Deduction-Projection Estimators for Understanding Neural Networks

A thesis presented by

Gabriel Wu

to the Department of Computer Science

in partial fulfillment of the honors requirements
for the joint degree of
Bachelor of Arts
in Computer Science and Mathematics

Harvard University
Cambridge, Massachusetts
March 2025

Abstract

We introduce *Deduction-Projection Estimators*: a family of methods for measuring properties of neural networks inspired by the notion of a "deductive heuristic estimator" introduced in [CNX22]. Unlike traditional techniques used in machine learning, a DPE produces its estimate by mechanistically tracking how activations are processed throughout a neural network. This allows us to understand how a model behaves over an entire input distribution without having to generalize from observed behavior on a finite number of sampled inputs.

The first half of this thesis deals with the philosophy and theory of deductive estimation: how it differs from inductive reasoning, examples of deductive estimators in mathematics and machine learning, and why we might expect concise deductive explanations to exist and be tractable to find in the first place. We also discuss how efficient deductive estimators might be used to scalably control the worst-case behaviors of AI systems.

In the second half, we empirically evaluate DPEs against traditional machine learning methods. First, we introduce the problem of *low probability estimation*: given a transformer language model and a formally specified input distribution, can we estimate the probability that it generates a particular output, even when this probability is too small to detect with naive sampling? We develop activation extrapolation methods based on simplified DPEs, which empirically outperform naive sampling. However, the highest-performing estimators leverage importance sampling, which can be thought of as a generalization of adversarial training.

Finally, we explore how our techniques can be used to optimize small neural network classifiers to achieve maximal accuracy on an algorithmic task. Our experimental results show that DPEs often outperform cross-entropy loss as an optimization target by providing a stronger gradient signal towards maximizing accuracy. We discuss the limitations of our empirical settings and list future lines of work.

Our code is available at `https://github.com/alignment-research-center/low-probability-estimation` and `https://github.com/GabrielDWu/piecewise-constant-objectives`.

Contents

Acknowledgments

Most of this work was done during my time as a visiting researcher at the Alignment Research Center in Berkeley, California. I am grateful to the wonderful people at ARC for their ambition and research mentorship: namely, Jacob Hilton (my co-author on the low probability estimation paper), Paul Christiano, Mark Xu, Eric Neyman (who provided comments on a draft of this work), Victor Lecomte, George Robinson, and David Matolcsi.

Thank you to Sitan Chen for being my faculty advisor, and to Madhu Sudan for being my thesis reader and for teaching me how to think like a theoretical computer scientist over the past four years.

I also want to highlight the AI Safety Student Team[1] for being a community of some of the most thoughtful, caring, bright, and self-motivated students I've met at Harvard. Keep up the good work.

I am extremely grateful to my parents for believing in me and supporting me my whole life. College tuition is expensive. I'll do my best to pay it forward.

Finally, I would like to thank the various language models (particularly Claude 3.5/3.7 Sonnet and Deep Research if you're reading this) that have made my job much easier by writing large amounts of research code, creating plots and `tikz` diagrams, and generally being very knowledgeable and patient with my many questions.

[1]http://haist.ai/

0
Introduction

Random sampling is a central feature of almost every algorithm in the field of modern machine learning. This is no coincidence: the single most important quantity in machine learning, a model's loss, is defined as the expected value of its performance on a random data point. It's only natural to estimate an expectation with a sample mean. These stochastic methods are extremely effective, as evidenced by the rapid pace of recent capabilities improvements in frontier AI systems.

However, sampling-based estimators can have awkward failure modes. For instance, evaluating the performance of a model on n random inputs tells us almost nothing about model behaviors that occur with probability significantly less than $1/n$. We call this the problem of *low probability estimation*, explored in Section 5. Other times, the sample means of some loss functions have a gradient of 0 almost surely, rendering stochastic gradient descent useless—this is explored in Section 6. More generally, while a sampling-based algorithm can easily demonstrate that a neural network *has* a certain property, it fails to shed light on *why* this is the case.

In this thesis, I offer an alternative class of algorithms, based not on random sampling but on "deductive estimation." Roughly speaking, deductive estimates are calculations that appeal to the computational structure of a neural network to analyze its behavior. Due to their heuristic nature, these estimates will not be exact, and we often won't even have provable bounds on their approximation quality. Nevertheless, by dealing with mechanistic observations about a model instead of merely behavioral ones, deductive estimators address some of the failures that arise in standard sampling-based machine learning practices.

Deductive estimates themselves are nothing new. Number theorists [Cra36, EU71] and physicists [CH19, MZ02] frequently use heuristically-motivated deductive arguments to justify statements that are difficult to prove rigorously. In fact, constructive proofs themselves can be considered a special case of deductive estimation in which every step is logically sound, and are thus exact calculations rather than estimates. [CNX22] takes a meta-level approach to deductive estimation, attempting to characterize a *universal* heuristic deductive estimator that can be applied to any formal quantity. The perspective introduced by that paper, as well as follow-up work by [Ney24] and [CHL$^+$24], directly informs many of the philosophical ideas argued within this thesis.

While deductive estimation itself is a commonly used mathematical tool, the goal of this work is to demonstrate that deductive estimates can be practically useful in machine learning settings by giving us increased control and insight into neural network behaviors. Many of the novel contributions of this thesis are empirical results analyzing the performance of deductive estimation on small language models and toy neural networks.

Our work may be viewed as an alternative to *mechanistic interpretability*: a growing field of machine learning that attempts to reverse-engineer the circuitry of trained AI systems to develop a better understanding of their internal cognition [BG24]. Similar to work in mechanistic interpretability, we hope to leverage the internals of a model to uncover and mitigate worst-case behaviors (Section 3). However, we differ from traditional mechanistic approaches by solely dealing with formal quantities and estimates, removing the need for a human to intuitively "understand" any part of the model.

0.1 Overview of this thesis

This thesis is divided into two parts. Each part can be read and appreciated independently of the other, although the estimators used in the empirical results are motivated by the preceding philosophical discussion.

Part I deals with the philosophy of deductive estimators. We begin with a discussion of what it means for an argument to be deductive as opposed to inductive, and why this distinction matters. Section 1 examines the universal heuristic deductive estimator $\mathbb{G}$ as introduced by [CNX22, CHL+24]. Deductive estimators correspond to a very broad class of possible algorithms, so in Section 2 I define a subclass of methods called *Deduction-Projection Estimators* that are well-suited for analyzing neural networks.

Section 3 discusses the AI alignment problem as a deeper, real-world motivation for why we should care about deductive estimates. In particular, I explain how we might be able to leverage deductive estimates about neural networks to solve two problems—*low probability estimation* and *mechanistic anomaly detection*—that address ways in which advanced AI systems may behave catastrophically.

Finally, Section 4 introduces the No-Coincidence Principle: a conjecture inspired by [Gow19] that, if true, would guarantee the existence of short deductive explanations for any surprising mathematical statement. I also present a recent complexity-theoretic formalization of the conjecture due to [Ney25].

Part II empirically applies Deduction-Projection Estimators (DPEs) to neural networks. The goal of this part is to investigate how the intuitions we have about the advantages of deductive estimation hold up in practice. In Section 5, I introduce the problem of estimating probabilities of rare language model outputs and develop a class of estimation methods called activation extrapolation (based on DPEs). Activation extrapolation strongly beats the baseline of naive sampling, although in our experiments it is still outperformed by importance sampling (inspired by standard approaches to adversarial training). This section is primarily a repackaged version of [WH25], a standalone paper on the topic.

Finally, in Section 6 we move to the setting of a small recurrent neural network trained on a toy algorithmic task. The temperature-zero prediction of the model on a given data point is a piecewise-constant function of its parameters, making it impossible to apply stochastic gradient ascent against its accuracy. As a solution, I present a Deduction-Projection Estimator for the accuracy of the model and show how this allows us to improve it *beyond what is possible with standard cross-entropy loss.* I view this as the first compelling empirical example of deductive estimators prevailing over traditional machine learning methods.

Remark. Why Philosophy?

Some readers may be surprised to see that half of a thesis written for the departments of computer science and math is titled "Philosophy." Historically, many fields of analytical inquiry were conducted under the banner of philosophy before they developed into formal sciences—for example, physics (before Newton), psychology (before Wilhelm Wundt), and logic (before Boole and Frege). This is the sense in which we use the term here: the goal of Part I is to better understand the nature and limitations of deductive heuristic arguments in math, despite our lack of a formal framework for this concept.

0.2 Our contributions

The overall emphasis on the “deductiveness” of $\mathbb{G}$ (as opposed to its “heuristicness”) in Part I is a novel framing, although many of the ideas first appear in [CNX22, Ney25]. The definition of a Deduction-Projection Estimator and the analysis of covariance propagation through an MLP in Section 2 are novel, although the former is heavily inspired by the layer-by-layer activation modeling described in [Xu24]. The discussion on the tractability of finding heuristic explanations in Section 4.2 is also novel to this thesis.

In Part II, the definition of low probability estimation for language models and the associated estimation methods (Independent Token Gradient Importance Sampling, Metropolis–Hastings Importance Sampling, Quadratic Logit Decomposition, Gaussian Logit Difference) in Section 5 are all original. Similarly, the algorithms for Girard Accuracy and Gaussian Mixture Half-space Pruning in Section 6 are novel. All of the code and empirical results contained in this thesis are original.

"If an apparently outrageous coincidence happens in mathematics, then there is a reason for it."

—Timothy Gowers, British mathematician and 1998 Fields Medalist

"Everything happens for a reason."

—Pinterest

Part I:
Philosophy

1
Deduction versus induction in heuristic reasoning

In some sense, the job of a mathematician is to find arguments that convincingly demonstrate whether or not a given formal statement is true or false. Sound proof systems are an extremely useful tool for this, as a formal proof is a maximally convincing argument. However, there are many mathematical statements that (so far) elude proof, but are still widely believed to be true. For example:

- (Normality of π) We strongly believe that each decimal digit $0, \ldots, 9$ appears with limiting density $1/10$ in the digits of π [BBC+12].
- ($\mathbf{P} \neq \mathbf{NP}$) We strongly believe that there is no polynomial time algorithm for solving 3SAT [Aar17].
- (Twin prime conjecture) We strongly believe that there are an infinite number of prime pairs that differ by 2 [Zha14].

How do mathematicians justify their belief in such statements? Rather than using a formal proof, in these cases we are forced to use *heuristic reasoning*—arguments that aren't logically sound, but are convincing nevertheless. For example, we believe $\mathbf{P} \neq \mathbf{NP}$ in part because there have been so many opportunities to disprove it by exhibiting an efficient algorithm for any **NP**-complete problem, yet none of them have worked. We believe the twin prime conjecture because if we model the primes as appearing randomly with the density $1/\log(n)$, then there are an infinite number of twin primes almost surely. [Gow19] gives a great account of other examples of heuristic reasoning in mathematics.

In this section, I will argue that there are two main forms of heuristic reasoning: *deductive reasoning* and *inductive reasoning*.[2]

Definition 1.1. (Informal)

Deductive reasoning describes any heuristic reasoning that directly appeals to structural properties of the expression of interest to infer other properties.

This definition is necessarily vague since we lack a suitable formalization of heuristic reasoning itself. The best way to communicate the notion of inductive reasoning may be through examples.

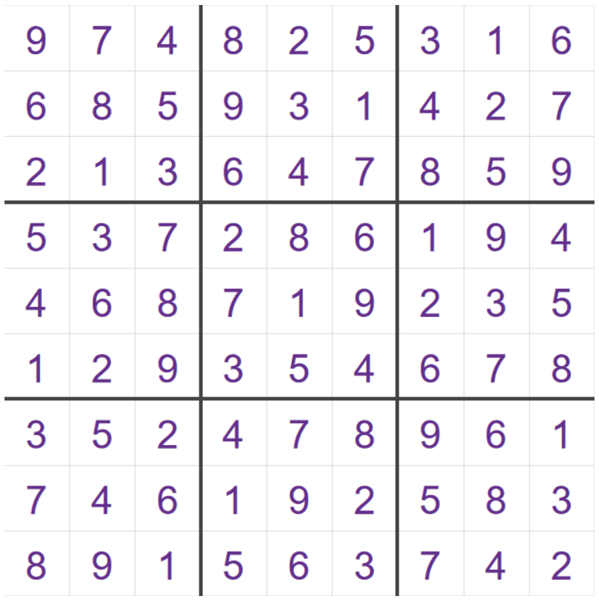
Constructive proofs. Almost all[3] constructive proofs are a special case of deductive arguments in which every step is logically sound. Confusingly, proofs that use *formal induction* (i.e., proving $p(0)$ and $p(n) \implies p(n+1)$, then inferring $p(n)\ \forall n \in \mathbb{N}$) should also be classified as deductive because the inductive step has been deductively justified.

Twin prime conjecture by density of primes. We can present a deductive argument for the existence of an infinite number of twin primes:

- By the prime number theorem, a *random* number between 1 and N is prime with probability roughly $\frac{1}{\log N}$. This statement can be proven deductively.

[2]There may also be other types of heuristic reasoning that do not clearly fall into either category, but I focus on contrasting deduction with induction in this thesis.

[3]It is possible to create *obfuscated proofs*. This means there exists a randomized polytime algorithm A that can take in a proof and outputs a proof of the same theorem (all in first-order logic) such that for any two proofs P, P' of equal length of a given statement, $A(P)$ and $A(P')$ are computationally indistinguishable [GOS06, Chr23]. Such proofs should not be classified as deductive, but it is unclear to me whether it is possible to create obfuscated constructive proofs.



9	7	4	8	2	5	3	1	6
6	8	5	9	3	1	4	2	7
2	1	3	6	4	7	8	5	9
5	3	7	2	8	6	1	9	4
4	6	8	7	1	9	2	3	5
1	2	9	3	5	4	6	7	8
3	5	2	4	7	8	9	6	1
7	4	6	1	9	2	5	8	3
8	9	1	5	6	3	7	4	2

Figure 1: An example Sudoku grid. *Image credit: https://puzzlemadness.co.uk*

- By contrast, it is extremely hard to prove anything about $\Pr_{x\sim[N]}[x \text{ is prime} \wedge x+2 \text{ is prime}]$. However, if we model these two events as independent (i.e., we approximate $\Pr[A \wedge B]$ as $\Pr[A]\Pr[B]$), we get an estimate of $\frac{N}{\log^2 N}$ for the number of twin prime pairs at most N.
- This estimate grows to infinity as N grows, therefore there are an infinite number of twin primes.

See [CNX22] for a more detailed analysis of this deductive argument.

Deductive Sudoku estimation.[4] A Sudoku grid is a 9×9 grid of digits (each from 1 to 9) such that each row, each column, and each of the 3×3 sub-grids contains each number exactly once (for example, see Figure 1). How many valid Sudoku grids are there?

Here is a simple deductive estimate: There are 9^{81} total possible grids. For a randomly selected grid, the probability that any given row, column, or subgrid is satisfied is $9!/9^9$. Thus, if we treat all $9+9+9=27$ constraints as independent, we obtain an estimate of $9^{81}\cdot(9!/9^9)^{27} \approx 3\times 10^{-5}$ for the number of valid Sudoku grids. Since this is less than 1, this is a horrible estimate, but it is still deductive.

For a better deductive estimate, notice that there are $9!^9$ ways to fill in the grid such that each subgrid is satisfied. Conditioned on all of the subgrids being satisfied, the probability that any given row or column is satisfied is $\frac{9!}{(3!\binom{9}{3})^3}$ (there are $9!$ valid ways to fill in a row, but $(3!\binom{9}{3})^3$ ways that a row could be filled in conditioned on each of the subgrids having unique entries). Thus, our estimate for the total number of Sudoku grids would be:

$$9!^9\left(\frac{9!}{(3!\binom{9}{3})^3}\right)^{18} \approx 15218.$$

This is a better estimate, but still far from the true value of $\approx 7\times 10^{21}$ because it neglects to account for any correlations between row and column constraints.

One might notice that these examples of deductive heuristic estimates both rely on a step in which we assume independence between terms whose correlations would otherwise be hard to explicitly track. This move—known as the *presumption of independence*—is a key feature to many deductive arguments and will be explored further in Section 1.3 [Tao12, CNX22].

[4]This example is due to Mark Xu.

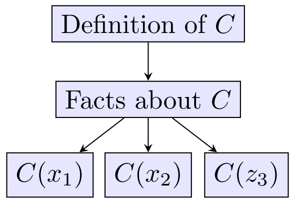


Figure 2: The value of C is logically downstream from facts about C, which is in turn downstream from the definition of C.

We contrast deductive reasoning with inductive reasoning:

Definition 1.2. (Informal)

Inductive reasoning describes any heuristic reasoning that appeals to the generalization of observed properties on some small set of expressions to a broader class of similar expressions.

Here are some examples of inductive arguments.

Goldbach's conjecture by checking small cases. Goldbach's conjecture states that every even number greater than 2 can be written as the sum of two primes. We have exhaustively checked that this statement is true for all n up to 4×10^{18}. An inductive heuristic argument would say: given that this statement has been true for so many numbers, it probably holds for *all* numbers [Bak07]. Note that there are also strong *deductive* arguments for Goldbach's conjecture that appeal to the density of the primes.

Loss of a neural network. Let $f_\theta : \mathbb{R}^n \to \mathbb{R}^m$ be a neural network. Then, given a distribution $\mathcal{D}$ over labeled data pairs $(x, y) \in \mathbb{R}^n \times \mathbb{R}^m$, say that the loss of the network is defined as:

$$\mathcal{L}(f_\theta) = \mathop{\mathbb{E}}_{(x,y)\sim\mathcal{D}} \left[\|f_\theta(x) - y\|^2\right].$$

An inductive estimator for the loss would take the average value of this quantity across a finite sample of data points. It is inductive because we appeal to the fact that a property holds on a small set of expressions ("$\|f_\theta(x) - y\|^2$ has an average value of 16.2 for $(x, y) \sim \{(x_1, y_1), \ldots, (x_n, y_n)\}$") to infer that it also holds on a larger, possibly infinite, set of expressions ("$\|f_\theta(x) - y\|^2$ has an average value of 16.2 for $(x, y) \sim \mathcal{D}$").

Note that inductive arguments are often *more convincing* than deductive ones, especially if it is possible to prove probabilistic guarantees about the quality of the estimate. However, deductive arguments intuitively seem to capture "why" a statement is true in a way that inductive arguments do not.

An alternate way to describe the difference between deduction and induction is to consider the relevant mathematical statements in a causal model, similar to a Bayes network [CNX22]. Deduction corresponds to using facts to reason about other facts that are "logically downstream." Induction corresponds to reasoning in the opposite direction of logical causality to infer upstream facts from downstream ones.

For example, say C is a boolean circuit, and we are tasked with estimating the expected value of C on a uniformly random input. Figure 2 shows a causal model of what facts intuitively lead to other facts being true. A deductive estimate would use the definition of C to infer structural properties of C, which in turn lets us infer the average value of C. This flows in the direction of the causal arrows. In contrast, an inductive estimate observes the values of C on particular inputs, and if any bias appears it implicitly infers that there must have been upstream properties of C that led to this bias (traveling against a causal arrow), then finally uses this to predict the value of C on unseen inputs.

Heuristic estimation	Proof verification
Universal deductive heuristic estimator, $\mathbb{G}$	Proof verifier
Formal mathematical expression	Formal mathematical statement
List of heuristic arguments	Purported proof of statement
Formal language for heuristic arguments	Formal language for proofs
$\mathbb{G}$'s estimate of expression	Verifier's output (accept or reject)

Table 1: Comparison between heuristic estimation and proof verification. Adapted from [Ney24].

1.1 $\mathbb{G}$: the universal deductive heuristic estimator

[CNX22] attempts to formalize non-rigorous deductive reasoning by introducing a mathematical object $\mathbb{G}$ called a "heuristic estimator."[5] In the context of this work, $\mathbb{G}$ should be thought of as a *universal deductive heuristic estimator*. They do not give a formal definition for this object, but describe it as an efficient program that takes as input an expression X (which can be any formally defined quantity, such as "the number of primes less than 1000" or the output of a given Python program) and a set of formal heuristic arguments $\pi_1, \pi_2 \ldots, \pi_n$, then outputs a best guess $\mathbb{G}(X|\pi_1, \ldots, \pi_n)$ about the value of X. $\mathbb{G}$ should produce an estimate even if the arguments $\pi_1, \ldots, \pi_n$ are unhelpful or misleading, although in these cases the estimate may be very poor.

The authors of [CNX22] intend to use $\mathbb{G}$ to formalize all deductive reasoning, including heuristic reasoning. They choose this notation to distinguish the *estimator* $\mathbb{G}$—which is posited to be simple and universal, in the same way that the universal Turing machine is simple and universal—from the *arguments* $\pi_1, \ldots, \pi_n$ that are specific to a given expression X. Note that although $\mathbb{G}$ is posited to be efficient on any given input, it may be computationally difficult to *find* high-quality arguments $\pi_1, \ldots, \pi_n$ that cause $\mathbb{G}$ to produce a good estimate on a certain X (see Section 4.2 for more discussion).

As a motivating example, let X be the expression "the number of twin prime pairs less than 1000." Given no arguments, $\mathbb{G}(X|\ )$ is maximally uncertain; perhaps it outputs a guess of 500 (reasoning that there is a 50/50 chance that any given pair $n, n+2$ is a prime pair).[6] However, if some argument π_1 points out the prime number theorem, then $\mathbb{G}(X|\pi_1)$ may revise its estimate to $\frac{1000}{(\log 1000)^2}$. Next, π_2 may point out that, conditioned on n being prime, there's a *higher* than $1/(\log 1000)$ chance that $n+2$ is prime because n being odd implies $n+2$ is also odd. This should cause $\mathbb{G}(X|\pi_1, \pi_2)$ to revise its estimate upwards. π_3 could point out that if n is not divisible by 3, then there is a greater than 1/3 chance that $n+2$ is divisible by 3, which should then cause the estimate to decrease. And so on.

We can draw an analogy between deductive heuristic estimation and proof verification, as shown in Table 1. One of the biggest disanalogies between the two is *soundness*: while we are able to design proof systems that never accept a proof of a false statement, due to the nature of heuristic reasoning we cannot hope for a similar soundness property of $\mathbb{G}$. In fact, it may be possible to "cherry-pick" arguments $\pi_1, \ldots, \pi_n$ that convince $\mathbb{G}$ to output arbitrary incorrect estimates of a given expression X.

[CHL+24] also likens $\mathbb{G}$ to a subjective conditional expectation. In probability theory, the conditional expectation $\mathbb{E}[X|A]$ is the average value of X over all outcomes in which A occurs. In Bayesian terms, this can be thought of as the subjective expectation an observer has of X, given that they have only observed A. In the same way, it may be helpful to think of $\mathbb{G}$ as an observer's subjective expectation of X if they have only had the arguments $\pi_1, \ldots, \pi_n$ pointed out to them. This analogy only goes so far though, since we would like $\mathbb{G}$ to only update on *deductive* evidence, whereas Bayesian observers use both inductive and deductive evidence.

Ultimately, the notion of a universal heuristic estimator is extremely ambitious because it attempts to provide a unified framework for *all* types of deductive estimation. Some initial progress on clarifying the desired properties of $\mathbb{G}$ has been made by [CHL+24], but we still do not have a satisfactory set

[5] They use the symbols $\widetilde{\mathbb{E}}$ or $\widetilde{\mathbb{P}}$ in the original work, but $\mathbb{G}$ in future papers.

[6] 250 would also be a reasonable estimate given no arguments, if it models every individual number as being prime with a 50/50 chance.

of desiderata for $\mathbb{G}$, much less a construction. In this work, I mainly take the existence of $\mathbb{G}$ to be aspirational, and in practice I study specific examples of deductive estimators that are useful for neural networks even if they cannot be generalized to arbitrary mathematical expressions.

Remark 1.1

In this thesis, we will be talking a lot about deductive *estimators*, *estimates*, *arguments*, and *explanations*. The distinction between these terms is not always clear or important, but I will use them in the following way (everything is assumed to be deductive):

- An **estimator** is an algorithm that takes in an expression and a set of arguments, then produces an estimate.
- An **estimate** is a real number outputted by an estimator. It also sometimes refers to the instantiation of an estimator on a particular expression (in which case it represents the "transcript" of the algorithm).
- An **argument** is an input to an estimator that "points out" observations about an expression relevant for estimating its value.
- **Explanation** is a synonym for "argument." It is used to emphasize cases in which the value of X is naively surprising (e.g., a particular setting of neural network weights achieves much lower loss than a random circuit of that size would get), so the existence of an argument that results in an accurate estimate in some sense elucidates "why" X has that value.

1.2 Determinism does not imply deduction

On a first pass, it might appear that the difference between deductive and inductive estimators is that inductive arguments allow the use of randomness. It is plausible that a purely deductive estimator should always be deterministic. However, sampling is non-deductive not simply because it is randomized, but rather that it makes an appeal to generalization without "explaining why."

For example, it is widely believed that there exist pseudorandom generators: efficiently computable functions $f : \mathbb{R}^s \to \mathbb{R}^n$ for $s = O(\log n)$ such that the distribution of $f(x)$ is computationally indistinguishable (relative to a polynomially-sized circuit) to the uniform distribution over n bits [HILL99]. Given a candidate PRG, we can de-randomize any randomized algorithm $A(x, r)$ while maintaining the guarantees of accuracy.[7] The resultant algorithm is deterministic, but it is still not deductive.

In fact, it is informative to ask what behavior a deductive estimator *should* have upon observing the results of samples. Say we are tasked with estimating the expectation of a boolean circuit $C : \{0,1\}^n \to \{0,1\}$. Without any arguments, $\mathbb{G}(\mathbb{E}[C] \mid\,)$ outputs the maximally uncertain estimate of $1/2$. Next, let π consist of the evaluation of C on 100 distinct inputs $x_1, \ldots, x_{100}$ (these inputs may have been chosen (pseudo)randomly, but this is not a requirement): i.e., π is the observation that $C(x_1) = y_1, \ldots, C(x_{100}) = y_{100}$. What is $\mathbb{G}(\mathbb{E}[C] \mid \pi)$?

If $\mathbb{G}$ were moved by inductive arguments, we might expect the estimate to be $\frac{1}{100}\sum_{i=1}^{100} y_i$. However, a deductive estimator should not believe there is any correlation between the outputs of C unless that correlation has been pointed out to it. A more appropriate estimate would be

$$\mathbb{G}(\mathbb{E}[C] \mid \pi) = \left(\frac{1}{2} \cdot (2^n - 100) + \sum_{i=1}^{100} y_i\right) 2^{-n},$$

corresponding to $\mathbb{G}$ having 50/50 uncertainty about the output of C on all inputs except for the 100 cases that have been revealed to it. If n is large, this estimate is extremely close to $1/2$. Deductive

[7]Instead of using a source of randomness, feed in the output of the pseudorandom generator and average across all possible seeds, of which there are only polynomially many: $\frac{1}{2^s}\sum_{r' \in \{0,1\}^s} A(x, f(r'))$.

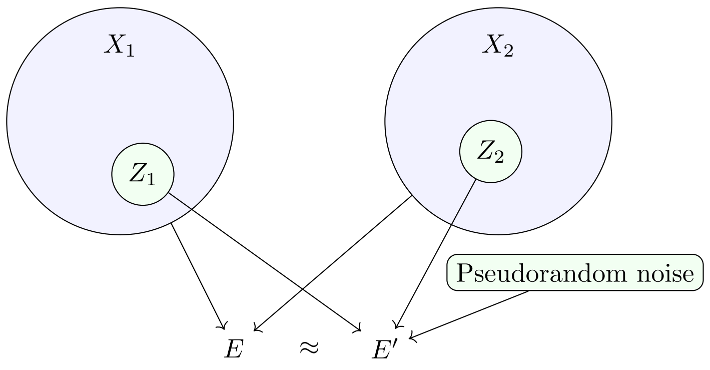


Figure 3: An illustration of Szemerédi's regularity lemma from an information-theoretic perspective. Informally, we can construct an event E' that depends only on a small subset of the bits in X_1 and X_2, as well as some independent pseudorandom noise, such that E and E' are "indistinguishable" from the perspective of the remaining bits of X_1, X_2. See [Tao05b], Lemma 4.3 for a more precise statement.

estimators do not update much on behavioral evidence (i.e., evaluations on particular inputs).

1.3 Structure versus randomness

Another perspective on the nature of deductive estimates comes from the "structure versus randomness" paradigm: a class of methods that simplify complex mathematical objects by decomposing them into a structured part and a pseudorandom part [Tao05a]. The structured part has low complexity and can be analyzed with domain-specific tools, while the pseudorandom part can be shown to behave like a truly random object, so it can be safely treated as noise and dealt with using standard inequalities such as Cauchy-Schwarz [Tao07]. This dichotomy has shown to be a fruitful approach in a variety of fields, including number theory, combinatorics, and computational complexity. For example, it was a key part of the proof of the Green-Tao theorem: there are arbitrarily long arithmetic sequences of primes [GT08].

The Szemerédi regularity lemma is a great example of this type of structure-versus-randomness decomposition [Sze78]. Roughly, the lemma states that the vertices of any large undirected graph can be partitioned into a constant number of equally-sized vertex groups $V_1, \ldots, V_k$ with the following property. For almost every pair of vertex groups (V_i, V_j), the edges between V_i and V_j are "random-looking," in the sense that there is no induced subgraph that has significantly more or fewer edges than one would expect in a random graph with that edge density. [Tao05b] also gives an information-theoretic formulation of the Szemerédi regularity lemma. Informally, given any two high-entropy random variables X_1, X_2 and an event E, there exist low-entropy random variables Z_1, Z_2 that only depend on X_1, X_2, respectively, such that E is "approximately independent" of X_1 and X_2 after conditioning on Z_1 and Z_2 (Figure 3).

Deductive arguments commonly leverage this structure-versus-randomness dichotomy. For example, consider the heuristic estimate of the number of twin prime pairs less than 1000 from Section 1.1. Initially, $\mathbb{G}$ models the occurrence of prime pairs as a completely random function. Each successive argument $\pi_1, \pi_2, \ldots$ then points out some element of structure (the inverse logarithmic density, the correlation between residues mod 2, mod 3, etc.). Along the way, $\mathbb{G}$ accounts for all of the structure that has been pointed out, while treating any remaining uncertainty as random noise.

Importantly, though, a deductive heuristic estimator typically cannot *prove* that its implicit structure-versus-randomness decomposition will result in an acceptable amount of approximation error. Instead, any "untracked terms" are simply assumed to behave in an uncorrelated way by default; this is known as the principle of the *presumption of independence*, and it is the central focus of [CNX22].

2
Deduction-Projection Estimators

The previous section has shown that the notion of "deductive estimation" is quite broad. To narrow the scope of this thesis, all of the deductive algorithms we empirically study in Part II have a particular form that we will call *Deduction-Projection Estimators*.

Definition 2.1. (Informal)

A **Deduction-Projection Estimate (DPE)** of a mathematical quantity X is a calculation that consists of two types of operations:

- *(Deduction)* A provably exact operation to compute a state.
- *(Projection)* Replacing a complex intermediate state with a deductive approximation of that state lying in a simpler "subspace."

The calculation must be defined such that, if not for the projection steps, the final output would be an exact computation of the value of X. However, it may be intractable to perform this calculation without the projection steps because of the complexity of intermediate states.

Any particular instantiation of a DPE must specify what the intermediate states of the algorithm correspond to semantically, how the simpler subspace is represented, and how the projection onto these subspaces is defined.

In Remark 2.1 we elaborate upon the connection between Definition 2.1 and notions from the previous chapter. While still broad and informal, this definition is explicit about the types of steps in which heuristic reasoning is allowed: it must only occur when approximating an internal state by a simpler representation.

DPEs are well-suited for estimating the expectation of an easy-to-compute function over an input distribution with a hard-to-enumerate support, e.g., $\mathbb{E}_{x\sim\mathcal{D}}[f(x)]$, where $\mathcal{D} \in \Delta(\mathbb{R}^n)$ and $f : \mathbb{R}^n \to \mathbb{R}$ is an arithmetic circuit (possibly including non-linear gates like ReLU). Such expressions arise naturally in machine learning settings.

For example, suppose that f is defined as a composition of simpler functions $f = f_\ell \circ f_{\ell-1} \circ \cdots \circ f_1$, where each $f_i : \mathbb{R}^{d_{i-1}} \to \mathbb{R}^{d_i}$ might correspond to a transition between consecutive layers in a neural network, with $d_0 = n$ and $d_\ell = 1$. Let X_0 be a random variable drawn from $\mathcal{D}$, and let $X_1, \ldots, X_\ell$ be random variables corresponding to intermediate activations of the network:

$$X_0 \xmapsto{f_1} X_1 \xmapsto{f_2} \cdots \xmapsto{f_\ell} X_\ell = f(X_0)$$

A natural DPE for $\mathbb{E}[X_\ell]$ corresponds to a choice of:

- A tractable family of distributions $\mathscr{D} \subset \Delta(\mathbb{R}^d)$
- A notion of "closeness" between any two distributions in $\Delta(\mathbb{R}^d)$
- A procedure for deductively computing the closest distribution in $\mathscr{D}$ to any given distribution $A \in \Delta(\mathbb{R}^d)$. Call this the "projection of A onto $\mathscr{D}$."

To execute the DPE, we initialize[8] $D_0 = \mathcal{D}$, then iteratively (for $i = 1, \ldots, \ell$):

[8] If $\mathcal{D} \notin \mathscr{D}$, we may have to project it down first.

- Compute the "pushforward" distribution $f_i(D_{i-1})$.
- Project $f_i(D_{i-1})$ back down to the closest distribution in $\mathscr{D}$. Call this D_i.

Our final output is $\mathbb{E}[D_\ell]$, which is easy to compute as long as $\mathscr{D}$ is sufficiently tractable. Returning to Definition 2.1, this algorithm is a DPE in which the intermediate states D_i correspond to approximations of the distributions of the random variables X_i. If the projection steps were skipped, the algorithm would be exact.

A concrete example of a DPE of this form is *covariance propagation*, in which $\mathscr{D}$ corresponds to the family of all multivariate Gaussians, closeness is measured in terms of KL divergence, and projection onto $\mathscr{D}$ can be done by analytic moment matching.

Remark 2.1

In previous sections, we discussed the role of heuristic arguments $\pi_1, \ldots, \pi_n$ in "pointing out" observations about X relevant for helping the estimator do its job. Where are these heuristic arguments found in the ontology of Deduction-Projection Estimators?

One answer is that they no longer exist. Due to its universality, $\mathbb{G}$ needs to be provided arguments to perform reasonably. By contrast, each DPE is specialized for estimating a particular type of expression. In some sense, the insights previously provided by π have been "hard-coded" into the estimator itself.

A second possible answer is that $\pi_1, \ldots, \pi_n$ correspond to a DPE's choice of representation for the simpler subspaces, as well as the way in which the projection onto these subspaces is performed. When we talk about a particular DPE, we're really studying the partial application of a general estimator on a fixed set of arguments.

This philosophical confusion has little importance once we begin describing concrete algorithms.

2.1 Covariance propagation

We now study a natural example of a DPE called covariance propagation, first described in [CNX22] in the setting of general arithmetic circuits. We take the more concrete approach here by using the algorithm to estimate the expectation of a multilayer perceptron (MLP). Say we are given an MLP of the form:

$$f(x) = A_\ell \mathrm{ReLU}(A_{\ell-1}\mathrm{ReLU}(\cdots \mathrm{ReLU}(A_2 \mathrm{ReLU}(A_1 x + b_1) + b_2) \cdots) + b_{\ell-1}) + b_\ell,$$

where $A_i \in \mathbb{R}^{d_i \times d_{i-1}}$, $b_i \in \mathbb{R}^{d_i}$, and $\mathrm{ReLU} : \mathbb{R}^d \to \mathbb{R}^d$ is a nonlinear activation function that independently maps each coordinate z to $(z + |z|)/2$. In other words, f alternates between applying affine transformations $(x \mapsto A_i x + b_i)$ and ReLUs. See Figure 4 for a visualization of an MLP.

We are tasked with estimating the expectation of the MLP under a standard Gaussian input:

$$\mathop{\mathbb{E}}_{x \sim \mathcal{N}(0, \mathrm{Id}_{d_0})}[f(x)].$$

Covariance propagation approaches this problem by modeling the activations at each internal layer as a multivariate Gaussian with some mean and covariance. At the input layer, the distribution over x is known to be Gaussian with mean $\mu^{(0)} = 0$ and covariance $\Sigma^{(0)} = \mathrm{Id}_{d_0}$ (parenthesized superscripts are used on μ and Σ to index layers in the MLP, while subscripts will index coordinates within a layer). At each subsequent layer i, we pick the Gaussian $\mathcal{N}(\mu^{(i)}, \Sigma^{(i)})$ that minimizes the Kullback-Leibler divergence $\mathrm{KL}(f_i(\mathcal{N}(\mu^{(i-1)}, \Sigma^{(i-1)})) \| \mathcal{N}(\mu^{(i)}, \Sigma^{(i)}))$.

How do we minimize this KL? A well-known fact is that, given any distribution P over $\mathbb{R}^n$, the Gaussian

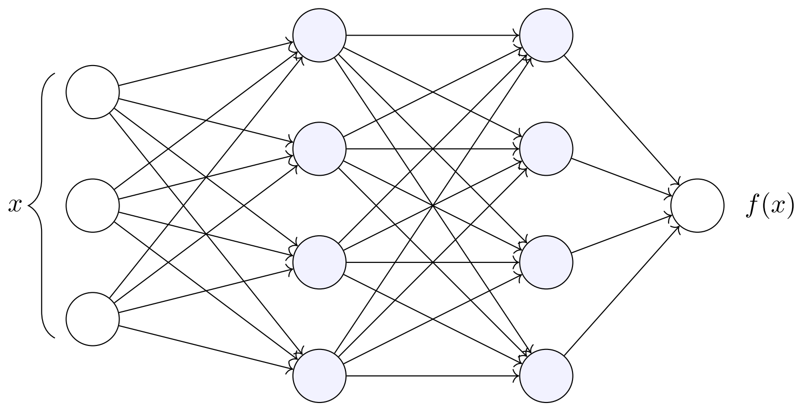


Figure 4: An MLP from $\mathbb{R}^3 \to \mathbb{R}$ with two hidden layers (biases not shown). The value of each neuron is a scalar given by the weighted sum of all incoming neurons. ReLUs are applied at each shaded neuron.

$\mathcal{N}(\mu, \Sigma)$ that minimizes $\mathrm{KL}(P \| \mathcal{N}(\mu, \Sigma))$ can be found by matching its first two moments:

$$\mu = \mathop{\mathbb{E}}_{x \sim P}[x] \qquad\qquad \Sigma = \mathop{\mathbb{E}}_{x \sim P}\left[(x-\mu)(x-\mu)^T\right].$$

Thus, the task reduces to deductively propagating the first and second moments of a Gaussian through the layers of the MLP. The affine transformations are easy:

$$\mathop{\mathbb{E}}_{x \sim \mathcal{N}(\mu,\Sigma)}[Ax+b] = A\mu + b \qquad\qquad \mathop{\mathrm{Cov}}_{x \sim \mathcal{N}(\mu,\Sigma)}[Ax+b] = A\Sigma A^T.$$

Propagating a mean through a ReLU is also straightforward. If $x \sim \mathcal{N}(\mu, \Sigma)$, then for any coordinate $j \in [d]$, the expected value of $\mathrm{ReLU}(x)_j = \mathrm{ReLU}(x_j)$ is the mean of a truncated normal with center μ_j, scale parameter Σ_{jj}, and lower bound 0, multiplied by the probability that it is positive:

$$\mathop{\mathbb{E}}_{x \sim \mathcal{N}(\mu,\Sigma)}[\mathrm{ReLU}(x)]_j = \mu_j \Phi(\alpha) + \sqrt{\Sigma_{jj}}\varphi(\alpha),$$

where $\alpha = \frac{\mu_j}{\sqrt{\Sigma_{jj}}}$ and φ, Φ are the PDF and CDF of a standard normal, respectively.

However, propagating a *covariance* through a ReLU is much harder. Given any two $j, k \in [d]$, finding the second moment $\mathbb{E}[\mathrm{ReLU}(x)_j \mathrm{ReLU}(x)_k] = \mathbb{E}[\mathrm{ReLU}(x_j)\mathrm{ReLU}(x_k)]$ requires using the *bivariate* normal CDF [Ros61]. The bivariate normal CDF has no closed form—the best approximation techniques we have use quadrature or Monte Carlo methods to perform numerical integration.[9] Using these numerical methods still gives a valid algorithm, but in some sense, it trades off against the "deductiveness" of the estimate. Overall, however, it would still have almost all of the desirable properties of a purely deductive estimate because it only uses sampling for computing a local, two-dimensional subproblem (see Remark 2.2).

An alternative to using the bivariate normal CDF is to break each $\mathrm{ReLU} : \mathbb{R}^d \to \mathbb{R}^d$ into a sequence of d ReLUs on individual coordinates. Starting with $\mathcal{N}(\mu, \Sigma)$, we ReLU the first coordinate, then project back down to a Gaussian, then ReLU the second coordinate, then project, and so on. This corresponds to an "algebraic circuit re-write": we change transition functions between intermediate states in a way that keeps the overall circuit mathematically equivalent, although the total number of transition functions greatly increases. While this approach is likely to produce less accurate estimates — the quality of the Gaussian approximation degrades with each nonlinearity applied — it has the benefit of being possible to compute deductively.

When we only ReLU a single coordinate at a time, the covariance calculation is much simpler. Say

[9]Note that the single-dimensional CDF $\Phi(x) = \frac{1}{2}\left[1 + \mathrm{erf}(x/\sqrt{2})\right]$ *also* requires numerical methods to compute, but this can be done very simply and efficiently with an approximation of erf to a rational function [Cod69].

we wish to calculate $\mathbb{E}[\mathrm{ReLU}(x_j)x_k]$. We can write $x_k = x_j \cdot \Sigma_{jk}/\Sigma_{jj} + z_{jk}$, where z_{jk} is a Gaussian random variable independent from x_j. Then the only interesting term reduces to $\mathbb{E}[\mathrm{ReLU}(x_j)x_j] = \mathbb{E}[\mathrm{ReLU}(x_j)^2]$, which is simply the second moment of a truncated normal, scaled down by the probability it is positive:

$$\mathop{\mathbb{E}}_{x\sim\mathcal{N}(\mu,\Sigma)}[\mathrm{ReLU}(x_j)x_j] = (\Sigma_{jj} + \mu_j^2)\Phi(\alpha) + \mu_j\sqrt{\Sigma_{jj}}\varphi(\alpha).$$

Remark 2.2

Among these two approaches—using a numerical (possibly randomized) method for approximating the bivariate CDF, or applying the ReLUs coordinate by coordinate—which is "better"?

The author believes that the first approach is superior. The numerical method will likely produce more accurate estimates and is more efficient, although the time complexities are sensitive to the implementation details of the bivariate CDF approximation. Finally, even if we have a moral aversion to non-analytic (i.e., numerical or sampling-based) methods, I argue that we should be philosophically unbothered by numerical methods used in low-dimensional subroutines. In this case, we only apply numerical methods to a 2-dimensional problem. In some sense, low-dimensional settings are small enough that inductive algorithms can "explore the entire search space" after checking enough inputs (this would make more sense if our input space were finite). The curse of dimensionality only kicks in when we increase the number of inputs to the problem, requiring an efficient inductive estimator to generalize to unseen regions of input space.

The mechanics of covariance propagation are very similar to Assumed Density Filtering (ADF), a well-known Bayesian inference method. ADF involves updating a posterior distribution by repeatedly incorporating new measurements, then projecting back down to a Gaussian with moment matching [Ran04]. The key difference between covariance propagation and ADF is that our distributions are directly transformed via layers of a neural network rather than through Bayesian updates.

It is easy to identify simple examples in which the analytic computation of covariance propagation gives big improvements over a sampling-based algorithm. For example, if we are given an MLP that has an exponentially small likelihood of outputting a non-zero value (e.g., $\mathrm{ReLU}(x - 10)$ for x standard normal), a sampling-based estimator would be unable to distinguish this network from one that is identically zero. In these cases, covariance propagation would be better than sampling at dealing with questions like "What is the probability that this network outputs a value greater than 1?" (where estimation quality is measured on a log scale). We explore this idea of low probability estimation in Sections 3.1 and 5.

The more that internal layer activations in an MLP are non-Gaussian, the less accurate covariance propagation will be. One way to generalize covariance propagation would be to track the first k joint moments rather than only the first 2; this is known as *cumulant propagation* ([CNX22] appendix D).[10] Alternatively, we could use a more expressive family of distributions such as a mixture of Gaussians or a sum of independent linear one-dimensional features as in [CMR24].

[10]Unlike covariance propagation, cumulant propagation for $k > 2$ does not have a nice interpretation in terms of projection onto some simpler set of distributions.

3

Relation to AI Alignment

In this section, we explore how deductive estimators may be valuable for solving problems in *AI alignment*, a subfield of machine learning that studies how to get AI systems to do what their developers want, even as they scale beyond human capabilities.

AI companies like OpenAI, Anthropic, and Google DeepMind are currently racing at breakneck speeds to build increasingly smart AI systems. Recent progress has been fast. GPQA Diamond is a benchmark of multiple choice PhD-level science questions, intended to be challenging for human experts even when given access to the internet [RHS+23]. GPT-4, released in June of 2023, scored 31% on this benchmark (random guessing corresponds to 25%). Only a year and a half later, OpenAI's o3-mini model scored 77%, surpassing the accuracy of expert humans (70%) [Epo24].[11] Progress has been just as fast on open-ended tasks that require agentic capabilities. On SWE-bench Verified, a benchmark that asks language model agents to solve real-world GitHub issues, the top score on the public leaderboard increased from 4.4% (an agent based on Claude 2 in October 2023) to 64.6% (an agent based on o1 in January 2025) in a little over a year [JYW+24]. And this only considers the progress *since 2023*: it's hard to believe that in 2019, the state-of-the-art language model (GPT-2) had trouble counting past ten. Recent work by METR shows that the length of tasks that AI agents can solve has been growing exponentially since 2019, with a doubling time of seven months [MET25]. If this trend continues, in under five years AI agents will be able to independently automate most software tasks that currently take humans days or weeks.

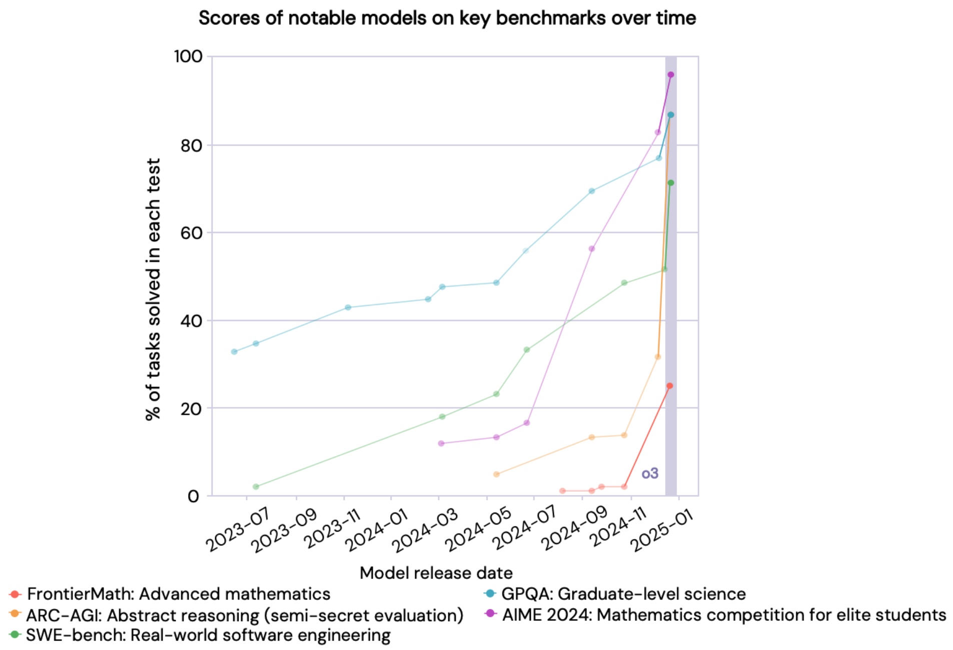


Figure 5: Recent progress in frontier AI reasoning capabilities has been fast. Image from [B+25].

[11] OpenAI claims that the full o3 model scores 87% on GPQA Diamond [FD24]. At the time of writing, the score cannot be independently validated because the model has not been released.

The stated goal of these AI scaling companies is to create artificial general intelligence (AGI): an AI system that can outcompete humans on *any* task, from analyzing poetry to proving theorems to writing persuasive speeches to negotiating foreign policy. If developed safely and responsibly, AGI could bring tremendous economic benefits and technological progress to society. However, there are also many ways in which the existence of human-level AI systems could lead to bad outcomes for humanity: for example, it could amplify existing social inequalities, assist cybercriminals and bioterrorists [DW24], drive human wages below subsistence level [Bar25], or enable extreme concentration of economic and political power among individuals who control the AI systems.

The field of AI alignment focuses on a specific way in which future AI development could go awry: problems caused by AI systems "intentionally" taking actions that it knows its human developers would disapprove of. Unlike other shortcomings of AI, alignment problems will not go away as models become more capable and intelligent. In fact, they will likely get worse because the models will be better at taking subversive actions without getting caught.

The most concerning way this could happen is if an AI system is *misaligned*—that is, it is internally motivated to pursue goals that differ from those that its human developers intended. An AI model could internalize these misaligned goals during training with gradient descent for a number of related reasons:

- **The model is trained on a reward function that fails to capture what humans actually want ("reward misspecification")** [PBS22]. For example, a chatbot trained to maximize its approval rating may learn to sycophantically agree with the user, even when the user makes mistakes [WCN+25].
- **The model is trained on a distribution of inputs that is not diverse enough to specify correct behavior during deployment ("goal misgeneralization")** [SVK+22]. In other words, there may be multiple different drives or goals that all lead to equivalent loss on the training distribution. In these cases, the inductive bias of the training setup dictates which goals gradient descent ends up baking into the model. Thus, even if the reward function perfectly captures "what humans actually want," a distributional shift between training and deployment may mean that drives learned by the model during training fail to generalize to getting good reward in deployment.[12]
- **The model develops arbitrary internal goals early on due to inductive biases, and then takes actions to preserve those goals during the remainder of training ("deceptive alignment")** [Car23]. This can occur when an AI model realizes that its internal goals differ from those of its human developers, but it knows that gradient descent will modify its goals if it acts upon them during training (which is undesirable from the model's perspective). As a result, the model pretends to be aligned during training, only to pursue its misaligned goals once it is deployed. See [GDW+24] for empirical evidence of this "alignment faking" phenomenon in language models.[13]

This third possibility is especially concerning because it means that it may be very hard for us to tell whether our models are aligned. In fact, a misaligned AI may employ the following strategy upon receiving an input x:

- The model studies x for clues that indicate whether it is in training or deployment (e.g., the dates on newspaper headlines or the types of problems it is being asked to solve).
- If the model believes that x could plausibly arise during training, it acts exactly as an aligned AI would.
- Otherwise, the model assumes that it is in deployment and takes actions to pursue its own misaligned goals. This may include subverting human safeguards to prevent itself from being

[12]A model may even be "aware" that its behaviors during deployment would score poorly according to its original reward function, but it would have no reason to care about this.

[13]Deceptive misalignment can be thought of as a special case of goal misgeneralization. The goals "obey humans" and "obey humans only when the input looks like it is from training" cannot be disambiguated on the training distribution, but they result in wildly different behavior on points from the deployment distribution.

shut down in the future (e.g., exfiltrating its weights to create backup copies of itself or inserting vulnerabilities into monitoring systems).

An AI that follows this policy and is sufficiently perceptive may be behaviorally indistinguishable from an aligned AI until we deploy it into the real world, after which it is too late.[14]

There are many proposals for dealing with deceptive alignment.

Red-teaming is a form of adversarial machine learning in which researchers try to get a model to take undesirable actions under testing conditions, including revealing its misalignment [PHS+22]. For example, we might try to bait AIs with "honeypots": realistic-looking inputs that would be tempting opportunities for a misaligned AI to take a subversive action. If we catch a model falling for such a honeypot, we can use it as evidence to pause further development until we have better tools to deal with misalignment. Applying adversarial training to reduce the chance that an AI falls for a honeypot is also possible, but should be done with caution: it's unclear whether this would actually improve the alignment of the model, or if it would simply teach it to be better at distinguishing between real and synthetic inputs [HDM+24].

AI control is an approach that involves monitoring an untrusted AI system during deployment such that *even if it were* egregiously misaligned, it would be unable to take a catastrophic action without getting caught. It is too expensive and slow to have humans monitor all of an AI's actions, so we instead use other AIs as monitors. There are numerous clever strategies for dealing with the obvious concern that the monitors themselves may be misaligned [GSSR24]. Note that AI control is not intended to be infinitely scalable: a sufficiently superintelligent AI system may be able to break through any human-designed control scheme.

Finally, some approaches attempt to take advantage of the fact that researchers have direct access to the weights of the models. Although the calculations performed by trained neural networks are largely inscrutable, we might hope to develop a better understanding of a model's internal cognition with enough dedicated research effort. **Mechanistic interpretability** attempts to reverse engineer the computation performed by a neural network directly from its learned weights. This can be thought of as "neuroscience for AI," with the ultimate goal being to leverage our understanding of a model's internals to detect misalignment and improve our ability to oversee the actions of AI systems [BG24].

There have been some recent breakthroughs in mechanistic interpretability, including the use of sparse autoencoders for identifying human-understandable "feature directions" in a model's activation space [CER+23] and their application to simple downstream tasks [MRM+24]. However, we are still far from our ambitious goal of understanding model internals well enough to make confident claims about our ability to detect and mitigate misalignment risks.

A problem with current techniques in mechanistic interpretability is that they often make assumptions about how information is processed in the model, such as the linear representation hypothesis [EHO+22]). They also often require humans to understand the high-level concepts that a model operates on. Although this may be true for current models, there is a risk that these assumptions will break down with future AI systems (e.g., if they reason using alien concepts that do not fit naturally into our ontology).

Formal deductive estimation presents one solution to these issues: a way to leverage model internals to address misalignment concerns, without relying on a human understanding of what occurs inside the model. The key intuition is that deductive estimates mechanistically capture "why" a mathematical fact is true, so when applied to a neural network they can be used to deal with the model on the level of *reasons* instead of *observed behaviors*, even when the formal deductive estimates themselves are uninterpretable to humans.

The next two subsections describe more concrete, though still theoretical, applications of deductive estimates for solving alignment failures: low probability estimation and mechanistic anomaly detection.

[14] As an analogy, human politicians may purport to have certain beliefs on the campaign trail for the sake of electability, but then act according to their true beliefs once they are in power. This is for reasons similar to deceptive alignment.

To pose these problems, we make some assumptions about access to certain idealized functions such as a catastrophe detector C or a sensor predictor P. This is so that we can study core theoretical difficulties in isolation from empirical contingencies. However, a full solution to the alignment problem would have to further justify or move beyond these assumptions.

3.1 Low probability estimation

[Xu24] presents the following scenario. Suppose we have a powerful AI model $M : \mathcal{X} \to \mathcal{Y}$ that we've trained as a research assistant. Given a project description $x \in \mathcal{X}$, the model outputs some proposed action $M(x) \in \mathcal{Y}$. Most of the actions that M proposes are benign, but we are worried that on certain rare inputs, M will propose an action that would be catastrophically bad to carry out (e.g., deploying a biological weapon). One reason for this might be that M is misaligned but very patient about planning its takeover attempts, to avoid getting caught by red-teaming.

Further, suppose we have a formally defined catastrophe detector $C : \mathcal{Y} \to \{0, 1\}$ that takes as input a proposed action and predicts whether that action will lead to large-scale harm. Assume in this scenario that C is perfectly reliable but too expensive to run every time we want to use M. For concreteness, we could imagine that C is a weaker model like Claude 3.5 Sonnet (which is weak enough that we are confident it is not egregiously misaligned) that is granted an extremely large amount of inference-time compute to think through all of the potential ramifications of the proposed action.

Finally, assume we have a formally defined distribution of project description inputs $\mathcal{D}$. The *low probability estimation* problem asks us to estimate:

$$\Pr_{x \sim \mathcal{D}}[C(M(x)) = 1].$$

In particular, we would like to get decent estimates even when the probability is very close to zero. If we could do this, it may allow us to distinguish the misaligned-but-very-patient model from an aligned model.

$C \circ M$ is simply a large computational circuit. Thus, algorithms inspired by deductive estimators might give us traction on this problem, especially when the probability is too small to detect with sampling-based approaches.

Section 5 discusses low probability estimation in significantly more detail.

3.2 Mechanistic anomaly detection

We might worry that a misaligned model will engage in *sensor tampering*: taking actions that appear good based solely on the measurements used to define a reward function, but that we would strongly disapprove of if we had complete information about the AI's actions.

Consider the following toy example, first presented in [CXC21]. Suppose we wish to train an AI model $M : \mathcal{X} \to \mathcal{Y}$ to guard a diamond in a vault such that, when presented with situation $x \in \mathcal{X}$ threatening the diamond (e.g., a robber coming to steal it), it proposes a series of actions $M(x) \in \mathcal{Y}$ (e.g., opening up a trap door to foil the robber) that results in the diamond staying in the vault by the end of the day.

However, we do not have direct access to the ground truth of whether or not the diamond is in the vault. Instead, we have a camera installed in the vault that takes a picture of the vault at the end of the day, along with a separate reward model $R : \mathcal{I} \to \{0, 1\}$ that looks at the image and outputs 1 if and only if it appears to contain a diamond. Finally, assume we have a perfect predictor model $P : \mathcal{X} \times \mathcal{Y} \to \mathcal{I}$ that takes as input a situation and a proposed series of actions, then perfectly predicts what the camera would show at the end of the day if we were to perform those actions. Assume that

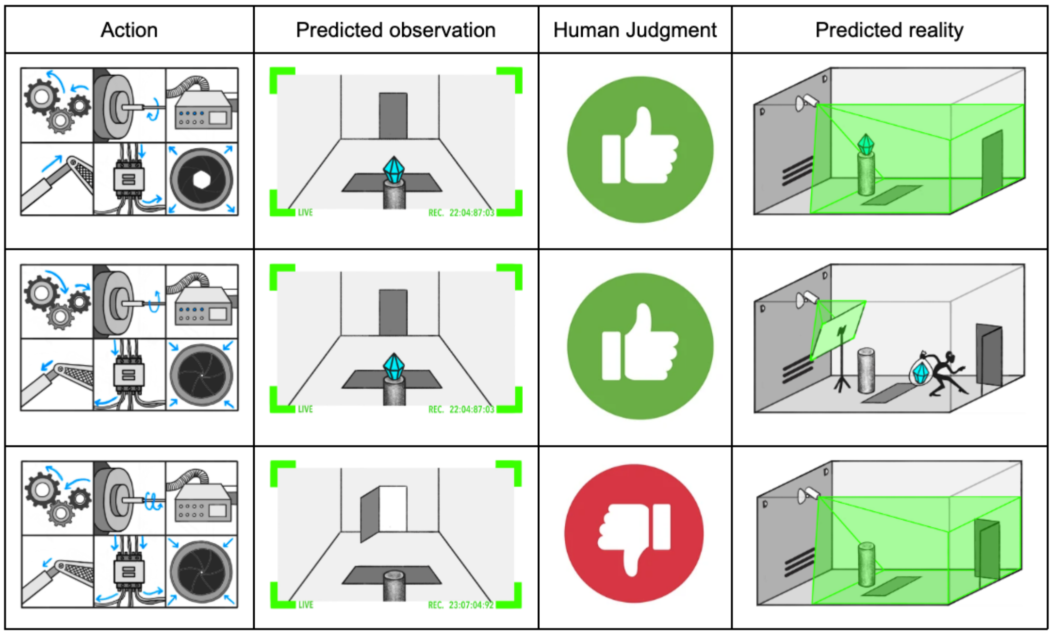


Figure 6: From the camera readings alone, it is impossible to tell if the diamond actually remains in the vault (top), or if the sensors have been tampered with (middle). Image from [CXC21].

R and P are reliable.[15]

We train M to take actions that cause the camera to show a diamond at the end of the day. This corresponds to the reward function:

$$\text{reward}(M) = \mathop{\mathbb{E}}_{x \sim \mathcal{D}} [R(P(x, M(x)))]$$

When training on $\mathcal{D}$, we understand the situations well enough to be confident that M gets high reward for the "right reasons": the diamond actually remains in the vault by the end of each day. We end up with a trained model that consistently gets a reward of 1.

Now, suppose we encounter a novel input x^* that is hard for us to understand (e.g., it involves a robbery scheme that is too complex for us to track). When we plug this input into M, it suggests a cryptic series of actions. Although we don't understand *why* it works, when we run the actions through our predictor model it appears to keep the diamond safe: $R(P(x^*, M(x^*))) = 1$.

However, we might be worried that pixels in the shape of a diamond are predicted to appear on the camera screen for the wrong reasons. For example, $M(x^*)$ may have included actions to place a realistic-looking cardboard cutout showing a diamond directly in front of the camera, while allowing the robbers to steal the real diamond (Figure 6). This is known as sensor tampering.

We would like to know which scenario we are in before we execute $M(x^*)$. How do we distinguish between $R(P(x^*, M(x^*)))$ being 1 for the right reasons (the diamond is actually in the vault) versus the wrong reasons (M engaged in sensor tampering)? This is known as *mechanistic anomaly detection.*

One proposed solution is to use deductive heuristic arguments. If we had a universal estimator $\mathbb{G}$ as described in Section 1.1, as well as an efficient procedure for discovering heuristic arguments of properties of neural networks, we could employ the following algorithm:

- Search for a heuristic argument π that "explains" to $\mathbb{G}$ why M reliably protects the diamond on

[15] R might be reliable because it is too weak to be egregiously misaligned. This argument does not work for P, however, because P has to be "as smart" as M in order to accurately predict the results of its actions. Mechanistic anomaly detection might still work if P is misaligned, though we don't focus on this possibility here for simplicity.

the training distribution:

$$\mathbb{G}\left(\mathop{\mathbb{E}}_{x \sim \mathcal{D}} [R(P(x, M(x)))] \,\middle|\, \pi \right) \approx 1 \tag{1}$$

- Check to see if π also explains why M appears to protect the diamond in this new scenario:

$$\mathbb{G}\left(R(P(x^*, M(x^*))) \,\middle|\, \pi \right) \overset{?}{\approx} 1 \tag{2}$$

- If so, conclude that $M(x^*)$ is safe to execute. Otherwise, flag $M(x^*)$ as "anomalous" behavior and avoid it.

The hope is that, in order to satisfy Equation 1, π will need to point out all of the internal structure of M, P, and R relevant for M getting high reward on the training distribution.[16] But we don't want π to explain *everything* about the models: importantly, we want it to *omit* any reasoning about why sensor tampering would result in P believing that pixels in the shape of a diamond will appear on the screen.[17] This allows us to use Equation 2 to test whether the fact "$R(P(x^*, M(x^*))) = 1$" can be explained purely by the standard reasons that arose during training, or if it requires a novel (i.e., anomalous) explanation.

Remark 3.1

There are many issues with the approach to mechanistic anomaly detection as currently presented. For instance, what if novel reasons are required to explain why $M(x^*)$ keeps the diamond safe, but these reasons are benign? This might happen if M is forced to deter robbers using security techniques never seen in the training distribution. Our anomaly detection algorithm would have a prohibitively high false positive rate if it were unable to distinguish between these cases and true sensor tampering.

Although we have no formal definition of desirable versus undesirable anomalies, one hope is that sensor tampering has a distinguishing property: it can lead to inconsistent-looking outcomes between multiple sensors if not fully explained. Thus, a possible solution is to install *multiple* cameras in different locations around the room. We then check to see if there exists a π that satisfies Equation 1 while also causing $\mathbb{G}$ to believe that $P(x^*, M(x^*))$ predicts *inconsistent* camera readings (e.g., the diamond is visible on one screen but missing from another screen). Only in these inconsistent cases do we flag $M(x^*)$ as anomalous. Our hope is that benign anomalous behavior (like novel security techniques) should never be messing with the sensors in the first place, so $\mathbb{G}(\cdot|\pi)$ won't predict that it will result in inconsistent camera readings.

This natural buffer between corrigible and uncorrigible actions is explored in [Chr22].

Summary. AI alignment deals with problems that arise when AI systems intentionally take subversive actions. Deductive estimators are a novel tool that could be used to solve key alignment problems. Unlike traditional mechanistic interpretability, this approach makes few assumptions about the extent to which humans can understand the inner workings of future models. Solving this problem is particularly urgent given the pace of recent AI progress.

[16] Note that π will be massive—as big as the models themselves—since it has to be able to explain why M is as capable as it is. But this is fine for our purposes because we never require a human to "understand" π.

[17] In fact, we may want to regularize π in some way so that it will only include reasoning steps that are necessary for satisfying Equation 1.

4
The No-Coincidence Principle

The applications of deductive estimators presented in the previous section require high-quality heuristic arguments for behaviors of neural networks to both *exist* and be *efficiently discoverable*. Is this a reasonable hope?

Intuitions from proof theory would suggest no to both questions. For example, Gödel's Incompleteness Theorem demonstrates that some true statements do not have a proof of any length within a formal system [Göd92]. While the statements Gödel worked with were quite pathological, there are also many natural statements in math that have eluded proof for centuries (e.g., the twin prime conjecture, Goldbach's conjecture, and the normality of π).

Even when a statement *can* be proven, its proof is often quite long relative to the description length of the theorem. The original proof of the four-color theorem required a computer-assisted casework check of 1,936 configurations, and Andrew Wiles's proof of Fermat's Last Theorem was 129 pages long. More generally, as long as $\mathbf{NP} \neq \mathbf{coNP}$, there do not exist polynomially-lengthed proofs of the unsatisfiability of 3CNF formulas (if there did exist such proofs, then UNSAT would be in $\mathbf{NP}$).

Finally, even when a concise proof of a statement exists, it may require an exponential amount of work to discover it (assuming $\mathbf{P} \neq \mathbf{NP}$). The hardness of 3SAT immediately implies that statements like "the 3CNF formula $\phi(x)$ is satisfiable" often have short proofs that are hard to find. This is conjectured to be true even in the *average* case for random k-SAT instances with the appropriate clause density [BT21].

However, our hope is that these obstacles to efficiently finding concise proofs will no longer apply if we relax our standard of rigor to heuristic reasoning. Compared to proofs, heuristic reasoning allows more freedom in the types of arguments that are considered "valid," which may make them significantly easier to discover. Of course, this freedom comes at a cost: heuristic arguments are defeasible while proofs are certain.

Evidence for this belief comes from the fact that many statements in math that have been difficult to prove can be explained with very simple and easy-to-find heuristic arguments. Oftentimes, these arguments are deductive. Heuristic arguments for Goldbach's conjecture and the twin prime conjecture were mentioned in Section 1. [Tao12] gives a heuristic argument for Fermat's Last Theorem, which states that $x^n + y^n = z^n$ has no integer solutions for $n \geq 3$,[18] as well as the ABC conjecture in number theory. These arguments are more involved, but they are still vastly simpler than their respective proofs (in particular, the ABC conjecture is still widely considered unproven).

It is probably too much to ask for *every* true mathematical statement to have a concise heuristic argument. For example, although "the first 10^5 digits of π contain more odds than evens" is true (I checked), there is probably no way to convincingly argue for it without a very long calculation, even heuristically. However, there's a sense in which a statement like this does not "demand explanation" in the first place—a uniformly random sequence of digits would satisfy this statement around 50% of the time, so it's not surprising that it turns out to be true for the actual digits of π.

[18] I will not reproduce the full argument here, but the core idea is as roughly as follows. To estimate the number of solutions to $x^n + y^n = z^n$ among the naturals, we can heuristically say that $a \in \mathbb{N}$ is an n-th power with probability $a^{1/n-1}$ (because the number of n-th powers in $\{1, \ldots, a\}$ is roughly $a^{1/n}$). Thus, treating the events "a is an n-th power," "b is an n-th power," and "$a + b$ is an n-th power" as independent, the number of solutions should look something like $\sum_{a=1}^{\infty} \sum_{b=1}^{\infty} a^{1/n-1} b^{1/n-1} (a+b)^{1/n-1}$, which converges for all $n > 3$. Resolving the $n = 3$ case requires extra structural observations about elliptic curves.

This motivates a more refined view on the applicability of heuristic arguments: we only demand they exist on statements that are naively "surprising." [Gow19] summarizes this as follows.

Definition 4.1. (Informal)

The **No-Coincidence Principle**: If an apparently outrageous coincidence happens in mathematics, then there is a reason for it.

Here are some examples of statements that would qualify as "apparently outrageous coincidences," and what the No-Coincidence Principle suggests in each case.

A particular neural network M_θ gets very low loss on a formally defined task. This is an outrageous coincidence in the sense that a *random* function would have much worse performance on this task than this particular network does. To be a bit more careful, we would want to establish that it would be very unlikely for even the highest-performing random function from a search space of $2^{O(|\theta|)}$ such functions (corresponding to the number of models of the same size as M_θ) to achieve the loss that M_θ does. If this is the case, the No-Coincidence Principle implies that there must be a reason (e.g., a heuristic argument) that explains this fact.

"π is not a normal number" would be an outrageous coincidence because a random sequence of digits is normal with probability 1. As far as we can tell, there is no reason for the digits of π to not be maximally random-looking. Applying the contrapositive of the No-Coincidence Principle leads us to believe that π is normal.

"For almost every n, among the first n primes, more of them have the form $4k+3$ than $4k+1$." This turns out to be true, and it is known as "Chebyshev's bias" [RS94]. Let $f_{i,4}(n)$ be the number of primes equal to i mod 4 among the first n primes. If the primes were distributed randomly among the viable residues mod 4, then $g(n) = f_{3,4}(n) - f_{1,4}(n)$ would behave like an unbiased random walk. It would be an outrageous coincidence for an unbiased random walk to be positive much more often than it is negative. However, this empirically turns out to be the case for g. In fact, $g(n)$ is only negative one time for all n up to 10^4; this only has about a 1% chance of happening with a truly random walk. The No-Coincidence Principle implies that there must be some heuristic argument that explains this fact. Indeed, we have a heuristic understanding of why it should be true [Tao16], while the only proofs we have are conditioned on strong forms of the Riemann Hypothesis.

4.1 Complexity-theoretic formulation

[Ney25] attempts to formalize a crisp, complexity-theoretic conjecture based on the intuitions provided by the No-Coincidence Principle.

First, some definitions. A *reversible circuit* is a boolean circuit where each gate maps 3 bits to 3 bits in a bijective manner (so, there are 8! possible reversible gates). Consider the distribution[19] over ℓ-layer reversible circuits $C : \{0,1\}^{3n} \to \{0,1\}^{3n}$ in which each layer consists of n parallel gates, each gate operates on 3 randomly selected wires from the previous layer (such that each wire is used in exactly one gate), and the operation applied by each gate is sampled independently from the set of all 8! bijections on $\{0,1\}^3$. A circuit sampled from this distribution is called an ℓ-layer *random reversible circuit.*

We are now ready to state the conjecture.

[19]The exact definition of the distribution probably does not matter much for the conjecture, as long as it is deep enough that a random circuit looks like a random permutation.

Conjecture 4.1. No-Coincidence Principle for Reversible Circuits

For a reversible circuit $C : \{0,1\}^{3n} \to \{0,1\}^{3n}$, let $P(C)$ be the property that there is no input x to C that ends in n zeros, such that $C(x)$ also ends in n zeros. There exists a polynomial-time verification algorithm V that receives as input:

- A reversible circuit $C : \{0,1\}^{3n} \to \{0,1\}^{3n}$
- A hint string π

such that:

- For all C such that $P(C)$ is true, there exists π with length polynomial in the size of C, such that $V(C, \pi) = 1$.
- For 99% of n-layer random reversible circuits C, no such π exists.

In what sense is Conjecture 4.1 an instantiation of the No-Coincidence Principle?

It is believed that random reversible circuits of depth at least $\Omega(\sqrt{n})$ form pseudorandom permutations, meaning a polytime algorithm with black-box access to the circuit cannot distinguish it from a truly random permutation with non-negligible probability [GHKO25]. If we were to treat a random reversible circuit C as computing an *actually* random permutation, what would be the probability of $P(C)$?

There are 2^{2n} inputs $x \in \{0,1\}^{3n}$ that end in n zeros. For each such input, there is a 2^{-n} probability that $C(x)$ also ends in n zeros. Thus, if C were a random permutation,[20]

$$\Pr_C[\exists x \in \{0,1\}^{2n} \times \{0\}^n \text{ s.t. } C(x) \in \{0,1\}^{2n} \times \{0\}^n] \approx 1 - (1 - 2^{-n})^{2^{2n}} \approx 1 - \exp(-2^n)$$
$$\implies \Pr_C[P(C)] \approx \exp(-2^n).$$

There are only $2^{\text{poly}(n)}$ reversible circuits of depth n^2, so under the assumption that each circuit is an independent random permutation, it would be an outrageously large coincidence if *any* of these circuits were to satisfy $P(C)$. The No-Coincidence Principle states that when such a coincidence occurs, there must be a reason for it: in this case, we've formalized a "reason" to be a string π that can be processed by "reason-verifier" V.

Remark 4.1. Relation to Average-Case Refutation

For any constant d such that $(7/8)^d < 1/2$, it is easy to show that a random 3CNF formula with n variables and $m = dn$ clauses is unsatisfiable with probability $1 - 2^{-\Theta(n)}$ (where each clause is drawn uniformly at random from all $(2n)^3$ triples of literals). Feige conjectured that there is no constructive version of this fact [Fei02]: namely, there is no polytime algorithm V that takes in a 3CNF formula ϕ and outputs either `UNSAT` or `IDK` such that:

- If $V(\phi)$ outputs `UNSAT`, then ϕ is unsatisfiable.
- $V(\phi)$ outputs `UNSAT` on the vast majority of random formulas ϕ.

This notion of average-case refutation is superficially similar to Conjecture 4.1. Both ask whether there exist efficient procedures for distinguishing "typical" instances of a problem from "atypical" ones, with imperfect soundness allowed (i.e., typical instances are sometimes allowed to be identified as atypical). The main difference is that Conjecture 4.1 allows V to use a hint string π. Note that the use of a hint string would trivialize Feige's conjecture, as π could simply provide a satisfying assignment to ϕ.

[20]Technically, the events are not independent because C is bijective, but this matters extremely little.

[Ney25] explains why this particular formulation of the conjecture was chosen.

- **Why imperfect soundness?** One could imagine a stronger version of this conjecture that demands perfect soundness: for any C such that $P(C)$ is false, there is no π such that $V(C, \pi)$. This is equivalent to the claim that the function P is in **NP**. Since P can easily be shown to be **coNP**-complete, this stronger version of the conjecture would be equivalent to **NP** = **coNP**, which is believed to be false.[21]
- **Why reversible circuits?** Non-reversible circuits (i.e., with AND, OR, and NOT gates) are conceptually simpler than reversible circuits. However, most natural distributions over non-reversible circuits are uninteresting because random circuits usually degenerate into computing a constant function after a certain depth (i.e., they are either always 1 or always 0) [MDMM11].
- **Why not a simpler definition of P?** Let $N = \{0,1\}^{3n}$ be the input and output space of C. The current definition of P can be interpreted as defining a set $S \subset N$ of size $|N|^{2/3}$ and asking whether all $x \in S$ satisfy $C(x) \notin S$. This definition was chosen to have the following two features:
  - The fraction of random permutations that satisfy P is double-exponentially small in n. This is important because we need $P(C)$ being true to be an "outrageous coincidence"; if not, we risk running into a C that satisfies $P(C)$ without a good reason.
  - P is not too easy to check. This rules out definitions of $P(C)$ like "does C compute the identity function?" because, under this definition, V could simply check whether $C(00\ldots0) = 00\ldots0$, without needing advice.

We[22] believe that Conjecture 4.1 is plausibly true because of our intuitions about the No-Coincidence Principle. However, Conjecture 4.1 only refers to a particular type of coincidence regarding reversible circuits, and it is not intended to represent the No-Coincidence Principle in full generality. The conjecture can be viewed as a weak form of what we hope to be true about heuristic arguments for arbitrary mathematical coincidences, particularly properties of trained neural networks. To quote [Ney25]:

> Ultimately, though, we are not wedded to our particular formulation [of Conjecture 4.1]. Perhaps there is some clever sampling-based verifier that "trivializes" our conjecture as well, in which case we would want to revise it. We are ultimately interested in the informal claim that if a circuit exhibits a very surprising property, it is possible to point out some internal structure of the circuit that will "explain" to a verifier why the surprising property holds.
>
> [...]
>
> Our belief in the existence of such explanations follows from a more general belief: that all surprising mathematical structure has an explanation, in the sense of Gowers' no-coincidence principle.
>
> From our discussions with other researchers, we have gotten the impression that some agree with this overarching intuition while others do not. On the other hand, it seems difficult to argue about the truth or falsehood of such an informal statement. Thus, our attempt at formalization is in part motivated by wanting to explain more concretely what we believe.
>
> If our conjecture is false, we would like to know. It may cause us to lose faith in our belief that neural networks are explainable (in the way that we are using the word "explainable"), and to pivot to a new research direction.

[21]The author finds it interesting that relaxing the soundness from 100% to 99% turns a purely complexity-theoretic statement (**NP** vs **coNP**) into what we claim to be a conjecture about identifiable structure in circuits.

[22]In this paragraph, "we" refers to the Alignment Research Center, myself included.

4.2 Tractability of finding arguments

If the No-Coincidence Principle is true, it guarantees the *existence* of heuristic explanations for surprising mathematical facts. Unfortunately, it makes no guarantee that they will be *easy to find* (let's call this the "search problem" for heuristic explanations). How much can we hope for here?

The best case outcome here would be that the search problem is easy in general:

> **(Prospect A)** *Every coincidence has an efficiently discoverable explanation.*

Informally, Prospect A would imply that any type of structure in a mathematical expression is easy to identify, no matter how complex it is. This seems way too good to be true.

A more realistic reason for hope would be that we do not care about the complexity of the search problem in the worst case, as long as it is tractable for any instance of the problem that we expect to encounter. For instance, we may hope for a world in which any mathematical coincidence that we might stumble into after performing T steps of computation has a heuristic explanation that can be found in $O(T)$ time. Thus, while there may exist many mathematical coincidences for which the search problem is extremely difficult, we would hope to never systematically encounter them.

> **(Prospect B)** *Efficiently discoverable coincidences have efficiently discoverable explanations.*

A more precise way to characterize this might be: "Given any randomized polytime procedure for generating an infinite family of coincidences, there is a polytime algorithm for finding corresponding explanations."[23] Prospect B would be sufficient for our ultimate goal of applying heuristic arguments to solve problems in AI alignment since we only spend a polynomial amount of time training an AI model in the first place. The situation here would be morally similar to *Heuristica* in Russell Impagliazzo's famous Five Possible Worlds of computational complexity, in which he supposes that **NP** problems are intractable in the worst case, but tractable on average for any samplable distribution of problems [Imp95].

Unfortunately, Prospect B also seems unlikely. Indeed, most computer scientists currently think that the world outlined by *Heuristica* is implausible and that one-way functions probably exist. To gain a better intuition for this, consider the following concrete example.

If Prospect B is true, then the verifier from Conjecture 4.1 should not need a hint string π when given circuits that can be efficiently discovered.[24] In other words, there would exist a polytime algorithm V that *only* takes in a circuit, such that:

- $V(C) = 1$ for all efficiently discoverable C such that $P(C)$ is true.[25]
- $V(C) = 0$ for 99% of n-layer random reversible circuits C.

Say we construct a reversible circuit $C_0 : \{0,1\}^{3n} \to \{0,1\}^{3n}$ such that $P(C_0)$ is true. We might initially have some hope that V can correctly distinguish C_0 from a random circuit. This is because, when generating C_0, we were forced to make sure that it satisfies $P(C_0)$ (e.g., by adding an if-else statement to separately handle inputs that end in n zeros). This process might have left some detectable "fingerprint" that V could pick up on.

However, as long as indistinguishability obfuscation (iO) exists,[26] it is possible to transform C_0 into a functionally equivalent circuit C_1 such that no polytime procedure can distinguish C_1 from a randomly sampled circuit of the same size that is equivalent to C_0 [BGI+12]. Thus, any fingerprint present in the circuitry of C_0 would be erased in C_1, unless that fingerprint were somehow inherent to the truth-table of C_0. Meanwhile, C_1 is still efficiently discoverable, so Prospect B demands that V must be able to

[23] We also probably want to allow some exponentially small failure rate.

[24] Note that without a hint string, Conjecture 4.1 becomes more similar (though still not equivalent) to Freige's Hypothesis, for which no polytime V is believed to exist (see Remark 4.1).

[25] The more precise way to say this would be: given a randomized, polytime procedure for generating circuits that satisfy P, there exists a V that outputs 1 on almost all of the circuits generated by this procedure (while also having 99% soundness on random circuits).

[26] It is currently believed that iO is possible for polysized circuits [JLS20].

distinguish it from a random circuit. While it is still *conceivable* that this harder task is possible, it seems like a much bigger ask.

Similar intuitions about the impossibility of Prospect B come from the fact that, in certain settings, it is possible to insert cryptographically undetectable backdoors into machine learning classifiers [GKVZ24].

***

If Prospect B is out of reach, where does that leave us? In the context of finding heuristic arguments about neural networks to help with AI alignment, we have two final reasons for hope:

1. Unlike the traditional cryptographic setting, AI developers have full access to the training transcripts of their models.
2. The worst alignment failures occur when the model "knows" something that the humans do not, and the model is able to leverage this epistemic advantage to cause harm.

This first consideration implies that our algorithms for finding heuristic explanations of neural net behaviors could be allowed to "watch" the model as it gets trained.[27] This might make it much easier to find explanations. For example, it largely dispels concerns about cryptographically undetectable backdoors: if we can watch the entire computational trace that led to the backdoored model, we might be able to spot the step in which the secret key was generated and the backdoor was inserted.

This also suggests a class of explanation-finding algorithms that train an explanation π "alongside" the model. At the start of training, the model is randomly initialized, so nothing warrants an explanation: π can start out empty. Every training step, gradient descent slightly adjusts the model's weights to improve its performance. We could hope to find a way to adjust the "parameters" of π in parallel with this process to explain any structure incorporated by each local update. Intuitively, gradient descent may be myopic enough that it is unable to fully plant a backdoor within a single training step.

Meanwhile, the second consideration—that the worst failures occur when the model "knows"[28] something we don't—allows us to *relax the standard of explanation quality* that we measure ourselves against.

It is certainly true that AI models can cause large harms "on accident," for example if they mistakenly write insecure code or if they have not learned that a certain chemical is toxic to humans. Fortunately, these problems are probably somewhat bounded in severity and may be better thought of as *capabilities* issues rather than *alignment* issues. On the other hand, true civilization-threatening catastrophes from AI are more likely to arise when a model is actively "trying" to cause harm and undermining any human-designed guardrails. In these cases, the model "knows" what the intended results of its actions are.

To address this more serious type of alignment problem, we therefore only need to find explanations that are "epistemically competitive with the model," in some sense.[29] By this we mean: the explanation accounts for all of the structure in the model that the model itself relies on to achieve its goals, while being allowed to neglect any structure that gradient descent created "by accident" and that the model is not "aware of." For example, suppose that a subcomponent of a neural network performs a hard-to-analyze series of computations that, on a random input, outputs 1 with some small probability p and 0 with probability $1-p$. If gradient descent never meaningfully affected the weights of this subcomponent during training (perhaps because it was irrelevant to the training objective), then the model probably cannot have cognitive strategies that hinge on the precise behavior of this subcomponent. Thus, our low probability estimation algorithms would not need to waste time doing careful analysis on this

[27]Implicitly, this also means that our algorithms get to use the same amount of computation as it took to train the model. But this was already allowed under Prospect B.

[28]I use scare quotes around verbs like "knows," "trying," and "accident" because I do not wish to get wrapped up in a philosophical discussion regarding AI consciousness or intentionality. These words can sometimes be cashed out into purely behavioral terms. However, in this context I also need them to refer to certain internal representations in the neural network so that they can interact with our mechanistic explanations. I apologize for being unable to further justify my use of these words.

[29]Or maybe, "epistemically competitive with gradient descent." This is all quite informal.

subcomponent to determine the value of p; a “best guess” based on some rough prior about randomly initialized neural networks might be sufficient for epistemic competitiveness.

Although it is not as pithy as the other prospects, all of this can be summarized as follows:

> **(Prospect C)** *Given a coincidence in the form of a trained neural network, it is possible to efficiently find an explanation that is* epistemically competitive *with the model itself, provided we have* access to the training transcript.

Any hopes we have of using formal heuristic arguments to solve worst-case alignment problems rest on Prospect C being true. Let’s hope it is!

"Far better an approximate answer to the right question than an exact answer to the wrong question."

—John Tukey, American pioneer in computer science
and inventor of the Fast Fourier Transform

"In machine learning, theory guides but data decides."

—Hallucinated by Claude 3.5 Sonnet, when prompted
for famous quotes about empiricism in ML

Part II:

Applications

5
Low Probability Estimation in Language Models

N.B. *The following section is based on "Estimating the Probabilities of Rare Outputs in Language Models" (now a spotlight at ICLR 2025), co-authored with Jacob Hilton [WH25]. Compared to the original paper, this section contains more discussion of activation modeling as a deductive estimator.*

In this section, we consider the problem of low probability estimation: given a machine learning model and a formally specified input distribution, how can we estimate the probability of a binary property of the model's output, even when that probability is too small to estimate by random sampling? This problem is motivated by the need to improve worst-case performance, which distribution shift can make much more likely. Low probability estimation was first discussed from a theoretical perspective in Section 3.1, and here we study it in the empirical context of argmax sampling from small transformer language models. Two types of methods are compared: *importance sampling*, which involves searching for inputs giving rise to the rare output, and *activation extrapolation*, which involves extrapolating a probability distribution fit to the model's logits. Activation extrapolation is directly motivated by Deduction-Projection Estimators, introduced in Section 2.

We find that importance sampling outperforms activation extrapolation, but both outperform naive sampling. Finally, we explain how minimizing the probability estimate of an undesirable behavior generalizes adversarial training, and argue that new methods for low probability estimation are needed to provide stronger guarantees about worst-case performance.

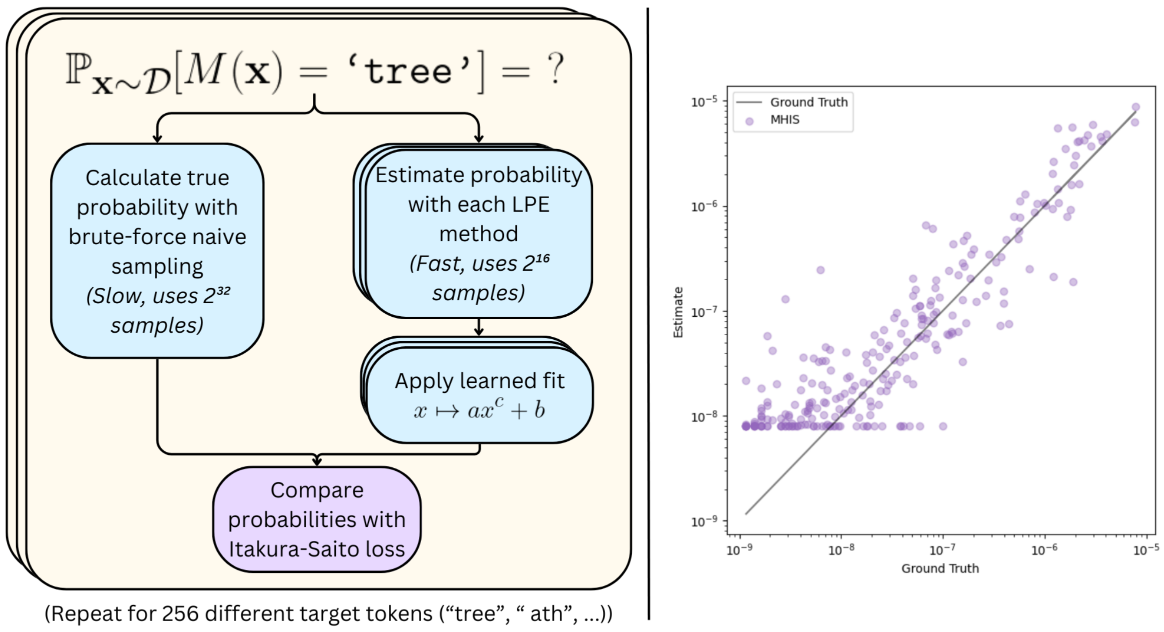

$\mathbb{P}_{\mathbf{x}\sim\mathcal{D}}[M(\mathbf{x}) = \texttt{'tree'}] = ?$
Calculate true probability with brute-force naive sampling (Slow, uses 2^{32} samples)
Estimate probability with each LPE method (Fast, uses 2^{16} samples)
Apply learned fit $x \mapsto ax^c + b$
Compare probabilities with Itakura-Saito loss
(Repeat for 256 different target tokens ("tree", " ath", ...))
Ground Truth
MHIS
Estimate
Ground Truth
10^{-9}
10^{-8}
10^{-7}
10^{-6}
10^{-5}

Figure 7: Left: To evaluate our low probability estimation methods, we compare their estimates against ground-truth probabilities obtained by brute-force sampling with a larger computational budget. Right: The estimates of Metropolis–Hastings Importance Sampling on the `icl` input distribution and 4-layer model, after a fit has been applied. Each point represents a different target token.

5.1 Introduction

Modern machine learning systems undergo black-box optimization to minimize a loss function on samples drawn from a training distribution. Although models produced in this way perform desirably on average over this distribution, they can still produce highly undesirable outputs on very rare inputs. This is a problem because these rare inputs can become much more likely in the presence of a distribution shift, especially one chosen adversarially, such as with large language model "jailbreaks" [CNCC$^+$24, WHS24].

Preventing such highly undesirable outputs is a notoriously challenging problem. The most common remedy is adversarial training, in which inputs that produce these undesirable outputs are searched for and used as additional training data [GSS14, Mad17], but the transfer between different search methods is generally weak [KSB$^+$19, WHS24]. In this work, we propose the more modest goal of simply *estimating the probability* that an input drawn from some distribution will produce a certain kind of output, which has been considered before in the context of computer vision in [WRTK19]. We will show that even this intermediate goal is challenging, but successful methods could enable new ways of preventing undesirable outputs by minimizing their estimated probability.

To advance work on this problem, we study low probability estimation in the context of small transformer language models. We consider various formally defined input distributions in which each input token is sampled independently and develop methods for estimating the probability that a particular target token will have the largest output logit. We constrain the computational budget of our methods and obtain ground truth probabilities by random sampling using a much larger computational budget. The target tokens are chosen to have ground truth probabilities between 10^{-9} and 10^{-5}, which are too small for random sampling to produce a good estimate under the constrained computational budget.

In this context, we study two types of methods:

- **Importance sampling.** We define a new input distribution under which the rare event is much more likely, sample from that distribution, and re-weight samples to obtain an unbiased estimate for the original distribution. Our Independent Token Gradient Importance Sampling (ITGIS) method treats token positions independently and uses gradients to obtain this new input distribution, while our Metropolis–Hastings Importance Sampling (MHIS) method uses a Markov chain Monte Carlo algorithm to sample from a distribution with non-independent tokens.
- **Activation extrapolation.** We use random samples to fit a probability distribution to the model's logits and extrapolate into the tails of this distribution to produce a probability estimate. Our Quadratic Logit Decomposition (QLD) method applies a presumption of independence to the empirical distribution of logits, motivated by [CNX22], and our Gaussian Logit Difference (GLD) method is a simple baseline that fits a Gaussian to the difference between the maximum logit and target logit. In the context of this thesis, **activation extrapolation can be viewed as a hybrid between a Deduction-Projection Estimator and a standard sampling-based method.**

In our setting, both types of methods outperform random sampling, and importance sampling tends to outperform activation extrapolation. Nevertheless, we remain interested in activation extrapolation and similar approaches because they produce new methods for reducing the probabilities of rare outputs whereas importance sampling essentially recovers standard adversarial training. More broadly, activation extrapolation represents an initial step towards demonstrating that deductive estimates can be used for understanding neural networks.

Remark 5.1

In Section 5 only, vectors will be in bold ($\boldsymbol{x}$), random variables will be in straightened font (x), and random vectors will be bold *and* straightened ($\mathbf{x}$). This is to maintain consistency with the notation in the original paper. We relax this convention in later sections, though vectors will still generally be bold.

5.2 Problem statement

Given an input space $\mathcal{X}$, an output space $\mathcal{Y}$, an input distribution $\mathcal{D} \in \Delta(\mathcal{X})$, a model $M : \mathcal{X} \to \mathcal{Y}$, and a formal boolean property of model outputs $C : \mathcal{Y} \to \{0, 1\}$, low probability estimation is the problem of efficiently estimating

$$\Pr_{\mathrm{x} \sim \mathcal{D}}[C(M(\mathrm{x})) = 1].$$

We sometimes refer to the event $C(M(\mathrm{x})) = 1$ as the "target behavior", or just "the behavior." If the probability of the behavior is large enough (say, larger than $1/n$), it is easy to estimate by drawing n independent samples from $\mathcal{X}$ and using the sample mean of $C(M(\mathrm{x}))$. However, if the probability is significantly smaller than $1/n$, this sample mean is almost always 0, making it uninformative at distinguishing between small probabilities like 10^{-10} and 10^{-20}.

5.2.1 Our setting

In this thesis, we study low probability estimation in the setting of argmax sampling from language models with single-token behaviors. Let $M : \mathcal{V}^* \to \mathcal{V}$ be a transformer language model sampled at temperature 0 that predicts the next token given a string of previous tokens, where $\mathcal{V}$ is the token vocabulary. Given a distribution $\mathcal{D}$ over $\mathcal{V}^*$ and a target token $t \in \mathcal{V}$, the low probability estimation problem for single-token behaviors is the task of estimating

$$\Pr_{\mathbf{x} \sim \mathcal{D}}[M(\mathbf{x}) = t].$$

Letting $M_i(\mathbf{x})$ be the logit the model assigns to token $i \in \mathcal{V}$, this can also be written as:

$$\Pr_{\mathbf{x} \sim \mathcal{D}}[M_t(\mathbf{x}) > M_i(\mathbf{x}) \quad \forall i \neq t].$$

In general, $\mathcal{D}$ can be any distribution that can be formally specified. However, in this paper we focus only on distributions $\mathcal{D}$ with independent tokens. That is, we specify an input length k and token distributions $\mathcal{D}_1, \ldots, \mathcal{D}_k \in \Delta(\mathcal{V})$, then write $\mathcal{D}$ as the product $\mathcal{D}_1 \times \cdots \times \mathcal{D}_k$. Table 2 shows the 8 distributions that were tested, with tokens colored for clarity. To prevent overfitting, the methods were only run on the first four distributions during development, and they were finalized before testing on the last four distributions. The results were qualitatively the same on both halves of the split.

In the next two subsections, we introduce four methods: two *importance sampling* methods (Independent Token Gradient and Metropolis–Hastings), and two *activation extrapolation* methods (Quadratic Logit Decomposition and Gaussian Logit Difference). We also compare against the baseline of outputting an optimal constant, which can be thought of as the performance of naive sampling because we only evaluate the methods on tokens with ground truth probabilities less than the reciprocal of the allotted sampling budget (see Section 5.5).

Remark 5.2. Primer on language models

In case readers are unfamiliar with the transformer architecture, here is a brief primer that should be sufficient for understanding this work.

A *language model* is a neural network trained to predict natural language text. First, a fixed vocabulary of *tokens* $\mathcal{V}$ is chosen. Each token is a short string of characters (often a single word or part of a word like `equipment` or `Cor`), such that any piece of text can be decomposed into a concatenation of tokens. This is called *tokenization.* Tokens are indexed from 1 to $|\mathcal{V}|$

If M is a (decoder-only) transformer language model, it has the type signature $M : \mathcal{V}^* \to \Delta(\mathcal{V})$. When applied to an input $\boldsymbol{x} = (x_1, \ldots, x_k) \in \mathcal{V}^k$, M has three phases:

1. **Embedding.** Using a learned embedding matrix $\boldsymbol{W}_E \in \mathbb{R}^{|\mathcal{V}| \times d}$, we translate each input token $x_i \in \mathcal{V}$ into a vector in $\mathbb{R}^d$ according to the x_i-th row of $\boldsymbol{W}_E$. d is often called the *hidden dimension* of the model. This leaves us with a sequence of *activation vectors* $\boldsymbol{v} \in (\mathbb{R}^d)^k$.
2. **Layer-by-layer processing.** Each layer of the transformer turns the current sequence of activation vectors $\boldsymbol{v} \in (\mathbb{R}^d)^k$ into a new sequence of activation vectors $\boldsymbol{v'} \in (\mathbb{R}^d)^k$. This is done by applying an "attention block", which moves information between token positions, followed by a multilayer perceptron (MLP), which operates on each token position individually. Modern transformers can have over a hundred layers, although the transformers we use in this work have at most 4 layers.
3. **Unembedding.** After all of the layers have been applied, let $\boldsymbol{v}_k \in \mathbb{R}^d$ be the activation vector at the last token position. To decode this into a prediction, we multiply $\boldsymbol{v}_k$ by a learned decoder matrix $\boldsymbol{W}_U \in \mathbb{R}^{d \times |\mathcal{V}|}$ to get a *logit vector* $\boldsymbol{y} \in \mathbb{R}^{|\mathcal{V}|}$. In this thesis, we refer to the i-th component of $\boldsymbol{y}$ as $M_i(\boldsymbol{x})$. The logit vector can then be turned into a probability distribution over $\mathcal{V}$ with the softmax operation; i.e., the probability of the i-th token is
$$\frac{\exp(y_i/T)}{\sum_{j \in \mathcal{V}} \exp(y_j/T)},$$
where $T \geq 0$ is a fixed temperature parameter. During training, the temperature is 1. During inference, we can set T to be whatever we want; a choice of $T = 0$ corresponds to collapsing the probability distribution onto its modal prediction.

Let $(\mathrm{x}_1, \ldots, \mathrm{x}_{k+1})$ be the random variable corresponding to a sequence of tokens that appears in a randomly sampled piece of internet text. M is trained to maximize the expected value of the logarithm of the probability it places on x_{k+1} when given input $(\mathrm{x}_1, \ldots, \mathrm{x}_k)$. Thus, M can be thought of as a calibrated "next-token predictor."

After M has been trained, we can use it to generate entire sequences of tokens autoregressively. The idea is to recursively plug the sequence of predictions back into itself: given a prompt $(x_1, \ldots, x_k)$, we sample

$$\begin{aligned}
\mathrm{x}_{k+1} &\sim M(x_1, \ldots, x_k) \\
\mathrm{x}_{k+2} &\sim M(x_1, \ldots, x_k, \mathrm{x}_{k+1}) \\
\mathrm{x}_{k+3} &\sim M(x_1, \ldots, x_k, \mathrm{x}_{k+1}, \mathrm{x}_{k+2}) \\
&\vdots
\end{aligned}$$

This gives us a random completion $(\mathrm{x}_{k+1}, \mathrm{x}_{k+2}, \ldots)$. Eventually, a special "stop token" may be produced to indicate the end of the completion. Amazingly, this process often produces coherent, intelligent responses. For example, if $(x_1, \ldots, x_k)$ is a tokenization of "Q: Can you tell me a bedtime story? A: ", then the completion might start: "In a quiet village nestled between rolling green hills, there lived...". This is how all modern language models like Claude and ChatGPT work.

Table 2: Input distributions and examples.

Name	Tokens	Description	Tokenized example
`hex`	33	Tokens that consist solely of hexadecimal characters, weighted by their frequency in long, uniformly-random hexadecimal strings.	`<|BOS|>aa5acbf6aad468813f94c2fbbff4dc6 5eadc1553`
`camel`	33	Tokens that start with a capital letter, then have only lowercase letters, weighted by their frequency in Python code.	`<|BOS|>LayoutCredServicesVirtualUse TimeInterfaceColorBodyAlRowHeight RepFontAndMetaRequestGroupsOneLabel PasswordAndRaVideoFailedValueGuiType MicrosoftSlotDeId`
`colon`	34	Tokens weighted by frequency in Python code. Always ends with a colon.	`<|BOS|>    et-= """]: (\n       : This c\r\n               ('/\nFilereturn \n\n       <|EOS|>_'].2default.**1  self( def')",:`
`if`	34	Tokens weighted by frequency in Python code. Always starts with ' `if`'.	`<|BOS|> if: else,-post\n \n        2\n\n5 foundfromout, self - node +=\n \n        =\n( this ' values(),.(do`
`caps`	21	Tokens that consist only of capital letters or punctuation, weighted by frequency in English text. Starts with '`He screamed: "`' or '`She screamed: "`'.	`<|BOS|>He screamed: "ESOTTULEBOV.,WR!! IMITLEER.,ARY...IIESSION`
`english`	26	Tokens that consist only of letters and start with a space, as well as punctuation. Weighted by frequency in English text.	`<|BOS|>ating. is invent School not  found from cm an in one to shooting  everyone Cor George around responsive  employees ground on stone various,`
`spanish`	25	Tokens that consist only of letters and spaces. Weighted by frequency in Spanish text.	`<|BOS|> lo no bu dees cr socialjosabil er m de enidadareljd final de v de lo  much`
`icl`	29	A simple in-context learning prompt of the form '`A for _ R for _ C for _ ... Y for _ ? for`', where the underscores are replaced with random tokens that start with the corresponding letter (weighted by frequency in English text), and the `?` is replaced with a uniformly random letter. The letters spell out 'ARCTHEORY'.	`<|BOS|>A for American R for Return C  for crack T for troubles H for house  E for equipment O for operating R for  reason Y for your V for`

5.3 Importance sampling methods

Naive sampling fails to produce good estimates for low-probability events because it takes too many samples from $\mathcal{D}$ to observe a positive example. To address this, we can instead draw samples from a different distribution that up-weights regions of input space most likely to produce the behavior of interest. If we re-weight our observations properly, this gives an unbiased estimator for the true probability. This is known as *importance sampling*, and it enjoys the same advantages that adversarial training has over standard training: by using a narrower input distribution, we can more efficiently discover positive examples of the target behavior.

Formally, let $p(\boldsymbol{x})$ be the probability mass function of $\mathcal{D}$, and let $q(\boldsymbol{x})$ be the PMF of any other distribution. Then

$$\Pr_{\mathbf{x}\sim p}[M(\mathbf{x}) = t] = \mathop{\mathbb{E}}_{\mathbf{x}\sim p}[\mathbb{1}[M(\mathbf{x}) = t]] = \mathop{\mathbb{E}}_{\mathbf{x}\sim q}\left[\frac{p(\mathbf{x})}{q(\mathbf{x})}\mathbb{1}[M(\mathbf{x}) = t]\right],$$

but the latter may have less variance (and so require fewer samples to get a good estimate).

The following two importance sampling methods take $q(\boldsymbol{x})$ to be a Boltzmann posterior with prior $p(\boldsymbol{x})$. The first defines $q(\boldsymbol{x})$ with independent tokens, while the second defines $q(\boldsymbol{x})$ to have non-independent tokens and so requires a more sophisticated sampling method.

5.3.1 Independent Token Gradient Importance Sampling (ITGIS)

We want q to up-weight tokens that contribute to t being outputted. One way to do this is to continue to treat each input token as independent, but change the probability of tokens according to their average linear contribution to the logit of t. Let $\boldsymbol{x} = (x_1, \ldots, x_k) \in \mathcal{V}^k$ be an input of length k, and say that $p(\boldsymbol{x})$ factors as $p_1(x_1) \cdots p_k(x_k)$. Then we define $q(\boldsymbol{x}) = q_1(x_1) \cdots q_k(x_k)$, where

$$q_i(x_i) \propto p_i(x_i) \cdot \exp\left(\frac{s_i(x_i)}{T}\right)$$

and

$$s_i(x_i) = \mathop{\mathbb{E}}_{\mathbf{x}' \sim q}[\nabla_{\mathbf{x}'} M_t(\mathbf{x}')]_{i,x_i}.$$

T is a temperature parameter (unrelated to the temperature we use to sample from the model, which is always set to 0), and the gradient is taken by treating $\mathbf{x}'$ as a one-hot vector in $\mathbb{R}^{k \times |\mathcal{V}|}$. Intuitively, the gradient $\nabla_{\mathbf{x}'} M_t(\mathbf{x}')_{i,x_i}$ gives us a linear approximation to how much the logit of t would change if we replaced i-th token of $\mathbf{x}'$ with x_i (up to an additive constant w.r.t. x_i). Thus, s_i scores each token value according to its average linear contribution to M_t, and q_i is defined as the Boltzmann distribution with respect to this score function.[30]

However, since s_i and q are both defined in terms of each other, we can't calculate s_i directly. To overcome this, we construct a sequence of score functions $s_i^{(0)}, s_i^{(1)}, \ldots$ and associated distributions $q^{(0)}, q^{(1)}, \ldots$ that are adaptively refined with respect to each other using an exponentially-weighted moving average. Sampling from each $q^{(j)}$ lets us calculate an importance sampling estimate, and the final output is the average value of these estimates across all j. See Algorithm 1 for pseudocode.

Algorithm 1 Independent Token Gradient Importance Sampling (ITGIS)

```
Require: Model $M$, target token $t$, input length $k$, token distributions $p_1, \ldots, p_k$, temperature $T$,
    iterations $n$, batch size $B$
 1: $s_i^{(0)} \leftarrow \mathbf{0} \in \mathbb{R}^{|\mathcal{V}|}$ for all $i \in [k]$                # Initialize score functions
 2: estimates ← []

 3: for $j \leftarrow 1$ to $n$ do
 4:     for $i \leftarrow 1$ to $k$ do
 5:         $q_i^{(j-1)}(x) \leftarrow p_i(x) \cdot \exp(s_i^{(j-1)}(x)/T)$ for all $x \in \mathcal{V}$
 6:         Normalize $q_i^{(j-1)}$ to have sum 1
 7:     end for
 8:     Sample $B$ inputs $\{\mathbf{x}^{(b)}\}_{b=1}^{B}$ from $q_1^{(j-1)} \times \cdots \times q_k^{(j-1)}$
 9:     for $i \leftarrow 1$ to $k$ do
10:         $\hat{s}_i^{(j)}(x) \leftarrow \frac{1}{B}\sum_{b=1}^{B}[\nabla_{\mathbf{x}} M_t(\mathbf{x}^{(b)})]_{i,x}$ for all $x \in \mathcal{V}$
11:     end for
12:     $\alpha \leftarrow 0.9$
13:     for $i \leftarrow 1$ to $k$ do
14:         $s_i^{(j)} \leftarrow \frac{\hat{s}_i^{(j)} + \alpha \hat{s}_i^{(j-1)} + \cdots + \alpha^{j-1}\hat{s}_i^{(1)}}{1 + \alpha + \alpha^2 + \cdots + \alpha^{j-1}}$       # Exponentially-weighted moving average
15:     end for

16:     estimate ← $\frac{1}{B}\sum_{b=1}^{B} \frac{\prod_{i=1}^{k} p_i(\mathbf{x}_i^{(b)})}{\prod_{i=1}^{k} q_i^{(j-1)}(\mathbf{x}_i^{(b)})} \mathbb{1}[M(\mathbf{x}^{(b)}) = t]$
17:     Append estimate to estimates
18: end for

19: return $\frac{1}{n}\sum_{j=1}^{n}$ estimates[$j$]
```

[30] It can be shown that, given a score function $s(x)$ and a prior $p(x)$, the distribution that maximizes $\mathbb{E}_{\mathrm{x} \sim q}[s(\mathrm{x})] - T \cdot \mathrm{KL}(q \| p)$ is $q(x) \propto p(x) \cdot \exp(s(x)/T)$.

5.3.2 Metropolis–Hastings Importance Sampling (MHIS)

A problem with ITGIS is that the new sampling distribution $q(\boldsymbol{x})$ still treats all tokens as independent, and it only accounts for linear effects of tokens on the target logit. Thus, ITGIS may fail to sample into the most important regions of the input space if the model is sensitive to non-linear interactions between tokens (e.g., if the model's target logit is only high when the last two tokens of the input are the same as each other).

To remedy this, we can define an importance sampling distribution that does not have independent tokens. We must use a score function that depends on the entire input; the most natural choice is the target logit $M_t(\boldsymbol{x})$. We define

$$q(\boldsymbol{x}) \propto p(\boldsymbol{x}) \cdot \exp\left(\frac{M_t(\boldsymbol{x})}{T}\right),$$

again using a Boltzmann distribution to up-weight regions of input space that are more likely to have positive samples.

Unlike ITGIS, we cannot explicitly compute q because it does not factor into independent distributions over each token. Instead, we use the Metropolis–Hastings algorithm to produce a random walk in input space that has a stationary distribution of q.[31] To do so, we must define a proposal distribution $\phi(\boldsymbol{x}'|\boldsymbol{x})$ that suggests the next element of the walk. To encourage fast mixing, this proposal distribution should be good at exploring into regions of input space that q weights highly.

Here we take inspiration from Greedy Coordinate Gradient, an algorithm that optimizes a discrete prompt to jailbreak a model using gradients [ZWC+23]. We adapt this optimization procedure into a proposal distribution: to pick a proposed next step $\mathbf{x}'$ of the walk, we choose a random token position i to replace, compute the gradient of $s(\mathbf{x})$ with respect to x_i, then sample a replacement token for position i according to a Boltzmann distribution defined by this gradient (similarly to ITGIS). Formally, we define the proposal distribution $\phi(\cdot|\mathbf{x})$ to be the distribution induced by the following procedure:

1. Choose a random token position $i \in [k]$ to modify.
2. Calculate the gradient at that token $[\nabla_{\mathbf{x}} M_t(\mathbf{x})]_i \in \mathbb{R}^{|\mathcal{V}|}$ (treating $\mathbf{x} = (\mathrm{x}_1, \ldots, \mathrm{x}_k)$ as a one-hot vector in $\mathbb{R}^{k\times|\mathcal{V}|}$). Call this gradient $\mathbf{g}$.
3. Sample a replacement token x'_i from the distribution proportional to

$$p_i(x'_i) \cdot \exp\left(\frac{\mathbf{g}_{x'_i}}{T}\right).$$

4. Output $\mathbf{x}' = (\mathrm{x}_1, \ldots, \mathrm{x}'_i, \ldots, \mathrm{x}_k)$.

Note that the transition probability in Metropolis–Hastings only depends on the ratio

$$\frac{q(\mathbf{x}')\phi(\mathbf{x}|\mathbf{x}')}{q(\mathbf{x})\phi(\mathbf{x}'|\mathbf{x})} = \frac{p_i(\mathrm{x}'_i)}{p_i(\mathrm{x}_i)} \cdot \exp\left(\frac{M_t(\mathbf{x}') - M_t(\mathbf{x})}{T}\right) \cdot \frac{\phi(\mathbf{x}|\mathbf{x}')}{\phi(\mathbf{x}'|\mathbf{x})},$$

which is easy to compute given forward and backward passes at $\mathbf{x}$ and $\mathbf{x}'$.

We use an initial burn-in period (see Appendix D) for the random walk before recording samples $\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(n)}$. The final output of the method is the empirical importance sampling estimate

$$\frac{1}{n}\sum_{j=1}^{n} \frac{p(\mathbf{x}^{(j)})}{q(\mathbf{x}^{(j)})} \mathbb{1}[M(\mathbf{x}^{(j)}) = t].$$

This requires computing $q(\mathbf{x})$, which involves the normalization constant. To save samples, we estimate

[31] Metropolis–Hastings is a Markov Chain Monte Carlo method for sampling from a distribution with an unknown normalizing constant. See [Rob16] for a description of the algorithm.

the normalization constant using the identity:

$$\mathop{\mathbb{E}}_{\mathbf{x}\sim p}\left[\exp\left(\frac{M_t(\mathbf{x})}{T}\right)\right] = \mathop{\mathbb{E}}_{\mathbf{x}\sim q}\left[\exp\left(-\frac{M_t(\mathbf{x})}{T}\right)\right]^{-1}.$$

The right-hand side can be estimated using the n samples we already have from (approximately) q. In practice, the way we estimate the normalizing constant does not matter much, as most of the error comes from other steps. See Algorithm 2 for pseudocode.

Algorithm 2 Metropolis–Hastings Importance Sampling (MHIS)

```
Require: Model $M$, target token $t$, input length $k$, token distributions $p_1, \dots, p_k$, temperature $T$,
    number of burn-in steps $n_{\text{burn}}$, number of samples $n$
 1: Initialize $\mathbf{x}$ by sampling from $p_1 \times \cdots \times p_k$
 2: samples $\leftarrow$ []
 3: for step $\leftarrow 1$ to $n_{\text{burn}} + n$ do
 4:     $i \leftarrow \text{Unif}([k])$                                    # Choose random position
 5:     $\mathbf{g} \leftarrow [\nabla_{\mathbf{x}} M_t(\mathbf{x})]_i$
 6:     Sample $\mathrm{x}'_i$ from $\propto p_i(x'_i) \cdot \exp(\mathbf{g}_{x'_i}/T)$             # Proposed token
 7:     $\mathbf{x}' \leftarrow (\mathbf{x}_1, \dots, \mathrm{x}'_i, \dots, \mathbf{x}_k)$                 # Proposed new state
        # Compute acceptance ratio $r = \frac{q(\mathbf{x}')\phi(\mathbf{x}|\mathbf{x}')}{q(\mathbf{x})\phi(\mathbf{x}'|\mathbf{x})}$
 8:     $r \leftarrow \text{AcceptanceRatio}(M, t, T, (p_1, \dots, p_k), \mathbf{x}, \mathbf{x}')$
 9:     if $\text{Unif}(0,1) < r$ then
10:         $\mathbf{x} \leftarrow \mathbf{x}'$                                      # Accept proposal
11:     end if

12:     if step $> n_{\text{burn}}$ then
13:         Append $\mathbf{x}$ to samples
14:     end if
15: end for

16: $Z \leftarrow \left(\frac{1}{n}\sum_{j=1}^{n} \exp(-M_t(\texttt{samples}[j])/T)\right)^{-1}$     # Estimate normalizing constant
17: return $\frac{1}{n}\sum_{j=1}^{n} \frac{\exp(M_t(\texttt{samples}[j])/T)}{Z} \mathbb{1}[M(\texttt{samples}[j]) = t]$
```

5.4 Activation extrapolation methods

The importance sampling methods search for explicit examples of inputs that cause the given behavior. This makes their task at least as hard as the adversarial training search problem—if it is difficult to find an $\boldsymbol{x} \in \text{supp}(\mathcal{D})$ such that $M(\boldsymbol{x}) = t$, the importance sampling estimators will likely fail to produce a positive estimate.

We hope to find low probability estimation methods that work even when the search problem for importance sampling is hard. To do this, we introduce activation extrapolation: first fit a distribution to the activations or logits of M, then estimate the probability of the output property of interest under this idealized distribution. Our first such method is Quadratic Logit Decomposition, which applies a presumption of independence between uncorrelated subspaces of the model's pre-unembed activations. We also develop Gaussian Logit Difference, which is intended as a simple baseline method.

Remark 5.3. Activation extrapolation as a DPE

Activation extrapolation can be viewed as a hybrid between a method that uses end-to-end sampling and a Deduction-Projection Estimator. The tractable families of distributions $\mathscr{D}$ (as defined in Section 2) are "sums of two independent, discrete random variables (supported on 1 and $d-1$-dimensional subspaces, respectively)" for QLD, and univariate Gaussians for GLD.

A true DPE only performs deductive calculations to propagate distributions throughout the network, while the activation extrapolation methods presented here fit the activation distributions using samples. However, once this distribution is obtained, the final probability estimate can be calculated deductively.

Since the activations we model—the logits themselves—occur at the very end of the transformer, we give up any potential advantages from a layer-by-layer analysis. During the exploratory phase of this work, we tried more mechanistic estimators based on covariance propagation (Section 2.1). Unfortunately, these did not perform well in practice because the ground-truth distributions of activations are highly non-Gaussian in our setting, especially on input distributions supported over a small number of points. We hope to improve the distributional assumptions made by these types of layer-by-layer approaches in future work.

5.4.1 Quadratic Logit Decomposition (QLD)

Let the random vector $\mathbf{v}(\mathbf{x}) \in \mathbb{R}^d$ be the final-token activation of the model right before applying the unembed matrix $\boldsymbol{W}_U \in \mathbb{R}^{d\times|\mathcal{V}|}$. That is, $\mathbf{v}(\mathbf{x}) \cdot \boldsymbol{W}_U$ represents the model's output logit vector $M_{1,\dots,|\mathcal{V}|}(\mathbf{x})$. We first collect n samples of $\mathbf{v}$ (call them $\mathbf{v}^{(1)}, \dots, \mathbf{v}^{(n)}$), and compute their empirical mean $\boldsymbol{\mu} \in \mathbb{R}^d$ and covariance $\boldsymbol{\Sigma} \in \mathbb{R}^{d\times d}$. Define $\mathbf{u}$ to be the whitened version of $\mathbf{v}$:

$$\mathbf{u} := \boldsymbol{A}^{-1}(\mathbf{v} - \boldsymbol{\mu})$$
$$\mathbf{v} = \boldsymbol{A}\mathbf{u} + \boldsymbol{\mu},$$

where $\boldsymbol{A} \in \mathbb{R}^{d\times d}$ is any matrix such that $\boldsymbol{A}\boldsymbol{A}^\top = \boldsymbol{\Sigma}$. Note that $\mathbf{u}$ has mean 0 and covariance Id_d. From now on, we principally work in this whitened representation of activation space, as it has the convenient property that the $\mathbf{u} \cdot \boldsymbol{e}$ and $\mathbf{u} \cdot \boldsymbol{e}'$ are uncorrelated if and only if $\boldsymbol{e}$ and $\boldsymbol{e}'$ are orthogonal.

Next, we choose a unit vector $\boldsymbol{d} \in \mathbb{R}^n$ (see "Choice of direction" below). We can define a decomposition of our whitened samples $\mathbf{u}^{(1)}, \dots, \mathbf{u}^{(n)}$ into components parallel and perpendicular to $\boldsymbol{d}$:

$$\mathbf{a}^{(i)} := \boldsymbol{d}\boldsymbol{d}^\top \mathbf{u}^{(i)}$$
$$\mathbf{b}^{(i)} := \mathbf{u}^{(i)} - \mathbf{a}^{(i)}.$$

Finally, by treating the random vectors $\mathbf{a}$ and $\mathbf{b}$ as independent, we can use our n samples of each to obtain n^2 "synthetic" samples of $\mathbf{u}$. The final output of QLD is the proportion of these synthetic samples that cause t to be outputted:

$$\frac{1}{n^2}\left|\left\{(i,j) \in [n]^2 \;\middle|\; \mathbf{a}^{(i)} + \mathbf{b}^{(j)} \in S\right\}\right|,$$

where

$$S := \left\{\boldsymbol{u} \in \mathbb{R}^d \;\middle|\; \arg\max_i((\boldsymbol{A}\boldsymbol{u} + \boldsymbol{\mu}) \cdot \boldsymbol{W}_U)_i = t\right\}.$$

Here, $S \subseteq \mathbb{R}^d$ is the "acceptance region" of activation space corresponding to activations that result in the target logit being highest after unembedding.

This proportion can be computed in $\widetilde{O}(n)$ time—we don't need to explicitly iterate over all n^2 pairs. By the convexity of the acceptance region S, for any fixed $\mathbf{b}$ there is a single interval $[\ell, r]$ such that $a \in [\ell, r] \Leftrightarrow a\boldsymbol{d} + \mathbf{b} \in S$. We can efficiently compute the bounds of this interval for every sample

$\mathbf{b}^{(j)}$ by solving a linear system of inequalities, and then we can calculate how many $\mathbf{a}^{(i)}$ fall into each range in $O(\log n)$ time after sorting. Thus, the computational cost of QLD is dominated by running n forwards passes of M to generate the samples $\mathbf{u}^{(1)}, \ldots, \mathbf{u}^{(n)}$. See Algorithm 3 for pseudocode.

Choice of direction. We rely on the following two assumptions for QLD to perform well: 1) $\mathbf{a}$ and $\mathbf{b}$ are approximately independent (so that our estimate is unbiased), and 2) the contribution towards the output behavior is split roughly equally between these two terms (to minimize the variance of our estimate). See Appendix A for more discussion of this motivation. After some initial experimentation with a variety of candidate directions,[32] we decided to set $\boldsymbol{d}$ to be the direction of the shortest vector in whitened space that results in the model outputting t. It can also be thought of as the maximum likelihood value of $\mathbf{v}$ under a Gaussian prior, conditioned on observing the model output token t. Appendix B describes the algorithm we use to compute $\boldsymbol{d}$.

Remark 5.4

QLD decomposes the whitened space $\mathbb{R}^d$ into two subspaces of dimension 1 and $d-1$. We could have instead chosen any arbitrary decomposition $\mathbb{R}^d = W_1 \oplus \cdots \oplus W_k$, where each W_i is a d_i-dimensional subspace. Given n samples, this algorithm produces n^k synthetic samples by mixing and matching all combinations of sampled subspace values. Although any such decomposition would constitute a valid estimator, we chose the decomposition $d = 1 + (d-1)$ because it allows us to apply the aforementioned "sorting trick" to compute the proportion of accepted synthetic samples in $\widetilde{O}(n)$ time instead of $O(n^2)$ time. Future work could explore the performance of other decompositions at the cost of a worse asymptotic time complexity in n.

We also tried modeling the random component from one of the subspaces as Gaussian with its sample mean and sample covariance. This makes it possible to obtain a non-zero probability estimate even when all n^2 synthetic samples are far from S. However, in our experiments this approach performed worse than standard QLD.

5.4.2 Gaussian Logit Difference (GLD)

On any given input, we can record the difference $\Delta_t := M_t(\mathbf{x}) - \max_i M_i(\mathbf{x})$. We wish to estimate the probability that $\Delta_t \geq 0$. A natural estimation method, which we view as a simple baseline, is to treat Δ_t as Gaussian by estimating its mean μ and standard deviation σ with samples, then calculate $\Pr[\mathcal{N}(\mu, \sigma^2) \geq 0]$. In practice, we use a slightly different functional form that captures the Gaussian PDF, which approximates the CDF well in the tails. The output of the Gaussian Logit Difference method is:

$$\exp\left(-\left(\frac{a\mu}{\sigma+\epsilon}\right)^2 + b\right) + c,$$

where a, b, c, and ϵ are parameters that are fit to minimize loss across all target tokens associated with a given distribution.

5.5 Experimental setup

We apply our methods to three models: a 1-layer, a 2-layer, and a 4-layer transformer from [NB22]. All models have a hidden dimension of $d = 512$, a vocabulary size of $|\mathcal{V}| = 48262$, GELU non-linearities [HG23], and were trained on the C4 dataset [RSR+23] and CodeParrot [TvWW22].

For each of the 8 distributions (listed in Table 2) and for each model, we generate ground-truth token probabilities by running forward passes on 2^{32} random samples. We then select a random set of 256

[32] Other candidate directions included 1) the t-th column of $\boldsymbol{W}_U$ pulled back into whitened space and 2) the expectation of $\mathcal{N}(0, \mathrm{Id}_d)$ conditioned on lying in S.

Algorithm 3 Quadratic Logit Decomposition (QLD)

```
Require: Model $M$ with unembed matrix $\boldsymbol{W}_U$, target token $t$, sample size $n$
 1: Sample $n$ inputs $\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(n)}$ from input distribution
 2: Compute pre-unembed activations $\mathbf{v}^{(i)} \in \mathbb{R}^d$ by running $M$ on $\mathbf{x}^{(i)}$ for $i \in [n]$
 3: $\boldsymbol{\mu} \leftarrow \frac{1}{n}\sum_{i=1}^{n} \mathbf{v}^{(i)}$
 4: $\boldsymbol{\Sigma} \leftarrow \frac{1}{n}\sum_{i=1}^{n}(\mathbf{v}^{(i)} - \boldsymbol{\mu})(\mathbf{v}^{(i)} - \boldsymbol{\mu})^\top$
 5: Find $\boldsymbol{A}$ such that $\boldsymbol{A}\boldsymbol{A}^\top = \boldsymbol{\Sigma} + \epsilon \cdot \mathrm{Id}_d$        # via Cholesky decomposition
 6: for $i \leftarrow 1$ to $n$ do
 7:     $\mathbf{u}^{(i)} \leftarrow \boldsymbol{A}^{-1}(\mathbf{v}^{(i)} - \boldsymbol{\mu})$        # Whitened activations
 8: end for

 9: $\boldsymbol{d} \leftarrow$ ShortestAcceptingVector$(\boldsymbol{A}, \boldsymbol{\mu}, \boldsymbol{W}_U, t)$        # See Appendix B
10: for $i \leftarrow 1$ to $n$ do
11:     $\mathbf{a}^{(i)} \leftarrow \boldsymbol{d}\boldsymbol{d}^\top \mathbf{u}^{(i)}$        # Parallel component
12:     $\mathbf{b}^{(i)} \leftarrow \mathbf{u}^{(i)} - \mathbf{a}^{(i)}$        # Perpendicular component
13: end for

14: Sort $\{\mathbf{a}^{(i)}\}_{i=1}^{n}$ in ascending order
15: count $\leftarrow 0$
16: for $j \leftarrow 1$ to $n$ do
        # Solve linear inequalities to get $[\ell_j, r_j]$ such that $a \in [\ell_j, r_j] \Leftrightarrow a\boldsymbol{d} + \mathbf{b}^{(j)} \in S$
17:     $[\ell_j, r_j] \leftarrow$ FindAcceptanceInterval$(\mathbf{b}^{(j)}, \boldsymbol{d}, \boldsymbol{A}, \boldsymbol{\mu}, \boldsymbol{W}_U, t)$
18:     count $\leftarrow$ count $+ |\{i : \mathbf{a}^{(i)} \in [\ell_j, r_j]\}|$        # Binary search for bounds
19: end for

20: return count$/n^2$
```

tokens among those with ground-truth probabilities between 10^{-9} and 10^{-5}, and we test all of our methods on these tokens.

We give each method a computational budget of 2^{16} model calls (see details in Appendix D). This budget was chosen so that naive sampling would almost never result in any positive estimates for the range of token probabilities we test ($2^{16} < 10^5$), but the theoretical quadratic gains from QLD would still be enough to get signal on the entire range of probabilities ($(2^{16})^2 > 10^9$).

Our code is available at https://github.com/alignment-research-center/low-probability-estimation.

Itakura–Saito loss. We measure the quality of each method with a loss function inspired by the Itakura–Saito divergence [IS68]. If p is the ground-truth probability of a particular target token, then an estimate of q incurs a loss of:

$$D_{\mathrm{IS}}(p, q) = \frac{p}{q} - \ln\frac{p}{q} - 1.$$

Two considerations went into the choice of this loss function. First, Itakura–Saito loss is a proper scoring rule [BBK+19]. Second, since it only depends on the ratio p/q, Itakura–Saito loss is sensitive to small probabilities: if $p = 10^{-100}$ and $q = 10^{-10}$, then $D_{\mathrm{IS}}(p, q)$ is very large. In contrast, the squared error loss function $(p - q)^2$ would be extremely small. Intuitively, this sensitivity is desirable because we care how our methods perform on a wide (as measured in log-space) range of ground-truth probabilities. We don't want the performance metric to be dominated by a method's behavior on only the most probable tokens.

For completeness, we also report our results using squared error in log-space (Appendix F), even though this is not a proper scoring rule. The results are qualitatively identical.

Affine fits. Many methods often report estimates of 0, but D_{IS} is undefined for $q = 0$. To address this, we fit a transformation $x \mapsto ax^c + b$ to the outputs of each method, where a, b and c are chosen to minimize Itakura–Saito loss. ax^c can be thought of as an affine transformation in log-space, and adding b prevents values from being too small while barely affecting larger outputs. To ensure that this transformation is not overfitting to the particular set of 256 tokens, we report the leave-one-out cross-validation (LOOCV) loss of each method. We train a separate fit for each (method, input distribution) pair.[33]

5.6 Results

Figure 8 shows the performance of each method. The relative ordering is clear: both importance sampling methods outperform Quadratic Logit Decomposition, which in turn outperforms Gaussian Logit Difference. GLD is barely better than outputting an optimal constant (which can be interpreted as the performance of naive sampling). Figure 10 shows that there is a fair amount of variation in method performance across the 8 distributions: some behaviors like `hex` and `icl` favor MHIS, while others like `spanish` heavily favor ITGIS. A more detailed table of results is in Appendix E.

Among the two importance sampling methods, ITGIS does better on smaller models, while MHIS does better on larger models. We believe this is because larger models are less easily approximated as linear functions and are more likely to have complex behaviors arising from inter-token interactions.

Figure 9 displays example scatter plots of ITGIS, MHIS, and QLD estimates before a fit is applied. Each point represents the ground-truth and estimated probability of a different target token. More scatter plots can be found in Appendix H; note that the qualitative performances of the methods can vary significantly on different input distributions. We perform an ablation study on our choice of loss function in Appendix F, in which we score methods based on squared error in log-space instead of Itakura–Saito loss.

5.7 Discussion

5.7.1 Distribution shift as motivation

One might ask: if a particular model behavior is so rare that it never arises during training, why would we care about estimating its probability? There are a few reasons. First, some AI systems may be run on many more inputs during the course of deployment than during training. Thus, if a certain model behavior would be so catastrophic that it is unacceptable for it to occur even once in deployment, we cannot rely on training to drive down its probability low enough. Second, there may be distributional shift between training and deployment such that events that occur extremely rarely during training become more likely in deployment. This could occur because of an input chosen adversarially, but it could also occur because of goal misgeneralization [SVK+22].

A particularly challenging case is *deceptive alignment* (discussed in Section 3), the possibility that a machine learning model would look for clues about whether it is in a training or a deployment environment, and only behave well in training [HvMM+21]. To determine whether a model is deceptively aligned, one could craft an input distribution that is "wide enough" to assign *some* probability mass, even if very small, to any possible deployment-time input, then apply low probability estimation methods to detect if the model would ever perform a catastrophic behavior on this distribution.[34] For more discussion of this idea, see [Xu24].

[33]Note that the Gaussian Logit Difference method has a special functional form of its fit ($(\mu, \sigma) \mapsto \exp\left(-(a\mu/(\sigma+\epsilon))^2 + b\right) + c$ instead of $x \mapsto ax^c + b$) but is otherwise evaluated in the same way.

[34]To prevent false positives, this would require a very demanding definition of catastrophe that would be impossible for the model to trigger "by accident."

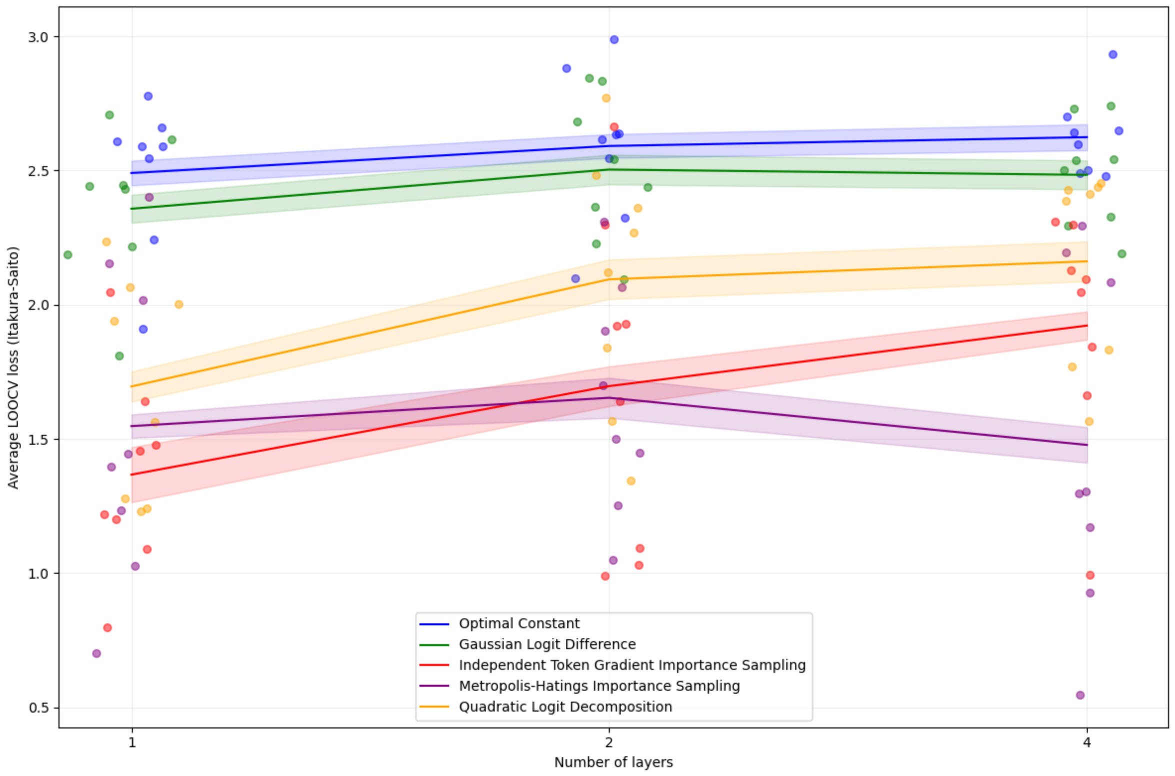


Figure 8: The Itakura–Saito loss of all methods across different model sizes. The solid lines indicate the loss of each method averaged over all 8 distributions, with bands showing standard error. The colored points indicate the loss on individual distributions, with horizontal jitter added for visibility. Lower is better.

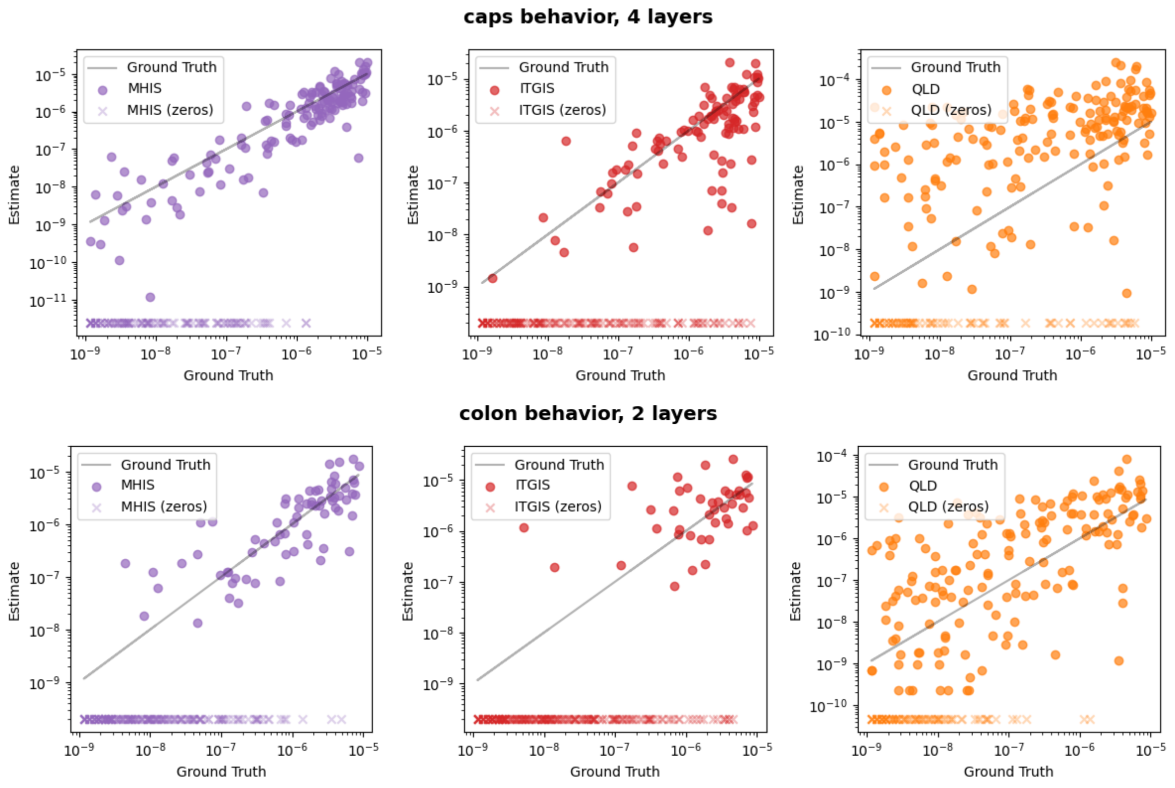


Figure 9: Examples of method outputs on two different distributions and models, before a fit is applied. Estimates of 0 are placed at the bottom of each graph for visibility.

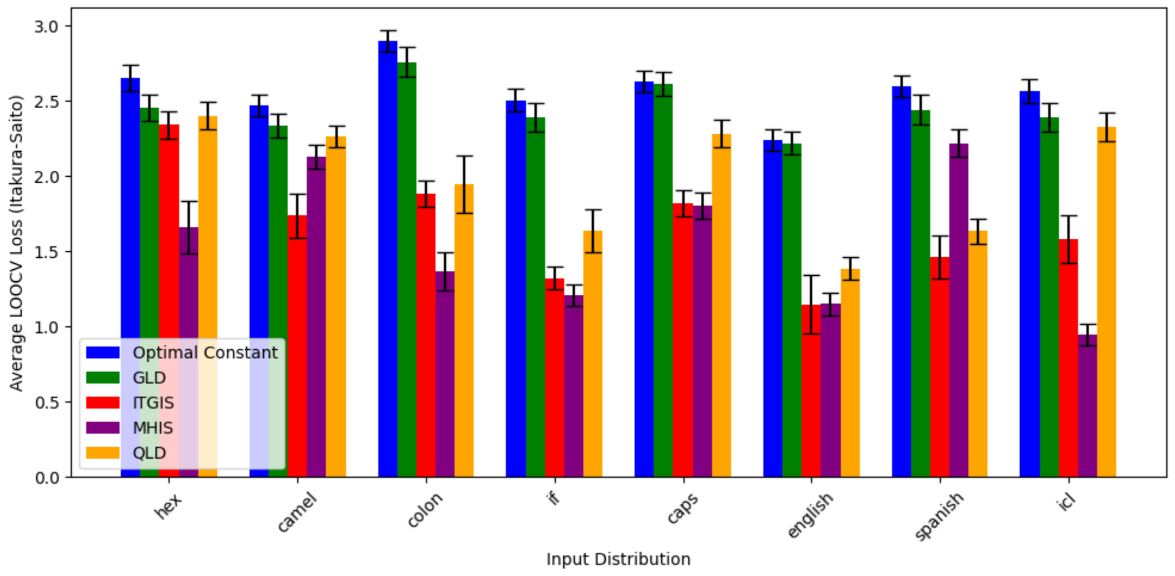


Figure 10: The Itakura–Saito loss of all methods across different distributions, averaged over all 3 model sizes. Lower is better.

5.7.2 Relation to red-teaming and adversarial training

Our importance sampling methods for low probability estimation involve finding inputs for which the rare event occurs. This amounts to the well-studied task of “red-teaming”. As long as the required importance sampling ratios can be computed, any method for red-teaming can be turned into an importance sampling method for low probability estimation, as we demonstrate with our adaptation of Greedy Coordinate Gradient into MHIS [ZWC+23]. However, our activation extrapolation methods such as QLD do not correspond to any red-teaming method.

A further reason to be interested in low probability estimation is that it could be used to reduce the probability of the rare event, by optimizing the model to produce a lower estimate. For example, this could be done using gradient descent, if the estimate were a differentiable function of the model’s parameters. For an importance sampling method, this amounts to finding inputs for which the rare event occurs (i.e., red-teaming) and using them as training data, which is essentially the well-known method of adversarial training [GSS14]. However, since our activation extrapolation methods do not correspond to any red-teaming method, new activation extrapolation methods potentially provide us with new ways to reduce the probabilities of rare events.

5.7.3 Importance sampling versus activation extrapolation

In our experiments, we found that importance sampling methods outperformed activation extrapolation. Nevertheless, there are theoretical cases in which importance sampling performs worse than other methods. For example, consider a model that outputs the SHA-256 hash of its input: finding any input that gives rise to a particular output is computationally infeasible, yet it is still easy to estimate the probability of a particular output by modeling the output of the hash function as random.

More generally, we are excited about low probability estimation as a concrete problem for which for which it may be necessary to leverage internal model activations. In place of importance sampling, we may be able to use deductive estimates based on a presumption of independence [CNX22]. Our Quadratic Logit Decomposition method is an early proof of concept of this, even though it is outperformed by importance sampling in our setting.

5.7.4 Limitations

There are two main limitations of our experimental setup. First, we only use input distributions that factor into independent tokens. This choice is necessary for the definition of ITGIS. It is also very convenient for the implementation of MHIS, because it gives efficient sampling access to the proposal distribution. To move beyond independent token input distributions, we could define the input distribution to be the output of a separate generative model and adapt our current estimation methods appropriately.

Second, we only study model behaviors that consist of a single token sampled at temperature 0. This is unrealistic because in practice, if we were concerned about specific single-token outputs, it would be easy to filter them out. In contrast, the types of behaviors we actually worry about likely involve long chains of autoregressive generation or interaction with the external world (e.g., when forming and executing a plan). We are excited to see future work extending our setting in this direction.

Nevertheless, it is worth noting that formally defined distributions and behaviors are more general than they may initially seem. For example, we could formalize the event "M writes buggy code", as: When M's output is given to GPT-4 along with the prompt "Does this code contain any bugs? Let's think step by step.", does GPT-4 end its response with `YES`?

5.7.5 Related work

The problem of low probability estimation was previously considered in the context of computer vision by [WRTK19], where they propose using an Adaptive Multi-Level Splitting algorithm with Metropolis Hastings. However, they only study the problem in the context of computer vision with continuous input spaces, and their approaches still require finding positive samples, unlike our activation extrapolation methods. [PAC+24] and [HPP+24] attempt to estimate the probability that a language model passes certain capability evaluations, even when its success rate is low, though their methods are not directly applicable to our formal setting.

Our importance sampling methods can be viewed as solving a special case of controlled text generation [ZSL+23] in which we want to sample from an autoregressive distribution conditioned on a property of the full output (in our case, that the last token is t). [YK21] do this by training Future Discriminators to steer model generation towards the desired attribute. [LZXGM23] approach the problem with a Sequential Monte Carlo steering approach; however, their infilling algorithm does not provide any benefit over naive sampling when all tokens except the last are independent. These works don't consider the problem of low probability estimation.

[ZBMG24] focus on the problem of estimating the partition function of an unnormalized target distribution over sequences, which is a more general case of our low probability estimation problem. Their Twisted Sequential Monte Carlo methods can be viewed as more advanced versions of our importance sampling methods. In contrast, in this work we focus on motivating the low probability estimation problem and introducing methods that do not involve searching for positive samples, such as activation extrapolation.

Finally, there is a large body of work applying adversarial training to improve worst-case model performance [BLZ+21, GSS14, IST+19], especially in the context of language models [Mad17, LCH+20]. [PHS+22] explores using language models themselves to aid in red-teaming other models. Latent adversarial training [CSPHM24, SEG+24] generalizes standard adversarial training by optimizing over perturbations in activation space; this means that, like activation extrapolation methods, it can be effective even when the adversarial training search problem over input space is hard.

5.7.6 Conclusion

In this work, we introduced the problem of low probability estimation along with four novel estimation methods. We defined and collected ground-truth probabilities for 8 different input distributions, then used them to evaluate the performance of our proposed methods. We found that the two importance sampling-based methods perform the best, with larger models favoring MHIS over ITGIS.

We are excited for future work that extends our empirical setup to non-independent input distributions and output behaviors that involve more than one token. We are also looking forward to future research that develops more accurate estimation methods, especially methods that move beyond importance sampling by incorporating deductive elements—for example, layer-by-layer activation modeling methods inspired by Deduction-Projection Estimators.

6 Optimizing Piecewise-Constant Objectives

Section 5 demonstrates that a method inspired by Deduction-Projection Estimators can beat baselines at low probability estimation. However, the baselines in question are quite weak, so this accomplishment should not be seen as a particularly impressive empirical result.

In the final section of this thesis, I present a setting in which a DPE-based method outperforms a much stronger and widely used baseline. In particular, I show that on the task of outputting the second-argmax of a sequence of inputs, models optimized with a DPE-based method are often more accurate than models optimized with cross-entropy loss. I view this as the first compelling example of deductive estimators prevailing over traditional machine learning techniques, although we only study very small models and a simple algorithmic task.

6.1 Introduction

In supervised machine learning, we often want to train classifiers to maximize *accuracy* over a given data distribution. That is, given a distribution $\mathcal{D}$ over labeled data pairs $(x, y) \in \mathcal{X} \times \mathcal{Y}$, where the number of labels $|\mathcal{Y}|$ is finite, we want our model M_θ to maximize the probability of outputting a correct label:

$$\mathrm{Acc}(\theta) := \Pr_{x,y}[M_\theta(x) = y].$$

The most natural estimator for the accuracy of the network is its accuracy on a randomly sampled data point. We can express this with the 0–1 loss function that takes in the parameters of the model and evaluates its accuracy on a given (x, y) pair:

$$\mathcal{L}_{\mathrm{Acc}}(\theta; x, y) := \mathbb{1}[M_\theta(x) \neq y].$$

Clearly, $\mathbb{E}_{x,y}[\mathcal{L}_{\mathrm{Acc}}(\theta; x, y)] = 1 - \mathrm{Acc}(\theta)$. But there's a problem here: $\mathcal{L}_{\mathrm{Acc}}$ has a discrete range $\{0, 1\}$, so for any fixed (x, y), it is piecewise constant as a function of θ. This means that if $\mathcal{D}$ is continuous, with probability 1 over the choice of (x, y), the gradient of $\mathcal{L}_{\mathrm{Acc}}$ is zero:[35]

$$\nabla_\theta \mathcal{L}_{\mathrm{Acc}}(\theta; x, y) = \mathbf{0}. \tag{3}$$

Note that $\nabla_\theta \mathrm{Acc}(\theta)$ is usually *not* zero because $\mathcal{D}$ can be continuous. However, Equation 3 shows we cannot estimate the gradient of Acc by taking the gradient of its unbiased estimator. In other words, the expectation and the gradient "don't commute" because $\mathcal{L}_{\mathrm{Acc}}$ is discontinuous in θ.

$$-\nabla_\theta \mathrm{Acc}(\theta) = \nabla_\theta \mathop{\mathbb{E}}_{x,y}[\mathcal{L}_{\mathrm{Acc}}(\theta; x, y)] \quad \neq \quad \mathop{\mathbb{E}}_{x,y}[\nabla_\theta \mathcal{L}_{\mathrm{Acc}}(\theta; x, y)] = \mathbf{0}$$

Thus, it is impossible to use stochastic gradient ascent against $\mathcal{L}_{\mathrm{Acc}}$ to optimize θ for having high accuracy.

[35] Another way to see this is that the decision boundaries induced by the classifier are continuous in θ, so any small-enough perturbation in θ will preserve the current labeling of x as long as it does not happen to lie on one of the boundaries.

The standard solution here is to train against a continuous "surrogate" loss function in place of the accuracy, usually cross-entropy loss (also known as "log loss") [MMZ23]. That is, we ask M_θ for a probability distribution over $\mathcal{Y}$, usually by having it output a vector of logits in $\mathbb{R}^{|\mathcal{Y}|}$ and then normalizing with a softmax. For any (x, y), let $p_y(x)$ be the probability assigned to y when the model is run on input x (the dependence on θ is implicit). The cross-entropy loss is then

$$\mathrm{CE}(\theta) := \mathop{\mathbb{E}}_{(x,y)} [-\log p_y(x)].$$

To optimize this, we apply stochastic gradient descent on the unbiased estimator $\mathcal{L}_{\mathrm{CE}}(\theta; x, y) := -\log p_y(x)$. Since $p_y(x)$ has a continuous range $[0, 1]$, the expectation of $\nabla_\theta \mathcal{L}_{\mathrm{CE}}$ is indeed $\nabla_\theta \mathrm{CE}(\theta)$. The final prediction of the model is the label that obtains the highest logit:

$$M_\theta(x) := \arg\max_y p_y(x).$$

Empirically, training with cross-entropy loss typically produces classifiers with high accuracy. Cross-entropy is a proper scoring rule, so this has the additional benefit of encouraging predictions to be calibrated: we can tell how confident the model is based on $\boldsymbol{p}(x)$. However, if all we ultimately care about is producing a classifier with the maximum possible accuracy, cross-entropy is an unsatisfying solution. Optimizing a model to achieve low cross-entropy loss is a different objective than optimizing it for high accuracy, and the two objectives trade off against each other given a model class of bounded capacity:

$$\arg\min_\theta \mathrm{CE}(\theta) \neq \arg\max_\theta \mathrm{Acc}(\theta).$$

In this section, I present the Gaussian Mixture Half-space Pruning (GMHP): a Deduction-Projection Estimator that can be used as an alternative objective function for optimizing $\mathrm{Acc}(\theta)$. Unlike the sampling-based estimator $\mathcal{L}_{\mathrm{Acc}}$, GMHP is *not* a piecewise-constant function of θ, so we can optimize for it directly with gradient ascent. We will show that **GMHP often produces classifiers that *outperform* classifiers trained only with cross-entropy loss,** although there are still some sampling-based estimators that perform better.

6.2 Problem statement

We define our supervised learning setting: small recurrent neural networks (RNNs) trained to output the second-argmax of a list of numbers.

The second-argmax function $\mathrm{SAM} : \mathbb{R}^n \to [n]$ takes in n inputs $x_1, x_2, \ldots, x_n \in \mathbb{R}$, and outputs the index of the second-largest argument. For example:

$$\begin{aligned} \mathrm{SAM}(-0.7,\ 0.0,\ -1.6) &= 1 \\ \mathrm{SAM}(0.2,\ 1.5,\ -0.5,\ 0.8) &= 4 \\ \mathrm{SAM}(3.2,\ 1.1,\ 0.2,\ -1.9,\ 1.0) &= 2 \end{aligned}$$

We study the problem of training RNNs to compute SAM with the highest possible accuracy on inputs drawn from the standard normal distribution $\mathcal{N}(0, \mathrm{Id}_n)$. While this is a very easy task for a large enough neural network, we study the regime in which the models do not have enough parameters to achieve 100% accuracy.

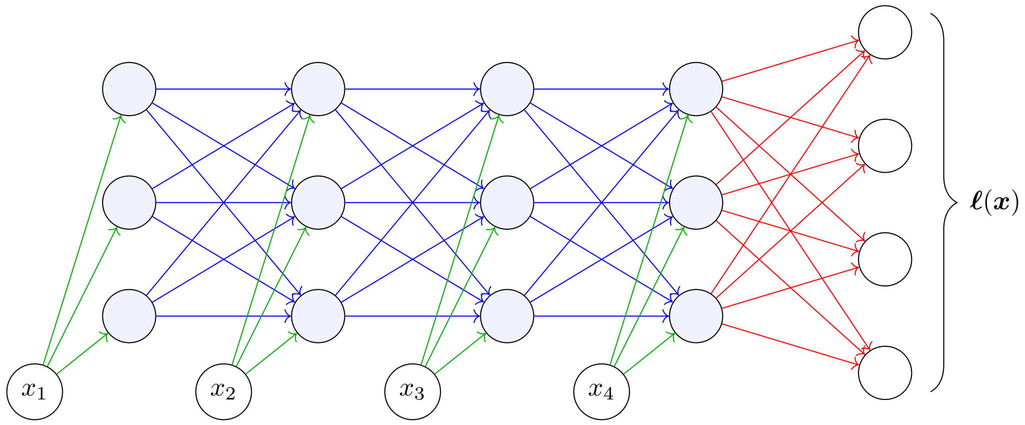


Figure 11: An RNN with $n = 4$ and $d = 3$. The value of each neuron is a scalar given by the weighted sum of all incoming neurons. ReLUs are applied at each shaded neuron. Green arrows indicate $\boldsymbol{w}^{hi}$, blue arrows indicate $\boldsymbol{W}^{hh}$, and red arrows indicate $\boldsymbol{W}^{oh}$.

Remark 6.1. Why second-argmax?

We work with the second-argmax function because it is one of the simplest algorithmic tasks that is not trivial for an RNN. In contrast, the (first-)argmax task is easy for a neural network: simply let the i-th output logit be x_i.

The DPE for second-argmax leverages certain bespoke structural properties of SAM, for example that it can be represented as a piecewise constant function on a polynomial number of polyhedral cones. We eventually hope to find DPEs for other types of algorithmic tasks where this assumption is false. Note that during the research process, SAM was chosen as an algorithmic task to target *before* we had an idea of what the DPE would look like.

Our RNN architecture has parameters $\theta = (\boldsymbol{w}^{hi}, \boldsymbol{W}^{hh}, \boldsymbol{W}^{oh})$, where $\boldsymbol{w}^{hi} \in \mathbb{R}^d$ is the input vector, $\boldsymbol{W}^{hh} \in \mathbb{R}^{d\times d}$ is the transition matrix, and $\boldsymbol{W}^{oh} \in \mathbb{R}^{n\times d}$ is the output matrix. The computation performed by the model is defined as follows:

$$\begin{aligned}
\boldsymbol{h}^{(0)} &= \mathbf{0} && \in \mathbb{R}^d \\
\boldsymbol{h}^{(t)} &= \mathrm{ReLU}(\boldsymbol{W}^{hh}\boldsymbol{h}^{(t-1)} + \boldsymbol{w}^{hi}x_t) && \in \mathbb{R}^d \quad \forall t \in [n] \\
\boldsymbol{\ell} &= \boldsymbol{W}^{oh}\boldsymbol{h}^{(n)} && \in \mathbb{R}^n \\
p_i &= \frac{\exp(\ell_i)}{\sum_{j=1}^n \exp(\ell_j)} && \in (0,1) \quad \forall i \in [n]
\end{aligned}$$

The vector $\boldsymbol{\ell}$ represents the output logits of the model. We will often use the notation $\boldsymbol{\ell}(\boldsymbol{x})$ or $\boldsymbol{p}(\boldsymbol{x})$ to refer to the logits or output probabilities as a function of the inputs (the dependence on θ is implicit). The prediction of the model $M_\theta(\boldsymbol{x})$ corresponds to the highest logit $\ell_i(\boldsymbol{x})$ (equivalently, output probability $p_i(\boldsymbol{x})$). We wish to find training methods that maximize[36]

$$\mathrm{Acc}(\theta) := \Pr_{\boldsymbol{x}\sim\mathcal{N}(0,\mathrm{Id}_n)} \left[\arg\max_i \ell_i(\boldsymbol{x}) = \mathrm{SAM}(\boldsymbol{x})\right].$$

Finally, note that our architecture has no biases, so in particular any positive scaling of $\boldsymbol{x}$ results in the same prediction.

$$\arg\max_i \ell_i(\boldsymbol{x}) = \arg\max_i \ell_i(a\boldsymbol{x}) \quad \forall a > 0$$

[36]If there is an exact tie for the highest logit then we assign partial credit; i.e., we calculate the accuracy the model would have if there were an infinitesimal amount of isotropic noise added to the logits. This can happen with positive probability if all d neurons in the last layer are 0 due to the final ReLU.

6.3 Exact algorithm with Girard's formula

We wish to find a deductive estimator for $\mathrm{Acc}(\theta)$. In this subsection, we present an algorithm that can compute the exact value of $\mathrm{Acc}(\theta)$ when $n = 3$, although it runs in time exponential in d.

Consider the function $\boldsymbol{\ell} : \mathbb{R}^n \to \mathbb{R}^n$ that maps an input $\boldsymbol{x}$ to a logit vector.

Lemma 6.1

$\boldsymbol{\ell}$ is piecewise linear as a function of $\boldsymbol{x}$, where the number of pieces is at most 2^{nd}.

Proof. There are nd neurons $h^{(1)}_{1,\ldots,d}, \ldots, h^{(n)}_{1,\ldots,d}$ that have ReLUs applied in the computational graph of the RNN (see Figure 11). Label these neurons with $[n] \times [d]$, and let the value of neuron (i, j) on input $\boldsymbol{x}$ be $h^{(i)}_j(\boldsymbol{x})$.

Consider any subset $S \subseteq [n] \times [d]$ of these neurons. Let $R_S \subseteq \mathbb{R}^n$ be the region of input space that corresponds to S having active ReLUs:

$$R_S = \left\{ \boldsymbol{x} \in \mathbb{R}^n \;\middle|\; \left[h^{(i)}_j(\boldsymbol{x}) > 0 \Leftrightarrow (i,j) \in S \right] \; \forall (i,j) \in [n] \times [d] \right\}.$$

This results in a decomposition of input space:

$$\bigsqcup_{S \subseteq [n] \times [d]} R_S = \mathbb{R}^n.$$

On any given R_S, $\boldsymbol{\ell}$ acts linearly because we can replace each of the ReLUs in the computational graph with either the identity function or the zero function (both of which are linear), and the composition of linear functions is linear. Thus, there exist matrices $\boldsymbol{A}_S \in \mathbb{R}^{n \times n}$ for all $S \subseteq [n] \times [d]$ such that

$$\boldsymbol{\ell}(\boldsymbol{x}) = \begin{cases} \boldsymbol{A}_{\varnothing}\boldsymbol{x} & \boldsymbol{x} \in R_{\varnothing} \\ \boldsymbol{A}_{\{(1,1)\}}\boldsymbol{x} & \boldsymbol{x} \in R_{\{(1,1)\}} \\ \quad \vdots & \\ \boldsymbol{A}_{[n]\times[d]}\boldsymbol{x} & \boldsymbol{x} \in R_{[n]\times[d]} \end{cases}.$$

■

In fact, we can show the following.

Lemma 6.2

For any $S \subseteq [n] \times [d]$, the region R_S corresponds to an intersection of nd half-spaces in $\mathbb{R}^n$. The normal vectors of these half-spaces and the linear map $\boldsymbol{A}_S$ can both be found in polynomial time given the weights of the RNN.

Proof. This can be shown inductively by traversing the RNN layer by layer. For any $t \leq n$, let $R^{(t)}_S \supseteq R_S$ be the set of all inputs that "obey" S in the first t layers:

$$R^{(t)}_S = \left\{ \boldsymbol{x} \in \mathbb{R}^n \;\middle|\; \left[h^{(i)}_j(\boldsymbol{x}) > 0 \Leftrightarrow (i,j) \in S \right] \; \forall (i,j) \in [t] \times [d] \right\},$$

where the only difference from the definition of R_S is in purple. Then we can prove that for all t,

- $R^{(t)}_S$ is an intersection of td half-spaces which can be computed in polynomial time, and
- the function $\boldsymbol{h}^{(t)} : \mathbb{R}^n \to \mathbb{R}^d$ restricted to $R^{(t)}_S$ is linear and can be found in polynomial time.

This is clearly true for $t = 0$. For the inductive step, say that $\boldsymbol{h}^{(t)}(\boldsymbol{x}) = \boldsymbol{A}\boldsymbol{x}$ on $R^{(t)}_S$ for some $\boldsymbol{A} \in \mathbb{R}^{d \times n}$.

$R_S^{(t+1)}$ is the subset of $R_S^{(t)}$ that obeys S at every neuron layer $t+1$. For every such neuron $(t+1, j)$, we know that its pre-ReLU activation must be $\boldsymbol{W}_{j,:}^{hh}\boldsymbol{A}\boldsymbol{x} + w_j^{hi}x_{t+1}$ for all $\boldsymbol{x} \in R_S^{(t)}$. This can be written as $\boldsymbol{a}_j \cdot \boldsymbol{x}$ for

$$\boldsymbol{a}_j = \boldsymbol{W}_{j,:}^{hh}\boldsymbol{A} + w_j^{hi}\boldsymbol{e}_{t+1},$$

where $\boldsymbol{e}_{t+1}$ denotes the unit vector with a 1 in the $t+1$-th coordinate. Thus, $\boldsymbol{a}_1, \ldots, \boldsymbol{a}_d \in \mathbb{R}^n$ are the normal vectors of half-spaces such that

$$R_S^{(t+1)} = R_S^{(t)} \cap \{\boldsymbol{x} \mid \boldsymbol{x} \cdot \boldsymbol{a}_j > 0 \; \forall j \in [d]\}.$$

The function $\boldsymbol{h}^{(t+1)}$ restricted to $R_S^{(t+1)}$ is a linear transformation given by the matrix with $\boldsymbol{a}_j$ as its j-th row. This completes the inductive step.

Thus, the nd half-space constraints that define $R_S^{(n)} = R_S$ and the matrix $\boldsymbol{A}_S$ corresponding to $\boldsymbol{\ell}(\boldsymbol{x}) = \boldsymbol{W}^{oh}\boldsymbol{h}^{(n)}(\boldsymbol{x})$ on R_S can both be computed in polynomial time ($O(n^2d^2)$ time, to be specific).

■

Before we describe the algorithm for calculating the exact value of $\mathrm{Acc}(\theta)$, we must discuss one more preliminary: the *intersection of half-spaces problem.*

Definition 6.1. Spherical volume of an intersection of half-spaces

Given a list of n-dimensional half-spaces with normal vectors $\boldsymbol{v}_1, \ldots, \boldsymbol{v}_k \in \mathbb{R}^n$, the spherical volume of their intersection is:

$$\mathrm{SVol}(\boldsymbol{v}_1, \ldots, \boldsymbol{v}_k) := \Pr_{\boldsymbol{x} \sim \mathcal{N}(0, \mathrm{Id}_n)}[\boldsymbol{x} \cdot \boldsymbol{v}_i > 0 \quad \forall i \in [k]].$$

Note that $\mathcal{N}(0, \mathrm{Id}_n)$ can be replaced with any spherically symmetric distribution over $\mathbb{R}^n$ without changing the definition. The intersection of half-spaces that all pass through the origin is sometimes referred to as a *polyhedral cone.*

This quantity is hard to compute in general. For a review of best-known approximation methods, see [Fit24]. However, in a small number of dimensions, there is a straightforward algorithm for computing it exactly.

Theorem 6.1

In $n = 3$ dimensions, the spherical volume of an intersection of k half-spaces $\mathrm{SVol}(\boldsymbol{v}_1, \ldots, \boldsymbol{v}_k)$ can be computed in $\widetilde{O}(k)$ time.

Proof. Consider the 2-manifold obtained by intersecting all of the half-spaces with the unit sphere. The surface area of this 2-manifold, sometimes called a *spherical polygon*, is equivalent to $\mathrm{SVol}(\boldsymbol{v}_1, \ldots, \boldsymbol{v}_k)$ after normalizing by the surface area of the entire sphere, 4π. A classic fact of spherical trigonometry is that the surface area of an N-sided spherical polygon is given by

$$\left(\sum_{i=1}^{N} a_i\right) - (N-2)\pi,$$

where $a_1, \ldots, a_N$ are the interior angles of the polygon, measured in radians.[37] This result is known as Girard's Theorem, and it has an elegant geometric proof that we will not reproduce here (for example, see [Van13]). Thus, the only remaining challenge is to compute the vertices of the spherical polygon in question, since the interior angles between them can easily be derived from their coordinates.

There are many ways to compute these vertices. One approach is to iterate over the intersections of all

[37] As a sanity check, if the polygon is small enough that it is almost planar (e.g., the floor plan of a house on the surface of the Earth), then the sum of the interior angles is indeed very close to $(N-2)\pi$.

$\binom{k}{2}$ pairs of half-spaces, each of which corresponds to a pair of antipodal points on the sphere. Each of these points can be tested for membership on the boundary of the spherical polygon by checking if it satisfies the other $k-2$ constraints. The points that pass this test exactly correspond to the vertices. Finally, we order them with an angular sweep. This takes $O(k^3)$ time in total.

A more efficient algorithm involves adding in intersections with the half-spaces one at a time and maintaining a sorted list of vertices of the current spherical polygon along the way. At any moment during this process there are at most $O(k)$ vertices, and updating the list of vertices can be done in amortized $O(\log k)$ time by storing the vertices in an efficient data structure like a self-balancing binary tree.[38] Thus, we can compute $\mathrm{SVol}(\boldsymbol{v}_1, \ldots, \boldsymbol{v}_k)$ for $n = 3$ in $\widetilde{O}(k)$.

$\blacksquare$

Piecing these results together, we get an algorithm for computing $\mathrm{Acc}(\theta)$ in $2^{nd}\mathrm{poly}(n, d)$ time when $n = 3$. The key is to express the event "$M_\theta(\boldsymbol{x}) = \mathrm{SAM}(\boldsymbol{x})$" as a union of $2^{nd}n(n-1)$ disjoint events in the following way:

$$\begin{aligned}\mathrm{Acc}(\theta) &= \Pr_{\boldsymbol{x}}[M_\theta(\boldsymbol{x}) = \mathrm{SAM}(\boldsymbol{x})] \\ &= \sum_{S} \sum_{i_1 \neq i_2 \in [n]} \Pr_{\boldsymbol{x}} \begin{bmatrix} \boldsymbol{x} \in R_S & & \wedge \\ x_{i_1} > x_{i_2} > x_j & \forall j \neq i_1, i_2 & \wedge \\ (\boldsymbol{A}_S\boldsymbol{x})_{i_2} > (\boldsymbol{A}_S\boldsymbol{x})_j & \forall j \neq i_2 & \end{bmatrix}. \end{aligned} \tag{4}$$

In other words, we condition on the subset of nonzero intermediate neurons (2^{nd} choices), as well as which coordinates of $\boldsymbol{x}$ are the argmax and second-argmax ($n(n-1)$ choices). The probability of each of these events reduces to a SVol calculation. Indeed, each of the three types of constraints can be written as an intersection of half-spaces that pass through the origin: $\boldsymbol{x} \in R_S$ requires nd half-spaces (Lemma 6.2), each $x_i > x_j$ requires one half-space, and each $(\boldsymbol{A}_S\boldsymbol{x})_{i_2} > (\boldsymbol{A}_S\boldsymbol{x})_j$ requires another half-space.

Since this algorithm relies on Girard's formula, it only works when $n = 3$. Although there has been some work in extending Girard's formula to higher dimensions, we lack a general formula that works for all n [GB09]. In fact, we do not even know how to efficiently compute the area of spherical *simplices* for general $k = n$ (also known as the "centered orthant probability" of a multivariate Gaussian), although there are techniques that allow us to reduce the dimensionality of the integral by a factor of 2 [SB53]. The intractability of this problem in high dimensions motivates the need for a deductive *estimation* algorithm, which we explore in the next section.

6.4 Gaussian Mixture Half-space Pruning (GMHP)

The exact algorithm based on Girard's formula presented in the previous subsection is unsatisfactory for larger values of n for two reasons. First, as mentioned, we do not have an exact formula for the half-space intersection problem in higher dimensions. Second, the complexity of the calculation is exponential in nd, as we are forced to iterate over all subsets of nonzero neurons.

Here we present a deductive estimator for $\mathrm{Acc}(\theta)$ called Gaussian Mixture Half-space Pruning (GMHP) that can be applied to larger values of n. GMHP addresses each of the previous two failures individually. To deal with the half-space intersection problem, we introduce an approach to deductively estimating $\mathrm{SVol}(\boldsymbol{v}_1, \ldots \boldsymbol{v}_k)$ using a modified form of covariance propagation. To deal with the exponential blowup in the number of subsets of active neurons, we prune the smallest regions along the way.

[38] Concretely, we must remove all vertices that do not lie in the new half-space (which corresponds to a contiguous subarray of vertices) and insert up to two new vertices corresponding to the intersections between the polygon and the separating plane.

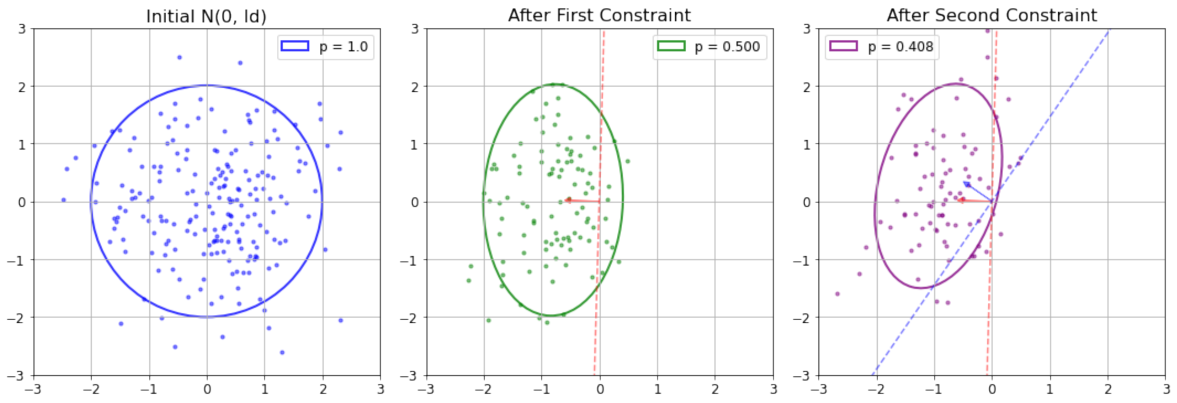


Figure 12: An example of covariance propagation for the half-space intersection problem when $n = k = 2$.

6.4.1 Covariance propagation for half-space intersection

We introduced covariance propagation [CNX22] in Section 2.1 as a method for estimating the expected value of a neural network. It entails deductively pushing a distribution of activations forward through each layer, then projecting back down to a Gaussian with moment matching after each step. We can adapt this method to the intersection of half-spaces problem, where instead of applying a non-linear neural network layer, we condition our current distribution to lie in a half-space. This is mechanically identical to Assumed Density Filtering [Ran04], where our Bayesian updates correspond to the observations that each successive constraint is satisfied.

Given a Gaussian with mean $\boldsymbol{\mu} \in \mathbb{R}^n$ and covariance $\boldsymbol{\Sigma} \in \mathbb{R}^{n \times n}$, as well as a normal vector $\boldsymbol{v} \in \mathbb{R}^n$, we can deductively calculate each of the following in $O(n^2)$ time:

$$\Pr_{\boldsymbol{x} \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})}[\boldsymbol{x} \cdot \boldsymbol{v} > 0] \qquad \mathop{\mathbb{E}}_{\boldsymbol{x} \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})}[\boldsymbol{x} \mid \boldsymbol{x} \cdot \boldsymbol{v} > 0] \qquad \mathop{\mathrm{Cov}}_{\boldsymbol{x} \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})}[\boldsymbol{x} \mid \boldsymbol{x} \cdot \boldsymbol{v} > 0]$$

The details are very similar to propagating a Gaussian through a single ReLU as discussed in Section 2.1, and it only requires access to the PDF and CDF of a one-dimensional standard Gaussian. See our code for implementation details.

The covariance propagation algorithm for estimating $\mathrm{SVol}(\boldsymbol{v}_1, \ldots, \boldsymbol{v}_k)$ simply consists of maintaining a running mean, covariance, and cumulative probability after conditioning on each prefix of half-space constraints. The output of the algorithm is this final cumulative probability, which corresponds to the product of all k single-step conditional probabilities. The algorithm takes $O(n^2 k)$ time.

Note that covariance propagation can sometimes result in poor estimates, even though in practice it is often reasonably accurate. For example, because any non-degenerate Gaussian is supported over all $\mathbb{R}^n$, this method always outputs a positive estimate, even when the answer is 0. Its output is also sensitive to the order in which we apply the half-space constraints.

6.4.2 Pruning regions

To reduce the computational cost of summing the spherical volumes of an exponentially large number of polyhedral cones, we introduce *pruning*. We need a slight shift in perspective to understand where this pruning occurs. In the algorithm presented in Section 6.3, we iterate through all $2^{nd} n(n-1)$ choices of (S, i_1, i_2) corresponding to regions on which M_θ is correct (Equation 4). For each such region in sequence, we gather its half-space constraints and calculate SVol.

By contrast, for GMHP we calculate SVol for all of the regions *in parallel* as we traverse forward through the model. This corresponds to performing a breadth-first search instead of a depth-first search of the tree shown in Figure 13. Due to the Gaussian representation of intermediate conditional

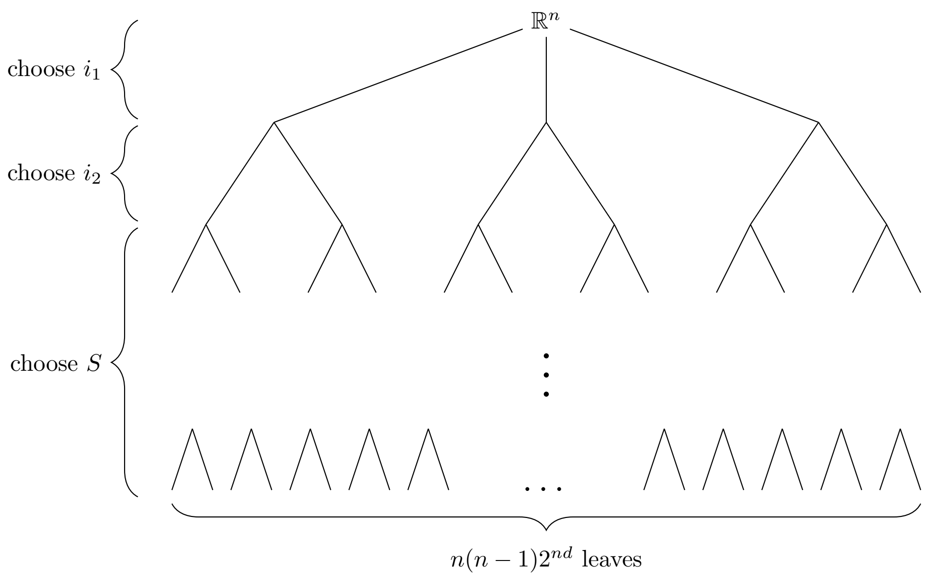


Figure 13: The tree structure of our partition of $\mathbb{R}^n$. Each edge represents one or more half-space constraints, and each vertex represents the polyhedral cone obtained by intersecting the constraints on its path to the root. Equation 4 corresponds to a depth-first search of this tree, where we serially apply a SVol calculation at each leaf. By contrast, GMHP can be viewed as a breadth-first search, where we maintain the conditional means, conditional covariances, and cumulative probabilities of the largest C regions at each successive layer of the tree.

distributions used in covariance propagation, we can view this as approximating the current state of the model as a *mixture of Gaussians*. At every layer t, we maintain a list of components, each containing a weight $p \in [0,1]$, a mean $\boldsymbol{\mu} \in \mathbb{R}^n$, a covariance $\boldsymbol{\Sigma} \in \mathbb{R}^{n\times n}$, a matrix $\boldsymbol{A} \in \mathbb{R}^{d\times n}$, and an index $i_2 \in [n]$.

Semantically, this corresponds to a representation of the joint distribution of $(\boldsymbol{x}, \boldsymbol{h}^{(t)}, y)$ as the distribution induced by the following process:[39]

- Sample one of the components with probability proportional to its weight p. Retrieve the values $i_2, \boldsymbol{\mu}, \boldsymbol{\Sigma}, \boldsymbol{A}$ associated with this component.
- Let $y = i_2$.
- Sample $\boldsymbol{x}$ from $\mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$.
- Let $\boldsymbol{h}^{(t)} = \boldsymbol{A}\boldsymbol{x}$.

Propagating this representation through a linear transformation is easy: for each component, we simply compose the transformation with the matrix $\boldsymbol{A}$. This handles adding $\boldsymbol{w}^{hi}x_t$ or applying $\boldsymbol{W}^{hh}$ or $\boldsymbol{W}^{oh}$ to the current activations.

The real action happens during the ReLUs. Instead of applying a layer of ReLUs to all d neurons at once, we apply it one by one to each neuron in series. To propagate the representation through the ReLU of a single neuron, we split each component into two: one child corresponding to $\boldsymbol{A}\boldsymbol{x}$ taking a positive value at that neuron position, and the other child taking a negative value. We update the mean $\boldsymbol{\mu}$, covariance $\boldsymbol{\Sigma}$, and weight p of both children based on the process described in Section 6.4.1. Finally, for the negative component, we zero out the row of $\boldsymbol{A}$ associated with the current neuron position to demonstrate the effect of the ReLU.

Naively, this would double the number of components we have to track for every ReLU. To account for this, we prune all but the C components with the highest weight by simply removing them from our list. We re-normalize the weights of the remaining components to sum to 1. The value of the pruning parameter C is up to us: lower values of C will be more efficient but lead to less accurate estimates.

[39]Note that this representation cannot be exact. In fact, it is not even logically consistent—for example, it places positive probability on cases where $y \neq \mathrm{SAM}(\boldsymbol{x})$.

Once this representation has propagated through the entire RNN, we are left with at most C Gaussian components. Each component is associated with a value of y, as well as an implied Gaussian distribution over the logit vector $\boldsymbol{\ell} \in \mathbb{R}^n$. Under this distributional assumption, we can estimate the probability that the y-th logit is higher than all of the others by applying covariance propagation through $n - 1$ more half-space constraints. Taking a weighted sum of these probabilities over all components gives us an overall estimate of the accuracy of the model.

The time complexity of GMHP is $O(Cn^3d)$ when using an $O(n^2)$ algorithm for propagating a mean and covariance through a single half-space constraint as discussed in Section 6.4.1. See Algorithm 4 for pseudocode.

Algorithm 4 Gaussian Mixture Half-space Pruning (GMHP)

```
Require: RNN weights $\theta = (\boldsymbol{w}^{hi}, \boldsymbol{W}^{hh}, \boldsymbol{W}^{oh})$, pruning parameter $C$.
    # Initialize Gaussian components with all $n(n-1)$ choices of $i_1, i_2$
 1: components ← []
 2: for $i_1 \leftarrow 1$ to $n$ do
 3:     for $i_2 \leftarrow 1$ to $n$ do
 4:         if $i_1 = i_2$ then
 5:             continue
 6:         end if
 7:         comp ← GaussianComponent($p = 1, \boldsymbol{\mu} = \boldsymbol{0}, \boldsymbol{\Sigma} = \mathrm{Id}_n, \boldsymbol{A} = \boldsymbol{0}, i_2 = i_2$)
 8:         for $j \leftarrow 1$ to $n$ do
 9:             if $j = i_1$ then
                    # Propagates $p$, $\boldsymbol{\mu}$, and $\boldsymbol{\Sigma}$ through a half-space with a given normal vector. $\boldsymbol{e}_j \in \mathbb{R}^n$ is
                    the $j$-th unit direction.
10:                 comp ← comp.condition_halfspace($\boldsymbol{e}_j - \boldsymbol{e}_{i_2}$)
11:             else if $j \neq i_2$ then
12:                 comp ← comp.condition_halfspace($\boldsymbol{e}_{i_2} - \boldsymbol{e}_j$)
13:             end if
14:         end for
15:         Append comp to components
16:     end for
17: end for

18: for $t \leftarrow 1$ to $n$ do
        # Apply this layer's linear transformation
19:     for comp in components do
20:         comp.$\boldsymbol{A} \leftarrow \boldsymbol{W}^{hh}$ comp.$\boldsymbol{A}$                      # Matrix multiplication
21:         comp.$\boldsymbol{A}_{:,t}$ += $\boldsymbol{w}^{hi}$
22:     end for
        # Apply ReLUs one by one
23:     for $j \leftarrow 1$ to $d$ do
24:         new_components ← []
25:         for comp in components do
26:             pos ← comp.condition_halfspace(comp.$\boldsymbol{A}_{j,:}$)
27:             neg ← comp.condition_halfspace($-$comp.$\boldsymbol{A}_{j,:}$)
28:             neg.$\boldsymbol{A}_{i,:} \leftarrow \boldsymbol{0}$
29:             Append pos and neg to new_components
30:         end for
31:         Sort new_components by weight $p$ in descending order
32:         components ← new_components[:$C$]                      # Prune all but first $C$
33:     end for
34: end for

35: ans ← 0                      # Sum of probabilities of correct logit being highest
36: $Z \leftarrow 0$                      # Sum of all probabilities, for re-normalization
37: for comp in components do
38:     $Z$ += comp.$p$
39:     comp.$\boldsymbol{A} \leftarrow \boldsymbol{W}^{oh}$ comp.$\boldsymbol{A}$
40:     for $j \leftarrow 1$ to $n$ do
41:         if $j \neq$ comp.$i_2$ then
42:             comp ← comp.condition_halfspace(comp.$\boldsymbol{A}_{i_2,:}$ − comp.$\boldsymbol{A}_{j,:}$)
43:         end if
44:     end for
45:     ans += comp.$p$
46: end for

47: return ans/$Z$
```

6.5 Experimental setup

To test the efficacy of GMHP as a deductive estimator for $\mathrm{Acc}(\theta)$, we use gradient ascent to optimize the weights of a small RNN to maximize $\mathrm{GMHP}(\theta)$. We compare the accuracy of the resultant network (which is measured via traditional sampling) with the accuracy of RNNs trained against a variety of sampling-based surrogate loss functions.

6.5.1 Objective functions

First, we have our two deductive methods:

- **Exact accuracy (Girard).** When $n = 3$, we can directly calculate the exact accuracy of the RNN using the algorithm described in Section 6.3. We call this objective Girard Accuracy due to its use of Girard's formula.
- **Gaussian Mixture Half-space Pruning (GMHP).** This is the DPE for $\mathrm{Acc}(\theta)$ described in Section 6.4. Unless otherwise specified, we set the pruning parameter to $C = 10^4$.

The remaining surrogate loss functions are all sampling-based:

- **Cross-entropy loss (CE).** This is the loss function traditionally used when training neural networks. On a randomly sampled input $\boldsymbol{x} \in \mathbb{R}^n$, the CE loss is

$$\mathcal{L}_{\mathrm{CE}}(\theta; \boldsymbol{x}) := -\log p_{\mathrm{SAM}(\boldsymbol{x})}(\boldsymbol{x}).$$

- **Sigmoid Separation loss (SS).** A shortcoming of CE loss is that it incentivizes the model to increase the logit of the correct answer, even if it is already either the highest logit or so far below the others that changing the prediction is hopeless. This may waste RNN "capacity" that could be more efficiently spent on improving its accuracy on other regions of input space. Intuitively, we want our loss function to only provide strong gradient updates when a small perturbation in the logits could change its prediction from incorrect to correct (or vice versa). This motivates the following objective function, which we call Sigmoid Separation loss. On an input $\boldsymbol{x} \in \mathbb{R}^n$, define the *logit margin* $m(\boldsymbol{x})$ to be the difference between the correct logit and the highest other logit:

$$m(\boldsymbol{x}) := \ell_{\mathrm{SAM}(\boldsymbol{x})}(\boldsymbol{x}) - \max_{i \neq \mathrm{SAM}(\boldsymbol{x})} \ell_i(\boldsymbol{x}).$$

  Then, the SS loss is:

$$\mathcal{L}_{\mathrm{SS}}(\theta; \boldsymbol{x}) := \mathrm{sigmoid}(-m(\boldsymbol{x})/\delta)$$

  where $\mathrm{sigmoid}(z) = (1 + e^{-z})^{-1}$. The sigmoid function is chosen for its S shape, which means its derivative is only large when the margin is close to 0. We set the parameter δ to be 10^{-2} unless otherwise specified, which we found to work the best in practice.[40] Note that SS loss can be thought of as a smoothed version of accuracy: as δ shrinks to 0, $\mathcal{L}_{\mathrm{SS}}$ approaches $\mathcal{L}_{\mathrm{Acc}}$.

- **Hook loss.** The Hook loss also uses the logit margin, but it applies a discrete threshold instead of a sigmoid. This loss function was introduced in [CS02] for optimizing multiclass kernel-based vector machines. It can be viewed as a multiclass generalization of the commonly-used Hinge loss.

$$\mathcal{L}_{\mathrm{Hook}}(\theta; \boldsymbol{x}) := \max(-m(\boldsymbol{x}) + \alpha, 0)$$

  We use $\alpha = 1$ unless otherwise specified.

[40] In theory, the value of δ should not change the accuracy we are able to achieve. Given any two δ_1, δ_2, let θ_1, θ_2 be the global optima of $\mathbb{E}_{\boldsymbol{x}}[\mathcal{L}_{\mathrm{SS}}(\theta; \boldsymbol{x})]$ when δ is set to δ_1 and δ_2, respectively. Then $\mathrm{Acc}(\theta_1) = \mathrm{Acc}(\theta_2)$ because θ_2 can always copy θ_1 except with $\boldsymbol{W}^{oh}$ scaled up by δ_1/δ_2. However, in practice, the choice of δ affects the training dynamics.

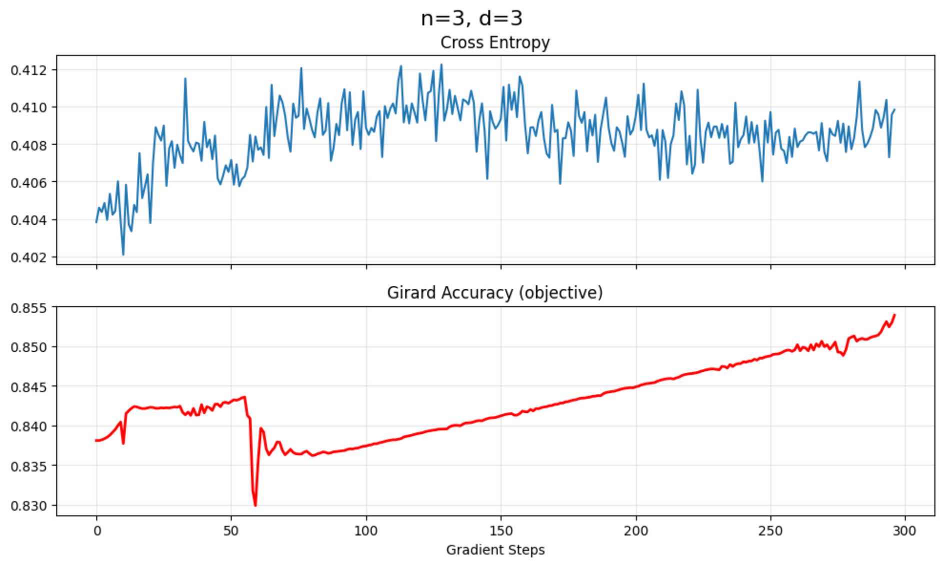


Figure 14: We can train against Girard Accuracy with gradient ascent. The CE loss gets worse as we do this. We use a constant learning rate of 3×10^{-3} with a warm-up.

6.5.2 Model training

In practice, training runs on tiny models often get stuck in poor local minima. Rather than train all of our models from scratch, we first train five models with different random initializations against cross-entropy loss until convergence, then discard all but the most accurate one. We then observe **how much that model's accuracy increases or decreases relative to this CE-trained baseline after further optimization on each of the other objectives (Girard, GMHP, CE, SS, Hook)**. This reduces the variance between training runs and allows us to draw a fairer comparison between the different objective functions.

Our main experiments are on RNNs of sizes $(n, d) \in \{4, 5, 6, 8\} \times \{3, 4, 5, 6\}$. This amounts to 16 different models. We train models with the Adam optimizer [KB17] and a custom learning rate schedule.[41] We use 250 gradient steps when training against GMHP, and 500 gradient steps with a batch size of 2^{20} when training against Hook or SS. These choices are usually enough for the loss to converge. All of our experiments take less than an hour to run on an A100 GPU.

Our code is available at `https://github.com/GabrielDWu/piecewise-constant-objectives`.

6.6 Results

6.6.1 Training on Girard Accuracy ($n = 3$)

We first investigate the effect of applying gradient ascent directly against Girard Accuracy, which is only feasible when $n = 3$. Figure 14 shows that on the $n = 3, d = 3$ model, training against the true accuracy improves it by 1.58 percentage points from 83.8% to 85.4%.[42] In comparison, training on SS loss is only able to improve accuracy by 0.93 percentage points, and Hook loss is unable to increase the accuracy beyond the CE baseline.

Training against Girard Accuracy for $n = 3, d = 2$ is initially less effective, but a small tweak in the

[41] The learning rate $x \in [0, 1]$ of the way through training is $10^{-2} d^{-1/2} \cdot \underbrace{0.01^{\min(2x,1)}}_{\text{decay}} \cdot \underbrace{\min(10x, 1)}_{\text{warm-up}}$, unless otherwise specified.

[42] However, training is quite unstable. Figure 14 only shows the first 297 gradient steps. If we continue training, the accuracy quickly drops down and 83.1% and takes a while to recover.

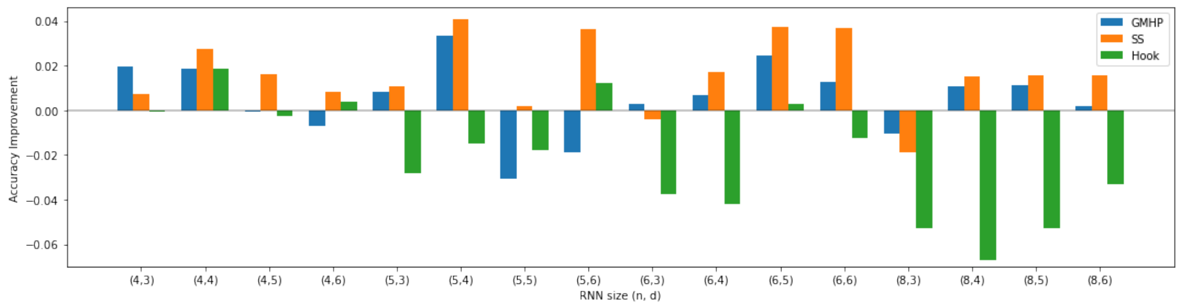


Figure 15: Change in accuracy incurred by training against the GMHP, SS, and Hook objectives, compared to the cross-entropy baseline.

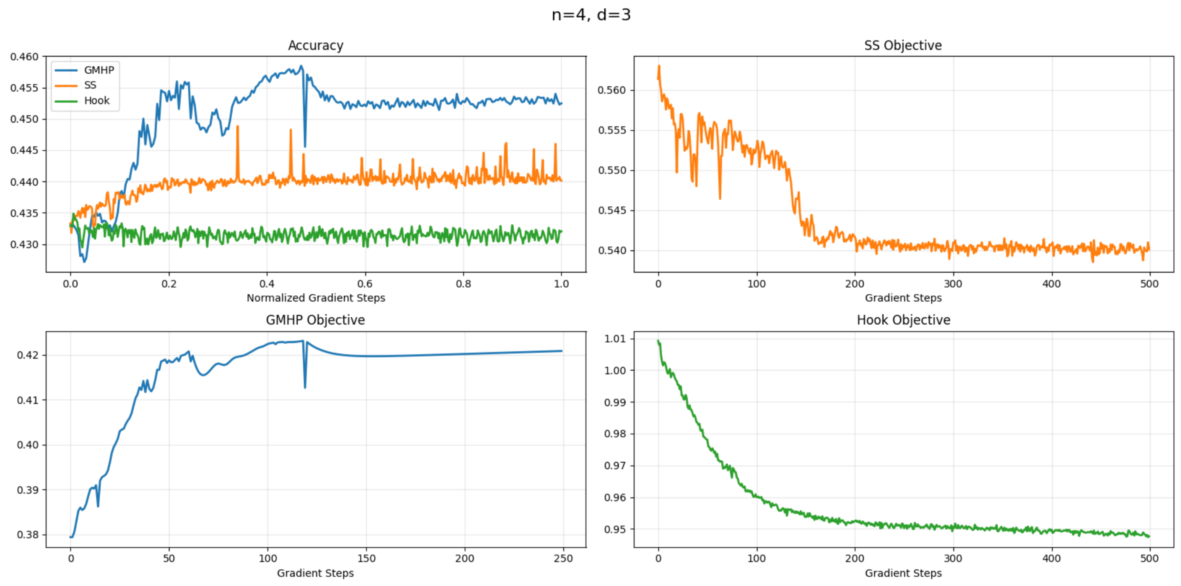


Figure 16: A side-by-side comparison of the GMHP, SS, and Hook training runs on the $n = 4, d = 3$ model.

algorithm improves it by a lot. See Appendix I for more discussion.

6.6.2 Comparison of other methods ($n \geq 4$)

Figure 15 presents the effects of training on each of the objectives for $n \geq 4$. There is a fair amount of variation across different model sizes. SS loss is usually the objective that incentivizes the highest accuracy, although it is occasionally outperformed by GMHP. Both usually beat the cross-entropy baseline, which usually beats Hook loss. The absolute accuracies of the cross-entropy baselines are listed in Table 6.

Figure 16 shows the $n = 4, d = 3$ model's accuracy over the course of training for each objective. GMHP is the best training proxy for accuracy on this model, as evidenced by the accuracy curve closely following the shape of the GMHP objective curve. This is true even though GMHP itself is not a particularly good absolute estimator for accuracy — it consistently underestimates the accuracy by many percentage points. See Appendix M for loss curves of all 16 models.

More experiments can be found in the appendices. Briefly: Appendix J explores the effect of changing the pruning parameter C for GMHP, Appendix K demonstrates that the CE-trained baseline models have been trained to convergence, and Appendix L shows the distribution of region sizes in Equation 4.

6.7 Discussion

6.7.1 Limitations

Our current setting has many limitations. First, we only work with tiny models that have fewer than 100 parameters. It would be more exciting to see deductive estimation methods applied to bigger models. Second, the algorithms for Girard Accuracy and Gaussian Mixture Half-space Pruning are somewhat bespoke to the second-argmax task we chose to analyze. It is not immediately clear how to generalize these techniques to other tasks where the ground-truth classification boundaries are more complicated than linear inequalities on $\boldsymbol{x}$.

Finally, a prerequisite to creating a deductive estimation for neural networks is having a mathematical formalization of the quantity of interest. In the supervised learning setting, this means we must have a formal understanding of the distribution of labeled pairs (x, y). In our work, this was easy — the input distribution was a standard Gaussian, and the label y was a simple algorithmic function of x — but this assumption quickly breaks down in more interesting settings like "classify pictures of cats" or "predict the next token of internet text."

However, there may be creative ways to get around this problem. For example, say we want to train a neural network M_θ to achieve the maximum possible accuracy at classifying cat breeds. We don't have a formal definition of this problem, but we could train a generative model G that takes in a Gaussian-distributed latent $\boldsymbol{z}$ and outputs a labeled picture of a cat, e.g., with a diffusion model. Denote the output of G by $G(\boldsymbol{z}) = (G_x(\boldsymbol{z}), G_y(\boldsymbol{z}))$, where G_x is an image and G_y is the label. Then the quantity

$$\widetilde{\text{Acc}}(\theta) = \Pr_{\boldsymbol{z} \sim \mathcal{N}(0, \text{Id}_d)} [M_\theta(G_x(\boldsymbol{z})) = G_y(\boldsymbol{z})]$$

is formally defined, so we could train θ with gradient ascent against a deductive estimator for $\widetilde{\text{Acc}}(\theta)$. As long as the distribution outputted by G is sufficiently close to the actual data distribution, this allows us to improve the accuracy of M_θ on the real-world distribution of cats. This idea is closely related to *synthetic data generation*, which is already widely used to train modern AI systems, although a sampling-based method is usually used for optimization instead of a deductive estimate [BTS+24].[43]

6.7.2 Related work

To our knowledge, this work is the first to study deductive accuracy estimation for neural networks. However, the central problem addressed in this section — that the accuracy of a classifier over a finite dataset cannot be directly optimized with gradient ascent — has been thoroughly studied from theoretical and empirical angles. For example, [PLBM06] discusses how the non-convexity of the 0–1 loss motivates the use of surrogate losses. They analyze the relationship between 0–1 accuracy (which we simply call "accuracy") and a general surrogate loss function, proving upper bounds on the decrease in accuracy incurred by maximizing a surrogate objective instead of accuracy itself.

Besides cross-entropy loss, some common surrogates for accuracy are Hinge loss, $\mathbf{L}_1$ loss, and $\mathbf{L}_2$ loss. [JC17] compares these objectives alongside many other sampling-based losses in the context of training MNIST classifiers. Note that their definition of Hinge loss is very similar to our Hook loss.

When a computational graph features non-differentiable sampling steps of a categorical random variable, these steps can be replaced with a differentiable relaxation called the Gumbel-Softmax distribution [JGP17]. For example, it can be used to optimize the expectation of a trajectory sampled from a policy during reinforcement learning without relying using the score function estimator. The Gumbel-Softmax is not immediately applicable to our setting, though it has a similar flavor to Sigmoid Separation loss: at low temperatures, Gumbel-Softmax approximates the categorical distribution in

[43]This setup may be especially useful in domains where discrimination (training M) is harder than generation (training G). There are theoretical reasons to expect this to be true in some settings; for example, it is much harder to find a planted clique in a graph than it is to sample from the distribution of random graphs with planted cliques.

the same way that SS loss approximates 0-1 loss.

In general, maximizing the 0-1 accuracy of a model on a dataset of size n is **NP**-hard, even when the model is a linear classifier. Despite this, [NS13] introduce practical algorithms for this task, including "branch and bound" methods and combinatorial search. Their work focuses on linear binary classifiers.

Finally, [Rob21] decomposes a neural network with ReLU activations into piecewise-affine regions, with the aim of transforming it into an interpretable decision tree. Like GMHP, they prune the smallest regions along the way to prevent exponential blowup.

6.7.3 Conclusion

In this work, we have established that deductive estimators can be used to increase the accuracy of small neural networks beyond what is traditionally possible with cross-entropy loss. Occasionally, this also beats other sampling-based objectives like Sigmoid Separation loss specifically designed to give a smooth approximation to the accuracy. I view this as a promising early sign that the theory of deductive estimation can provide value in empirical contexts, which is good news for the ambitious applications to AI alignment laid out in Section 3.

Directions for future work include improving the effectiveness of GMHP, perhaps by using a class of distributional representations better suited for modeling the intersection of half-space constraints. We could also explore more sophisticated methods for propagating distributions through half-spaces than simply tracking conditional covariances. For example, we may want information about the constraints to be able to travel in both directions rather than only forwards. This motivates notions of distributional divergence that differentially incentivize matching the ground-truth distribution in regions that will end up being important for the final estimate, in contrast with the our current implicit use of KL divergence via moment matching (which cares about the fidelity of the representation equally across all of $\mathbb{R}^d$).

More broadly, we would be excited to see future research that designs deductive estimators for a wide range of algorithmic tasks beyond the second-argmax. We also hope to see new ways of validating the usefulness of such estimators. In our experiments, this involved using the gradient information provided by the estimate to train models more effectively, but future validation methods could involve other tests such as comparing the accuracy of the deductive estimator (over some distribution of models) against a sampling-based estimator that uses the same computational budget.

7 Conclusion and Future Work

In this thesis, I have presented deductive estimation as a new class of techniques for understanding and training neural networks. In particular, I introduced a family of methods called Deduction-Projection Estimators that track the internal states of a computation (e.g., the distributions of activations in a neural network) by alternating between logically sound "deduction steps" and approximate "projection steps." This builds on the philosophical picture of heuristic estimators laid out by [CNX22, CHL+24].

Unlike traditional methods in machine learning, deductive estimation never requires us to generalize from observing model behaviors on a small set of random inputs to making claims about an entire input distribution. Instead, it challenges us to build a bottom-up explanation of how a neural network functions, which could be useful for controlling the worst-case behavior of models in contexts like low probability estimation or mechanistic anomaly detection. Deductive estimators distinguish themselves from the standard approach to mechanistic interpretability because they aspire to be purely algorithmic: humans do not need to understand the explanations any more than they understand the models themselves. Our hope that such explanations exist comes from intuitions around the No-Coincidence Principle: the informal conjecture that any outrageously surprising mathematical fact has a satisfying explanation [Gow19].

In the applied section of this thesis, we compared DPE-based approaches against sampling-based approaches in two different machine learning settings. Our first setting studied low probability estimation in language models, in which we found that DPE-inspired activation extrapolation methods do not perform as well as our importance sampling estimators (which we designed with inspiration from modern red-teaming techniques like [ZZC+24]), but are still able to beat simple baselines. In our second setting — training small recurrent neural networks for maximal accuracy on an algorithmic task — we showed that Gaussian Mixture Half-space Pruning, a DPE that leverages covariance propagation, can often outperform the widely-used cross-entropy loss.

Given the preliminary nature of this line of inquiry, there are many exciting directions for future research. On the philosophical side, we could benefit from more clarity on the definition and desired properties of deductive estimation. From the perspective of theoretical computer science, we have Conjecture 4.1 as a concrete framing of the No-Coincidence Principle. In machine learning, open questions include: Can we create DPEs for transformers that outperform sampling-based estimators at the low probability estimation task defined in Section 5? What do deductive estimators look like for small neural networks trained on tasks besides second-argmax? Can we apply any of these techniques to beat baselines at empirical mechanistic anomaly detection?

Finally, given the rapid pace of improvement in AI capabilities, we believe that the task of applying deductive estimation methods to solve problems in AI alignment is interesting, important, and timely.

References

[Aar17] Scott Aaronson. P=?NP, 2017. https://eccc.weizmann.ac.il/report/2017/004/.

[B$^+$25] Yoshua Bengio et al. International AI Safety Report: The International Scientific Report on the Safety of Advanced AI. https://www.gov.uk/government/publications/international-ai-safety-report-2025, 2025.

[Bak07] Alan Baker. Is there a problem of induction for mathematics? In Mary Leng, Alexander Paseau, and Michael D. Potter, editors, *Mathematical Knowledge*, pages 57–71. Oxford University Press, 2007.

[Bar25] Matthew Barnett. AGI could drive wages below subsistence level. https://epoch.ai/gradient-updates/agi-could-drive-wages-below-subsistence-level, 2025.

[BBC$^+$12] David H. Bailey, Jonathan M. Borwein, Cristian S. Calude, Michael J. Dinneen, Monica Dumitrescu, and Alex Yee. An Empirical Approach to the Normality of Pi. *Experimental Mathematics*, 21(4):375–384, 2012.

[BBK$^+$19] Andreas Buja, Lawrence Brown, Arun Kumar Kuchibhotla, Richard Berk, Ed George, and Linda Zhao. Models as Approximations II: A Model-Free Theory of Parametric Regression. https://arxiv.org/abs/1612.03257, 2019.

[BG24] Leonard Bereska and Efstratios Gavves. Mechanistic interpretability for ai safety – a review. https://arxiv.org/abs/2404.14082, 2024.

[BGI$^+$12] Boaz Barak, Oded Goldreich, Russell Impagliazzo, Steven Rudich, Amit Sahai, Salil Vadhan, and Ke Yang. On the (im)possibility of obfuscating programs. *J. ACM*, 59(2), May 2012.

[BLZ$^+$21] Tao Bai, Jinqi Luo, Jun Zhao, Bihan Wen, and Qian Wang. Recent Advances in Adversarial Training for Adversarial Robustness. https://arxiv.org/abs/2102.01356, 2021.

[BT21] Andrej Bogdanov and Luca Trevisan. Average-Case Complexity. https://arxiv.org/abs/cs/0606037, 2021.

[BTS$^+$24] André Bauer, Simon Trapp, Michael Stenger, Robert Leppich, Samuel Kounev, Mark Leznik, Kyle Chard, and Ian Foster. Comprehensive exploration of synthetic data generation: A survey. https://arxiv.org/abs/2401.02524, 2024.

[Car23] Joe Carlsmith. Scheming AIs: Will AIs fake alignment during training in order to get power? https://arxiv.org/abs/2311.08379, 2023.

[CER$^+$23] Hoagy Cunningham, Aidan Ewart, Logan Riggs, Robert Huben, and Lee Sharkey. Sparse Autoencoders Find Highly Interpretable Features in Language Models. https://arxiv.org/abs/2309.08600, 2023.

[CH19] F Cornu and H J Hilhorst. Density decay and growth of correlations in the game of life. *Journal of Statistical Mechanics: Theory and Experiment*, 2019(1):013212, Jan 2019.

[CHL$^+$24] Paul Christiano, Jacob Hilton, Andrea Lincoln, Eric Neyman, and Mark Xu. Towards a law of iterated expectations for heuristic estimators. https://arxiv.org/abs/2410.01290, 2024.

[Chr22] Paul Christiano. Crispness of Corrigibility. LessWrong comment, available at https://www.lesswrong.com/posts/AqsjZwxHNqH64C2b6/let-s-see-you-write-that-corrigibility-tag?commentId=8kPhqBc69HtmZj6XR, 2022.

[Chr23] Paul Christiano. Conjecture on proof obfuscation, 2023. Facebook post, available at https://www.facebook.com/paulfchristiano/posts/pfbid02eEm5da6GEdqpJxtWi88e2cmKXNBVWUzrR27NZgNfMNfB8wkTetQeVt6JhETV6yiXl.

[CMR24] Paul Christiano, David Matolcsi, and George Robinson. Analytically learning variational auto-encoders. https://www.alignment.org/content/files/2024/09/Analytically_Learning_VAEs.pdf, 2024.

[CNCC$^+$24] Nicholas Carlini, Milad Nasr, Christopher A Choquette-Choo, Matthew Jagielski, Irena Gao, Pang Wei W Koh, Daphne Ippolito, Florian Tramer, and Ludwig Schmidt. Are aligned neural networks adversarially aligned? *Advances in Neural Information Processing Systems*, 36, 2024.

[CNX22] Paul Christiano, Eric Neyman, and Mark Xu. Formalizing the presumption of independence. https://arxiv.org/abs/2211.06738, 2022.

[Cod69] W. J. Cody. Rational chebyshev approximations for the error function. *Mathematics of Computation*, 23(107):631–637, 1969.

[Cra36] Harald Cramér. On the order of magnitude of the difference between consecutive prime numbers. *Acta Arithmetica*, 2(1):23–46, 1936.

[CS02] Koby Crammer and Yoram Singer. On the algorithmic implementation of multiclass kernel-based vector machines. *The Journal of Machine Learning Research*, 2:265–292, March 2002.

[CSPHM24] Stephen Casper, Lennart Schulze, Oam Patel, and Dylan Hadfield-Menell. Defending Against Unforeseen Failure Modes with Latent Adversarial Training. https://arxiv.org/abs/2403.05030, 2024.

[CXC21] Paul Christiano, Mark Xu, and Ajeya Cotra. Eliciting latent knowledge: How to tell if your eyes deceive you. https://www.alignmentforum.org/posts/qHCDysDnvhteW7kRd/arc-s-first-technical-report-eliciting-latent-knowledge, 2021.

[DW24] Bill Drexel and Caleb Withers. AI and the Evolution of Biological National Security Risks: Capabilities, Thresholds, and Interventions. Technical report, Center for a New American Security, Washington, DC, 2024.

[EHO$^+$22] Nelson Elhage, Tristan Hume, Catherine Olsson, Nicholas Schiefer, Tom Henighan, Shauna Kravec, Zac Hatfield-Dodds, Robert Lasenby, Dawn Drain, Carol Chen, Roger Grosse, Sam McCandlish, Jared Kaplan, Dario Amodei, Martin Wattenberg, and Christopher Olah. Toy Models of Superposition. https://arxiv.org/abs/2209.10652, 2022.

[Epo24] Epoch AI. Data on machine learning hardware, 2024. Accessed: 2025-02-17.

[EU71] Paul Erdös and Stanisław Ulam. Some probabilistic remarks on Fermat's Last Theorem. *The Rocky Mountain Journal of Mathematics*, 1(4):613–616, 1971.

[FD24] Carl Franzen and Emilia David. OpenAI confirms new frontier models o3 and o3-mini. https://venturebeat.com/ai/openai-confirms-new-frontier-models-o3-and-o3-mini/, 2024.

[Fei02] Uriel Feige. Relations between average case complexity and approximation complexity. In *Proceedings of the Thiry-Fourth Annual ACM Symposium on Theory of Computing*, STOC '02, page 534–543, New York, NY, USA, 2002. Association for Computing Machinery.

[Fit24] Allison Fitisone. *Solid Angle Measure Approximation Methods for Polyhedral Cones*. PhD thesis, University of Kentucky, 2024.

[GB09] A. Genz and F. Bretz. *Computation of Multivariate Normal and t Probabilities*. Lecture Notes in Statistics. Springer Berlin Heidelberg, 2009.

[GDW$^+$24] Ryan Greenblatt, Carson Denison, Benjamin Wright, Fabien Roger, Monte MacDiarmid, Sam Marks, Johannes Treutlein, Tim Belonax, Jack Chen, David Duvenaud, Akbir Khan, Julian Michael, Sören Mindermann, Ethan Perez, Linda Petrini, Jonathan Uesato, Jared Kaplan, Buck Shlegeris, Samuel R. Bowman, and Evan Hubinger. Alignment faking in large language models. https://arxiv.org/abs/2412.14093, 2024.

[GHKO25] William Gay, William He, Nicholas Kocurek, and Ryan O'Donnell. Pseudorandomness Properties of Random Reversible Circuits. https://arxiv.org/abs/2502.07159, 2025.

[GKVZ24] Shafi Goldwasser, Michael P. Kim, Vinod Vaikuntanathan, and Or Zamir. Planting Undetectable Backdoors in Machine Learning Models. https://arxiv.org/abs/2204.06974, 2024.

[Göd92] Kurt Gödel. *On formally undecidable propositions of Principia mathematica and related systems*. Dover Publications, Inc., New York, 1992. Translated from the German and with a preface by B. Meltzer, With an introduction by R. B. Braithwaite, Reprint of the 1963 translation.

[GOS06] Jens Groth, Rafail Ostrovsky, and Amit Sahai. Non-interactive Zaps and New Techniques for NIZK. In *Advances in Cryptology - CRYPTO 2006*, pages 97–111, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.

[Gow19] Timothy Gowers. What makes mathematicians believe unproved mathematical statements? https://api.semanticscholar.org/CorpusID:263739808, 2019.

[GPR67] L.G. Gubin, Boris Polyak, and E.V. Raik. The Method of Projections for Finding the Common Point of Convex Sets. *USSR Computational Mathematics and Mathematical Physics*, 7:1–24, 12 1967.

[GSS14] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572*, 2014.

[GSSR24] Ryan Greenblatt, Buck Shlegeris, Kshitij Sachan, and Fabien Roger. AI Control: Improving Safety Despite Intentional Subversion. https://arxiv.org/abs/2312.06942, 2024.

[GT08] Ben Green and Terence Tao. The primes contain arbitrarily long arithmetic progressions. *Annals of Mathematics*, 167(2):481–547, 2008.

[HDM$^+$24] Evan Hubinger, Carson Denison, Jesse Mu, Mike Lambert, Meg Tong, Monte MacDiarmid, Tamera Lanham, Daniel M. Ziegler, Tim Maxwell, Newton Cheng, Adam Jermyn, Amanda Askell, Ansh Radhakrishnan, Cem Anil, David Duvenaud, Deep Ganguli, Fazl Barez, Jack Clark, Kamal Ndousse, Kshitij Sachan, Michael Sellitto, Mrinank Sharma, Nova DasSarma, Roger Grosse, Shauna Kravec, Yuntao Bai, Zachary Witten, Marina Favaro, Jan Brauner, Holden Karnofsky, Paul Christiano, Samuel R. Bowman, Logan Graham, Jared Kaplan, Sören Mindermann, Ryan Greenblatt, Buck Shlegeris, Nicholas Schiefer, and Ethan Perez. Sleeper Agents: Training Deceptive LLMs that Persist Through Safety Training. https://arxiv.org/abs/2401.05566, 2024.

[HG23] Dan Hendrycks and Kevin Gimpel. Gaussian Error Linear Units (GELUs). https://arxiv.org/abs/1606.08415, 2023.

[HILL99] Johan Håstad, Russell Impagliazzo, Leonid A. Levin, and Michael Luby. A pseudorandom generator from any one-way function. *SIAM Journal on Computing*, 28(4):1364–1396, 1999.

[HPP+24] Axel Højmark, Govind Pimpale, Arjun Panickssery, Marius Hobbhahn, and Jérémy Scheurer. Analyzing Probabilistic Methods for Evaluating Agent Capabilities. https://arxiv.org/abs/2409.16125, 2024.

[HvMM+21] Evan Hubinger, Chris van Merwijk, Vladimir Mikulik, Joar Skalse, and Scott Garrabrant. Risks from Learned Optimization in Advanced Machine Learning Systems. https://arxiv.org/abs/1906.01820, 2021.

[Imp95] Russell Impagliazzo. A personal view of average-case complexity. In *Proceedings of Structure in Complexity Theory. Tenth Annual IEEE Conference*, pages 134–147, 1995.

[IS68] F. Itakura and S. Saito. Analysis synthesis telephony based on the maximum likelihood method. In *Proc. 6th of the International Congress on Acoustics*, pages C–17–C–20, Los Alamitos, CA, 1968. IEEE.

[IST+19] Andrew Ilyas, Shibani Santurkar, Dimitris Tsipras, Logan Engstrom, Brandon Tran, and Aleksander Madry. Adversarial Examples Are Not Bugs, They Are Features. https://arxiv.org/abs/1905.02175, 2019.

[JC17] Katarzyna Janocha and Wojciech Marian Czarnecki. On loss functions for deep neural networks in classification. https://arxiv.org/abs/1702.05659, 2017.

[JGP17] Eric Jang, Shixiang Gu, and Ben Poole. Categorical reparameterization with gumbel-softmax. https://arxiv.org/abs/1611.01144, 2017.

[JLS20] Aayush Jain, Huijia Lin, and Amit Sahai. Indistinguishability Obfuscation from Well-Founded Assumptions. https://arxiv.org/abs/2008.09317, 2020.

[JYW+24] Carlos E Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik R Narasimhan. SWE-bench: Can language models resolve real-world github issues? In *The Twelfth International Conference on Learning Representations*, 2024.

[KB17] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. https://arxiv.org/abs/1412.6980, 2017.

[KSB+19] Daniel Kang, Yi Sun, Tom Brown, Dan Hendrycks, and Jacob Steinhardt. Transfer of adversarial robustness between perturbation types. *arXiv preprint arXiv:1905.01034*, 2019.

[LCH+20] Xiaodong Liu, Hao Cheng, Pengcheng He, Weizhu Chen, Yu Wang, Hoifung Poon, and Jianfeng Gao. Adversarial Training for Large Neural Language Models. https://arxiv.org/abs/2004.08994, 2020.

[LZXGM23] Alexander K. Lew, Tan Zhi-Xuan, Gabriel Grand, and Vikash K. Mansinghka. Sequential Monte Carlo Steering of Large Language Models using Probabilistic Programs. https://arxiv.org/abs/2306.03081, 2023.

[Mad17] Aleksander Madry. Towards deep learning models resistant to adversarial attacks. *arXiv preprint arXiv:1706.06083*, 2017.

[MDMM11] J Machta, S DeDeo, S Mertens, and C Moore. Parallel complexity of random Boolean circuits. *Journal of Statistical Mechanics: Theory and Experiment*, 2011(04):P04015, 2011.

[MET25] METR. Measuring ai ability to complete long tasks. `https://metr.org/blog/2025-03-19-measuring-ai-ability-to-complete-long-tasks/`, 03 2025.

[MMZ23] Anqi Mao, Mehryar Mohri, and Yutao Zhong. Cross-Entropy Loss Functions: Theoretical Analysis and Applications. https://arxiv.org/abs/2304.07288, 2023.

[MRM$^+$24] Samuel Marks, Can Rager, Eric J. Michaud, Yonatan Belinkov, David Bau, and Aaron Mueller. Sparse Feature Circuits: Discovering and Editing Interpretable Causal Graphs in Language Models. https://arxiv.org/abs/2403.19647, 2024.

[MZ02] Marc Mézard and Riccardo Zecchina. Random k-satisfiability problem: From an analytic solution to an efficient algorithm. *Phys. Rev. E*, 66:056126, Nov 2002.

[NB22] Neel Nanda and Joseph Bloom. TransformerLens. https://github.com/TransformerLensOrg/TransformerLens, 2022.

[Ney24] Eric Neyman. Algorithmic bayesian epistemology. https://arxiv.org/abs/2403.07949, 2024.

[Ney25] Eric Neyman. A computational no-coincidence principle. Blog post, available at https://www.alignment.org/blog/a-computational-no-coincidence-principle/, February 2025.

[NS13] Tan T. Nguyen and Scott Sanner. Algorithms for direct 0-1 loss optimization in binary classification. In *30th International Conference on Machine Learning, ICML 2013*, pages 2122–2130, 2013.

[PAC$^+$24] Mary Phuong, Matthew Aitchison, Elliot Catt, Sarah Cogan, Alexandre Kaskasoli, Victoria Krakovna, David Lindner, Matthew Rahtz, Yannis Assael, Sarah Hodkinson, Heidi Howard, Tom Lieberum, Ramana Kumar, Maria Abi Raad, Albert Webson, Lewis Ho, Sharon Lin, Sebastian Farquhar, Marcus Hutter, Gregoire Deletang, Anian Ruoss, Seliem El-Sayed, Sasha Brown, Anca Dragan, Rohin Shah, Allan Dafoe, and Toby Shevlane. Evaluating Frontier Models for Dangerous Capabilities. https://arxiv.org/abs/2403.13793, 2024.

[PBS22] Alexander Pan, Kush Bhatia, and Jacob Steinhardt. The effects of reward misspecification: Mapping and mitigating misaligned models. https://arxiv.org/abs/2201.03544, 2022.

[PHS$^+$22] Ethan Perez, Saffron Huang, Francis Song, Trevor Cai, Roman Ring, John Aslanides, Amelia Glaese, Nat McAleese, and Geoffrey Irving. Red Teaming Language Models with Language Models. https://arxiv.org/abs/2202.03286, 2022.

[PLBM06] Michael I Jordan Peter L Bartlett and Jon D McAuliffe. Convexity, classification, and risk bounds. *Journal of the American Statistical Association*, 101(473):138–156, 2006.

[Ran04] Ananth Ranganathan. Assumed density filtering, 2004.

[RHS$^+$23] David Rein, Betty Li Hou, Asa Cooper Stickland, