The Uncanny Usefulness of Constructive Proofs of Pseudorandomness

Valentine Kabanets (SFU)

(M. Carmosino, R. Impagliazzo, V. Kolokolova)

Pseudorandom Objects

- Pseudorandom Generators (PRGs)
- Expanders
- Extractors
- Error-Correcting Codes
- Boolean functions of high circuit complexity

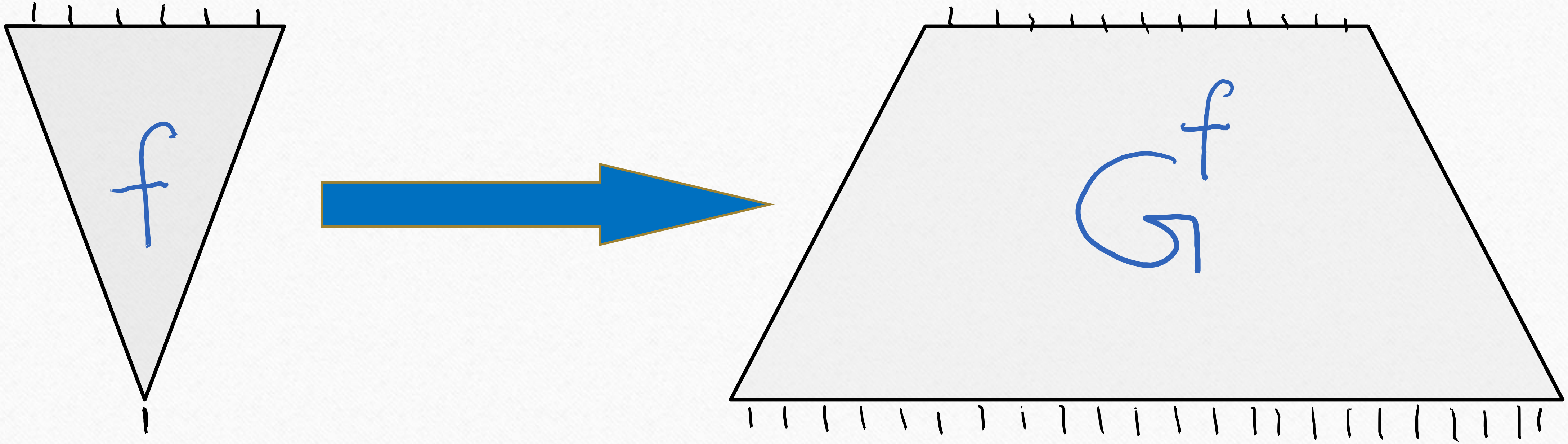
Pseudorandom Objects

- Pseudorandom Generators (PRGs)
- Expanders
- Extractors
- Error-Correcting Codes
- Boolean functions of high circuit complexity

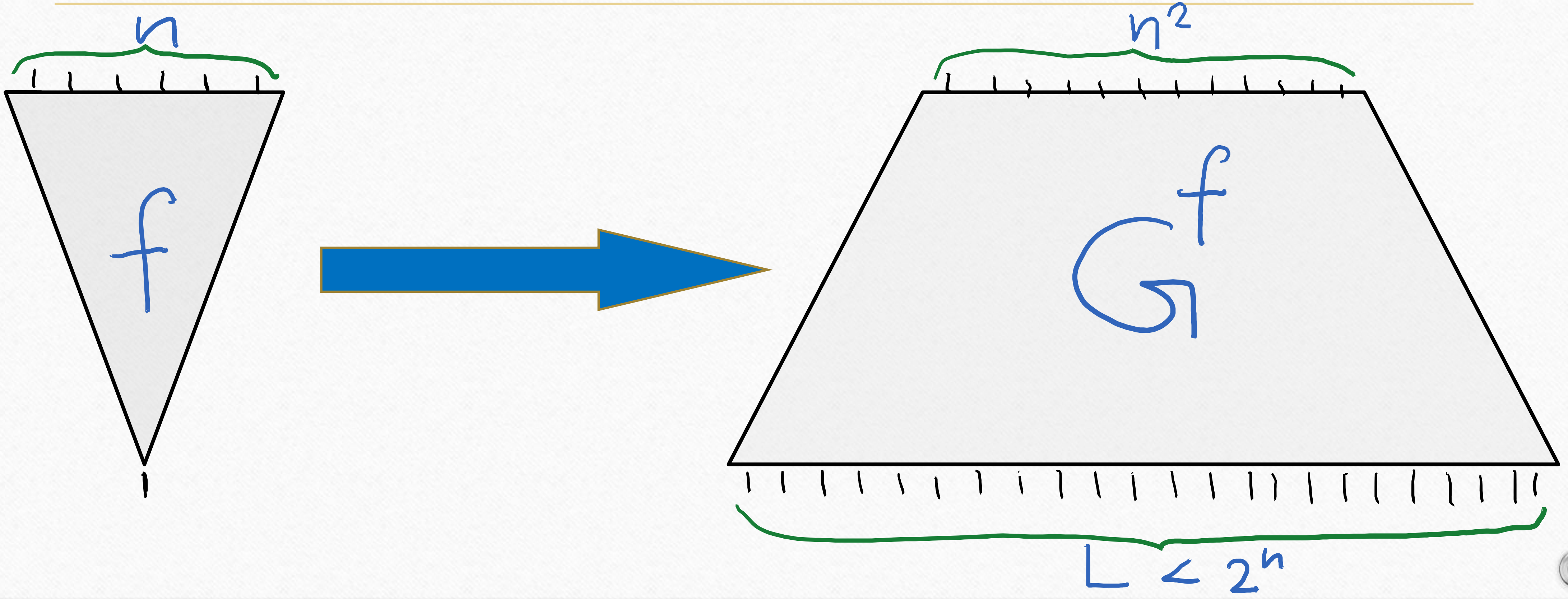
NW PRG: Two Viewpoints

- Construction: "hard" fn $\to$ PRG
- Analysis/Reconstruction: "breaking" PRG $\to$ small circuit

