

# How Does Machine Learning Manage Complexity?

Lance Fortnow\*

Illinois Institute of Technology

## Abstract

We provide a computational complexity lens to understand the power of machine learning models, particularly their ability to model complex systems.

Machine learning models are often trained on data drawn from sampleable or more complex distributions, a far wider range of distributions than just computable ones. By focusing on computable distributions, machine learning models can better manage complexity via probability. We abstract away from specific learning mechanisms, modeling machine learning as producing  $P/poly$ -computable distributions with polynomially-bounded max-entropy.

We illustrate how learning computable distributions models complexity by showing that if a machine learning model produces a distribution  $\mu$  that minimizes error against the distribution generated by a cryptographic pseudorandom generator, then  $\mu$  must be close to uniform.

## 1 Introduction

Ignorance of the different causes involved in the production of events, as well as their complexity, taken together with the imperfection of analysis, prevents our reaching certainty about the vast majority of phenomena. Thus there are things that are uncertain for us, things more or less probable, and we seek to compensate for the impossibility of knowing them by determining their different degrees of likelihood. So it was that we owe to the weakness of the human mind one of the most delicate and ingenious of mathematical theories, the science of chance or probability.

– Pierre-Simon Laplace [Lap91]

Modern machine learning has modeled complex behavior in ways that go far beyond our traditional logic approaches, making tremendous advances in language translation, computer vision, Chess and Go playing, protein folding and automated programming to name just a few.

In this paper we give a complexity lens to machine learning models, abstracting the model away from the specific technologies. We then argue that paradoxically these models derive their strength from the weakness of their distributions, modeling complex behavior by probabilistic outcomes, much the way Laplace has argued that humans do the same.

---

\*Much of this work was done while a visiting fellow at Magdalen College, University of Oxford.

In Section 4, we review the argument that describing a string with a simple distribution leads to minimizing the Kullback-Leibler of the original distribution  $D$  with a learned computable distribution  $\mu$ .

In Section 5, we abstract the mechanisms of machine learning to model the output of a machine learning model as a  $P/poly$ -computable distribution with polynomially-bounded max entropy. We make separate arguments for the various aspects.

- Computable distributions come from viewing next-token prediction as P-computable conditional probability functions which are equivalent to P-computable distributions.
- We can achieve polynomial time even though neural nets have low depth, through either reasoning or coding.
- The heavy use of non-uniformity, expressed through the weights of the model, captures data, time and capabilities.
- Finally, we argue that the softmax function guarantees polynomially-bounded max entropy, i.e., that every outcome has at least exponentially small probability of occurring.

In Section 6, we present our main theorem. Let  $\mu$  minimize  $\text{KL}(D||\mu)$  over the set of  $P/poly$ -computable distributions with polynomially-bounded max entropy, where  $D$  is a distribution generated by a cryptographic pseudorandom generator. We show this yields  $\text{KL}(U||\mu) < 1/n^c$ , for all  $c > 0$ , where  $U$  is the uniform distribution. While  $D$  and  $U$  are computationally indistinguishable, when you learn a computable distribution  $\mu$  from  $D$  then  $\mu$  and  $U$  are information-theoretically indistinguishable.

If we remove the max entropy condition, we can still show that  $U$  and  $\mu$  are statistically close.

We put it all together in Section 7, arguing that the power of modern machine learning models comes from both its ability to generate random guesses instead of specific answers, and by limiting the hypotheses to computable distributions, forces the model to represent complex behavior by probabilities.

## 2 Previous Work

Theoretical computer science has studied learning since the early days of the field. In 1967, Mark Gold [Gol67] developed inductive inference and in the 1980s Leslie Valiant [Val84] described PAC (Probably Approximately Correct) learning. Both these paradigms focused on learning deterministic algorithms and not probabilities. Kearns and Valiant [KV94] showed that learning circuits in general in the PAC model would break cryptography.

Shuichi Hirahara and Mikito Nanashima [HN23] show that in a world without cryptography one can learn certain distributions well. However, their work only learns distributions with relatively small initial state, where we want a theory that handles complex distributions.

This paper tries to distill ideas from a vast literature on machine learning to abstract to a relatively simple model of machine learning that we can directly study. In each of the sections that follow, we will cite some of the seminal work that shaped the author's thinking for this paper.

### 3 Preliminaries

This paper uses ideas, concepts and tools from information theory, Kolmogorov complexity and cryptography. For details on the results mentioned in this section and general background, the author recommends the textbooks of Cover and Thomas [CT06], Li and Vitányi [LV09] and Goldreich [Gol01] respectively.

A function  $f(n)$  is negligible if  $f(n) = o(1/n^k)$  for all  $k$ . We abuse notation to say  $f(n) \leq \text{negl}(n)$  if  $f(n)$  is negligible. All logarithms are taken base 2.

#### 3.1 Nonuniform Circuits

Every polynomial-time computable language  $L$  can be computed by a family  $C_0, C_1, \dots$  of polynomial-size circuits. In this framework,  $C_n$  has  $n$  inputs, and the number of gates of  $C_n$  is bounded by  $n^c$  for some constant  $c$ . A string  $x = x_1 \dots x_n$  is in  $L$  if and only if  $C_n(x_1, \dots, x_n)$  outputs 1.

