1 Recap — end of box fit

Currently in the course we are showing that we can approximate continuous functions on compact sets with piecewise constant functions; in other words, with boxes. We will finish this today.

Remaining in the representation section we still have: polynomial fit of continuous functions, functions we can fit succinctly, and GANs.

Remark 1.1 (Homework comment.). A brief comment about problem 2(c); it wasn’t stated clearly enough, so hardly anything was taken off, but it really wanted something about LIL and Hoeffding disagreeing. Something along the lines of there being a lower bound (anti-concentration) infinitely often sufficed.

Note one funny thing we can do with Hoeffding. Hoeffding by default holds for a fixed $n$, but we can instantiate it for each $(\delta_n)_{n=1}^{\infty}$ with $\delta_n := \delta/(n(n+1))$. Thus

$$\Pr \left[ \forall n \geq 1, |S_n| < \sqrt{4n \ln(2n/\delta)} \right] = 1 - \Pr \left[ \exists n \geq 1, |S_n| \geq \sqrt{4n \ln(2n/\delta)} \right] \geq 1 - \sum_{n=1}^{\infty} \Pr \left[ |S_n| \geq \sqrt{2n \ln(2/\delta_n)} \right] \geq 1 - \sum_{n=1}^{\infty} \delta_n = 1 - \delta.$$  

On the other hand, LIL has a dependence on $\ln(\ln(n))$ rather than $\ln(n)$! Where is the Hoeffding+Union approach losing efficiency?

The union bound can be tight: when it is controlling disjoint events! That is far from the case here: as an extremal example, if the random walk happens to be near 0, we don’t have to worry about it being near the envelop for the next $\sqrt{n}$ steps.

2 Further statements about decision trees

[ Picture drawn to recall decision trees (DT); see last lecture notes. ]

Last time we proved DT can approximate continuous functions on compact sets in the uniform norm. Before giving the remaining theorems, a few remarks.

Remark 2.1. • The representation size was exponential in dimension: to approximate an $L$-Lipschitz function to accuracy $\epsilon > 0$, we needed a tree with $O((L/\epsilon)^d)$ nodes.

• Decision trees will probably not be mentioned outside this part of the course! It’s a good time to mention an open problem for them: learning decision trees is a disaster. The best result I know is due to [Kearns and Mansour, 1999], which basically says boosting is better, the issue being that data is partitioned amongst leaves, so with $m$ leaves, most are trained with less than $1/m$ fraction of the data.

The last results are with regards to function classes built off of decision trees.

Theorem 2.2. Let any continuous $g : \mathbb{R}^d \rightarrow \mathbb{R}$ and any $\epsilon > 0$ be given. Let $\mathcal{B}$ denote standard decision trees with at most $2d$ nodes. Then there exists $f \in \text{span}(\mathcal{B})$ such that $\|f - g\|_u \leq \epsilon$.  

1
Remark 2.3. • This is the class of boosted decision trees. Popular implementations like xgboost are still in use (on kaggle I see them last night aggregating convnet features?!). Many old implementations used decision “stumps”: depth 1 standard decision trees.

Proof. First note that a decision tree of depth $2d$ can represent an indicator an arbitrary product of intervals, the tree having a chain of nodes for each constituent halfspace, anything failing any halfspace test resulting in a leaf with output 0, and the intersection of all halfspaces having leaf value 1, with some care taken for open halfspaces. [Picture drawn in class.]

By the lemma from the previous lecture, there exists an integer $k$ so that if we divide $[0, 1]^d$ into $N = k^d$ cubes $(R_i)_{i=1}^N$, each formed by $d$ intervals $\cap_{j=1}^d I_j$, where $I_j = [i/k, (i+1)/k)$ for some $i < k-1$ and $I_j := [(k-1)/k, 1]$ otherwise. By the same lemma, there exist scalars $(\alpha_i)_{i=1}^N$ so that $h := \sum_i \alpha_i I_{R_i}$ satisfies $\|h - g\|_u \leq \epsilon$. The proof is complete since $1_{R_i} \in \mathcal{B}$ by the preceding, thus $h \in \text{span}(\mathcal{B})$.