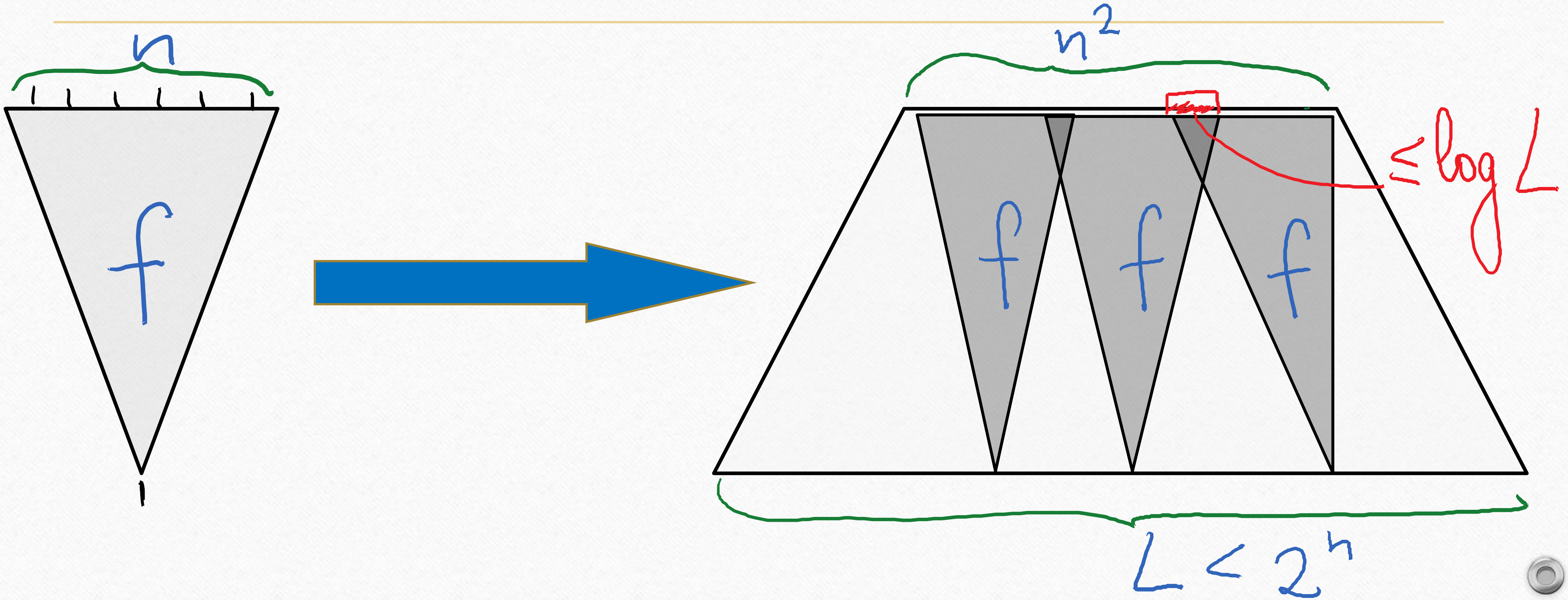
Nisan-Wigderson Generator



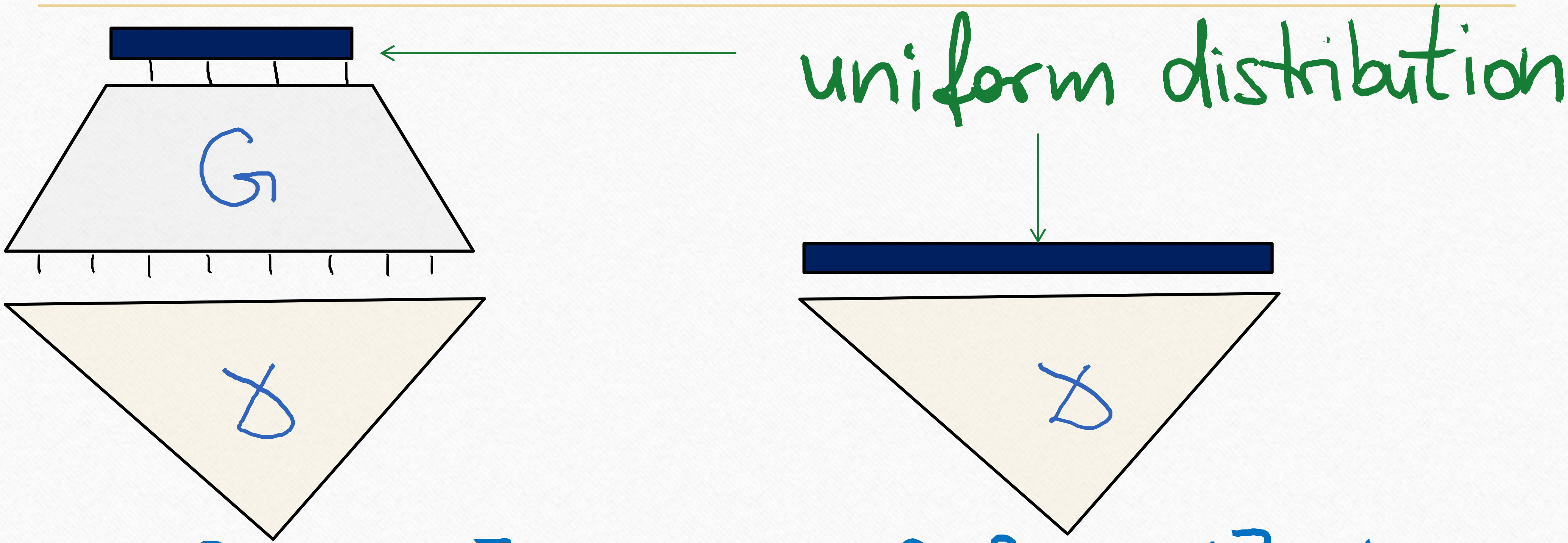
NW Generator



NW Generator



ε-PRG G against tests $\mathcal{D}$



$$\Pr[\bigcirc = 1] \approx \Pr[\bigcirc = 1] \pm \varepsilon$$

NW PRG: Hardness to Randomness

Thm[NW]: If $f:\{0,1\}^n \to \{0,1\}$ has correlation $\le \varepsilon/L$ with size-L^2 circuits, then $G^f:\{0,1\}^{n^2} \to \{0,1\}^L$ is ε-PRG against size-L circuits.

NW PRG:
Non-Randomness to Easiness

Thm [NW]: For $f:\{0,1\}^n \to \{0,1\}$, if $G^f:\{0,1\}^{n^2} \to \{0,1\}^L$ is not ε-PRG against size L-circuits, then f has $> \varepsilon/L$-correlation with L^2-size circuit.

Many Uses of the NW-Generator

- Derandomization [NW, BFNW, IW, ...]
- Extractors [Trevisan]
- Proof Complexity & Bounded Arithmetic [ABSRW, Kraj, Razb, Pich, ...]
- Circuit Lower Bounds (NEXP $\not\subseteq$ ACC^0) [Wil]

NW Generators.

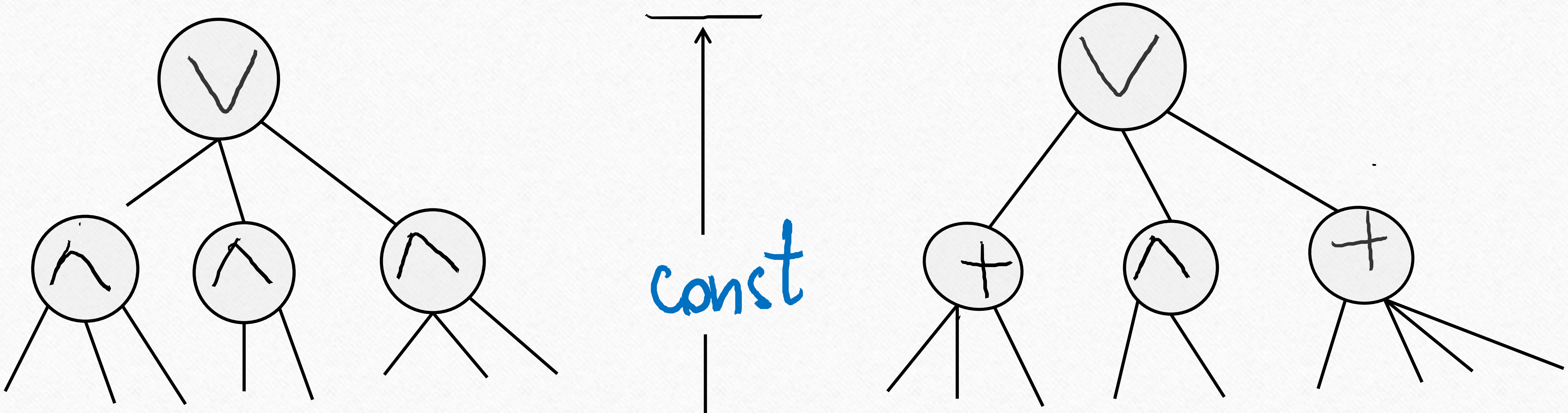
Is there anything they can't do?

Will use NW PRG

to learn

$AC^0[p]$ functions!

