CS4803/7643: Deep Learning
Fall 2018
Problem Set 2 – Architecture Theory – Solutions

Instructor: Dhruv Batra
TAs: Michael Cogswell, Harsh Agrawal, Nirbhay Modhe, Erik Wijmans
Discussions: https://piazza.com/gatech/fall2018/cs48037643

Due: Thursday October 18, 2018, 11:55pm

Instructions

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

   - Each problem/sub-problem should be on one or more pages. This assignment has 9 total problems/sub-problems, so you should have at least 9 pages in your submission.
   - When submitting to Gradescope, make sure to mark which page corresponds to each problem/sub-problem.
   - For the coding problem, please use the provided `collect_submission.sh` script and upload `hw2.zip` to the HW2 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.
   - 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.

     Please read https://stats200.stanford.edu/gradescope_tips.pdf for additional information on submitting to Gradescope.

2. LaTeX'd solutions are strongly encouraged (solution template available at cc.gatech.edu/classes/AY2019/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.

This assignment introduces you to some theoretical results that address which neural networks can represent what. It walks you through proving some simplified versions of more general results. In

particular, this assignment focuses on piecewise linear neural networks, which are the most common type at the moment. The general strategy will be to construct a neural network that has the desired properties by choosing appropriate sets of weights.

1 Logic and XOR

1. **[4 points]** Implement AND and OR for pairs of binary inputs using a single linear threshold neuron with weights $\mathbf{w} \in \mathbb{R}^2$, bias $b \in \mathbb{R}$, and $\mathbf{x} \in \{0,1\}^2$:

$$f(\mathbf{x}) = \begin{cases} 1 & \text{if } \mathbf{w}^T\mathbf{x} + b \geq 0 \\ 0 & \text{if } \mathbf{w}^T\mathbf{x} + b < 0 \end{cases} \tag{1}$$

That is, find $\mathbf{w}_{\texttt{AND}}$ and $b_{\texttt{AND}}$ such that

x_1	x_2	$f_{\texttt{AND}}(\mathbf{x})$
0	0	0
0	1	0
1	0	0
1	1	1

Also find $\mathbf{w}_{\texttt{OR}}$ and $b_{\texttt{OR}}$ such that

x_1	x_2	$f_{\texttt{OR}}(\mathbf{x})$
0	0	0
0	1	1
1	0	1
1	1	1

Solution:

There are many solutions, but here are two. Let

$$\mathbf{w}_{\texttt{AND}} = [1,1]^T \tag{2}$$
$$b_{\texttt{AND}} = -1.5 \tag{3}$$

and

$$\mathbf{w}_{\texttt{OR}} = [1,1]^T \tag{4}$$
$$b_{\texttt{OR}} = -0.1 \tag{5}$$

2. **[4 points]** Consider the XOR function

x_1	x_2	$f_{\texttt{XOR}}(x)$
0	0	0
0	1	1
1	0	1
1	1	0

Show that XOR can NOT be represented using a linear model with the same form as (1). [1]

Solution:

Proof. Let $g(x) = w^T x + b$ be the linear function preceeding the threshold in (1). For XOR to be representable there must be w and b such that

$$g([1,1]^T) < 0 \leq g([0,1]^T) \tag{6}$$
$$[1,1]^T w + b < [0,1]^T w + b \tag{7}$$
$$w_1 + w_2 + b < w_2 + b \tag{8}$$
$$w_1 < 0 \tag{9}$$

and

$$g([0,0]^T) < 0 \leq g([1,0]^T) \tag{10}$$
$$[0,0]^T w + b < [1,0]^T w + b \tag{11}$$
$$b < w_1 + b \tag{12}$$
$$0 < w_1. \tag{13}$$

This is a contradiction: $w_1 < 0$ and $0 < w_1$ cannot both be true. Thus XOR can't be represented with a single neuron of this type (or any neuron with linear activations for that matter). □

2 Piecewise Linearity

Consider a specific 2 hidden layer ReLU network with inputs $x \in \mathbb{R}$, 1 dimensional outputs, and 2 neurons per hidden layer. This function is given by

$$h(x) = W^{(3)} \max\{0, W^{(2)} \max\{0, W^{(1)} x + \mathbf{b}^{(1)}\} + b^{(2)}\} + b^{(3)} \tag{14}$$

with weights:

$$W^{(1)} = \begin{bmatrix} 0.5 \\ 0.5 \end{bmatrix} \tag{15}$$
$$b^{(1)} = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \tag{16}$$
$$W^{(2)} = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} \tag{17}$$
$$b^{(2)} = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \tag{18}$$
$$W^{(3)} = \begin{bmatrix} 1 & 1 \end{bmatrix} \tag{19}$$
$$b^{(3)} = 1 \tag{20}$$

An interesting property of networks with piecewise linear activations like the ReLU is that on the whole they compute piecewise linear functions. For each of the following points give the weight $W \in \mathbb{R}$ and bias $b \in \mathbb{R}$ (report the numerical values) which computes such that $Wx + b = h(x)$. Also compute the gradient $\frac{dh}{dx}$ evaluated at the given point.

