

HOMEWORK SET 1

CSCI 5980/8980 Think Deep Learning (Fall 2020)

Due 11:59 pm, Oct 15 2020

Instruction Please typeset your homework in \LaTeX and submit it as a single PDF file in Canvas. No late submission will be accepted. For each problem, you should acknowledge your collaborators if any. For problems containing multiple subproblems, there are often close logic connections between the subproblems. So always remember 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., \mathbf{a}) for vectors, and capital boldface letters (e.g., \mathbf{A}) for matrices. For a matrix \mathbf{A} , \mathbf{a}^i (superscripting) means its i -th row as a *row vector*, and \mathbf{a}_j (subscripting) means the j -th column as a column vector, and a_{ij} means its (i, j) -th element. \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) The standard perceptron is a single-layer, single-output neural network with the step function as the activation, i.e.,

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

where $\text{step}(z) = 1$ if $z \geq 0$ and 0 otherwise. Geometrically, f is a $\{0, 1\}$ -valued function with the hyperplane $\{\mathbf{x} : \mathbf{w}^\top \mathbf{x} + b = 0\}$ as the separating boundary between the 0- and the 1-region; see Fig. 1 (left).

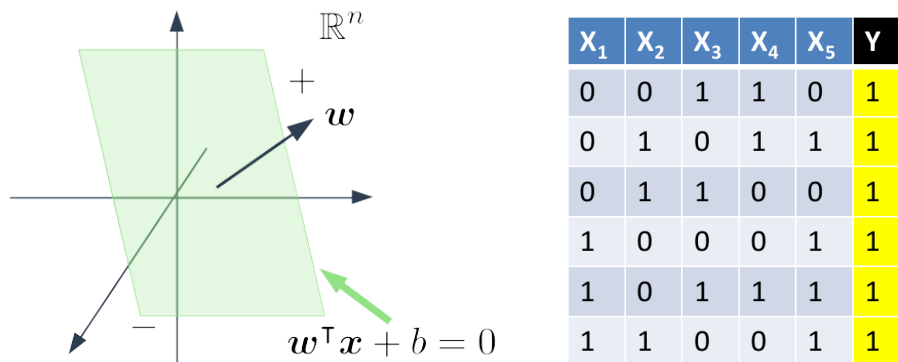


Figure 1: (left) Geometric illustration of the perceptron. (right) An example truth table.

Consider Boolean functions $\{0, 1\}^n \rightarrow \{0, 1\}$. 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. (0.5/12)
- (b) Consider the case $n = 2$ first. Show that the 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 questions in this problem. (1/12)

- (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? (1/12)
- (d) For general $n \geq 2$, we consider general AND functions that take n inputs, where each input is either x_i or \bar{x}_i . A typical such function looks like $x_1 \cdot \bar{x}_2 \cdot \bar{x}_3 \cdot x_{n-1} \cdot x_n$. Show that all general n -input AND function can be implemented using an n -input perceptron. (1/12)
- (e) Similar to (d), show that all n -input general OR function can be implemented using an n -input perceptron. (1/12)
- (f) Boolean functions are fully specified by a list of all variable combinations leading to output 1. This list is often tabulated and called the truth table. For example, in the truth table of Fig. 1 (right), the Boolean function represented reads

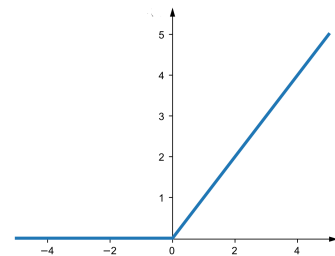
$$\bar{x}_1 \bar{x}_2 x_3 x_4 \bar{x}_5 + \bar{x}_1 x_2 \bar{x}_3 x_4 x_5 + \bar{x}_1 x_2 x_3 \bar{x}_4 \bar{x}_5 + x_1 \bar{x}_2 \bar{x}_3 \bar{x}_4 x_5 + x_1 \bar{x}_2 x_3 x_4 x_5 + x_1 x_2 \bar{x}_3 \bar{x}_4 x_5,$$

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. (1/12) In the worst case, how many hidden nodes are needed? (0.5/12)

Problem 2 (Universal approximation property of ReLU networks)

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 step-bump-sum roadmap to work out the argument.

- (a) Show that we can use two ReLU's, probably shifted versions of the standard, to construct an arbitrary good approximation to any step function. Please draw the construction as a neural network. (1/12)
- (b) Show that we can further construct "bump" functions out of the step functions. (0.5/12)
- (c) Now we are ready to sum up shifted copies of bumps to approximate arbitrary functions. How many (weight) layers do we have in our current construction? If it is not two, can you show how to reduce it to two without sacrificing the approximation capacity? (1/12)

Problem 3 (Jacobian, gradient and Hessian from expansions, or chain rule; optimality conditions) To derive Jacobian, gradient, and Hessian below, you can choose applying the chain rule, the Taylor expansion trick we learned in class, or any mixture of them. But, you may find 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 $f(\mathbf{x}) = \mathbf{x}^\top \mathbf{A}\mathbf{x} + \mathbf{b}^\top \mathbf{x}$ and remember to include the detailed steps. (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 is convex if and only if its Hessian is positive semidefinite. Apply this to $f(\mathbf{x})$ in (a) and state the condition for $f(\mathbf{x})$ being convex. (0.5/12) Do we have a unique *minimizer* or not for $f(\mathbf{x})$ and why? (0.5/12)
- (c) We talked of the first- and second-order optimality conditions for $\min_{\mathbf{x}} f(\mathbf{x})$ for a generic differentiable function f . What are the first- and second-order optimality conditions for $\max_{\mathbf{x}} f(\mathbf{x})$, 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

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

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

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

The first order optimality condition says there exists a $\boldsymbol{\lambda}$ so that $\nabla_{\mathbf{x}} \mathcal{L} = \mathbf{0}$ ($\nabla_{\boldsymbol{\lambda}} \mathcal{L} = g(\mathbf{x}) = \mathbf{0}$ for obvious reasons).

Now consider a constrained optimization problem

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

where \mathbf{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).