AC^0 & $AC^0[p]$



AND, OR, NOT gates

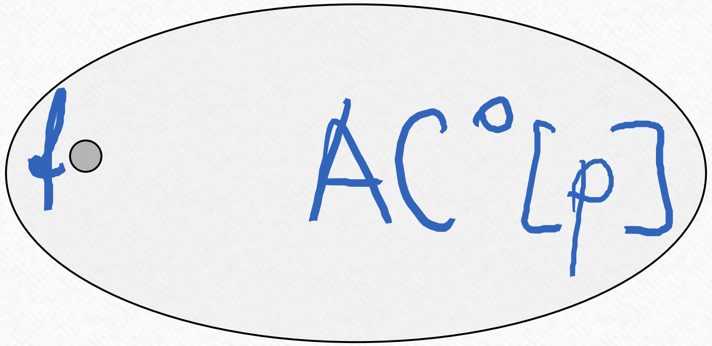
AND, OR, NOT, MOD p gates

Learning Algorithms [CIKK'16]

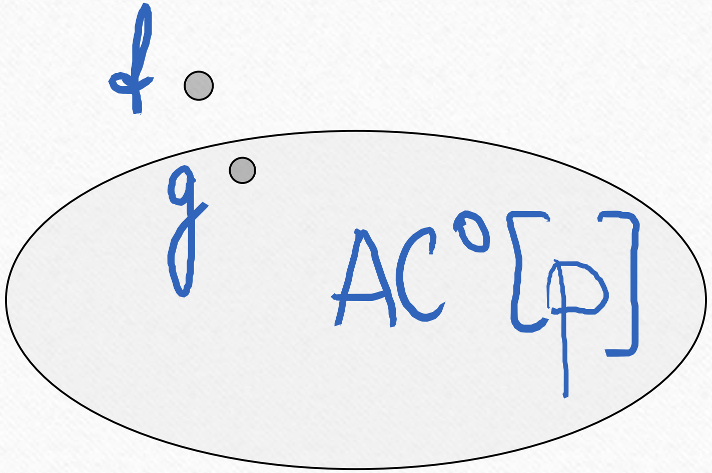
Thm: There is a randomized quasi-polytime algorithm that, given oracle access to $f \in AC^0[p]$, outputs a circuit C s.t.

$$\Pr_{x \sim U}\left[C(x) = f(x) \right] \geq 1 - 1/\mathrm{poly}.$$

Learning $AC^0[p]$



Agnostically Learning $AC^0[p]$



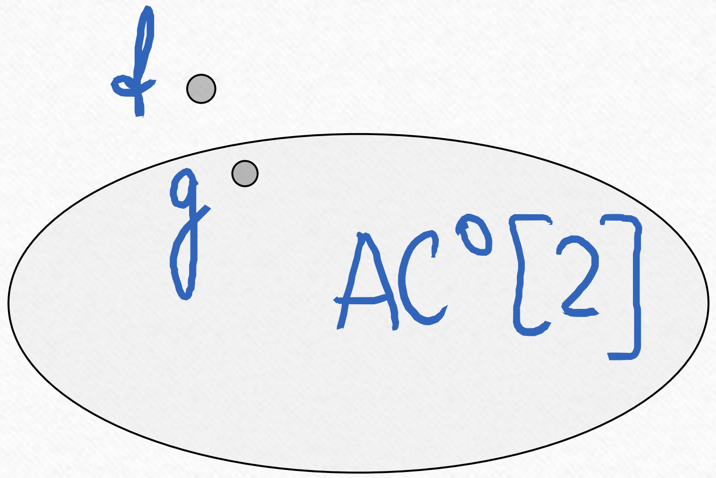
$$\Pr_{x\sim U}[f(x)\neq g(x)]\leq \beta$$

Agnostic Learning Algorithm

Learn a circuit C s.t.

$$\Pr_{x\sim U}[C(x)\neq f(x)] \le d\cdot\beta+\varepsilon$$

We only get $d=\text{polylog}$.



$$\Pr_{x\sim U}[f(x)\neq g(x)]\le\beta$$

Agnostic Learning [CIKK'17]

Thm: There is a randomized quasi-polytime algorithm that, given oracle access to f s.t. $\Pr_{x\sim U}[f(x)=g(x)] \geq 1-\beta$ for some $g \in AC^0[p]$ outputs C s.t. $\Pr_{x\sim U}[C(x)=f(x)] \geq 1-\beta\cdot \text{polylog}$.

Previous Work

[LMN'89]: Learning AC^0

[KSS'94, KKMS'08]: agnostic version

- Random examples $(x, f(x))$ suffice
- Analysis uses "Fourier concentration" of AC^0

Our Approach

Use NW Generators

NW PRG

algorithm	analysis
1. take hard f	1. take D not fooled by G^f
2. construct G^f	2. argue f is not hard
3. use G^f to fool any circuit D	

NW PRG

algorithm

1. take hard f
2. construct G^f
3. use G^f to fool any circuit D

Constructive analysis [IW'98]

1. take D not fooled by G^f
2. argue f is not hard

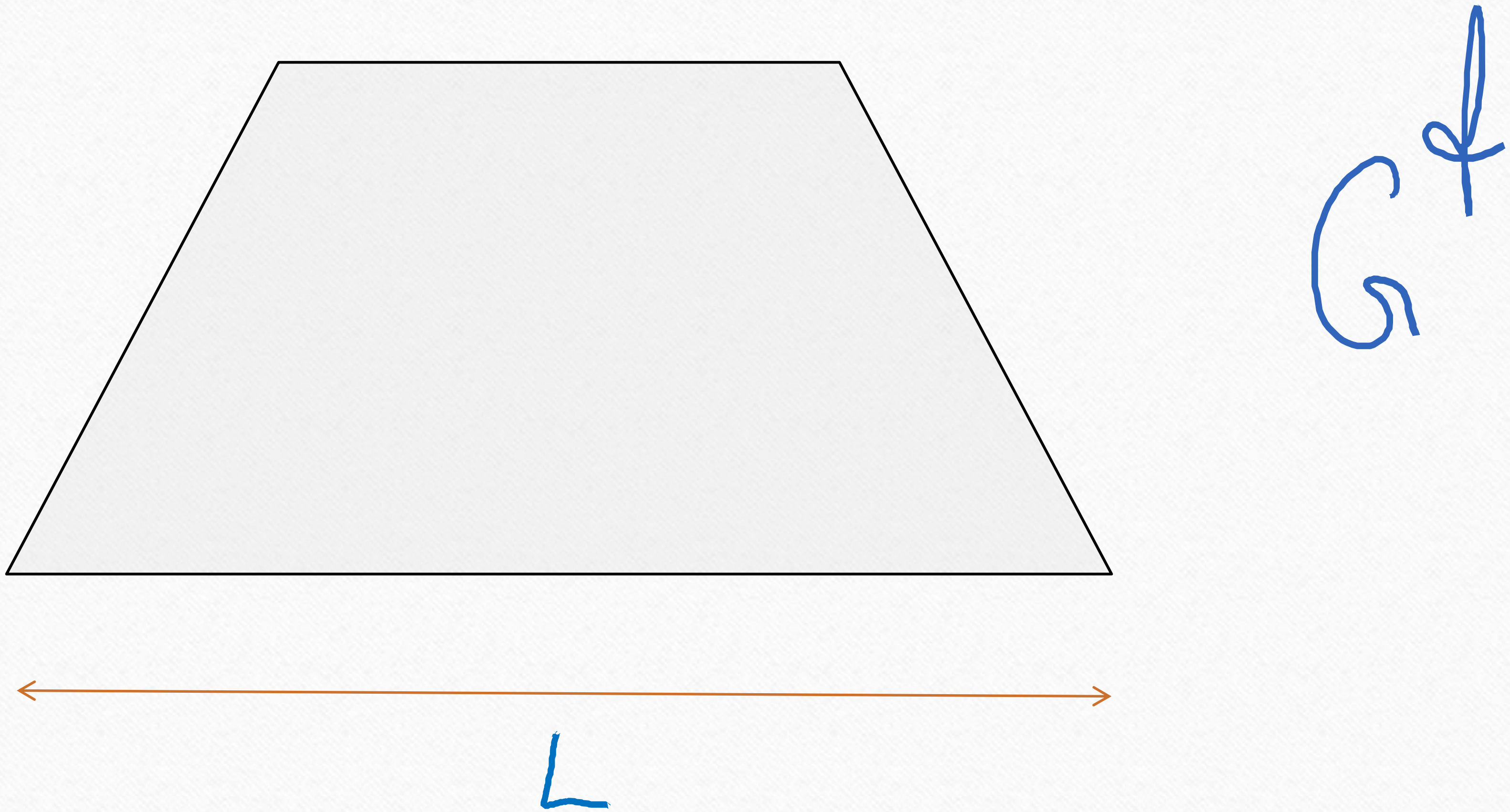
BPP^f algorithm builds circuit for f, given D