The converse is not true; there are languages  $L$  that have polynomial-size circuits but cannot be computed in polynomial time. This is because each  $C_n$  might not have any relation to  $C_m$  for  $n \neq m$ . We can achieve equivalence by adding a uniformity condition, requiring an easily computable function  $f$  such that  $f(1^n) = C_n$ . Alternatively, we can keep the nonuniformity to define a larger class called  $P/poly$ .

#### 3.2 Information Theory

In this paper all distributions will be over  $\{0, 1\}^n$ . We state the definitions and known relationships for polynomial-time but they all hold if we allow nonuniformity as well.

A distribution  $\mu$  is *polynomial-time sampleable* if there is a probabilistic polynomial-time algorithm  $A$  such that  $\mu(x)$  is the probability that  $A(1^n) = x$ .

A distribution  $\mu$  is *polynomial-time computable* if the CDF function  $f(x) = \sum_{y \leq x} \mu(y)$  is computable in polynomial time.

If  $\mu$  is polynomial-time computable then it is sampleable and  $\mu(x) = f(x) - f(x - 1)$  is also computable in polynomial time.

We also consider  $P/poly$  computable and sampleable distributions where we replace polynomial-time by non-uniform polynomial time. The same relationships hold.

If there are cryptographic one-way functions then there are  $P/poly$ -sampleable distributions that are not  $P/poly$ -computable, for example consider the distribution  $(f(x), x)$  where  $f$  is a one-way function and  $x$  is chosen at random. This distribution is sampleable and if it were computable we could find  $x$  from  $f(x)$ .

For a distribution  $\mu$ , the entropy of  $\mu$ , denoted  $H(\mu)$ , is  $\sum_{x \in \{0,1\}^n} \mu(x) \log 1/\mu(x)$ . The max-entropy of  $\mu$  is  $\max_x \log 1/\mu(x)$ . Polynomial-bounded max-entropy means the max-entropy is bounded by  $n^k$  for some  $k$ , equivalent to saying  $\mu(x) \geq 2^{-n^k}$ .

For two distributions  $\mu$  and  $\tau$  the KL-divergence of  $\tau$  from  $\mu$  is

$$\text{KL}(\tau \parallel \mu) = \mathbb{E}_{x \sim \tau} [\log(\tau(x)/\mu(x))]$$

The KL-divergence is not symmetric but it is always nonnegative and equal to zero only when  $\tau = \mu$ . It measures how well the distribution  $\mu$  models  $\tau$  as seen by the cross-entropy decomposition

$$\text{KL}(\tau\|\mu) = \mathbb{E}_{x\sim\tau}[\log(1/\mu(x))] - H(\tau).$$

We can also measure the statistical difference between  $\tau$  and  $\mu$  as

$$\Delta(\tau, \mu) = \frac{1}{2} \sum_x |\tau(x) - \mu(x)|$$

which is symmetric, nonnegative and zero only if  $\tau = \mu$ .

Pinsker's inequality allows us to bound the statistical difference from the KL-divergence.

$$\Delta(\tau, \mu) \leq \sqrt{\frac{1}{2} \text{KL}(\tau\|\mu)}$$

The other direction does not hold as the KL-divergence can get very large if  $\mu(x) \ll \tau(x)$  for some  $x$ , even infinite if for some  $x$ ,  $\tau(x) > \mu(x) = 0$ .

### 3.3 Cryptography

A *cryptographic pseudorandom generator* is a function  $G$  whose output on uniform inputs is not efficiently distinguishable from a uniform distribution [Yao82]. Formally,  $G$  takes seeds of length  $\ell$  to outputs of length  $n > \ell$  such that for all probabilistic P/poly algorithms  $A$ ,

$$\left| \Pr_{s \in \{0,1\}^\ell} (A(G(s)) = 1) - \Pr_{r \in \{0,1\}^n} (A(r) = 1) \right| \leq \text{negl}(n).$$

Håstad, Impagliazzo, Levin and Luby [HILL99] show that one can create a cryptographic pseudorandom generator from any one-way function.

### 3.4 Kolmogorov Complexity

For a string  $x$ , the polynomial-time Kolmogorov complexity of  $x$ ,  $K(x)$ , is the length of the shortest program that outputs  $x$ . We also consider  $K^{\text{poly}}(x)$  where the program runs in time a fixed polynomial in  $n = |x|$ . We will be using Kolmogorov complexity informally in this paper so we will ignore quantifiers on the polynomials and small additive factors in the inequalities.

## 4 Minimizing Kullback-Leibler Divergence

In this section we show that finding a  $\mu$  that minimizes the Kullback-Leibler divergence of a given distribution  $D$  to  $\mu$  arises naturally from the goal of finding the simplest explanation for  $D$ . This section is based on ideas from Jorma Rissanen [Ris78], Ray Solomonoff [Sol64], and Ming Li and Paul Vitányi [VL00].

For any finite set  $S$ ,  $K(x) \leq K(S) + \log |S|$ , i.e., you can describe  $x$  by the description of a set and the index of  $x$  in that set.

If we have equality then  $x$  is a random element of  $S$ . We can achieve equality with  $S = \{x\}$ , but consider the simplest set  $S$  (minimizing  $K(S)$  which maximizes  $|S|$ ) where we have equality. That breaks  $x$  into a structured part, the description of  $S$ , and a random part, the index of  $x$  in  $S$ . By an Occam's razor argument, the description of  $S$  is the best explanation for  $x$ .