Remark 2.4.
1. The size of the tree is relevant in two ways. First, shallow trees take less time to construct.
2. It is a disaster to learn even these trees! In fact, note that these trees are a single branch, and thus called “decision lists”, which are hard to learn agnostically [Feldman et al., 2009].

Open problem: find some way to circumvent the hardness barrier for instance via “improper learning”.

To close, let’s discuss an interesting modification of decision trees called Branching programs which allow tree nodes to have multiple incoming edges.

Example 2.5. This example is due to Kalai (2004). We can construct a succinct branching program for $x \mapsto \sum_i x_i$ for $x \in \{-1, +1\}$ as follows [in class, a picture was drawn]. The tree can be drawn as $d$ rows of predicate nodes; row $i$ has $i$ nodes, each containing the predicate $I_{R_i}$; Adjacent nodes in a row share a child, namely for any adjacent pair of nodes in a layer, the right child of the left node is the same as the left child of the right node. Row $i$ has $d + 1$ leaf nodes with the correct values.

By this construction, we can associate to every node in any row $i$ the count $\sum_{j<i} x_i$, and indeed maintain this property inductively. The lazy base case of $i = 1$ has no coordinates evaluated and the summation equal to 0. In the inductive step, consider any node in layer $i + 1$. Either we got here from our left parent moving right, which means $x_i = 1$, so our running sum is $\sum_{j<i} x_i + 1 = \sum_{j<i+1} x_i$. Alternatively, we came left from our right parent, and our running sum is $\sum_{j<i} x_i + 0 = \sum_{j<i+1} x_i$.

The only thing we’ll prove about branching programs is a succinctness result, thus looking ahead to a later part of this representation “chapter”.

Theorem 2.6. Consider $g(x) = \sum_i x_i$ for $x \in \{-1, +1\}$ as in the preceding example. Then there exists a branching program implementing this function with size $O(d^2)$ (indeed, the branching program from the example), whereas any decision tree equal to $g$ must have size $\geq 2^d$.

Proof. The discussion in the example established that the branching program has size $O(d^2)$ and correctly implements the function, so let’s focus on the possible decision trees. Note that this is the first representation lower bound we are proving; so far we’ve only done upper bounds!

To this end, note a special structure imposed by the axis-aligned nature of the predicates. We may tag every node in the decision tree with a system of constraints indicating the bit strings reaching that point; formally, write these strings as elements of $\{-1, +1, +\}^d$. Every predicate, due to being “axis-aligned”, can change at most one star to a $-1$ or $+1$ constraint (indeed, predicates of the form $I_{x_i \geq 10}$ leave a star).

As such, consider a tree with less than $2^d$ leaves. This means there must exist a leaf which is at depth strictly less than $d$, which in turn means it accepts strings which contain at least one star. Therefore this node, which outputs a fixed value, accepts a pair of strings which differ in exactly one bit, which thus necessarily have a different summation of their bits. Consequently, this tree with less than $2^d$ nodes is not equal to $g$. □

3 Neural networks

Standard setup:

\[
x \mapsto \sigma_L (A_L \sigma_{L-1} (A_{L-1} \cdots \sigma_1 (A_1 x + b_1) \cdots + b_{L-1}) + b_L).
\]

Remark 3.1. Some conventions.

- Old choice for \(\sigma_i\) is coordinate-wise it applies \(r \mapsto 1[r \geq 0]\) or \(r \mapsto \frac{1}{1+\exp(-r)}\).
- Contemporary choice is Lipschitz and continuous. E.g., either coordinate-wise ReLU \(r \mapsto \max\{0, r\}\), or max-pooling, which replaces groups of coordinates with their maximum [picture dawn in class]. AFAIK: popularized by ImageNet paper; ReLU might be fundamental to resurgence?
- \(\sigma_L\) typically identity.
- \(A_i\) really is just a linear operator. It may be written in a funny way though (e.g., convnet).
- Sometimes we’ll drop \(b_i\) for convenience; maybe have hwk question on this.
- Can also be interpreted as a graph/network. [picture drawn in class; “layers” defined, layer 0.]
- Obvious nice property: easy to adapt to complicated input/output domains.
- Real question (“non-mathematical”): why is this the function class that’s taking over? Not just a representation question.