What do you see?

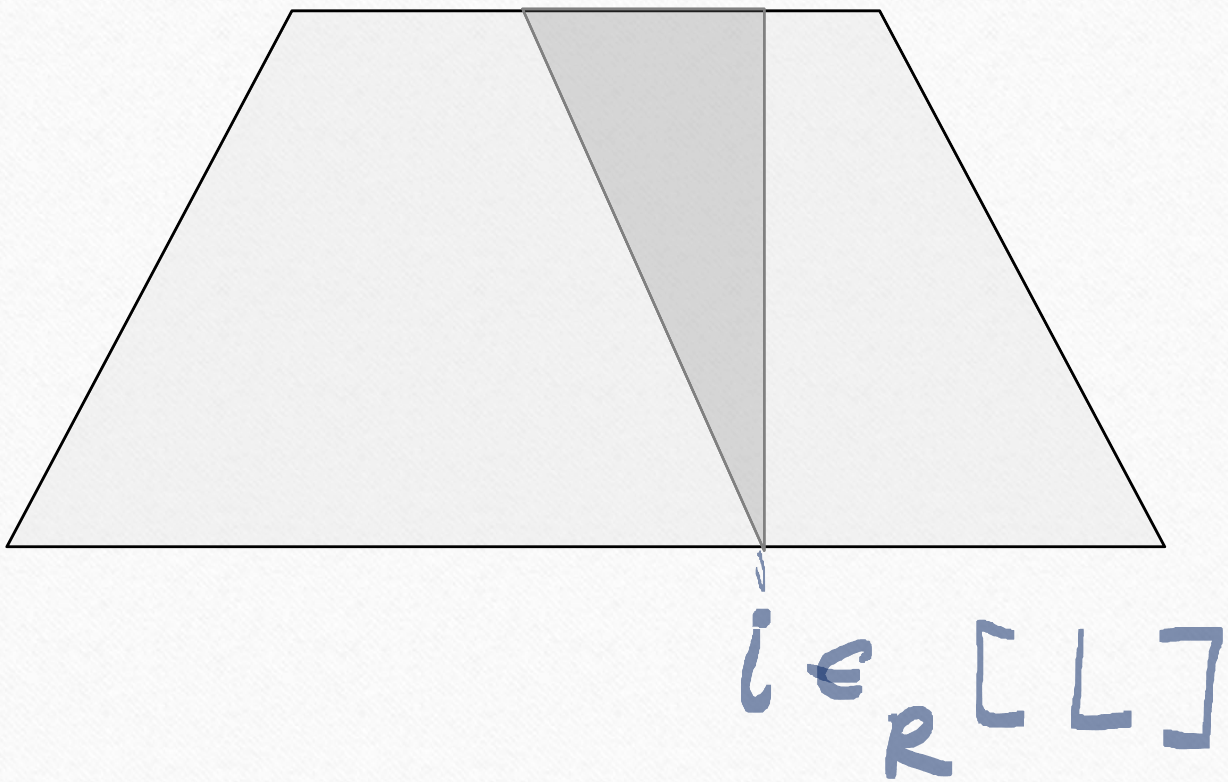
What do you see?

PRG	Learning algorithm
1. take hard f	1. take D not fooled by G^f
2. construct G^f	2. argue f is not hard
3. use G^f to fool any circuit D	BPP^f algorithm builds circuit for f, given D

