CS4803/7643: Deep Learning
Fall 2019
Problem Set 2

Instructor: Zsolt Kira

TAs: Yihao Chen, Sameer Dharur, Rahul Duggal, Patrick Grady, Harish Kamath
Yinquan Lu, Anishi Mehta, Manas Sahni, Jiachen Yang, Zhuoran Yu

Discussions: https://piazza.com/gatech/spring2020/cs4803dl7643a/home

Due: Wednesday, Februrary 26, 2020, 11:59pm

Instructions

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

   - Each subproblem must be submitted on a separate page. When submitting to Gradescope, make sure to mark which page(s) corresponds to each problem/sub-problem. For instance, Q3 has 3 subproblems, and the solution to each must start on a new page.

   - When submitting to Gradescope, make sure to mark which page corresponds to each problem/sub-problem. ALSO remember to append your notebook PDFs to the solutions for the problem set, as described in the instructions!

   - For the coding problem, please use the provided `collect_submission.sh` script and upload `cs7643_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 https://www.cc.gatech.edu/classes/AY2020/cs7643_spring/assets/sol2.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 Convolution Basics

The convolution layer inside of a CNN is intrinsically an *affine transformation*: A vector is received as input and is multiplied with a matrix to produce an output (to which a bias vector is usually added before passing the result through a nonlinearity). This operation can be represented as $y = Ax$, in which A describes the affine transformation.

1. **[2 points]** We will first revisit the convolution layer as discussed in the class. Consider a convolution layer with a 3x3 kernel W, operated on a single input channel X, represented as:

$$W = \begin{bmatrix} w_{(0,0)} & w_{(0,1)} & w_{(0,2)} \\ w_{(1,0)} & w_{(1,1)} & w_{(1,2)} \\ w_{(2,0)} & w_{(2,1)} & w_{(2,2)} \end{bmatrix}, X = \begin{bmatrix} x_{(0,0)} & x_{(0,1)} & x_{(0,2)} \\ x_{(1,0)} & x_{(1,1)} & x_{(1,2)} \\ x_{(2,0)} & x_{(2,1)} & x_{(2,2)} \end{bmatrix} \tag{1}$$

Now let us work out a **stride-2** convolution layer, with **zero-padding size of 1**. Consider 'flattening' the input tensor X in row-major order as:

$$X = \begin{bmatrix} x_{(0,0)}, x_{(0,1)}, x_{(0,2)}, ..., x_{(2,0)}, x_{(2,1)}, x_{(2,2)} \end{bmatrix}^{\top} \tag{2}$$

Write down the convolution as a matrix operation A that: $Y = AX$. Output Y is also flattened in row-major order.

2. **[2 points]** Recall that transposed convolution can help us upsample the feature size spatially. Consider a transposed convolution layer with a 2x2 kernel W operated on a single input channel X, represented as:

$$W = \begin{bmatrix} w_{(0,0)}, w_{(0,1)} \\ w_{(1,0)}, w_{(1,1)} \end{bmatrix}, X = \begin{bmatrix} x_{(0,0)}, x_{(0,1)} \\ x_{(1,0)}, x_{(1,1)} \end{bmatrix} \tag{3}$$

We 'flattened' the input tensor in row-major order as $X = [x_{(0,0)}, x_{(0,1)}, x_{(1,0)}, x_{(1,1)}]$.

Write down the affine transformation A corresponding to a transposed convolution layer with kernel W, **stride 2, no padding**. Output Y is also flattened in row-major order.

2 Logic

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 $\mathbf{x} \in \{0, 1\}^2$:

$$f(\mathbf{x}) = \begin{cases} 1 & \text{if } \mathbf{w}^T\mathbf{x} + b \geq 0 \\ 0 & \text{if } \mathbf{w}^T\mathbf{x} + b < 0 \end{cases} \tag{4}$$

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

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

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

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

2. **[4 points]** Consider the bidirectional implication "$\Leftrightarrow$" (i.e., "if and only if", often abbreviated as "iff"), which has the following truth table

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

Show that "$\Leftrightarrow$" can NOT be represented using a linear model with the same form as (4).

