Alon Rosen

Cryptography and Learning

Nobody expects the Spanish Inquisition![*]

Cryptography and learning often give an algorithm access to evaluations $(x, f(x))$ of an unknown function. In *PAC learning*, the goal is to use observed evaluations to predict the function on fresh inputs. In *pseudorandomness*, the task is to *distinguish* whether an oracle implements a keyed function or a truly random one.

In learning, the target function is unknown, and the input distribution need not itself be known. In cryptography, one sometimes considers a known distribution over functions while the sampled function is hidden. Such *pseudorandom functions* (PRFs) give natural examples of functions that are information-theoretically learnable but computationally hard to learn.

This chapter develops the connection. Low-complexity PRFs yield hardness even for simple circuits and bounded-depth neural networks.

The main points of this chapter are:

- PRFs can be learnable in principle while resisting efficient learning.
- Learning targets *accuracy*; cryptography combines *correctness* and *security*.
- Low-complexity PRFs imply hardness for simple circuits and bounded-depth neural nets.

[*] Monty Python, *Monty Python's Flying Circus*.

1 Pseudorandom functions

Consider an unknown function f accessed through an oracle: an algorithm A^f submits a query x and receives $f(x)$ in reply. A random function R answers each fresh query with an independent uniform value from its range and repeats the same value on repeated queries.

Definition 1 (Pseudorandom function family)**.** *Let $\{F_s{:}\{0,1\}^n \to \{0,1\}^\ell\}_{s\in\{0,1\}^k}$ be a keyed family whose evaluation is polynomial-time computable. It is a* (t,ε)-pseudorandom function family *if, for every oracle circuit A of size at most t,*

$$\left|\Pr_s[A^{F_s}=1]-\Pr_R[A^R=1]\right|\le\varepsilon,$$

where $s\leftarrow\{0,1\}^k$ and R is a uniformly random function from $\{0,1\}^n$ to $\{0,1\}^\ell$.

By default the oracle queries of A may be adaptive. The family is a *weak PRF* if A instead receives independent pairs $(x_i, F_s(x_i))$ for uniformly random x_i.

Any statement of computational hardness depends on a model of computation and the resource being measured. Since standard models simulate one another with polynomial overhead, the particular model is usually immaterial. This permits a technology-independent treatment. Unless stated otherwise, an efficient adversary has polynomial size and succeeds with nonnegligible probability or advantage; negligible means eventually below $1/p(n)$ for every polynomial p.

2 PAC learning

In the PAC model a learner receives labeled examples $(x, f(x))$, where x is drawn from an input distribution D, and produces a hypothesis that should predict f on a fresh sample from D.

Definition 2 (PAC learning)**.** *An algorithm* PAC learns *a class of functions if, for every target f in the class, every input distribution D, and every $\varepsilon,\delta>0$, given independent pairs $(x,f(x))$ with $x\leftarrow D$, it outputs with probability at least $1-\delta$ a circuit h such that*

$$\Pr_{x\leftarrow D}[h(x)\neq f(x)]\le\varepsilon.$$

With membership queries, the learner may also ask for $f(x)$ at chosen points. Efficient learning means running time $\mathrm{poly}(n,1/\varepsilon,1/\delta)$.

The learner may be required to output a hypothesis from the target class; this is called *proper* learning. If it may output an arbitrary efficiently evaluatable hypothesis, the learning is *improper*.

PAC learning concerns supervised prediction on the training distribution. It does not directly address generation, reinforcement learning, or distribution shift.

A useful example is the class of linear equations $b = f_{\mathbf{s}}(\mathbf{a}) = \mathbf{a}^\top\mathbf{s}$, indexed by an unknown parameter $\mathbf{s}$ over a field $\mathbb{F}$. From exact labeled examples, learning reduces to solving a linear system. Once the samples determine $\mathbf{s}$, Gaussian elimination recovers it, so $f_{\mathbf{s}}$ is properly learned.

Will recovering $\mathbf{s}$ be hard if the labels are perturbed? Over $\mathbb{R}$, additive noise can still be handled by least squares. Over $\mathbb{F}_2$, perturbing a label means flipping a bit, and Gaussian elimination no longer directly recovers the parameter. We return to this case in Chapter 2.

3 Hardness of PAC learning from PRFs

PRFs give a general hardness result on learning: if a secure PRF family lies in a class, then no efficient learner can learn that class. As we will see, PRFs can be based on a broad range of cryptographic assumptions.