Learning algorithm for f

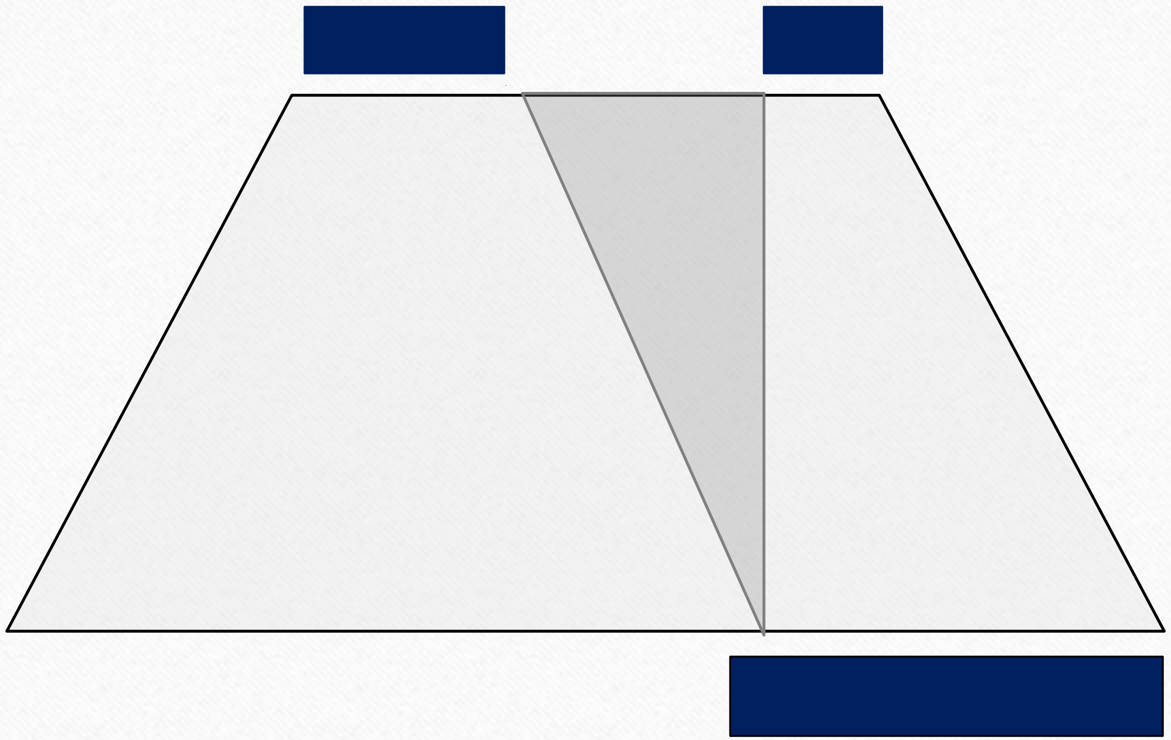


Learning algorithm for f



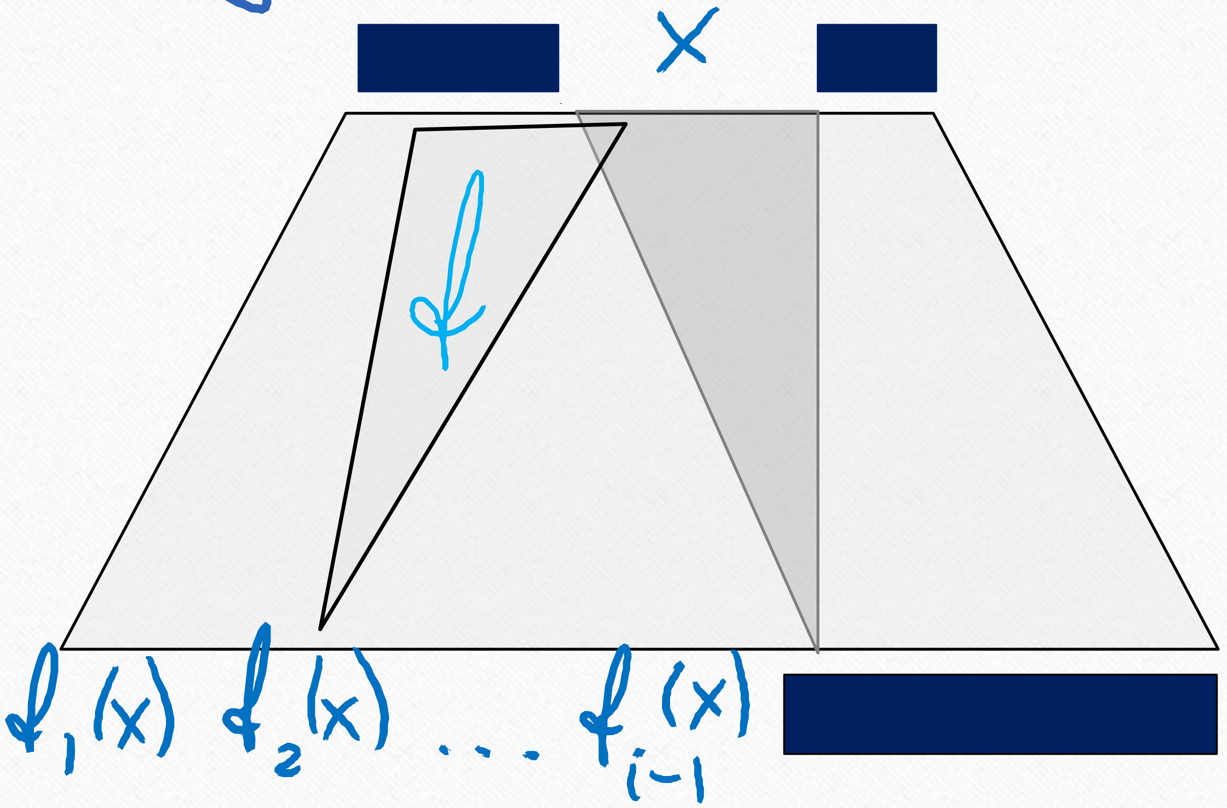
G^f

Learning algorithm for f



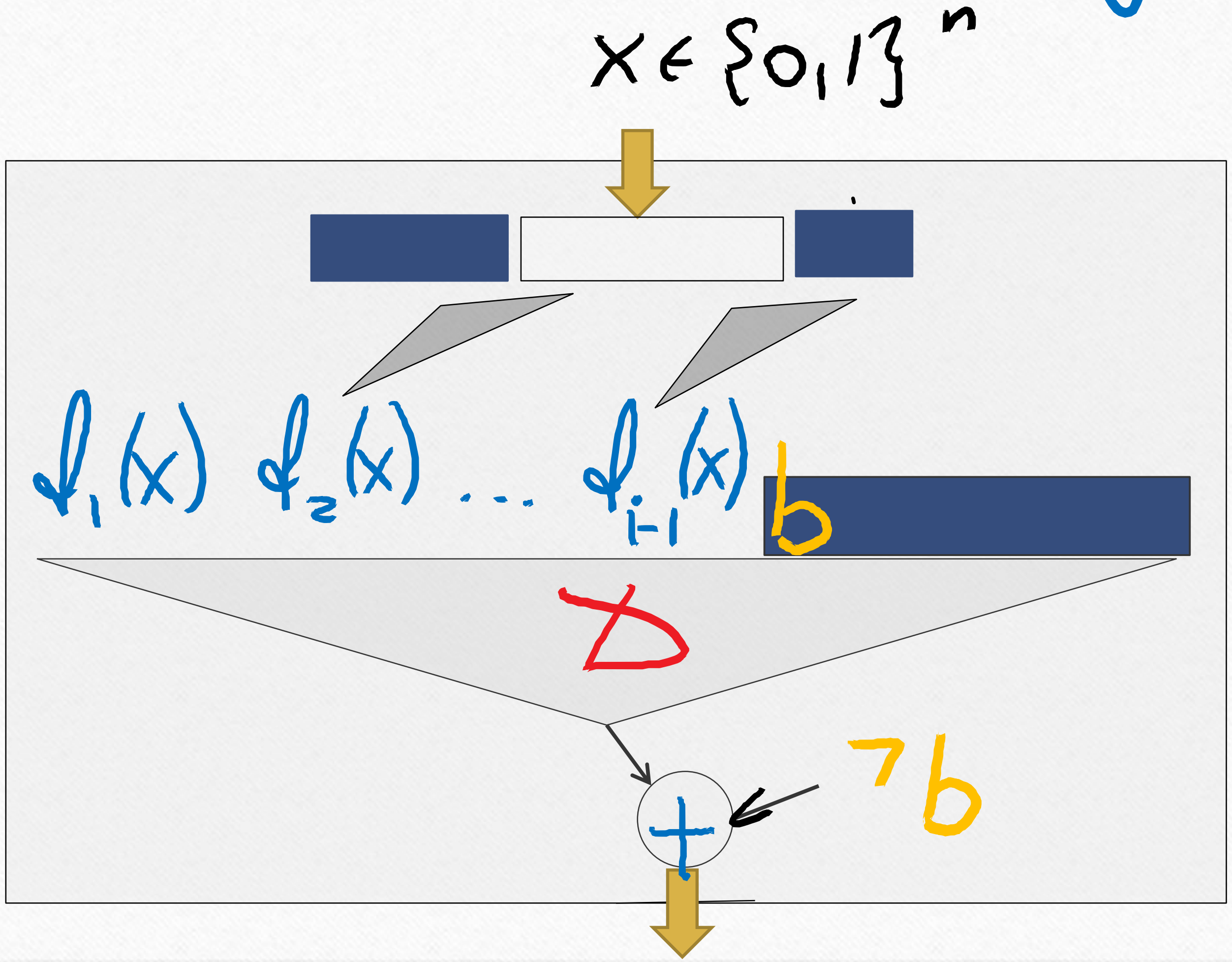
G^f

Learning algorithm for f



G^f

Circuit C for f, given distinguisher D



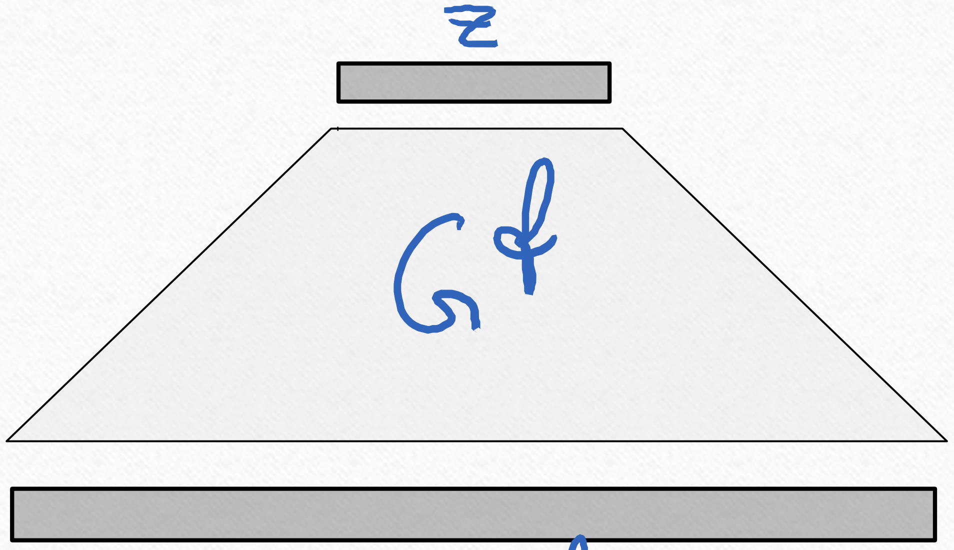
Need a Distinguisher D for G^f

"Locality" of G^f:

