ML Theory Lecture 6 — Succinct Representations 1

Matus Telgarsky

1 Miscellanea

- Hwk1 out tonight. Policy different than Hwk0: everyone still needs their own writeup, but can talk with up to three others.

This lecture we will start on *succinct representations*: the goal in this and the following lectures is to show that there are situations where a deep network can approximate a function much more efficiently than a shallow network.

2 Tent maps and fractional parts

Define the *fractional part* $\langle x \rangle := x - \lfloor x \rfloor$.

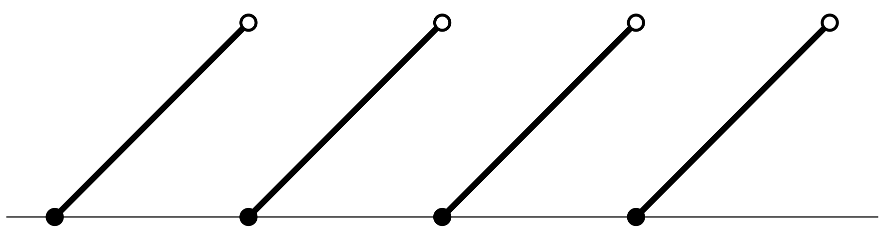


Figure 1: $\langle x \rangle$ along $[0, 4)$.

This is a hard function. Consider for instance polynomial approximation.

- Of course, $\langle x \rangle$ is discontinuous, so we have to give up on $\|\cdot\|_u$.
- Worse, consider $x \mapsto \langle x \rangle - 1/2$. On any interval $[0, n)$, this function has n crossings of 0. Therefore it degree at least n and at least n terms.

Don't underestimate this modest function. It can be found within *all* high-depth lower bounds for neural networks (both representation and VC bounds).

2.1 Tent maps

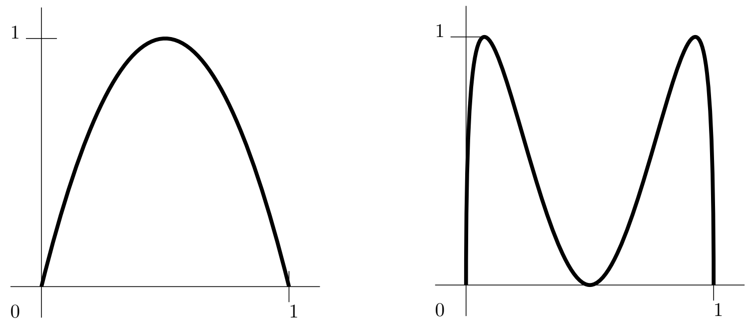
We can't represent $\langle x \rangle$ with ReLUs because it is discontinuous. But it turns out we can get really close...

Let us look at a function studied by Ulam and von Neumann (1947):

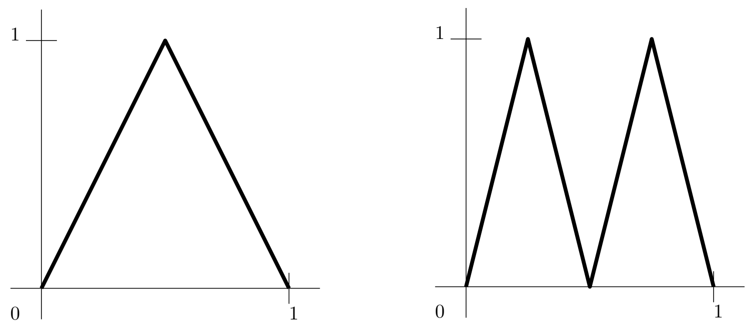
$$p(x) := 4x(1 - x).$$

These functions are heavily studied in the dynamical systems literature. Here is another one, which we *can* represent with the ReLU.

$$\Delta(x) := \sigma_r(2\sigma_r(x) - 4\sigma_r(x - 1/2)) = \begin{cases} 2x & x \in [0, 1/2], \\ 2(1 - x) & x \in (1/2, 1], \\ 0 & \text{otherwise.} \end{cases}$$



(a) p. (b) $p^2 = p \circ p$.



(a) Δ. (b) $\Delta^2 = \Delta \circ \Delta$.

The dynamical systems literature tells us p and Δ are "topologically equivalent". But at this point we must break from that literature, it doesn't seem to give us the lemmas we want...

Why are we studying tent maps? *They quickly grow in complexity.* Indeed, we will show:

- Δ^k has 2^{k-1} peaks; it consists of 2^{k-1} copies of Δ, each of width 2^{1-k}. Δ^k is "complex".
- Moreover, while Δ^k itself is a ReLU network with $O(k)$ layers, edges, and nodes, it is hard to approximate with shallow networks.

Let us begin making these claims rigorous.

