Benefits of depth, part 1

Administrative:

- Project info now on course webpage.
- Please keep giving feedback.

Are some function classes better than others?

The representation results presented so far say that various function classes used in machine learning are able to approximate continuous functions arbitrarily well. The following list summarizes these classes, and their "exponential blowup"/"curse of dimension".

To make the bounds a little more explicit than before, the target function g is taken to be *Lipschitz continuous*, meaning there exists $L < \infty$ so that

$$|g(x) - g(y)| \leq L\|x - y\|_\infty \qquad \forall x, y \in [0,1]^d.$$

(The choice of $\|\cdot\|_\infty$ is for simpliity, to work with cubes.) As such, if $\|x - y\|_\infty \leq \epsilon/L$, then $|g(x) - g(y)| \leq \epsilon$, so we can ϵ-approximate g by a function f which is constant on each of $(L/\epsilon)^d$ cubes forming a uniform partition of $[0,1]^d$. With this in mind, the following function classes from previous lectures have the following upper bounds on representation size.

- **Polynomials** have their proof deferred to homework, but there it will similarly be necessarily to interpolate polynomials through $\mathcal{O}((L/\epsilon)^d)$ points drawn from the cubes.
- **Decision trees** explicitly fit the partition, thus had $\mathcal{O}((L/\epsilon)^d)$ leaves.
- **Boosted decision trees** worked with trees of size $\mathcal{O}(d)$, but there had to be $\mathcal{O}((L/\epsilon)^d)$ of them.
- **3-layer neural networks** similar used $\mathcal{O}(d)$ nodes to approximate a single cube, but then needed to approximate $\mathcal{O}((L/\epsilon)^d)$ cubes. On one hand, the network has a tree-like structure and is not fully exploiting the architecture, but as will be sketched shortly, it's really hard to imagine needing less than exponential space.
- **2-layer neural networks** did not give an explicit upper bound, but the proof suggests and even worse $\mathcal{O}(2^{2d}(L/\epsilon)^d)$ as follows. The induction to rewrite products took a product of k terms into a summation of 2^k terms. An indicator on a cube is a product of $2d$ sigmoids (since it's an intersection of $2d$ halfspaces), meaning each cube expands to a summation of size $\mathcal{O}(2^{2d})$, and there are $\mathcal{O}((L/\epsilon)^d)$ cubes.

Perhaps some savings are possible, but the situation looks pretty grim, as the function class really seems to be able to pack some information into each of $(L/\epsilon)^d$ grid cells.

The hope then is to change the *goal*: to try to find some other class of functions which are *natural*, and then talk about the ability of different function classes to approximate these natural functions.

Unfortunately:

- It's hard to pin down natural functions.

- It's hard to compare the ability of two function classes on any reasonable definition of "natural functions".

Today we'll work towards a much more modest goal, which is to find *small* classes of functions which are easier for one function class than another. We'll be specifically interested in **the question of depth**: since 2-layer neural networks fit continuous functions, why bother with more layers?

Here is a list of results along these lines.

- Håstad (1986), in his thesis, gave a very nice proof (with nearly optimal bounds) that **parity on** d **bits** has a small-sized boolean circuit (AND, OR, NOT gates) of depth $\mathcal{O}(\ln(d))$, but requires circuits exponentially as big for constant depth. His thesis also had similar results for "Sipser functions"; notably, both circuits have tree structure.
- Rossman, Servedio, and Tan (2015) showed recently that for the same type of circuit, for any integer k one can construct a variant of the Sipser functions which can be written with $k+1$ layers, but can not be approximated on a .49-fraction of the inputs without exponential size.
- **[Todo (next time...):]** sum-product networks and arithmetic circuits (Raz 2010), (Ran Raz 2008).
- Telgarsky (2015) showed that for every k, there exists a ReLU network with $\mathcal{O}(k^2)$ layers and nodes which can not be approximated by networks with $\leq k$ nodes unless they have $\geq 2^k$ nodes. (A similar but more complicated/general version was later published (Telgarsky 2016).)
- Kane and Williams (2015) showed that with boolean inputs and nodes computing $y \mapsto \mathbf{1}[a^\top y \geq b]$, there exists a function in 3 layers that requires polynomially more nodes when approximated with 2 layers.
- Eldan and Shamir (2015) showed that for various types of networks, there is a function written with 3 layers which can not be approximated with 2 layers. **Open problem** / project idea: this proof is kindof long, perhaps it can be simplified?