Theorem 3.2. Consider standard 3 layer ReLU networks, meaning \(\sigma_L(r) = r\) (last layer has no nonlinearity), whereas other nodes use the ReLU nonlinearity \(r \mapsto \max\{0, r\}\). Then for every continuous function \(g\) and every \(\epsilon > 0\), there exists a function \(f\) written as a 3 layer ReLU network such that \(\|f - g\|_1 \leq \epsilon\).

Proof. [Proof had tons of pictures in class.] By the lemma discussed in the preceding lecture, it suffices to prove the following: given \(\tau > 0\) (suppose \(\tau \leq 1\) WLOG) and any rectangle \(R := [a_i, b_i] \times \cdots \times [a_d, b_d]\), there exists a two-layer network \(f_R\) with \(\|f_R - 1_R\| \leq \tau\). (The final construction is a linear combination of these two layer networks; the internal nodes thus have the shape of a tree; each \(f_R\) is a subtree, and does have a ReLU at the top.)

The idea of the proof is the following. It is easy to build an indicator for an interval (univariate rectangle) using a linear combination of nodes. If we try combining these for each dimension, we don’t get quite what we want, and we need to do some cleanup with another layer.

In more detail, magically choose \(\delta := \ln(1 + \tau)/(2d)\) (this will make sense later), fix a dimension \(i \in \{1, \ldots, d\}\), and define

\[
f_i(x) := \sigma\left(\frac{x_i - a_i}{\delta} + 1\right) - \sigma\left(\frac{x_i - a_i}{\delta}\right) - \sigma\left(\frac{x_i - b_i}{\delta}\right) + \sigma\left(\frac{x_i - b_i - \delta}{\delta}\right).
\]

Thus \(f_i(x) = 1\) when \(x_i \in [a_i, b_i]\), \(f_i(x) = 0\) when \(x_i \leq a_i - \delta\) or \(x_i \geq b_i + \delta\), and for the remaining strips around \([a_i, b_i]\), \(f_i\) linearly interpolates (and thus lies with \([0, 1]\).
To start with the multivariate case, consider what’s wrong with the mapping $x \mapsto \sum_i f_i(x)$. This is equal to $d$ within $R$, but it is large elsewhere. Note however that at least one $f_i$ is 0 whenever we are $\delta$ away from $R$ along any axis, and therefore $\sum_i f_i$ is at most $d - 1$ whenever we are at least $\delta$ away. Thus define

$$f_R(x) := \sigma\left((\sum_i f_i(x)) - (d - 1)\right).$$

(Picture drawn in $\mathbb{R}^2$ in class: without the outer $\sigma$, the function $\sum_i f_i$ is correct on $R$, but lots of slop elsewhere; adding a threshold and a ReLU cleans this up.) Summarizing what we said before,

$$f_R(x) = \begin{cases} 1 & x \in R, \\ \in [0, 1) & x \not\in R, \inf_{y \in R} \|x - y\|_\infty \leq \delta, \\ 0 & \text{otherwise}. \end{cases}$$

To calculate $\|f_R - 1_R\|_1$, it suffices to upper bound the volume of the region along the boundary of $R$ which $f_R$ maps within $[0, 1)$ as above. But this excess is no more than the excess given by the $\delta$-neighborhood of $[0, 1]^d$, meaning

$$\|f_R - 1_R\|_1 \leq \text{vol}([-\delta, 1+\delta]^d) - \text{vol}([0, 1]^d) = (1 + 2\delta)^d - 1 \leq \exp(2d\delta) - 1 = (1 + \tau) - 1 = \tau,$$

which used the tangent inequality $1 + x \leq e^x$, which holds for $x \in \mathbb{R}$, and the secant inequality $e^x \leq 1 + 10x$, which holds along $[0, 1]$.

