Last time:

- more hardness of learning based on public-key cryptography
  - (trapdoor 1-way permutations)
  - → our hardness assumption: "discrete cube roots" are hard to compute

Today:

- using this to show that even "simple" poly(n)-size Boolean circuits are hard to learn
  - → more precisely: poly(n) size Boolean formulas; equivalently, O(log n)-depth Boolean circuits

- peek at some other topics in CLT that we didn't cover this term

---

Reminder: final exam Fri 9am (check location online...) (Cumulative)
- → 90 min, closed book/notes
- → OH: exam review

Questions?

"Booleanizing" our hard-to-learn class $\mathcal{C}$ of

$f_N^{-1}(x) = x^d \bmod N$ functions:

$\hat{\mathcal{C}}$ = class of all $\{f_N^{-1}(x)_i\}$ $\quad i \in \{1, \ldots, n\}$

- bona fide concept class
- i^{th} bit of $f_N^{-1}(x)$.

Claim If DCRHA true, then $\hat{e}$ not eff PAC learnable.

Pf Sps have eff PAC alg for $\hat{e}$.

Given N, y, can generate rand ex. for $f_{N,1}^{-1}, \ldots, f_{N,n}^{-1}$ as follows:

- pick rand z
- compute $f_N(z)$
- use $(f_N(z), z_i)$ as the ex.
  $(x, f_N^{-1}(x)_i)$

Run PAC learner for $f_{N,1}^{-1}, \ldots, f_{N,n}^{-1}$ each time with acc $\varepsilon = 1/n^2$, conf $\delta = 1/n^2$

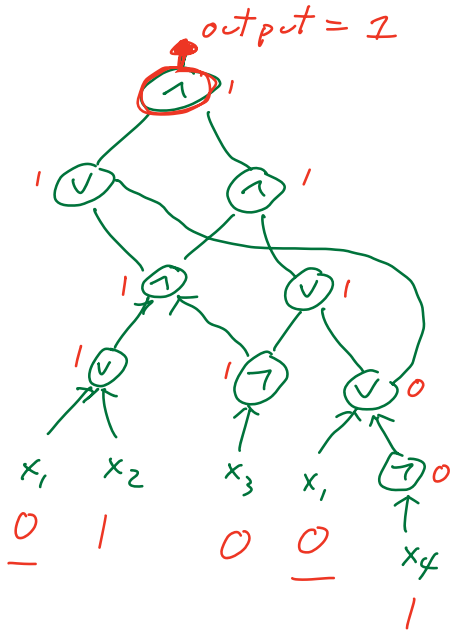
$1 - \frac{2}{n^2}$ chance h_i right on y for each $i = 1, \ldots, n$

VB $\rightarrow$ $(h_1(y), \ldots, h_n(y)) = f_N^{-1}(y)$ w.p. $\geq 1 - 2/n$.

Let's relate this to circuit complexity.

Bool ckt: rooted directed acyclic graph

Each gate: 



size of ckt = # gates

depth " " = length of longest directed path.

↳ parallel time

We know our $f_N^{-1}(x) = x^d \bmod N$

$f_N^{-1}(x) = x^{\frac{2(p-1)(q-1)+1}{3}} \bmod N$

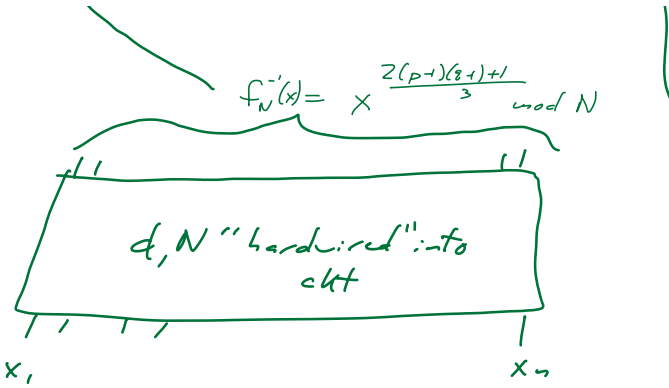
fns are hard to learn.

↳ what does this tells abt hardness of learning diff. types of ckts?

$f_N^{-1}(x) = x^d \bmod N$

$N = n\text{-bit \#} \approx 2^n$

$d = \dfrac{n\text{-bit \#}}{d \approx 2^n}$



→ how to compute?

View d as binary # $d = 1011010011101$

$8+4+1$

use repeated squaring:

$x \bmod N$

$x^2 \bmod N$

$x^4 = (x^2)^2 \bmod N$

$x^8 = (x^4)^2 \bmod N$

$\approx O(n)$ of these

☆ $O(n)$ mult (of the right coll. of these, corresp. to 1's in d) to get $x^d \bmod N$.

Easy

Fact: ☆ Can multiply two n-bit #s with a poly(n)-size, $O(\log n)$-depth ckt.

So, There's a ckt to compute f_N^{-1} that has $O(n \cdot \log n)$ depth & poly(n) size.

Dirty trick: Instead of $f_N^{-1}(x) = x^d \bmod N$ as the hard fn...

Consider setup where input is (!)

