HOMEWORK SET 1

CSCI5527 Deep Learning (Fall 2022)

Due 11:59 pm, Oct 16 2022

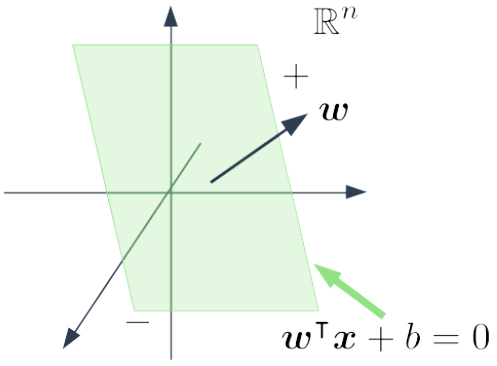
Instruction Your writeup, either typeset or scanned, should be a single PDF file. For problem requiring coding, please organize all codes for each problem into a separate Jupyter notebook file (i.e., `.ipynb` file). Your submission into Canvas/Gradescope should include the single PDF and all the notebook files—**Please do not zip them**! No late submission will be accepted. For each problem, your should acknowledge your collaborators if any. For problems containing multiple subproblems, there are often close logic connections between the subproblems. So whenever possible, try to build on previous ones, rather than work from scratch.

Notation We will use small letters (e.g., u) for scalars, small boldface letters (e.g., $\boldsymbol{a}$) for vectors, and capital boldface letters (e.g., $\boldsymbol{A}$) for matrices. $\mathbb{R}$ is the set of real numbers. $\mathbb{R}^n$ is the space of n-dimensional real vectors, and similarly $\mathbb{R}^{m\times n}$ is the space of $m \times n$ real matrices. The dotted equal sign $\doteq$ means defining.

Problem 1 (Neural networks can represent all Boolean functions; 6/15) The standard perceptron is a single-layer, single-output neural network with the step function as the activation, i.e.,

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

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).



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.

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. (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. 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? $(1/15)$

(d) For general $n \geq 2$, we consider general AND functions that take n inputs, where each input is either x_i or $\overline{x_i}$. 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. $(1/15)$

(e) Similar to (d), show that all n-input general OR function can be implemented using an n-input perceptron. $(1/15)$

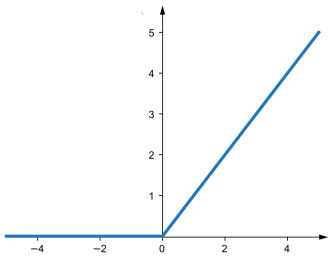
(f) Any Boolean function is fully specified by a list of all variable combinations that are evaluated to 1. Such list is often tabulated and the resulting table is called the truth table. For example, in the truth table of Fig. 1 (right), the Boolean function represented reads

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

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 all n-input Boolean functions can be represented by a two-layer neural network. $(1/15)$ In the worst case, how many hidden nodes are needed? $(0.5/15)$

Problem 2 (Universal approximation property of ReLU networks; $3/15$) 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 sum up the bumps to form the approximation.



Now let's work out a similar property for two-layer neural networks with the ReLU activation. The ReLU function is $\sigma(z) = \max(0, z)$, as shown in the right figure. We now follow the same step-bump-sum roadmap to work out the argument.

(a) Show that we can use two ReLU's, probably shifted versions of the standard one, to construct an arbitrary good approximation to the step function. Please draw the construction as a neural network. $(1/15)$

(b) Show that we can further construct "bump" functions out of the step functions. $(1/15)$

(c) Now we are ready to sum up shifted copies of bumps to approximate arbitrary functions. How many layers we need in total? $(1/15)$

Problem 3 (Jacobian, gradient and Hessian from expansions, or chain rule; optimality conditions; 4/15) To derive Jacobian, gradient, and Hessian below, you can choose applying the chain rule, the Taylor expansion trick, or any mixture of them. But, you may find that the Taylor trick mixed together with basic gradient rules (sum, product, quotient, and sometime chain rule) much easier and quicker than applying the chain rule alone.

(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. (0.5/12)

(b) We talked of the algebraic and geometric definition of convex functions in class. But it's often a tedious process to tell convex functions using the definition. To ease 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. (0.5/12) Do we have a unique *minimizer* or not for $h(\boldsymbol{x})$, and why? (0.5/12)

(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? (1/12)

(d) We will not think much of constrained optimization in this course. But let's play with a simple yet important one here. 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 there exists a $\boldsymbol{\lambda}$ so that $\nabla_{\boldsymbol{x}} \mathcal{L} = \boldsymbol{0}$ ($\nabla_{\boldsymbol{\lambda}} \mathcal{L} = g(\boldsymbol{x}) = \boldsymbol{0}$ for obvious reasons).

Now consider a constrained optimization problem

$$\max_{\boldsymbol{x}} \boldsymbol{x}^\intercal \boldsymbol{A}\boldsymbol{x} \quad \text{s.t. } \|\boldsymbol{x}\|_2^2 = 1,$$

where $\boldsymbol{A}$ is a symmetric matrix. Write down the first-order optimality condition (0.5/12) and what can you say about the solution (0.5/12; Hint: think of eigenvalue and eigenvectors)?

For those familiar with the second-order condition (https://en.wikipedia.org/wiki/Karush%E2%80%93Kuhn%E2%80%93Tucker_conditions#Necessary_conditions), you're encouraged to dig in and find the exact solution to the problem, but this is optional. This problem is closely connected to the famous Rayleigh quotient (https://en.wikipedia.org/wiki/Rayleigh_quotient).

Problem 4 (Deep learning problems are typically non-convex; $2/15$) Convex analysis and optimization has 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, single-hidden-node network with identity activation, i.e., $f(x) = w_2 w_1 x$. For a training set $\{(x_i, y_i)\}_{i=1,\dots,N}$, let's take the mean squared loss and set up a supervised learning objective

$$L(w_1, w_2) = \frac{1}{N} \sum_{i=1}^{N} (y_i - f(x_i))^2 . \tag{0.1}$$

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 $(-w_1^*, -w_2^*)$.) $(1/15)$

(**optional**) What if $f(x)$ consists of more than 2 layers? What if there are nonlinear activations, e.g., ReLU? What if the loss we choose is not the mean squared loss, e.g., cross-entropy? Does the argument still hold?