There is no efficient or even computable procedure for finding the best  $S$ . We'll now shift our focus to polynomial time and distributions.

Let  $\mu$  be a P-computable distribution with CDF function  $f(x)$ . Then for all  $x$ ,

$$K^{poly}(x) \leq \log\left(\frac{1}{\mu(x)}\right) + K^{poly}(\mu)$$

up to small additive factors. There is some integer  $w$  such that  $f(x-1) < w/2^k \leq f(x)$  where  $k = 1 + \lceil \log(1/\mu(x)) \rceil$ . Given a description of  $\mu$ ,  $k$  and  $w$  we can find  $x$  using binary search.

If we have equality, then  $\mu$  is a structured part of which  $x$  is a random example, i.e., random in  $\mu$ . We can get equality by a  $\mu$  that puts all its weight on  $x$ . We want to find the simplest  $\mu$ , i.e., that minimizes  $K(\mu)$  while achieving near equality.

Suppose  $x$  comes from a distribution  $D$  and we want to create the distribution  $\mu$  before  $x$  is drawn. Taking expected values of both sides

$$\mathbb{E}_{x \sim D}(K^{poly}(x)) \leq \mathbb{E}_{x \sim D}\left(\log \frac{1}{\mu(x)}\right) + K^{poly}(\mu) = H(D) + \text{KL}(D \parallel \mu) + K^{poly}(\mu)$$

by the cross-entropy decomposition.

Now  $H(D) \leq \mathbb{E}_{x \sim D}(K^{poly}(x))$  by the Shannon coding theorem [Sha48] giving

$$\text{KL}(D \parallel \mu) + K^{poly}(\mu) \geq 0.$$

By the Occam's razor principle, we can get the best model for  $D$  by finding the P-computable  $\mu$  that minimizes  $\text{KL}(D \parallel \mu) + K^{poly}(\mu)$ .

Let's connect this with neural nets. The value  $K^{poly}(\mu)$  is dominated by the number of weights in the model, a hyperparameter to be optimized. Fixing that size, the model will try to find the weights that minimize  $\text{KL}(D \parallel \mu)$ , equivalent to minimizing the cross-entropy loss, the standard training objective of neural nets.

## 5 Neural Nets and Computable Distributions

In this section, we argue that neural nets are roughly equivalent to P/poly-computable distributions with polynomially-bounded max entropy. We examine each aspect, computable distribution, polynomial-time, non-uniformity and max entropy, below.

The input length  $n$  corresponds to the size of the context window, typically about one million tokens for frontier models at the time of this writing [Ant26]. Each token averages about 3-4 bytes.

## 5.1 Computable Distributions

Neural nets predict the next token based on the tokens seen so far (the so-called “stochastic parrots” [BGMMS21]). Learning the next token function makes the learning process tractable. If we tried to train on entire strings, they would occur with such low probability that we wouldn’t have enough samples to make reasonable hypotheses. In this section, we show that learning next-token predictors means we limit our hypotheses to computable distributions.

The transformer model [VSP<sup>+</sup>17] gives neural nets the ability to see its input in order. Mathematically then we can describe these next token predictors as conditional probability functions. If we stick to the binary regime, a *conditional probability function* for a distribution  $\mu$  on  $n$  bits is a function  $f$  from  $\{0, 1\}^{<n}$  to a value  $p \in [0, 1]$  such that for any  $y$  in  $\{0, 1\}^{<n}$ ,  $f(y)$  is the conditional probability of the next bit being 1 given the prefix  $y$ . Formally,

$$f(y) = \Pr_{x \sim \mu} [x_{|y|+1} = 1 \mid x_{1..|y|} = y].$$

In terms of the probabilities assigned by  $\mu$ ,

$$f(y) = \frac{\mu(y1)}{\mu(y0) + \mu(y1)},$$

whenever  $\mu(y0) + \mu(y1) > 0$ . Here  $\mu(z)$  denotes the probability that the string drawn from  $\mu$  begins with the prefix  $z$ .

Neural nets output tokens instead of bits but we can think of them just outputting one bit at a time.

If we don’t restrict the complexity of  $f$ , then conditional probability functions can represent any distribution. In this section we explore what happens when we do restrict the complexity.

A *P/poly-computable conditional probability function* is a conditional probability function where the function  $f$  is computable in *P/poly*.

A distribution  $\mu$  is polynomial-time computable exactly when  $\mu$  can be described by a polynomial-time computable conditional probability function. We give a formal proof of this connection in the Appendix (Theorem A.1). The same holds for *P/poly-computable distributions* and *P/poly-computable conditional probability functions* with a similar proof.

In a large language model, we need to output a sequence based on a prompt, so we need a notion of conditionally sampleable.

A distribution  $\mu$  is *P/poly-conditionally sampleable* if there is a *P/poly-computable* probabilistic algorithm  $A$  such that  $A(1^n, y)$  outputs  $x$  with probability  $\Pr_{\mu}(x \mid y \sqsubseteq x)$  if  $y \sqsubseteq x$  ( $y$  is a prefix of  $x$ ) and zero otherwise.

You can computably sample from a conditional probability function by starting at  $y$  and choosing the next bits according to the prescribed probabilities. So every *P/poly-computable distribution* is *P/poly-conditionally sampleable*.

The converse isn’t likely true exactly, but we can approximate the values of the conditional probability function from a conditionally-sampleable distribution, so they are roughly equivalent.

