CS7643: Deep Learning
Fall 2017
Problem Set 2

Instructor: Dhruv Batra
TAs: Michael Cogswell, Abhishek Das, Zhaoyang Lv
Discussions: http://piazza.com/gatech/fall2017/cs7643

Due: Tuesday, Oct 10, 11:55pm

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. **[2 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

2. **[2 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]

2 Universal Approximator [Extra Credit]

In this section you'll show that an MLP is a universal approximator of "nice" functions. There are more general results that relax the "niceness" assumptions and the particular choice of activation function.

Let $f : \mathbb{R}^d \to \mathbb{R}$ be differentiable and Lipschitz continuous. This assumption will remain true throughout this section. That is, there is some $K \in \mathbb{R}$ such that for all $\mathbf{x}_1, \mathbf{x}_2 \in \mathbb{R}^d$

$$|f(\mathbf{x}_1) - f(\mathbf{x}_2)| \leq K\|\mathbf{x}_1 - \mathbf{x}_2\|. \tag{2}$$

1. **[Extra Credit: 1 points]** Recall that the Taylor expansion of f at a point $\mathbf{x}_0$ is

$$f(\mathbf{x}) = f(\mathbf{x}_0) + \nabla f(\mathbf{x}_0)^T(\mathbf{x} - \mathbf{x}_0) + \mathscr{R}(x) \tag{3}$$

where $\mathscr{R}(\mathbf{x}) = \frac{1}{2}(\mathbf{x}-\mathbf{x}_0)^T\nabla^2 f(\mathbf{x}_0)(\mathbf{x}-\mathbf{x}_0)+\cdots$ is the remainder of the linear part of the Taylor series. Assume $\|\mathbf{x} - \mathbf{x}_0\| \leq C$ for some positive real constant C. Show that the remainder (or error) $\mathscr{R}(\mathbf{x})$ is bounded by [2]

$$|\mathscr{R}(\mathbf{x})| \leq 2KC \tag{4}$$

2. **[Extra Credit: 2 points]**

Now you'll show that a particular kind of neural net can approximate any "nice" function. Consider the differentiable and Lipschitz continuous function $f(\cdot)$ with input dimension $d = 1$, Lipschitz constant K. Show that $f(\cdot)$ can be approximated by the neural net $f'(\mathbf{x})$ (specified below) on the input region (α, β) (with $\alpha > \beta$) using a finite (but large) number of neurons to any desired error ϵ. That is, pick some weights for $f'(\cdot)$[3] and show that

$$|f(x) - f'(x)| \leq \epsilon \;\; \forall x \in (\alpha, \beta) \tag{5}$$

In this version $f'(\cdot)$ uses activation functions which represent a segment of an input on a given interval $(u, v]$ (*e.g.*, Fig. 1):

$$g(\mathbf{h}, \mathbf{u}, \mathbf{v})_i = \begin{cases} 0 & h_i \leq u_i \\ h_i & u_i < h_i \leq v_i \\ 0 & v_i < h_i \end{cases} \tag{6}$$

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

[2] Hint: You don't need to write out $\mathscr{R}(\mathbf{x})$ explicitly using an infinite series and higher order derivatives.

[3] Note that these weights can and should depend on $f(\cdot)$.

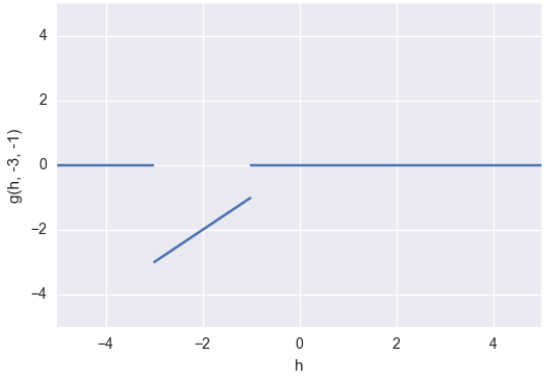

Figure 1

Now choose the number of hidden neurons H and the weights for the following neural net with a single hidden layer.