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

3 Depth - Composing Linear Pieces [Extra Credit for 4803; Regular Credit for 7643]

An interesting property of networks with piecewise linear activations like the ReLU is that on the whole they compute piecewise linear functions. For instance, each layer $h^{(i)}$ below computes a linear function of the previous layer, $h^{(i-1)}$, and the resulting function is piecewise-linear in x:

$$h^{(1)}(x) = W^{(1)}x + b^{(1)} \tag{5}$$
$$h^{(2)}(x) = W^{(2)}\max\{h^{(1)}(x), 0\} + b^{(2)} \tag{6}$$
$$h^{(3)}(x) = W^{(3)}\max\{h^{(2)}(x), 0\} + b^{(3)} \tag{7}$$

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.
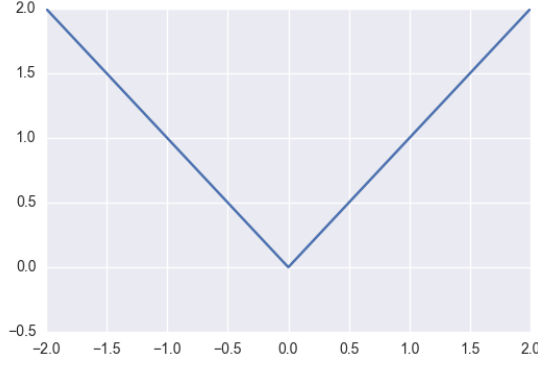
Figure 1

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. [1]

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 $\mathbf{x} \in \mathbb{R}^d$, $\mathbf{y} \in \mathbb{R}^d$ [2], and pick weights $W^{(1)} \in \mathbb{R}^{d \times d}$ and bias $\mathbf{b}^{(1)} \in \mathbb{R}^d$ as follows:

$$W_{ij}^{(1)} = \begin{cases} 2 & \text{if } i = j \\ 0 & \text{if } i \neq j \end{cases} \tag{8}$$

$$b_i^{(1)} = -1 \tag{9}$$

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

$$f_1(\mathbf{x}) = |W^{(1)}\mathbf{x} + \mathbf{b}| \tag{10}$$

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. [3]

---

[1] 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 $\mu(x) = y$.

[2] 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 $\mathbf{y}$.

[3] 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).

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 $(0,1)^d$?

3. **[6 points]** Finally consider a series of L layers identical to the one in question 4.1.

$$\mathbf{h}_1 = |W_1\mathbf{x} + \mathbf{b}_1| \tag{11}$$

$$\mathbf{h}_2 = |W_2\mathbf{h}_1 + \mathbf{b}_2| \tag{12}$$

$$\vdots \tag{13}$$

$$\mathbf{h}_L = |W_L\mathbf{h}_{L-1} + \mathbf{b}_L| \tag{14}$$

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.

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, the representation power of neural nets increases exponentially with the number of layers. 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 "simple" piecewise linear functions can be surprisingly powerful.

5 Paper Review

The second of our paper reviews for this course comes from some interesting work at Facebook AI Research (FAIR) last year exploring the limits of weakly supervised learning and improving upon their results on the ImageNet dataset by appending data from Instagram using hashtags as ground truth labels. The paper can be viewed here.
Guidelines: Please restrict your answers to no more than 350 words combined. The evaluation rubric for this section is as follows :

1. **[2 points]** The paper shows that training with very weak (noisy) labels still leads to robust learning of features that generalize well. Why is deep learning so robust to noise? Provide some conjectures, relating it to what you know about machine/deep learning and optimization.

2. **[2 points]** Another finding is that learning features using similar label spaces (i.e. on categories that overlap with what you are hoping to generalize to) is more successful than learning features on dissimilar label spaces. Why might that be the case?

6 Coding: Gradients With Respect to Input [65 points]

The coding part of this assignment will have you implement 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/AY2020/cs7643_spring/Z3o9P26CwTPZZMDXyWYDj3/hw2/.