For a PRF family, information is not the obstacle. There are at most 2^k possible functions, so an unbounded learner can enumerate the keys and find a consistent one; $O(k/\varepsilon)$ examples suffice at constant confidence. The question is whether this can be done efficiently.

For this argument it is enough to work under the uniform distribution. Since PAC learning must succeed for every input distribution, it must in particular succeed there. If it could, we could distinguish a PRF from a random function: run the learner using the oracle and check its prediction on one fresh uniform point. This gives the following theorem, which holds even under the uniform distribution, even improperly, and even with membership queries.

Theorem 1. *If $F = \{F_s\}$ is a PRF family and $\mathcal{C}$ is any class of functions containing F, then $\mathcal{C}$ is not efficiently PAC learnable.*

Proof. Suppose an efficient learner L for $\mathcal{C}$ exists. Fix $\varepsilon, \delta > 0$ such that $(1-\varepsilon)(1-\delta) > 1/2$, and let $m = \mathrm{poly}(n)$ bound the number of examples and queries used by L. We construct a distinguisher A whose oracle $\mathcal{O}$ is either F_s for a random key s or a random function R.

Run L under the uniform input distribution. To provide a labeled example, draw a uniform x and return $(x, \mathcal{O}(x))$; forward membership queries to $\mathcal{O}$. After L outputs a hypothesis h, draw an independent uniform x^* and accept if $h(x^*) = \mathcal{O}(x^*)$.

If $\mathcal{O} = F_s$, the learner's view is exactly distributed as in the PAC experiment, so

$$\Pr[A^{F_s} \text{ accepts}] \geq (1-\delta)(1-\varepsilon).$$

If $\mathcal{O} = R$, then unless x^* coincides with one of the at most m inputs seen during learning, $R(x^*)$ is uniform and independent of $h(x^*)$. Hence

$$\Pr[A^{R} \text{ accepts}] \leq 2^{-\ell} + \frac{m}{2^n}.$$

Thus A has advantage at least

$$(1-\delta)(1-\varepsilon) - 2^{-\ell} - \frac{m}{2^n},$$

which is a positive constant for all sufficiently large n since $\ell \geq 1$. The distinguisher has polynomial size, contradicting PRF security. □

This is a worst-case impossibility for the class: it rules out a learner guaranteed to succeed on every target. Most concepts may nevertheless be efficiently learnable.

This proof illustrates a typical cryptographic reduction: a hypothetical learner is converted into a distinguisher. The key requirement is to reproduce the learner's view exactly, not merely the format of its inputs. Here, choosing the uniform distribution and answering every sample and query through the same oracle places the learner exactly in its PAC experiment.

Cryptographic statements also make the adversarial model explicit: computational power, access, and success criteria are part of the claim. The construction itself is public. For a PRF, the family $\{F_s\}$, the key distribution, and the evaluation algorithm are known; only the sampled key s is secret. This is *Kerckhoffs' principle*: security should not rely on obscurity of the system.

4 PRFs, PRGs, and OWFs

Pseudorandom generators expand a short random seed into a longer string that no efficient test can distinguish from uniform, and lead to the construction of pseudorandom functions.

Definition 3 (Pseudorandom generator)**.** *Let $\ell = \ell(k) > k$. A polynomial-time computable $G{:}\{0,1\}^k \to \{0,1\}^\ell$ is an* (t,ε)-pseudorandom generator *if for all circuits A of size at most t,*

$$\left| \Pr_{s \leftarrow \{0,1\}^k}[A(G(s)) = 1] - \Pr_{r \leftarrow \{0,1\}^\ell}[A(r) = 1] \right| \le \varepsilon.$$

A PRF gives a PRG by evaluating it on fixed distinct inputs and concatenating the outputs; distinguishing the resulting string from uniform would distinguish the PRF from random.

For the construction of PRFs, it suffices to use a length-doubling PRG. For $\ell = 2k$, write $G(s) = G_0(s) \,\|\, G_1(s)$, where $G_0(s), G_1(s) \in \{0,1\}^k$, and define

$$F_s(x_1 \cdots x_n) = G_{x_n}\big(\cdots G_{x_1}(s) \cdots\big).$$

An efficient oracle algorithm that distinguishes this family from random can be turned into an efficient algorithm that distinguishes the output of G from uniform, contradicting PRG security.

A PRG must be hard to invert: if an efficient algorithm could, given $t = G(s)$, find s' with $G(s') = t$, then checking $G(s') = t$ would distinguish generator outputs from uniform. This is a special case of *one-wayness*: an efficiently computable function f for which, given $y = f(x)$ for random x, finding any preimage is hard. The converse, constructing PRGs from arbitrary one-way functions, is a landmark result of modern cryptography.