$$h(x) = W^{(2)} g(W^{(1)} x + \mathbf{b}^{(1)}, \mathbf{u}, \mathbf{v}) + b^{(2)} \tag{7}$$

with

$$W^{(1)} \in \mathbb{R}^{H \times 1} \tag{8}$$
$$\mathbf{b}^{(1)} \in \mathbb{R}^{H} \tag{9}$$
$$W^{(2)} \in \mathbb{R}^{1 \times H} \tag{10}$$
$$b^{(2)} \in \mathbb{R} \tag{11}$$
$$\mathbf{u} \in \mathbb{R}^{H} \tag{12}$$
$$\mathbf{v} \in \mathbb{R}^{H} \tag{13}$$
$$\tag{14}$$

3. **[Extra Credit: 1 point]** Note that adding a hidden layer allowed us to approximate the XOR function. Find specific weights (as for AND and OR in the previous section) for a 2-layer network which represent XOR.

3 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

$$h(x) = W^{(3)} \max\{0, W^{(2)} \max\{0, W^{(1)} x + \mathbf{b}^{(1)}\} + b^{(2)}\} + b^{(3)} \tag{15}$$

with weights:

$$W^{(1)} = \begin{bmatrix} 0.5 \\ 0.5 \end{bmatrix} \tag{16}$$

$$b^{(1)} = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \tag{17}$$

$$W^{(2)} = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} \tag{18}$$

$$b^{(2)} = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \tag{19}$$

$$W^{(3)} = \begin{bmatrix} 1 & 1 \end{bmatrix} \tag{20}$$

$$b^{(3)} = 1 \tag{21}$$

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. **[1 points]**

$$x = 1 \tag{22}$$

2. **[1 points]**

$$x = -1 \tag{23}$$

3. **[1 points]**

$$x = -0.5 \tag{24}$$

4 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.

Consider $y = \sigma(x) = |x|$ for scalar $x \in \mathcal{X} \subseteq \mathbb{R}$ and $y \in \mathcal{Y} \subseteq \mathbb{R}$ (Fig. 2). 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

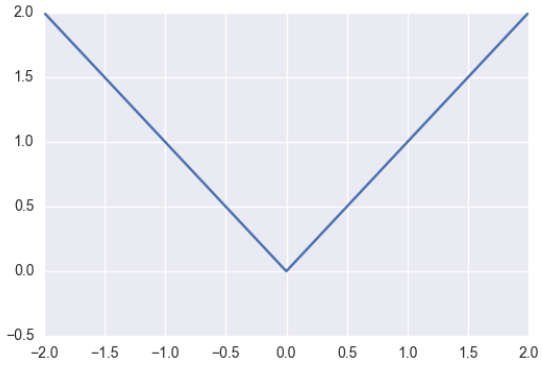

Figure 2

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 input regions $\mathscr{R} = \{R_1, \ldots, R_M\}$ (*e.g.*, $\mathscr{R} = \{(-1, 0), (0, 1)\}$) with $R_i \subseteq \mathscr{X}$ onto one output region $O \subseteq \mathscr{Y}$ (*e.g.*, $(0, 1)$) if for all $R_i \in \mathscr{R}$ there is a bijection from R_i to O. [4]

(a) **[2 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 $\mathbf{x} \in \mathbb{R}^d$, $\mathbf{y} \in \mathbb{R}^d$ [5], and pick weights $W^{(1)} \in \mathbb{R}^{d \times d}$ and bias $\mathbf{b}^{(1)} \in \mathbb{R}^d$ as follows:

$$W_{ij}^{(1)} = \begin{cases} 2 & \text{if } i = j \\ 0 & \text{if } i \neq j \end{cases} \tag{25}$$

$$b_i^{(1)} = -1 \tag{26}$$

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

$$f_1(\mathbf{x}) = |W^{(1)}\mathbf{x} + \mathbf{b}| \tag{27}$$

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. [6]

(b) **[2 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 $(0, 1)^d$?

(c) **[3 points]** Finally consider a series of L layers identical to the one in question 3.1.

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

[5] 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 $\mathbf{y}$.

[6] 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).

$$\mathbf{h}_1 = |W_1\mathbf{x} + \mathbf{b}_1| \tag{28}$$
$$\mathbf{h}_2 = |W_2\mathbf{h}_1 + \mathbf{b}_2| \tag{29}$$
$$\vdots \tag{30}$$
$$\mathbf{h}_L = |W_L\mathbf{h}_{L-1} + \mathbf{b}_L| \tag{31}$$

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.

5 Conclusion

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.