$f \in \text{Ckt-Size}(S) \Rightarrow$

$\forall z,\ F_z \in \text{Ckt-Size}(S + \text{poly}(n))$

[Razborov '02]



$F_z = G^f(z)$

truth table of Bool. fn

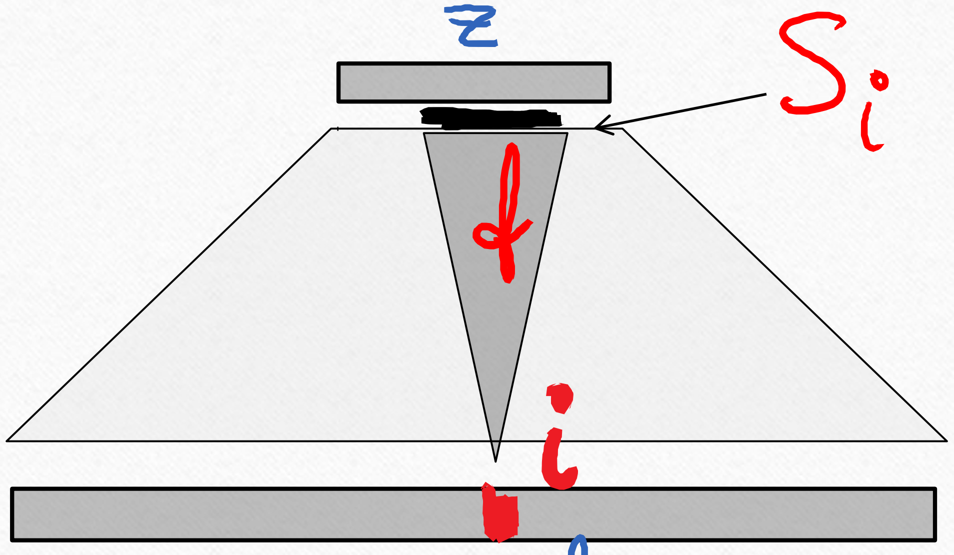
Need a Distinguisher $\mathcal{D}$ for G^f

"Locality" of G^f:

$f \in \text{Ckt-Size}(S) \Rightarrow$

$\forall z,\ F_z \in \text{Ckt-Size}(S + \text{poly}(n))$

[Razborov '02]



$F_z = G^f(z)$

truth table of Bool. fn

To get a learning algorithm for f,
it suffices to "break" G^f.
To "break" G^f,
it suffices to distinguish "easy" Boolean functions from random.

Learning circuit class $\mathcal{C}$ reduces to

Distinguishing $\mathcal{C}$-"easy" functions from random functions

To learn f,

we will make sure that

G^f is "broken"!

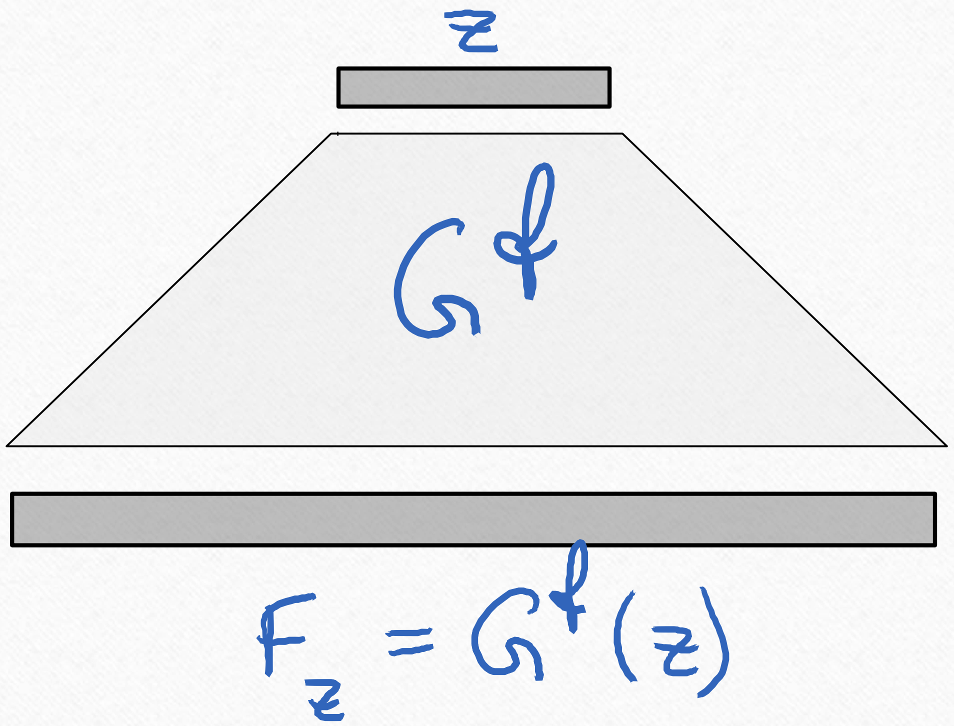
Play to Lose!

To break $G^f : \{0,1\}^{n^2} \to \{0,1\}^L$, & efficiently learn f, we choose parameter L carefully!