Theorem 2. *Pseudorandom function families exist if and only if one-way functions exist.*

The equivalence is striking: pseudorandom functions emulate a random function against every efficient oracle algorithm, yet their existence follows from one-way functions. Plausible one-way function candidates are plentiful, putting the theory of pseudorandomness on strong footing.

One-wayness gives a compelling form of computational asymmetry: choose random x, publish $y = f(x)$, and keep x. We know a solution by construction, while any efficient algorithm faces a hard inversion problem. If an AGI is vastly more capable than us but remains computationally bounded, such asymmetry could preserve an advantage. It may not suffice for protection, but some form of computational asymmetry appears necessary.

5 Circuit complexity

5.1 Circuit complexity of PRFs

Theorem 1 makes PRF complexity directly relevant to learning: the simpler the circuit class supporting PRFs, the simpler a class for which efficient PAC learning is ruled out. The generic PRG-to-PRF construction above has polynomial size but depth growing with the input length because its generator evaluations are sequential.

Circuit size measures total computation, while depth measures sequential layers. The class TC^0 consists of constant-depth polynomial-size circuits with threshold gates, where a threshold gate computes $\mathbf{1}[\sum_i w_i x_i \ge \theta]$.

Theorem 3. *Under standard cryptographic hardness assumptions, there exist pseudorandom function families computable in* TC^0.

The threshold gates are important. For AC^0, the class of constant-depth polynomial-size AND/OR/NOT circuits, the situation is different.

Theorem 4. *Any PRF family computable in* AC^0 *can be broken in quasipolynomial time.*

The reason is Fourier concentration. For a depth-d, size-s AC^0 circuit, all but ε of the Fourier mass lies on coefficients of degree at most $O(\log(s/\varepsilon))^d$. Thus AC^0 functions have accurate low-degree approximations under the uniform distribution, and for polynomial s and constant d the relevant coefficients can be estimated in quasipolynomial time. This yields a quasipolynomial-time learner and, by Theorem 1, a distinguisher against any PRF family in AC^0.

5.2 Neural nets as circuits

Neural networks can be viewed as circuits with real-valued wires and gates for arithmetic and non-linear operations. This is idealized: finite-precision implementations can be represented discretely over suitable finite rings or fields, making them compatible with cryptographic operations.

A width-m layer on $x \in \mathbb{R}^n$ can be written as

$$x \longmapsto \varphi(Ax), \qquad x \in \mathbb{R}^n, \quad A \in \mathbb{R}^{m \times n},$$

where $\varphi\colon \mathbb{R} \to \mathbb{R}$ acts coordinatewise. Typical choices are ReLU, sigmoid, and the step function; biases can be absorbed by appending a constant 1 to x.

In a fixed network, learned parameters are hardwired constants; operations determine size and layers determine depth. On Boolean inputs, a step neuron $x \mapsto \mathbf{1}[\sum_i w_i x_i \geq \theta]$ is exactly a threshold gate. At finite precision, ReLU has a threshold implementation: writing a pre-activation z with sign bit s and remaining bits z_i, its output bits are $y_i = z_i \wedge \neg s = \mathbf{1}[z_i - s \geq 1]$ (and the output sign bit is 0). Smooth activations such as sigmoid can be approximated on bounded ranges by polynomials evaluated with fixed-precision arithmetic, which can likewise be implemented by threshold circuits. Thus depth-$d(n)$ networks have comparable circuit depth; fixed-depth polynomial-size threshold networks are TC^0, so the low-depth PRFs above lie in a natural neural-network class.

There is one caveat for the discussion of parity in Chapter 2. Linear algebra over $\mathbb{F}_2$ is not the same as linear computation over the reals: even addition modulo 2 on two bits, namely XOR, cannot be computed by a single threshold unit. A small network with a hidden layer can compute XOR though, so this is a restriction of a single layer rather than of neural networks in general.

Under standard assumptions, Theorem 3 places PRFs inside constant-depth polynomial-size threshold networks. Combining this with Theorem 1 gives the following consequence.

Proposition 1. *Under standard cryptographic assumptions, constant-depth polynomial-size threshold networks are not efficiently PAC learnable.*

The hardness persists even for much larger, polynomial-size learned networks. Overparameterization cannot overcome the computational obstruction posed by PRFs.