In this paper, we assume that machine learning produces *P/poly-computable distributions*, both because next token predictors act this way and because we need such distributions if we want to give answers to inputs.

## 5.2 Polynomial-Time

We need to argue that we can consider P-computable distributions when the frontier machine learning models have low depth, typically about a hundred layers. The neural nets by themselves do not have large depth, but the frontier models allow reasoning and executing code, both of which allow us to achieve polynomial depth. This section builds on the chain-of-thought reasoning described by Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed Chi, Quoc V Le and Denny Zhou [WWS<sup>+</sup>22].

### Reasoning

Reasoning models allow neural nets to write down information and then take next steps based on what they wrote down. Alan Turing [Tur36] defined his eponymous machine after considering how a then-human computer processed information. Thus a reasoning model acts like a Turing machine.

More formally, the next state of a Turing machine can be computed from the current state via a small-depth circuit, even in  $NC^0$ . So a neural net could via reasoning in polynomial-time compute any polynomial-time function. Keeping all these configurations will take up a large part of the context window but we only need to remember the last one, similar to the way some models compress the discussion so far so they can continue their work without going past the context window.

### Coding

More directly, neural nets can write and execute their own code. For example, neural nets are notoriously bad at arithmetic operations like multiplying large numbers, but they know how to write code to multiply, not unlike how humans would tackle the same task.

A neural net could simply write code that simulates a polynomial-size circuit based on its own weights.

## 5.3 Nonuniformity

In computational complexity, nonuniformity allows having different programs for inputs of different lengths. The concept of nonuniformity comes up often in complexity, most notably for circuit complexity and pseudorandomness. Nonuniformity makes it more difficult to prove lower bounds as it can get in the way of straightforward diagonalization. For example, we know P, polynomial-time computable languages, are strictly contained in EXP, languages computable in exponential time, but for all we can prove, non-uniform polynomial-time could contain non-deterministic exponential time.

Nonuniformity was historically seen more as a technicality than as something that gave circuits significantly more power. Nearly all programs used in practice are highly uniform with a single algorithm that works on all input lengths. For example, standard algorithms for sorting, shortest path and matching work on every input length without modification. Rarely did we use nonuniformity in our programs, other than for random bits for randomized algorithms.

Machine learning puts a new face on uniformity. Machine learning typically works in two phases: the *training phase* where a model's weights are set based on data, and the *inference phase* where we use the model with its weights to make predictions based on new data. These weights are the nonuniform advice seemingly needed for problem solving. The latest frontier models are at their core neural nets with about a trillion parameters. Let's consider the input length to be the size of the context window, about a million tokens. The weights are somewhere between a square and a cube of the input length, like polynomial advice.

We have to be careful talking about nonuniformity and running time when working at a fixed length. Nevertheless there are good reasons why we should consider the weights as nonuniform advice.

### 1. Running Time

The inference phase typically runs quickly, a few seconds, or minutes in a reasoning model. A fully trained model is a straightforward circuit and computing its outcome can be done quickly, especially in parallel in large data centers. On the other hand, the training phase, where the model learns the weights via gradient descent and back propagation, often takes months training on hundreds of trillions of tokens. Parallelizing the training phase is possible but tricky and limited. It would be highly infeasible to run a training phase every time we want to do inference. It would be a stretch to say the training phase takes time exponential in the input length, but at best it is a much larger polynomial than the running time of the inference phase.

### 2. Data

As mentioned above, the neural net is trained on hundreds of trillions of tokens. The trained weights of the neural net directly depend on the data; in fact, the goal of the training is to be able to simulate that and similar data going forward. The weights form a sort of lossy compression of the data. So the trained weights are a nonuniform representation of the data.

### 3. Capabilities

There's something larger going on, though a bit harder to capture mathematically. As these models train on more and more data, their capabilities grow. The language seems more natural, the models reason and problem solve better and hallucinate less, as measured by various benchmarks. The new weights, i.e., the updated nonuniformity, create an algorithm that actually computes in a different way. So the computing itself actually improves as you update the nonuniformity in the circuits.

While the learning phase creates a non-uniform distribution from a potentially non-uniform distribution, the learning process itself is typically uniform.

## 5.4 Max Entropy

The KL-divergence can give a large penalty if the learned distribution puts very low weight on strings that have much larger weight on the true distribution.

Recalling the definition of KL-divergence of  $D$  from  $\mu$ ,

$$\text{KL}(D\|\mu) = \mathbb{E}_{x \sim D}[\log(D(x)/\mu(x))].$$

If for some  $x$ ,  $\mu(x) \ll D(x)$  then  $\text{KL}(D\|\mu)$  can be quite high, even infinite if  $D(x) > 0$  while  $\mu(x) = 0$ .

To avoid this issue, we would like to give our learned distribution a minimum weight, equivalently a bounded max entropy. Typical learning algorithms cooperate with us, probably because they try to minimize KL-divergence.

Recall the maximum entropy of a distribution  $\mu$  is

$$\max_x(\log(1/\mu(x)))$$

So a distribution  $\mu$  has polynomially-bounded max-entropy if there is some  $k$  such that  $\mu(x) \geq 2^{-n^k}$  for all  $x \in \{0, 1\}^n$  which is the requirement for Theorem 6.1.

