d}} \ \langle \boldsymbol{d}, \nabla f(\boldsymbol{x}) \rangle \quad \text{s.t. } \|\boldsymbol{d}\|_2^2 = 1$$

is $\boldsymbol{d}^* = -\frac{\nabla f(\boldsymbol{x})}{\|\nabla f(\boldsymbol{x})\|_2}$, i.e., a result we used in class when deriving the gradient descent method.

Problem 4 (Deep learning problems are typically non-convex; $4/15$) Convex analysis and optimization have dominated classical machine learning (e.g., the famous support vector machines, and lasso for variable selection), as with convexity most of the time we can focus on the modeling part and worry little about the possibility of finding a bad local solution for the resulting optimization problem. In deep learning, the optimization problems involved are almost always non-convex. Let's try to convince ourselves using two different arguments.

(a) Consider a simplistic two-layer all-scalar network with identity activation $f(x) = w_2 w_1 x$, where w_1, w_2, x are all scalars. For a training set $\{(x_i, y_i)\}_{i=1,\ldots,N}$, let's take the mean squared loss and set up a supervised learning objective

$$L(w_1, w_2) = \frac{1}{N} \sum_{i=1}^{N} (y_i - f(x_i))^2.$$

Show that $L(w_1, w_2)$ is non-convex by checking its Hessian. (Hint: recall how to check convexity from Problem 3(b) above; also, when a 2×2 matrix is positive semidefinite, both its trace and determinant are nonnegative. $1/15$)

(b) An alternative way to see L is non-convex is to prove by contradiction. Let's assume that L is indeed convex. In class, we recognized that for any (w_1, w_2), $L(w_1, w_2) = L(-w_1, -w_2)$. If a particular pair (w_1^*, w_2^*) is a global minimizer of L, what can we say about $(0, 0)$? Then conclude that L being convex will lead to trivial learning. (Hint: $(0, 0)$ is a convex combination of (w_1^*, w_2^*) and $(-w_1^*, -w_2^*)$, i.e., lying on the line segment connecting (w_1^*, w_2^*) and $(-w_1^*, -w_2^*)$.) $(1/15)$

(c) Is L convex if $f(x) = w_k w_{k-1} \dots w_2 w_1 x$, i.e., when the network is k-layer with $k \geq 3$? What happens when we replace the mean squared loss by the mean absolute error, i.e.,

$$L(w_1, w_2, \dots, w_{k-1}, w_k) = \frac{1}{N} \sum_{i=1}^{N} |y_i - f(x_i)|?$$

$(1/15)$

(d) Now let's move to realistic multi-layer perceptrons with multi-neuron hidden layers. To fix the notation, assume that the input dimension is n_0, and we have L hidden layers with $n_1, \dots, n_L$ hidden neurons, respectively, and n_f outputs, that is,

$$f(\boldsymbol{x}) = \boldsymbol{W}_{L+1} \circ \sigma \circ \boldsymbol{W}_L \circ \sigma \cdots \circ \boldsymbol{W}_2 \circ \sigma \circ \boldsymbol{W}_1 \boldsymbol{x},$$

where $\circ$ means the composition of the function (https://en.wikipedia.org/wiki/Function_composition; we use this neat notation to avoid nesting many parentheses), and activation σ is always applied elementwise. Suppose that we take the mean squared loss, i.e.,

$$L(\boldsymbol{W}_1, \dots, \boldsymbol{W}_{L+1}) = \frac{1}{N} \sum_{i=1}^{N} \|y_i - f(\boldsymbol{x}_i)\|_2^2,$$

argue that L is in general nonconvex to achieve nontrivial learning. (Hint: Assume σ is the identity for simplicity. We have that $\boldsymbol{W}_1 \boldsymbol{W}_0 \boldsymbol{x} = (\boldsymbol{W}_1 \boldsymbol{\Pi})(\boldsymbol{\Pi}^\intercal \boldsymbol{W}_0)\boldsymbol{x}$ for any permutation matrix $\boldsymbol{\Pi} \in \mathbb{R}^{n_1 \times n_1}$. What is the average of all permutation matrices?) $(1/15)$