[1] Hint: To see why, plot the examples from above in a plane and think about drawing a linear boundary that separates them.

1. **[2 points]**

$$x = 1 \tag{21}$$

Solution:

$$W = 2 \tag{22}$$
$$b = 3 \tag{23}$$
$$\frac{dh}{dx}|_1 = W = 2 \tag{24}$$

2. **[2 points]**

$$x = -1 \tag{25}$$

Solution:

$$W = 1 \tag{26}$$
$$b = 3 \tag{27}$$
$$\frac{dh}{dx}|_{-1} = W = 1 \tag{28}$$

3. **[2 points]**

$$x = -0.5 \tag{29}$$

Solution:

$$W = 1 \tag{30}$$
$$b = 3 \tag{31}$$
$$\frac{dh}{dx}|_{-0.5} = W = 1 \tag{32}$$

3 Depth - Composing Linear Pieces

Now we'll turn to a more recent result that highlights the *Deep* in Deep Learning. Depth (composing more functions) results in a favorable combinatorial explosion in the "number of things that a neural net can represent". For example, to classify a cat it seems useful to first find parts of a cat: eyes, ears, tail, fur, *etc*. The function which computes a probability of cat presence should be a function of these components because this allows everything you learn about eyes to generalize to all instances of eyes instead of just a single instance. Below you will detail one formalizable sense of this combinatorial explosion for a particular class of piecewise linear networks.

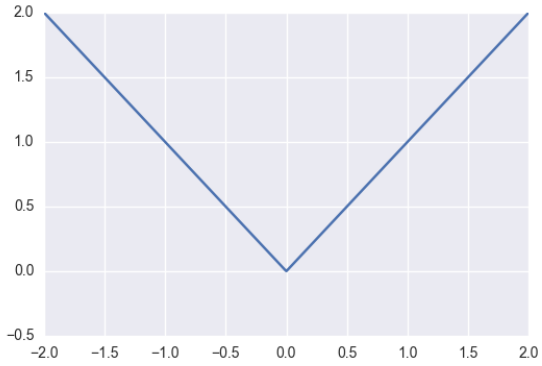

Figure 1

Consider $y = \sigma(x) = |x|$ for scalar $x \in \mathcal{X} \subseteq \mathbb{R}$ and $y \in \mathcal{Y} \subseteq \mathbb{R}$ (Fig. 1). It has one linear region on $x < 0$ and another on $x > 0$ and the activation identifies these regions, mapping both of them to $y > 0$. More precisely, *for each linear region of the input*, $\sigma(\cdot)$ is a bijection. There is a mapping to and from the output space and the corresponding input space. However, given an output y, it's impossible to tell which linear region of the input it came from, thus $\sigma(\cdot)$ *identifies* (maps on top of each other) the two linear regions of its input. This is the crucial definition because when a function identifies multiple regions of its domain that means any subsequent computation applies to all of those regions. When these regions come from an input space like the space of images, functions which identify many regions where different images might fall (*e.g.*, slightly different images of a cat) automatically transfer what they learn about a particular cat to cats in the other regions.
More formally, we will say that $\sigma(\cdot)$ identifies a set of M disjoint input regions $\mathcal{R} = \{R_1, \ldots, R_M\}$ (*e.g.*, $\mathcal{R} = \{(-1, 0), (0, 1)\}$) with $R_i \subseteq \mathcal{X}$ onto one output region $O \subseteq \mathcal{Y}$ (*e.g.*, $(0, 1)$) if for all $R_i \in \mathcal{R}$ there is a bijection from R_i to O. [2]

1. **[4 points]** Start by applying the above notion of identified regions to linear regions of one layer of a particular neural net that uses absolute value functions as activations. Let $\mathbf{x} \in \mathbb{R}^d$, $\mathbf{y} \in \mathbb{R}^d$ [3], and pick weights $W^{(1)} \in \mathbb{R}^{d \times d}$ and bias $\mathbf{b}^{(1)} \in \mathbb{R}^d$ as follows:

$$W_{ij}^{(1)} = \begin{cases} 2 & \text{if } i = j \\ 0 & \text{if } i \neq j \end{cases} \tag{33}$$

$$b_i^{(1)} = -1 \tag{34}$$

Then one layer of a neural net with absolute value activation functions is given by

$$f_1(\mathbf{x}) = |W^{(1)}\mathbf{x} + \mathbf{b}| \tag{35}$$

Note that this is an absolute value function applied piecewise and not a norm.