Neural nets, in the abstract, never give zero weight to any output. Typically a neural net will output token  $i$  with probability

$$p_i = \frac{e^{z_i}}{\sum_{j=1}^V e^{z_j}}$$

The  $z_i$  are called *logits* computed by the function

$$z_i = w_i \cdot h + b_i$$

where  $h$  is a vector representation of the input computed by the neural net,  $w_i$  are vectors of additional learned weights and  $b_i$  is a bias scalar. If all these values are bounded by a polynomial, as they typically are in neural nets, then the output distribution has polynomially-bounded max entropy. This even holds if there are exponentially many output tokens.

## 6 Complexity as Randomness

As we showed in Section 5, a trained machine learning model generates a computable distribution. Informally, our world creates distributions generated through a series of complex interactions of random and unpredictable events. Restricting to computable distributions forces the machines to treat this complexity as random events.

To illustrate this point consider the distribution  $D$  generated by a cryptographic pseudorandom generator and  $U$  a uniform distribution. The distribution  $D$  is a *P/poly*-sampleable distribution and an easy calculation shows that  $\text{KL}(D\|U)$  is  $n$  minus the key size of the generator. While  $D$  and  $U$  are indistinguishable by *P/poly* algorithms, they are far apart as distributions in an information theoretic sense.

We show that if *P/poly*-computable  $\mu$  is optimally learned being trained on  $D$ , then  $\mu$  is close to uniform in the information theoretic sense. Specifically we show that if  $\mu$  is a *P/poly*-computable distribution with polynomially-bounded max entropy such that the Kullback-Leibler divergence of

$D$  from  $\mu$  is at most its divergence from the uniform distribution, then the divergence of the uniform distribution from  $\mu$  is small. If we remove the max-entropy requirement, we still show that  $\mu$  is statistically close to uniform.

**Theorem 6.1.** *Let  $D$  be the output distribution of a cryptographic pseudorandom generator on  $\{0, 1\}^n$ , and let  $U$  denote the uniform distribution on  $\{0, 1\}^n$ .*

*Suppose  $\mu$  is a P/poly-computable distribution with polynomially-bounded max entropy such that  $\text{KL}(D\|\mu) \leq \text{KL}(D\|U)$ . Then  $\text{KL}(U\|\mu)$  is negligible in  $n$ , i.e.,  $\text{KL}(U\|\mu) \leq \frac{1}{n^c}$  for all  $c > 0$ .*

*Proof.* Define

$$f_\mu(x) = \log \frac{2^{-n}}{\mu(x)} = \log \frac{1}{\mu(x)} - n.$$

Since  $\mu$  has polynomially-bounded max entropy there is some  $k$  such that  $\log \frac{1}{\mu(x)} \leq n^k$  and  $-n \leq f_\mu(x) \leq n^k - n$  for all  $x \in \{0, 1\}^n$ .

$$\mathbb{E}_{x \sim U}[f_\mu(x)] = \sum_{x \in \{0, 1\}^n} 2^{-n} \log \frac{2^{-n}}{\mu(x)} = \text{KL}(U\|\mu).$$

On the other hand,

$$\mathbb{E}_{x \sim D}[f_\mu(x)] = \sum_{x \in \{0, 1\}^n} D(x) \log \frac{2^{-n}}{\mu(x)}.$$

Using

$$\text{KL}(D\|\mu) = \mathbb{E}_{x \sim D}[\log \frac{1}{\mu(x)}] - H(D) \quad \text{and} \quad \text{KL}(D\|U) = \mathbb{E}_{x \sim D}[\log \frac{1}{2^{-n}}] - H(D).$$

we obtain

$$\text{KL}(D\|\mu) - \text{KL}(D\|U) = \mathbb{E}_{x \sim D}[\log \frac{1}{\mu(x)}] - \mathbb{E}_{x \sim D}[\log \frac{1}{2^{-n}}] = \mathbb{E}_{x \sim D}[\log \frac{2^{-n}}{\mu(x)}] = \mathbb{E}_{x \sim D}[f_\mu(x)]$$

and thus

$$\mathbb{E}_{x \sim D}[f_\mu(x)] = \text{KL}(D\|\mu) - \text{KL}(D\|U) \leq 0.$$

by the assumption that  $\text{KL}(D\|\mu) \leq \text{KL}(D\|U)$ .

Because  $D$  is pseudorandom, it is computationally indistinguishable from  $U$ . Since  $f_\mu$  is P/poly-computable and polynomially bounded, this implies

$$|\mathbb{E}_{x \sim D}[f_\mu(x)] - \mathbb{E}_{x \sim U}[f_\mu(x)]| \leq \text{negl}(n).$$

Otherwise we could distinguish  $D$  from  $U$  by an algorithm  $A(x)$  that outputs 1 with probability  $(f_\mu(x) + n)/n^k$ , which is a valid probability in  $[0, 1]$  because  $f_\mu(x) + n = \log(1/\mu(x))$ , and  $0 \leq \log(1/\mu(x)) \leq n^k$ .

Hence

$$\mathbb{E}_{x \sim U}[f_\mu(x)] \leq \mathbb{E}_{x \sim D}[f_\mu(x)] + \text{negl}(n) \leq \text{negl}(n).$$

Recalling that

$$\mathbb{E}_{x \sim U}[f_\mu(x)] = \text{KL}(U \parallel \mu),$$

we conclude that

$$\text{KL}(U \parallel \mu) \leq \text{negl}(n).$$

Thus  $\text{KL}(U \parallel \mu)$  is negligible, as claimed.  $\square$

We need some sort of max-entropy bound for Theorem 6.1, because if  $\mu(x)$  was extremely small for some  $x$ ,  $\text{KL}(U \parallel \mu)$  could be very high. However we can use the proof of Theorem 6.1 to show that even if we remove the max-entropy condition, we still get  $\mu$  statistically close to uniform. First we need the following lemma.

**Lemma 6.2.** *Let  $D$  and  $\mu$  be distributions over  $\{0, 1\}^n$ . Define  $\mu'(x) = (1 - 2^{-n})\mu(x) + 2^{-2n}$  for all  $x \in \{0, 1\}^n$ , then*

$$\text{KL}(D \parallel \mu') \leq \text{KL}(D \parallel \mu) + \log\left(\frac{1}{1 - 2^{-n}}\right) \leq \text{KL}(D \parallel \mu) + \text{negl}(n).$$

The last inequality holds because  $\log\left(\frac{1}{1 - 2^{-n}}\right) = -\log(1 - 2^{-n}) \approx 2^{-n}$ .

*Proof.* First observe that  $\mu'$  is a probability distribution. Indeed,

$$\sum_{x \in \{0, 1\}^n} \mu'(x) = (1 - 2^{-n}) \sum_x \mu(x) + \sum_x 2^{-2n} = (1 - 2^{-n}) + 2^n \cdot 2^{-2n} = 1.$$

For every  $x$ , we have  $\mu'(x) = (1 - 2^{-n})\mu(x) + 2^{-2n} \geq (1 - 2^{-n})\mu(x)$ . Hence for any  $x$  with  $D(x) > 0$ ,

$$\log \frac{D(x)}{\mu'(x)} \leq \log \frac{D(x)}{(1 - 2^{-n})\mu(x)} = \log \frac{D(x)}{\mu(x)} + \log\left(\frac{1}{1 - 2^{-n}}\right).$$

Multiplying by  $D(x)$  and summing over  $x$  gives

$$\text{KL}(D \parallel \mu') = \sum_x D(x) \log \frac{D(x)}{\mu'(x)} \leq \sum_x D(x) \log \frac{D(x)}{\mu(x)} + \log\left(\frac{1}{1 - 2^{-n}}\right).$$

Thus  $\text{KL}(D \parallel \mu') \leq \text{KL}(D \parallel \mu) + \log\left(\frac{1}{1 - 2^{-n}}\right)$ .

Finally,  $\log\left(\frac{1}{1 - 2^{-n}}\right) = O(2^{-n})$ , which is negligible.  $\square$

If we do not restrict the max-entropy of  $\mu$ , we can still show that  $\mu$  that minimizes  $\text{KL}(D \parallel \mu)$  will be statistically close to uniform.

**Corollary 6.3.** *Let  $D$  be the output distribution of a cryptographic pseudorandom generator on  $\{0, 1\}^n$ , and let  $U$  denote the uniform distribution on  $\{0, 1\}^n$ .*

*Let  $\mu$  be a P/poly-computable distribution such that*

$$\text{KL}(D \parallel \mu) \leq \text{KL}(D \parallel U),$$

*then the statistical difference between  $U$  and  $\mu$  is negligible in  $n$ .*

*Proof.* Let  $\mu'$  be the distribution from Lemma 6.2 with  $\text{KL}(D\|\mu') \leq \text{KL}(D\|\mu) + \text{negl}(n)$ .  $\mu'$  is a P/poly-computable distribution with polynomially-bounded max entropy.

We have  $\text{KL}(D\|\mu) \leq \text{KL}(D\|U)$  so  $\text{KL}(D\|\mu') \leq \text{KL}(D\|U) + \text{negl}(n)$ . By the proof of Theorem 6.1, we have  $\text{KL}(U\|\mu') \leq \text{negl}(n)$ .

By Pinsker's inequality (see [CT06]), the statistical difference of  $U$  and  $\mu'$  is bounded by  $\sqrt{\text{KL}(U\|\mu')/2}$  which is  $\text{negl}(n)$  and since  $\mu$  and  $\mu'$  have negligible statistical difference, the statistical difference of  $U$  and  $\mu$  is negligible. □

## 7 Conclusions

In Section 5, we showed that we can model modern machine learning models as P/poly-computable distributions with polynomially-bounded max-entropy. The non-uniformity and full-depth polynomial circuits allow the flexibility to learn from complex distributions.

In Section 4, we showed that minimizing the KL-divergence over P/poly-computable distributions best captures the complexity of describing the distribution of strings, consistent with the distributions from Section 5.

Instead of rigidly forcing the hypothesis to give us a single answer, by looking at distributions we allow the models to make mistakes, and with polynomially-bounded max entropy the models never completely eliminate any possible output. This allows the modeling of complex behavior by probability, which we illustrate by Theorem 6.1 in Section 6, which shows that we can't do better at modeling the output of a pseudorandom generator than by a uniform distribution.

James Kurose, the noted networking researcher, has a saying that the Internet worked so well because it doesn't have to. In other words, because we don't require Internet protocols to guarantee delivery, we can create protocols that are far more robust and powerful. In this paper, we show a similar property for machine learning, because we don't require complete accuracy, that allows the models to capture far more complex behavior.