Today we'll do a (superior) version of the result by Telgarsky (2015), hopefully not due to vanity, but rather because it is much shorter than the others.

Disclaimer. It is incredibly tacky to present my own work. Therefore, here is a list of weaknesses and **open problems**.

1. As stated before, this is just a small class of functions, moreover pathological ones, not "natural functions".
2. The analysis breaks for the sigmoid $z \mapsto 1/(1+\exp(-z))$.
3. The analysis compares k and $\mathcal{O}(k^2)$ layers (or even $\mathcal{O}(k^3)$ in the bigger version) rather than k and $k+1$.
4. The analysis is not powerful enough to imply or give alternate proofs of the others in the list. Most notably it is essentially a univariate result.

Benefits of depth

Theorem. Fix nonlinearity $\sigma(x) := \max\{0, x\}$, let any integer $k \geq 1$ be given, and let S_k denote those ReLU networks mapping $\mathbb{R}$ to $\mathbb{R}$ with $\leq k$ layers and $\leq 2^k$ nodes. Then there exists a function $g : \mathbb{R} \to \mathbb{R}$ representable as a network with $2(k^2+3)$ layers and $3(k^2+3)$ nodes so that

$$\inf_{f \in S_k} \|f - g\|_1 \geq \frac{1}{32}.$$

In words: for every integer k, there exists a network of size and depth $\mathcal{O}(k^2)$ which can not be approximated by any network with size $\leq 2^k$ and depth $\leq k$. It is important that the degree of approximation is a constant independent of the other problem parameters (and the same proof can be used to push it arbitrarily close to $1/2$).

Before giving the proof, note that the preceding result directly implies a guarantee for networks mapping from $\mathbb{R}^d$ to $\mathbb{R}$.

Corollary. The preceding result holds if the domain of all networks is $[0,1]^d$, and the norm $\|\cdot\|_1$ is similarly taken over $[0,1]^d$.

Proof. Let $g_1 : \mathbb{R} \to \mathbb{R}$ be the function given by the preceding theorem, and define $g(x) := g(x_1)$. Similarly, let $S_{1,k}$ denote the univariate function class considered by the preceding theorem, and let S_k denote the multivariate function class for this corollary. Note that for any $f \in S_k$ and any $z \in \mathbb{R}^d$ with the notation $(x, y) = z$ where $x \in \mathbb{R}$ and $y \in \mathbb{R}^{d-1}$ and $\mathbf{e}_i$ denoting the ith standard basis vector, then the function

$$f_y(x) := f((x,y)) = f\left(x\mathbf{e}_1 + \sum_{i=1}^{d-1} y_i \mathbf{e}_{i+1}\right)$$

satisfies $f_y \in S_{1,k}$, since $y \in \mathbb{R}^{d-1}$ can be baked into the affine transformations forming g without introducing more nodes or layers. But this rewriting allows the theorem to be applied to g_1 and f_y, meaning

$$\begin{aligned}
\|g - f\|_1 &= \int_{[0,1]^d} |g(z) - f(z)| dz \\
&= \int_{[0,1]^{d-1}} \int_{[0,1]} |g((x,y)) - f((x,y))| dx dy \\
&= \int_{[0,1]^{d-1}} \int_{[0,1]} |g_1(x,y) - f_y(x))| dx dy \\
&\geq \int_{[0,1]^{d-1}} \frac{1}{32} dy = \frac{1}{32}.
\end{aligned}$$

□

With that out of the way, the proof of the theorem is based on the following principle.

When approximating Lipschitz functions above, it was necessary to tediously add together $(L/\epsilon)^d$ rectangles. Wouldn't it be great if we only had to do, say, $d \ln(L/\epsilon)$ work to build $(L/\epsilon)^d$ bumps?

So the idea of the proof is as follows:

1. Show that bumpiness can prove the result: namely, given one function with many bumps, and another with few bumps, the gap in $\|\cdot\|_1$ is large.
2. Explicitly construct a very bumpy function in $\mathcal{O}(k^2)$ layers.
3. Prove that every function with a small number of layers has few bumps.