How many regions of the input are identified onto $O = (0, 1)^d$ by $f_1(\cdot)$? Prove it. [4]

[2] Recall that a bijection from X to Y is a function $\mu : X \to Y$ such that for all $y \in Y$ there exists a **unique** $x \in X$ with $\mu(x) = y$.

[3] Outputs are in some feature space, not a label space. Normally a linear classifier would be placed on top of what we are here calling $\mathbf{y}$.

[4] Absolute value activations are chosen to make the problem simpler, but a similar result holds for ReLU units. Also, O could be the positive orthant (unbounded above).

Solution:

Proof. Consider the set $\mathscr{R}$ of hypercubes which partition the unit hypercube by slicing it in half with hyperplanes in every dimension:

$$\mathscr{R} = \{[\min\{v_1, 0.5\}, \max\{v_1, 0.5\}) \times \tag{36}$$
$$[\min\{v_2, 0.5\}, \max\{v_2, 0.5\}) \times \tag{37}$$
$$\vdots \tag{38}$$
$$\times (\min\{v_d, 0.5\}, \max\{v_d, 0.5\}) : \mathbf{v} \in \{0, 1\}^d\} \tag{39}$$

Pick an arbitrary hypercube $R_{\mathbf{v}} \in \mathscr{R}$. I want to show that $f_1(\cdot)$ is a bijection from $R_{\mathbf{v}}$ to O. Fix $\mathbf{y} \in O$. Then the following $\mathbf{x} \in R_{\mathbf{v}}$ maps to $\mathbf{y}$:

$$x_i = \begin{cases} \frac{-y_i+1}{2} & \text{if } v_i = 0 \\ \frac{y_i+1}{2} & \text{if } v_i = 1 \end{cases} \tag{40}$$

You can see this by computing the ith output of $f_1(\cdot)$

$$f_1(\mathbf{x})_i = |\sum_j W^{(1)}_{ij} x_i + b_i| \tag{41}$$
$$= |2x_i - 1| \tag{42}$$
$$= |2\frac{\pm y_i + 1}{2} - 1| \tag{43}$$
$$= |\pm y_i + 1 - 1| \tag{44}$$
$$= |y_i| \tag{45}$$
$$= y_i \tag{46}$$

where the last step follows because $\mathbf{y} \in (0, 1)^d$.

Suppose there where another $\mathbf{x}' \in R_{\mathbf{v}}$ with $\mathbf{x}' \neq \mathbf{x}$ such that $f_1(\cdot)$ mapped both points to $\mathbf{y}$. This would imply

$$f_1(\mathbf{x})_i = f_1(\mathbf{x}')_i \tag{47}$$
$$|2x_i - 1| = |2x'_i - 1| \tag{48}$$

Note that x_i is in either $(0, 0.5)$ or $(0.5, 1)$ and x'_i is in the same region as x_i. For any $\hat{x}_i \in (0, 0.5)$, $2\hat{x}_i - 1 < 0$. For any $\hat{x}_i \in (0.5, 1)$, $2\hat{x}_i - 1 > 0$. That means $2x_i - 1$ and $2x'_i - 1$ are either both positive or both negative. Therefore

$$2x_i - 1 = 2x'_i - 1 \tag{49}$$
$$2x_i = 2x'_i \tag{50}$$
$$x_i = x'_i \tag{51}$$

This is a contradiction because we assumed $\mathbf{x} \neq \mathbf{x}'$. That means $f_1(\cdot)$ is a bijection from $R_{\mathbf{v}}$ onto O.

Finally, there are $|\mathscr{R}| = |(0, 1)^d| = 2^d$ possible different choices of $R_{\mathbf{v}}$, so $f(\cdot)$ identifies 2^d input regions onto O. □

2. **[4 points]** Next consider what happens when two of these functions are composed. Suppose g identifies n_g regions of $(0,1)^d$ onto $(0,1)^d$ and f identifies n_f regions of $(0,1)^d$ onto $(0,1)^d$. How many regions of its input does $f \circ g(\cdot)$ identify onto $(0,1)^d$?