## 8 Future Work

In a 2022 CACM article [For22], the author observed that many of the good implications from  $P = NP$  are coming true while cryptography remains secure, a world he called "Optiland," the supposedly impossible counterpart to Russell Impagliazzo's "Pessiland" [Imp95]. This paper is a first step toward understanding how machine learning makes Optiland possible from a computational complexity viewpoint.

Further study will likely require a fuller understanding of the models, further refinements that better capture the true learning capabilities of modern machine-learning methods, and help us determine the computational power and limitations of these systems.

Some puzzles remain, in particular an efficient learning process cannot in general find the P/poly-computable distribution that has the smallest KL-divergence with a P/poly-sampleable distribution. Consider a one-way permutation  $f_k(x)$  (like AES) that is computable and invertible

with a given key. Then  $(f_k(x), x)$  is P/*poly*-sampleable and computable if the key is in the advice since the marginal  $y = f_k(x)$  is uniform. Learning the computable distribution, though, would break the one-way function.

## 9 Acknowledgments

Discussions with Matthew Gray, building on his unpublished work with Taiga Hiroka, helped formulate the statement of Theorem 6.1 and the counterexample at the end of Section 8 that we couldn't fully minimize the KL-divergence of a P/*poly*-computable distribution without breaking cryptography.

The author also thanks Rohan Acharya, Harry Buhrman, Om Mishra, Rahul Santhanam and Rakesh Vohra for helpful discussions, as well as countless others I have talked to about these questions over the past several years.

The author also had discussion on these topics with several AI models including Gemini, ChatGPT and Claude. ChatGPT helped with the proof and write-up of Theorem 6.1 and Theorem A.1. The author verified and takes responsibility for the proofs, and added details for clarity. The text of the article was fully written by the author, though Claude and Gemini were used for proofreading, stylistic suggestions, and some literature search.

## References

- [Ant26] Anthropic. Claude Opus 4.6 system card. Technical report, Anthropic, February 2026.
- [BGMMS21] Emily M. Bender, Timnit Gebru, Angelina McMillan-Major, and Shmargaret Shmitchell. On the dangers of stochastic parrots: Can language models be too big? In *Proceedings of the 2021 ACM Conference on Fairness, Accountability, and Transparency (FAccT)*, pages 610–623, 2021.
- [CT06] Thomas M. Cover and Joy A. Thomas. *Elements of Information Theory*. Wiley-Interscience, Hoboken, NJ, 2 edition, 2006.
- [For22] L. Fortnow. Fifty years of P vs. NP and the possibility of the impossible. *Communications of the ACM*, 65(1):76–85, January 2022.
- [Gol67] E. Mark Gold. Language identification in the limit. *Information and Control*, 10(5):447–474, 1967.
- [Gol01] Oded Goldreich. *Foundations of Cryptography: Volume 1, Basic Tools*. Cambridge University Press, 2001.
- [HILL99] J. Håstad, R. Impagliazzo, L. Levin, and M. Luby. A pseudorandom generator from any one-way function. *SIAM Journal on Computing*, 28(4):1364–1396, August 1999.

- [HN23] Shuichi Hirahara and Mikito Nanashima. Learning in pessiland via inductive inference. In *2023 IEEE 64th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 447–457, 2023.
- [Imp95] R. Impagliazzo. A personal view of average-case complexity theory. In *Proceedings of the 10th Annual Conference on Structure in Complexity Theory*, pages 134–147. IEEE Computer Society Press, 1995.
- [KV94] Michael Kearns and Leslie Valiant. Cryptographic limitations on learning boolean formulae and finite automata. *J. ACM*, 41(1):67–95, January 1994.
- [Lap91] Pierre-Simon Laplace. Recherches, 1<sup>o</sup>, sur l’intégration des équations différentielles aux différences finies, et sur leur usage dans la théorie des hasards. In *Oeuvres complètes de Laplace*, volume 8, pages 69–197. Gauthier-Villars, Paris, 1891. Originally published 1776; written 1773. Translation of pp. 144–145 in Charles Coulston Gillispie, *Pierre-Simon Laplace, 1749–1827: A Life in Exact Science*, Princeton University Press, 1997, p. 26.
- [LV09] Ming Li and Paul Vitányi. *An introduction to Kolmogorov complexity and its applications*. Springer Science & Business Media, 2009.
- [Ris78] Jorma Rissanen. Modeling by shortest data description. *Automatica*, 14(5):465–471, 1978.
- [Sha48] Claude E. Shannon. A mathematical theory of communication. *Bell System Technical Journal*, 27(3):379–423, 1948.
- [Sol64] Ray J. Solomonoff. A formal theory of inductive inference, Part I. *Information and Control*, 7(1):1–22, 1964.
- [Tur36] A. Turing. On computable numbers, with an application to the Entscheidungsproblem. *Proceedings of the London Mathematical Society*, 42:230–265, 1936.
- [Val84] L. Valiant. A theory of the learnable. *Communications of the ACM*, 27(11):1134–1142, 1984.
- [VL00] P.M.B. Vitanyi and Ming Li. Minimum description length induction, bayesianism, and kolmogorov complexity. *IEEE Transactions on Information Theory*, 46(2):446–464, 2000.
- [VSP<sup>+</sup>17] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 30, 2017. arXiv:1706.03762.