$(x,\ x^2 \bmod N,\ x^4 \bmod N,\ \ldots,\ x^{2^n} \bmod N)$

n^2-bit input

Consider Distrib. $\mathcal{D}_N^*$: a draw from $\mathcal{D}_N^*$ is

- pick rand x unif from $\mathbb{Z}_N^*$ (old $\mathcal{D}_N$)
- output (!) $d = \ldots b_2 b_1 b_0$

fn learning is $x^d \bmod N$

$= x^{2^{b_1}} \cdot x^{2^{b_2}} \cdot \ldots \bmod N$

$b_{i_1} =$ loc of first 1

$b_{i_2} =$ loc. of 2nd 1

$\vdots$

Since orig learning problem (just x as input) was hard, this one (! as input) must be hard too:

if had alg A for (!)-input problem,

could get an alg for orig problem by expanding each example & running A.

So following problem is hard to learn:

d, N:

input to fn is $z = (z_1, z_2, z_3, \ldots, z_n)$ — x, $x^2 \bmod N$, $x^4 \bmod N$

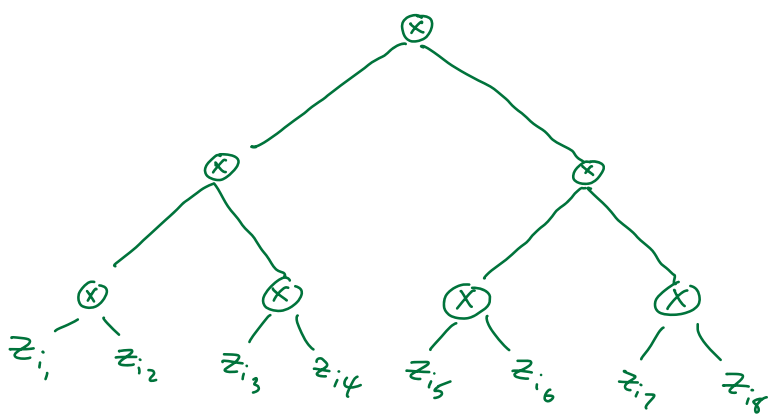
each z_i is n-bit string

$g(z)$

output is (j^{th} bit of)

$k \le n$

$z_{i_1} \cdot z_{i_2} \cdot \ldots \cdot z_{i_k} \bmod N.$



"iterated product mod N is hard to learn"

Need only $\log k$-depth tree of $\otimes$ ($k \le n$)

to compute prod. of k #s;

each $\otimes$ is computable by $O(\log n)$ depth ckt.

So $\mathcal{C} = \{$ poly(n)-size, $O(\log^2 n)$-depth ckts $\}$ is hard to learn, under DCRHA.

Hard

Fact: there is a ckt of depth $O(\log n)$ & size poly(n) computing product of n many n-bit #s.

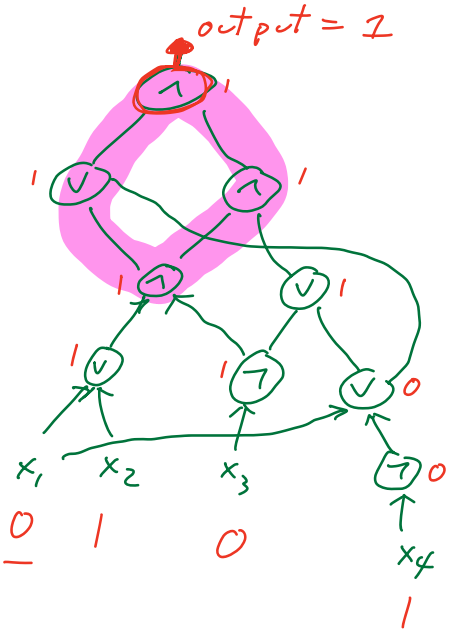
So $\mathcal{C} = \{$ poly(n)-size, $O(\log n)$-depth ckts $\}$ is hard to learn, under DCRHA.

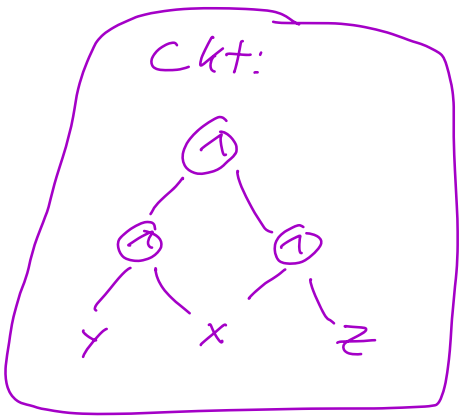
Easy fact: Every $O(\log n)$-depth ckt is computed by a poly(n)-size, $O(\log n)$-depth

Boolean formula

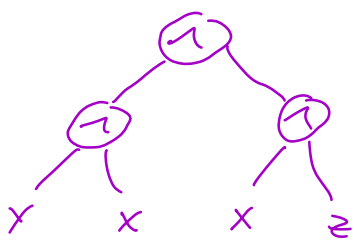
↳ "tree circuit"

$$\frac{-b \pm \sqrt{b \cdot b - 4 \cdot a \cdot c}}{2a}$$





Formula:



End of HOL

We Did:

- OCMB learning
- PAC learning

} specific algs for specific C's

- techniques for PAC learning:

- OLMB → PAC conversion
- CHF
- Occam thm

- sample cxity, VC dim
- boosting acc, confidence
- noise: — malicious
  — RCN
    ↳ SQ model
- rep. indep. HoL.

Other stuff learning theory people think about:

- learning rich/expressive C's:
  DTs? DNFs? const-depth ckts?
- MQ learning?
- agnostic learning? (noise model)
- active learning? (choose which unlab. ex. get labels)
- distrib. learning? (no target c)

Huge effort to bridge theory + practice.