   Solution:

   Proof. Denote the input regions identified by $g(\cdot)$ as $\mathscr{G}$ and the inputs to $f(\cdot)$ identified by $f(\cdot)$ as $\mathscr{F}$. Each $G \in \mathscr{G}$ has a sub-region which $g(\cdot)$ bijects onto F because $g(\cdot)$ bijects G onto $(0,1)^d \supset F$. Call this sub-region $G_F \subset G$.

   Since $f(\cdot)$ is a bijection between F and O, $f \circ g(\cdot)$ is a bijection bewtween G_F and O There are $|\mathscr{F}||\mathscr{G}| = n_f n_g$ choices of F and G, so $f \circ g(\cdot)$ identifies $n_f n_g$ input regions onto O. □

3. **[6 points]** Finally consider a series of L layers identical to the one in question 4 (a).

$$\mathbf{h}_1 = |W_1\mathbf{x} + \mathbf{b}_1| \tag{52}$$
$$\mathbf{h}_2 = |W_2\mathbf{h}_1 + \mathbf{b}_2| \tag{53}$$
$$\vdots \tag{54}$$
$$\mathbf{h}_L = |W_L\mathbf{h}_{L-1} + \mathbf{b}_L| \tag{55}$$

   Let $\mathbf{x} \in (0,1)^d$ and $f(\mathbf{x}) = \mathbf{h}_L$. Note that each $\mathbf{h}_i$ is *implicitly* a function of $\mathbf{x}$. Show that $f(\mathbf{x})$ identifies 2^{Ld} regions of its input.

   Solution:

   Proof. This can be accomplished by recursively applying the previous theorem. Start by making the layer-wise dependencies more explicit:

$$h_1(x) = |W_1 x + b_1| \tag{56}$$
$$h_2(h_1(x)) = |W_2 h_1(x) + b_2| \tag{57}$$
$$\vdots \tag{58}$$
$$h_L \circ h_{L-1} \cdots h_2 \circ h_1(x) = |W_L h_{L-1} \cdots h_2 \circ h_1(x) + b_L| \tag{59}$$

   I'll show that $f(x) = h_L \circ h_{L-1} \cdots h_2 \circ h_1(x)$ identifies 2^{Ld} regions by induction on L.

   For the base case I need to show $h_1(x)$ identifies $2^{1 \cdot d}$ regions, but this is already done in the first part of the section (question 4.1). Assume $g(x) = h_{L-1} \cdots h_2 \circ h_1(x)$ identifies $2^{(L-1)d}$ regions (inductive hypothesis). I'm interested in $f(x) = h_L(g(x))$. By the 2nd part of this section (question 4.2), this identifies $2^d \cdot 2^{(L-1)d} = 2^{Ld}$ input regions. Fortunately, that's exactly what I wanted to show. □

4 Conclusion to Theory Part

Now compare the number of identified regions for an L layer net to that of an $L-1$ layer net. The L layer net can separate its input space into 2^d more linear regions than the $L-1$ layer net. On

the other hand, the number of parameters and the amount of computation time grows linearly in the number of layers. In this very particular sense (which doesn't always align well with practice) deeper is better.

To summarize this problem set, you've shown a number of results about the representation power of different neural net architectures. First, neural nets (even single neurons) can represent logical operations. Second, neural nets we use today compute piecewise linear functions of their input. Third, neural nets with at least one hidden layer can represent arbitrary functions. Fourth, the representation power of neural nets increases exponentially with the number of layers.

Unfortunately, these results are not very practical. In the case of a 2-layer network an intractable number of hidden units may be required to represent the desired function well. Even if a particular architecture is capable of representing the function we're interested in, there's no guarantee that optimization will be able to find that solution. If an optimization method could find the solution then there's no reason to suggest that solution would generalize to test data.

The point of the exercise was to convey intuition that removes some of the magic from neural nets representations. Specifically, neural nets can decompose problems logically and piecewise linear functions can be surprisingly powerful.

5 Coding: Uses of Gradients With Respect to Input [50 points for CS7643, 40 points for CS4803]

The coding part of this assignment will have you implement 3 (CS4803) or 4 (CS7643) different ideas which all rely on gradients of some neural net output with respect to the input image. To get started go to https://www.cc.gatech.edu/classes/AY2019/cs7643_fall/hw2/.