Role of Stretch of G^f

$f \in \text{Ckt-Size}(S) \Rightarrow$

$\forall z,\ F_z \in \text{Ckt-Size}(S + \text{poly}(n))$

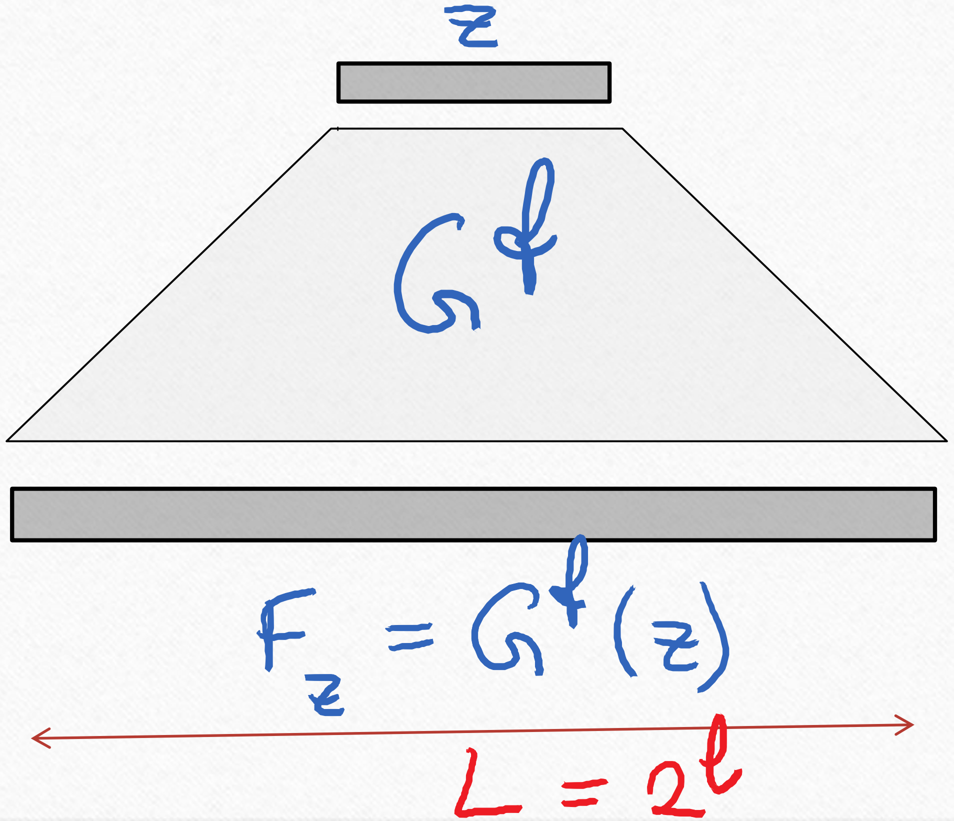


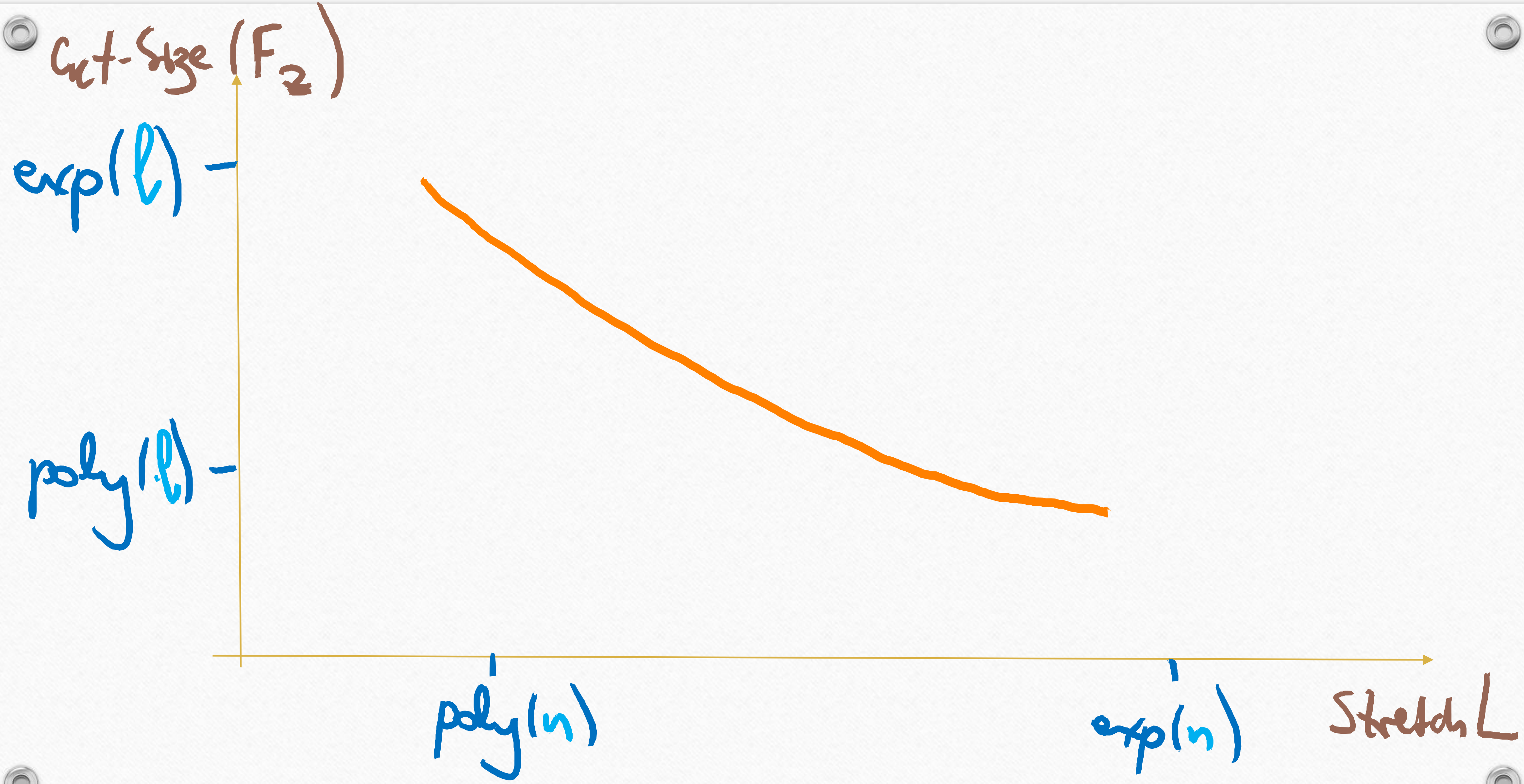
Role of Stretch of G^f

$f \in \text{Ckt-Size}(S) \Rightarrow$

$\forall z,\ F_z \in \text{Ckt-Size}(S + \text{poly}(n))$

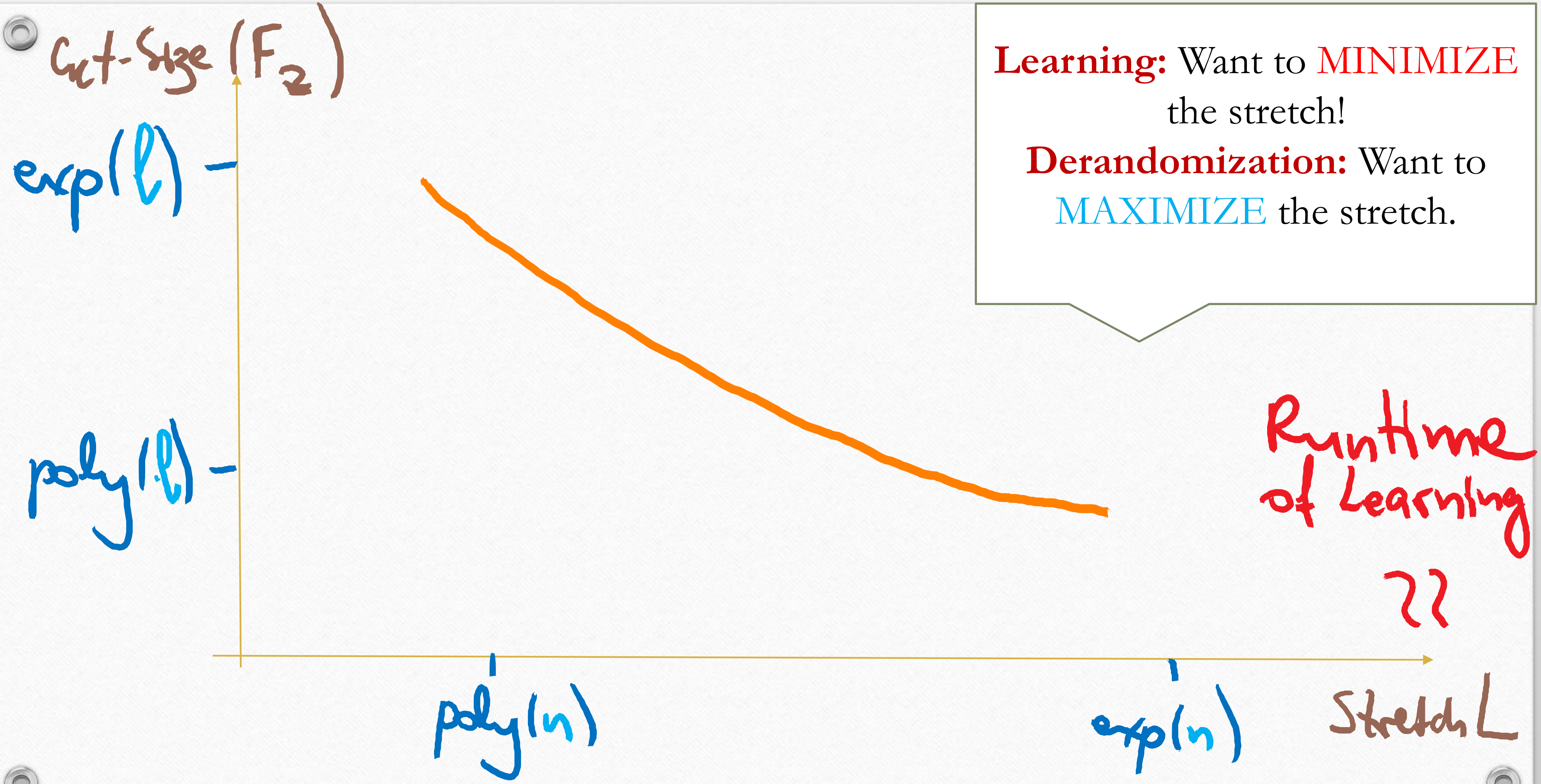
Note: $F_z : \{0,1\}^{\ell} \rightarrow \{0,1\}$