Remark 3.3. Since $f_R \to 1_R$ pointwise as $\delta \to 0$, we could have shortcut the proof.

One value to this explicit computation is to make the following comment about inequalities. It may seem strange to use loose inequalities on $e^x$, and moreover worrisome that machine learning in general relies on them so heavily. Note however that this inequality is tight in the regime we care about ($\delta$ small), and thus we are safe. This is a general rule of inequalities: keep in mind when they are tight, and try to stick to that regime.

4 Introduction to polynomial fit

The polynomial fit results — namely, saying we can approximate continuous functions in uniform norm with polynomials, linear combinations of RBFs, and 2 layer networks — will all rely on the following theorem. This result stems from the classical Weierstrass approximation theorem (Weierstrass, 1885), which states that polynomials can approximate continuous functions. The following generalization says that even function classes having some of the same structure as polynomials can approximate continuous functions.

First, a tiny piece of notation to make some theorems less tedious.

**Definition 4.1.** Say that a class of functions $\mathcal{F}$ is dense in $C_c(\mathbb{R}^d)$ if for every continuous $g$ and every $\varepsilon > 0$, there exists $f \in \mathcal{F}$ with $\|f - g\|_u \leq \varepsilon$. 

**Theorem 4.2** (Stone-Weierstrass; cf. (Folland, 1999, Theorem 4.45)). Let a class of functions $\mathcal{F}$ be given, satisfying the following properties.

- Each $f \in \mathcal{F}$ is continuous.
- $\mathcal{F}$ is an algebra, meaning it is closed under multiplication and under vector space operations.
- $\mathcal{F}$ separates points, meaning for every $x, y$ there exists $f \in \mathcal{F}$ with $f(x) \neq f(y)$. (Note that if this property failed, then we could pick some continuous function $g$ which differs on $x$ and $y$, and $\mathcal{F}$ couldn’t hope to approximate it well.)
- For every $x$, there exists $f \in \mathcal{F}$ with $f(x) \neq 0$.

Then $\mathcal{F}$ is dense in $C_c(\mathbb{R}^d)$. 
Next class, we’ll use Stone-Weierstrass to prove a bunch of approximation results. Let’s close with some remarks.

**Remark 4.3.**  
• As a sanity check, standard polynomials satisfy the conditions of the Stone-Weierstrass theorem. Indeed, we can write them as

\[
\text{polynomials} = \text{span}(\text{monomials}) = \text{span}(\prod_{i \in [d]} x_i).
\]

These can be viewed as linear functions over the basis of monomials.

• The proof that seems popular now for the Weierstrass theorem is to pick some \(n\) and consider the Bernstein polynomials

\[
x \mapsto \sum_{i_1=0}^{\binom{n}{d}} \cdots \sum_{i_d=0}^{\binom{n}{d}} g \left( \sum_{j=1}^{d} \frac{i_j e_j}{n} \right) \prod_{j=1}^{d} \Pr \left[ \binom{n}{x_j} = i_j \right],
\]

which is a polynomial since \(\Pr \left[ \binom{n}{x_j} = i_j \right] = \binom{n}{i_j} x_j^i (1-x_j)^{n-i} \). The proof then uses Chebyshev’s inequality (or the law of large numbers) to control the polynomials between the \(n^d\) interpolation points.

• Weierstrass original proof considers the mollification \(x \mapsto \mathbb{E}(f(x + \xi))\), where \(\xi \sim \mathcal{N}(0, \sigma^2)\). This mollified mapping approaches \(f\) as \(\sigma \downarrow 0\), and moreover it is analytic so it has a well-behaved Taylor expansion (a polynomial).

\(\diamondsuit\)

**References**