The same applies to black-box distillation: if a teacher is drawn from a PRF family, any efficient procedure producing an efficient student that agrees with it on fresh inputs would be an efficient learner. The difficulty is computational and does not apply to a white-box procedure given the teacher's internal parameters.

5.3 Are real-world neural nets in TC^0?

Asymptotic modeling is not far from deployed scale. Qwen3-235B-A22B has 94 transformer layers and 235 billion parameters; DINOv3 ViT-7B has 40 blocks and about 6.7 billion parameters. Relative to circuit size, treating these layer counts as fixed constants is reasonable. Under bounded-precision arithmetic, a fixed-depth softmax transformer can be implemented in uniform TC^0, so one forward pass is a very large constant-depth threshold circuit.

Architectural depth is not threshold-circuit depth: each transformer block expands into a constant-depth threshold subcircuit. Thus L blocks give depth $O(L)$, while width and parameter count mainly affect size; DINOv3's 40 blocks correspond schematically to $40c + O(1)$ threshold layers, not threshold depth 40.

For a vision encoder, one pass is essentially the whole computation. An autoregressive model instead outputs a next-token distribution, samples a token, and feeds it back; generating T tokens composes T passes. If T grows with the input length, the full process need not lie in TC^0.

For generic circuits, deciding whether two output distributions are statistically close is SZK-complete. Here the next-token distribution is explicit: with white-box access one can inspect every token probability rather than infer an implicit distribution from samples. This suggests that next-token distributions may be easier to understand than output distributions of general circuits.

6 Historical notes

Valiant introduced PAC learning [Val84]; Kearns and Valiant developed the cryptographic limitations used in Theorem 1 [KV94]; and Minsky and Papert established the single-threshold limitation for parity [MP69]. Golowich, Moitra and Rohatgi gave a cryptographic separation between reinforcement learning and associated supervised regression [GMR24].

Goldreich, Goldwasser and Micali constructed PRFs from PRGs [GGM86], while Håstad, Impagliazzo, Levin and Luby obtained PRGs from arbitrary one-way functions [HILL99]. The low-depth PRFs of Theorem 3 are from [NR04, BPR12], and Theorem 4 uses the Fourier concentration of [LMN93]; see [BR17] for a survey.

Subsection 5.3 follows Chiang's bounded-precision transformer analysis [Chi24]; Qwen3 and DINOv3 provide the scale examples [Y+25, S+25], and the circuit-output discussion uses Statistical Difference [SV03].

References

[BPR12] A. Banerjee, C. Peikert, and A. Rosen. Pseudorandom functions and lattices. In *EUROCRYPT*, 2012.

[BR17] A. Bogdanov and A. Rosen. Pseudorandom functions: Three decades later. In *Tutorials on the Foundations of Cryptography*, pages 79–158. 2017.

[Chi24] D. Chiang. Transformers in uniform TC^0. arXiv:2409.13629, 2024.

[GGM86] O. Goldreich, S. Goldwasser, and S. Micali. How to construct random functions. *JACM*, 33(4):792–807, 1986.

[GMR24] N. Golowich, A. Moitra, and D. Rohatgi. Exploration is harder than prediction: Cryptographically separating reinforcement learning from supervised learning. In *FOCS*, pages 1953–1967, 2024.

[HILL99] J. Håstad, R. Impagliazzo, L. A. Levin, and M. Luby. A pseudorandom generator from any one-way function. *SICOMP*, 28(4):1364–1396, 1999.

[KV94] M. Kearns and L. G. Valiant. Cryptographic limitations on learning Boolean formulae and finite automata. *JACM*, 41(1):67–95, 1994.

[LMN93] N. Linial, Y. Mansour, and N. Nisan. Constant depth circuits, Fourier transform, and learnability. *JACM*, 40(3):607–620, 1993.

[MP69] M. Minsky and S. Papert. *Perceptrons: An Introduction to Computational Geometry*. MIT Press, 1969.

[NR04] M. Naor and O. Reingold. Number-theoretic constructions of efficient pseudo-random functions. *JACM*, 51(2):231–262, 2004.

[S+25] O. Siméoni et al. DINOv3. arXiv:2508.10104, 2025.

[SV03] A. Sahai and S. Vadhan. A complete problem for statistical zero knowledge. *JACM*, 50(2):196–249, 2003.

[Val84] L. G. Valiant. A theory of the learnable. *CACM*, 27(11):1134–1142, 1984.

[Y+25] A. Yang et al. Qwen3 technical report. arXiv:2505.09388, 2025.