Cut-Size (F_2)
exp(ℓ)
poly(ℓ)
poly(n)
exp(n)
Stretch L

VVK1



Learning: Want to MINIMIZE the stretch!

Derandomization: Want to MAXIMIZE the stretch.

Distinguish $g:\{0,1\}^n \to \{0,1\}$ from random	Runtime of Learning algo
1. $g \in$ Ckt-Size$(2^{n/10})$	poly(n)
2. $g \in$ Ckt-Size$(2^{n^{1/10}})$	quasi-poly(n)
3. $g \in$ Ckt-Size$(n^{O(1)})$	subexp(n)

Distinguishing Easy Functions from Random Functions

Existing Circuit Lower Bounds are Constructive

... can be formalized in S_2^1 [Razborov]

Every such proof $\Rightarrow$ "Easy vs. Random fn" polytime distinguisher

To prove $f \notin \text{Ckt-Size}(S)$

Exhibit a property $\mathcal{P}$ of Boolean fns s.t.

(1) $\forall\, g \in \text{Ckt-Size}(S)$, $g \in \mathcal{P}$

(2) $f \notin \mathcal{P}$

Natural Property $\mathcal{P}$ Useful against Cut-Size(S)

(1) $\forall\, g \in$ Cut-Size(S), $g \in \mathcal{P}$

(2) $\mathcal{P}$ rejects $\geq 1/2$ of random fns

(3) $\mathcal{P}$ is polytime testable

Natural Property for $AC^0[2]$

[Razborov '87]

$f:\{0,1\}^n \to \{0,1\}$ ⇒ Boolean matrices

$$\left[A_{IJ} \right]_{\substack{I \in \binom{n}{a}, \\ J \in \binom{n}{b}}}$$

where $a = \frac{n}{2} - \sqrt{n}$,

& $0 \le b \le a$,

$A_{IJ} = \bigoplus f(x)$ over all x s.t. $x|_{I \cup J} = 0$.

Accept f ~~iff~~ $\forall A$,

$$\mathrm{rank}(A) \leq \frac{2^n}{140 \cdot n^2}.$$

Learning $AC^0[2]$

To Learn $f \in AC^0[2]$

1. Use the $AC^0[2]$ Natural Property as a distinguisher $\mathcal{D}$ for G^f
2. Run the BPP^f algorithm to learn a circuit for approximating f

To Learn $f \in AC^0[2]$

1. Use the $AC^0[2]$ Natural Property as a distinguisher $\mathcal{D}$ for G^f
2. Run the BPP^f algorithm to learn a circuit for approximating f (on only $1/2 + \varepsilon$ of all inputs)

Improved Learning of $f \in AC^0[2]$

1. Define $g(\vec{x}_1, \ldots, \vec{x}_k) = \bigoplus_{i=1}^{k} f(\vec{x}_i)$, $k = \mathrm{poly}(n)$.
2. Weakly learn g in BPP^g
   (by breaking G^g with the Natural property)
3. Strongly learn f
   (by decoding the XOR code)

Correctness

1. $f \in AC^0[2] \Rightarrow g \in AC^0[2]$
2. $g \in P^f$
3. Constructive proof of Yao's XOR Lemma

Agnostically

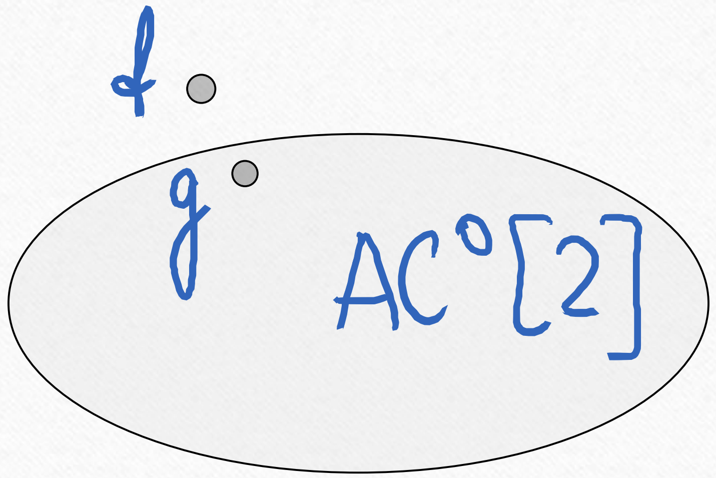
Learning

$AC^0[2]$

Agnostic Learning Algorithm

Learn a circuit C s.t.

$$\Pr_{x \sim U}[C(x) \neq f(x)] \leq \alpha \cdot \beta + \varepsilon$$



$$\Pr_{x \sim U}[f(x) \neq g(x)] \leq \beta$$

Same Approach Except

(1) Tolerant Natural Property

- accept all f "close" to $AC^0[2]$

(2) Modified NW Generator

- f "close" to $AC^0[2]$ ⇨ $G^f(z)$ "close" to $AC^0[2]$ for most z

(1) Tolerant Natural Property

- accept all f "close" to $AC^0[2]$

Existing properties are tolerant.

(2) Modified NW Generator

- f "close" to $AC^0[2]$ ⇒ $G^f(z)$ "close" to $AC^0[2]$ for most z

Add 2-wise Indepent Generator.

NW: $G^f(z) = f(z|_{S_1}), \ldots, f(z|_{S_L})$.

Generator π:

$\pi(w,1), \ldots, \pi(w,L)$ are 2-wise independent n-bit strings.

NW$^+$: $H^f(z,w)_i = f(z|_{S_i}) \oplus \pi(w,i))$

Summary

Constructive analysis of NW Generator

+

Constructive proofs of circuit lower bounds

↓

Learning algorithms for $AC^0[p]$

Main Open Question

A more intuitive/understandable learning algorithm?

Open Questions

- Natural Property useful against ACC^0?
- Learning algorithm without membership queries?
- Agnostically learning $AC^0[p]$ w/ smaller error?

It's all in Your Mind

algorithms

Correctness Proofs

Lower Bound Proofs