Before proceeding with this plan, there is a wrinkle, namely that it is not sufficient to have a very bumpy function. Suppose f is represented with few layers, and has few bumps, whereas g has many layers, and looks exactly like f for nearly all of $[0,1]$, but then packs a lot of bumps into some negligibly small portion of $[0,1]$; with this in mind, $\|f - g\|_1$ could be arbitrarily small. (*Picture drawn in class.*)

With this in mind, what's really needed is for g to not only have many bumps, but to space them uniformly. So let's construct this first.

To this end, define a very simple bump (*picture drawn in class*)

$$h(x) := \begin{cases} 2x & \text{when } x \in [0, 1/2], \\ 2(1-x) & \text{when } x \in (1/2, 1], \\ 0 & \text{when } x \notin [0,1]. \end{cases}$$

Observe that h can be written with 3 ReLUs in 2 layers: $h(x) = \sigma(\sigma(2x) - \sigma(4x - 2))$. The bumpy, regular function we use will be h^t for $t = \Theta(k^2)$, where

$$h^t(x) := \underbrace{h(h(h(\cdots(h(x)))))}_{t \text{ invocations}}.$$

Lemma. For any integer $i \in \{0, \ldots, 2^{t-1} - 1\}$ and any real $x \in [0, 1]$, h^t satisfies

$$h^t\left(\frac{1}{2^{t-1}}(x + i)\right) = h(x).$$

(Picture drawn in class.) In words, h^t looks like 2^{t-1} copies of h placed next to each other and squeezed together to fit in $[0, 1]$.

Proof. *(First a picture proof is given in class, constructing h^2, h^3, etc.)* The picture proof can be written out as follows. The basic idea is that we can decompose h^t as $h^{t-1} \circ h$, and h then "compresses" h^{t-1} along $[0, 1/2]$, and do a similar transformation along $[1/2, 1]$, ending with the same result by symmetry.

In symbols, the proof is an induction on t. The base case $t = 1$ leaves nothing to show. When $t > 1$, let $x \in [0, 1]$ and $i \in \{0, \ldots, 2^{t-1} - 1\}$ be given, and consider two cases.

- Suppose $i \in \{0, \ldots, 2^{t-2} - 1\}$, whereby $x' := (x + i)/2^{t-1} \in [0, 1/2]$. Then

  $$h^t(x') = h^{t-1} \circ h(x') = h^{t-1}(2x') = h^{t-1}\left(\frac{1}{2^{t-2}}(x + i)\right),$$

  the last step using the inductive hypothesis.

- Otherwise $i \in \{2^{t-2}, \ldots, 2^{t-1} - 1\}$, whereby $x' := (x + i)/2^{t-1} \in [1/2, 1]$. This case also uses the inductive hypothesis, but first some adjusting is required:

  $$\begin{aligned} h^t(x') &= h^{t-1} \circ h(x') = h^{t-1}(2 - 2x') \\ &= h^{t-1}\left(\frac{1}{2^{t-2}}(2^{t-1} - x - i)\right) = h^{t-1}\left(\frac{1}{2^{t-2}}((1 - x) + (2^{t-1} - 1 - i))\right) \\ &= h(1 - x) = h(x), \end{aligned}$$

  where the last step used symmetry of h around $1/2$. □

(At this point, the lecture ended; the next lecture picked up here, but started off with more discussion and intuition for h and h^t.)

References

Eldan, Ronen, and Ohad Shamir. 2015. "The Power of Depth for Feedforward Neural Networks."

Håstad, Johan. 1986. "Computational Limitations of Small Depth Circuits." PhD thesis, Massachusetts Institute of Technology.

Kane, Daniel, and Ryan Williams. 2015. "Super-Linear Gate and Super-Quadratic Wire Lower Bounds for Depth-Two and Depth-Three Threshold Circuits."

Ran Raz, Amir Yehudayoff. 2008. "Multilinear Formulas, Maximal-Partition Discrepancy and Mixed-Sources Extractors." In *FOCS*.

Raz, Ran. 2010. "Tensor-Rank and Lower Bounds for Arithmetic Formulas." In *STOC*.

Rossman, Benjamin, Rocco A. Servedio, and Li-Yang Tan. 2015. "An Average-Case Depth Hierarchy Theorem for Boolean Circuits." In *FOCS*.

Telgarsky, Matus. 2015. "Representation Benefits of Deep Feedforward Networks."

———. 2016. "Benefits of Depth in Neural Networks." In *COLT*.