- [WWS<sup>+</sup>22] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, brian ichter, Fei Xia, Ed Chi, Quoc V Le, and Denny Zhou. Chain-of-thought prompting elicits reasoning in large language models. In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh, editors, *Advances in Neural Information Processing Systems*, volume 35, pages 24824–24837. Curran Associates, Inc., 2022.
- [Yao82] A. Yao. Theory and applications of trapdoor functions. In *Proceedings of the 23rd IEEE Symposium on Foundations of Computer Science*, pages 80–91. IEEE, New York, 1982.

## A Appendix

**Theorem A.1.** *Let  $\mu$  be a distribution on  $\{0, 1\}^n$ . The following are equivalent.*

1. *The cumulative distribution function*

$$F_\mu(x) = \sum_{z \leq x} \mu(z)$$

*is computable in polynomial time, where  $\leq$  is the lexicographic order on  $\{0, 1\}^n$ .*

2. *The associated conditional probability function*

$$f_\mu(y) = \Pr_{x \sim \mu} [x_{|y|+1} = 1 \mid x_{1..|y|} = y]$$

*is in FP for all prefixes  $y \in \{0, 1\}^{<n}$ , with the convention that  $f_\mu(y) = 0$  if  $\mu(y0) + \mu(y1) = 0$ .*

*Proof.* For a prefix  $y \in \{0, 1\}^k$ , let

$$\mu(y) = \Pr_{x \sim \mu} [x_{1..k} = y]$$

denote the cylinder probability of  $y$ . Then

$$f_\mu(y) = \frac{\mu(y1)}{\mu(y)}$$

whenever  $\mu(y) > 0$ , and  $f_\mu(y) = 0$  otherwise.

We show each direction separately.

**(1)  $\Rightarrow$  (2).** Assume  $F_\mu \in \text{FP}$ . We first show that for every prefix  $y$ , the quantity  $\mu(y)$  is polynomial-time computable.

Let  $|y| = k$ . The set of strings in  $\{0, 1\}^n$  having prefix  $y$  is a contiguous interval in lexicographic order, namely

$$\{y0^{n-k}, y0^{n-k} + 1, \dots, y1^{n-k}\}.$$

Thus if we write

$$a_y = y0^{n-k} \quad \text{and} \quad b_y = y1^{n-k},$$

then

$$\mu(y) = \sum_{a_y \leq z \leq b_y} \mu(z).$$

Using the CDF, this becomes

$$\mu(y) = F_\mu(b_y) - F_\mu(a_y - 1),$$

where  $a_y - 1$  denotes the lexicographic predecessor of  $a_y$ , and if  $a_y$  is the minimum string  $0^n$ , then  $F_\mu(a_y - 1)$  is understood to be 0.

Similarly,

$$\mu(y1) = F_\mu(b_{y1}) - F_\mu(a_{y1} - 1),$$

so both  $\mu(y)$  and  $\mu(y1)$  are computable in polynomial time.

Therefore

$$f_\mu(y) = \begin{cases} \mu(y1)/\mu(y), & \text{if } \mu(y) > 0, \\ 0, & \text{if } \mu(y) = 0, \end{cases}$$

is polynomial-time computable as well. Hence  $f_\mu \in \text{FP}$ .

**(2)  $\Rightarrow$  (1).** Assume  $f_\mu \in \text{FP}$ . We show that  $F_\mu \in \text{FP}$ .

First observe that from  $f_\mu$  one can compute  $\mu(x)$  for every  $x = x_1x_2 \cdots x_n \in \{0, 1\}^n$ . Indeed, if  $x_{<i} = x_1 \cdots x_{i-1}$ , then

$$\mu(x) = \prod_{i=1}^n \Pr[x_i \mid x_{<i}] = \prod_{i=1}^n \begin{cases} f_\mu(x_{<i}), & \text{if } x_i = 1, \\ 1 - f_\mu(x_{<i}), & \text{if } x_i = 0. \end{cases}$$

This is polynomial-time computable since there are only  $n$  factors and each value  $f_\mu(x_{<i})$  is polynomial-time computable.

Now let  $x \in \{0, 1\}^n$ . We express  $F_\mu(x) = \sum_{z \leq x} \mu(z)$  as a sum of cylinder probabilities. Suppose

$$x = x_1x_2 \cdots x_n.$$

A string  $z \leq x$  either equals  $x$ , or is determined by the first position  $i$  at which  $z_i < x_i$ . The latter can happen only when  $x_i = 1$ , in which case  $z_i = 0$  and the previous bits agree with  $x$ . Hence

$$F_\mu(x) = \mu(x) + \sum_{i: x_i=1} \mu(x_1x_2 \cdots x_{i-1}0).$$

So it suffices to compute each cylinder probability  $\mu(y)$  in polynomial time. If  $y = y_1 \cdots y_k$ , then

$$\mu(y) = \prod_{i=1}^k \begin{cases} f_\mu(y_{<i}), & \text{if } y_i = 1, \\ 1 - f_\mu(y_{<i}), & \text{if } y_i = 0. \end{cases}$$

Thus  $\mu(x)$  and every term  $\mu(x_1 \cdots x_{i-1}0)$  are polynomial-time computable, and there are at most  $n+1$  such terms. Therefore  $F_\mu(x)$  is computable in polynomial time, and hence  $F_\mu \in \text{FP}$ .  $\square$