Theorem 2.1. *For any $x \in [0,1]$ and any $k \geq 1$,*

$$\Delta^k(x) = \Delta\left(\left\langle 2^{k-1}x \right\rangle\right).$$

Remark 2.2. Even though $x \mapsto \left\langle 2^{k-1}x \right\rangle$ is hard to approximate, Δ^k embeds it inside Δ (!). Later we will see how to use Δ to embed $\left\langle 2^{k-1}x \right\rangle$ within *other* functions as well (!!!). $\diamond$

Remark 2.3. (Geometric description and proof.) *[Many pictures drawn in class.]* Let's return to a geometric view of Theorem 2.1: Δ^k is 2^{k-1} smushed copies of Δ. Let's reason about this by induction. Write $\Delta^{i+1} = \Delta^i \circ \Delta$. Along $[0, 1/2]$, this is the function $x \mapsto \Delta^i(2x)$, meaning it is like Δ^i but it is squished to fit in $[0, 1/2]$. Similarly, along $(1/2, 1]$,

$$\Delta^{i+1} = \Delta^i \circ \Delta = \left(x \mapsto \Delta^i(2(1-x))\right) = \left(x \mapsto \Delta^i(2x-1)\right),$$

the last bit since Δ^i is symmetric about $1/2$, meaning $\Delta^i(1-z) = \Delta^i(z)$ (in this case with $z = 2x - 1$). But $2x - 1$ applied to $(1/2, 1]$ gives $(0, 1]$, so once again we get a full copy of Δ^i, squished to half width.

We can also develop a geometric picture while peeling the induction the other way. That is, write $\Delta^{i+1} = \Delta \circ \Delta^i$. By the inductive hypothesis, Δ^i is 2^{i-1} copies of Δ. Applying Δ to this will double the image of

Δ^i when it is within $[0, 1/2)$, and otherwise it will double it and subtract it from 2; equivalently, this operation replaces Δ^i with $2\Delta^i$, and then "folds downward" the part that exceeds 1. ◇

Proof. The proof is by induction on k; crucially it establishes that the claim holds for all $x \in [0, 1]$ at each level; a single fixed x is not assumed throughout.

When $k = 1$, $x \in [0, 1)$ implies $\langle x \rangle = x$ thus $\Delta^1(x) = \Delta^1(\langle x \rangle)$, whereas $x = 1$ means $\Delta^1(x) = 0 = \Delta^1(0) = \Delta^1(\langle x \rangle)$.

Now suppose the claim holds for some $k \geq 1$, and needs to be shown for $k + 1$. Let $x \in [0, 1]$ be given; there are two cases to consider.

- If $x < 1/2$, then $2x \in [0, 1]$, meaning $\Delta^k(2x) = \Delta(\left\langle 2^k \cdot 2x \right\rangle)$, thus

$$\Delta^{k+1}(x) = \Delta^k(\Delta(x)) = \Delta^k(2x) = \Delta(\langle 2x \rangle) = \Delta(\left\langle 2^k \cdot 2x \right\rangle) = \Delta(\left\langle 2^{k+1} x \right\rangle).$$

- Otherwise $x \geq 1/2$, whereby $2x - 1 \in [0, 1]$ and so $\Delta^k(2x - 1) = \Delta(\left\langle 2^k(2x - 1) \right\rangle) = \Delta(\left\langle 2^{k+1} x \right\rangle)$. Furthermore, since Δ is symmetric about $1/2$, meaning $\Delta(1 - y) = \Delta(y)$ for $y \in [0, 1]$, then together

$$\Delta^{k+1}(x) = \Delta^k(\Delta(x)) = \Delta^k(2(1-x)) = \Delta^{k-1}(\Delta(1-(2x-1))) = \Delta^{k-1}(\Delta(2x-1)) = \Delta^k(2x-1) = \Delta(\left\langle 2^{k+1} x \right\rangle).$$

□

Remark 2.4. Note that we already have some evidence that Δ^k is painful to approximate with shallow things. For instance, when fitting continuous functions with linear combinations of things, it seems we needed another linear combination term for each "bump". But Δ^k has 2^{k-1} bumps. . . ◇

2.2 "Applications" of tent maps

[I didn't get to cover this (except multiplication). Maybe we'll have time to return to it. . .]

Let's consider a few nice things we can do with tent maps.

- **Multiplication.** Next lecture we'll show how to implement $(x, y) \mapsto xy$. This is important since it can be used to approximate smooth functions and polynomials.
- **Parity.** Consider the boolean hypercube, $x \in \{-1, +1\}^d$, and suppose $d = 2^k$ for some positive integer k (for simplicity). Then, by direct inspection,

$$\text{parity}(x) = \prod_{i=1}^{d} x_i = \Delta^{k-1}\left(\frac{d + \sum_{i=1}^{d} x_i}{2d}\right).$$

  Of note here is the following size comparison.

  - Written as a ReLU network, we need $O(\ln(d))$ nodes and $O(d)$ wires.
  - A *branching program* was shown a few lectures ago to need $O(d^2)$ nodes and wires.
  - On the other hand, axis-aligned decision trees were shown to need $\Omega(2^d)$ leaves; indeed, they needed a path through the tree for every boolean sequence.

- **Replication of symmetric signals.** Consider a function $\phi : [0, 1] \to \mathbb{R}$ which is symmetric about $1/2$, meaning again that $\phi(x) = \phi(1 - x)$. Then

$$\phi(\Delta^k(x)) = \phi(\left\langle 2^k(x) \right\rangle).$$

  There are two things of note here: (a) we are getting 2^k copies of ϕ with only $O(k)$ added nodes in the network, (b) we are using Δ^k to embed $\left\langle 2^k x \right\rangle$ into *another* function, despite $\left\langle 2^k x \right\rangle$ being painful! *[In class, a picture was drawn where each affine piece of Δ^k is replaced with ϕ.]*

  As an example of when this replication property is nice, if ϕ is a discontinuous bump, then $\phi(\Delta^k(x - 2^{-k-1}))$ performs digit extraction. *[pictures drawn in class.]*

3 Complexity of piecewise affine functions

In the next lecture we will prove that shallow networks can not approximate Δ^k (unless they are very wide). So far we have constructed a "complex" function in many layers, Δ^k. It still remains to argue that shallow functions are "not complex", and that this gap in complexity implies a gap in approximation.

We will focus on ReLU networks. Note that the function computed by any ReLU network is piecewise affine; thus a natural notion of complexity is simply the number of pieces.

Definition 3.1. A function $f : \mathbb{R} \to \mathbb{R}$ is *piecewise affine* if $\mathbb{R}$ can be divided into finitely many intervals (1 or 2 of which might be unbounded) such that f is a fixed affine function along each interval. Let $N_A(f)$ denote the smallest possible number of intervals such that along any interval f is a fixed affine function (with $N_A(f) = \infty$ if this is not possible), and let P_A be any set of pieces with $N_A(f) = |P_A(f)|$ (in general $P_A(f)$ is not unique (consider boundaries), but it won't matter to us).

(The definition can be generalized to multivariate functions, but we won't need it.) ◇

For example, the ReLU σ_r satisfies

$$P_A(\sigma_r) = P_A(\sigma_r) = \{\mathbb{R}_{<0}, \mathbb{R}_{\geq 0}\} \text{ or } \{\mathbb{R}_{\leq 0}, \mathbb{R}_{>0}\}, \qquad N_A(\sigma_r) = |P_A(\sigma_r)| = 2.$$

The following lemma will be used to establish an upper bound on N_A of univariate ReLU networks.

Lemma 3.2. *Let univariate functions $f, g, (g_1, \ldots, g_t)$ and scalars $(a_1, \ldots, a_t, b)$ be given.*

1. $N_A(f + g) \leq N_A(f) + N_A(g)$.
2. $N_A(\sum_i a_i g_i + b) \leq \sum_i N_A(g_i)$.
3. $N_A(f \circ g) \leq N_A(f) \cdot N_A(g)$.
4. $N_A(x \mapsto f(\sum_i a_i g_i(x) + b)) \leq N_A(f) \cdot \sum_i N_A(g_i)$.

Note that the last piece of the lemma gives a bound on N_A for a single node of a network. The next lecture will apply this inductively to get a bound for full networks.

Proof. [Somewhat informal; geometric intuition stressed; lots of pictures drawn in class.]

1. Draw f and g and also vertical bars at the boundaries of the pieces of each. between any two adjacent bars, f and g are each a fixed affine function, and thus sum to a fixed affine function. There are less than $N_A(f) + N_A(g)$ vertical bars (e.g., this is clear if we process intervals in sorted order), so we are done.

2. Since $N_A(a_i g_i) \leq N_A(g_i)$ (inequality can be strict when $a_i = 0$) and since $N_A(g_1 + b) = N_A(g_1)$, it suffices to consider $N_A(\sum_i g_i)$, which is at most $\sum_i N_A(g_i)$ by applying the previous part inductively.

3. Fix any interval $U \in P_A(g)$. Since g is affine, then $g(U)$ is also an interval. Now turning to f, f is necessarily piecewise affine along the interval $g(U)$, in particular f along the interval $g(U)$ must still be piecewise affine with at most $N_A(f)$ pieces. Therefore

$$N_A(f \circ g) \leq \sum_{U \in P_A(g)} N_A(f) = N_A(g) \cdot N_A(f).$$

4. This follows by combining the last two parts.

□

References

Stanislaw Ulam and John von Neumann. On combination of stochastic and deterministic processes. *Bulletin of the American Mathematical Society*, 53:1120, 1947.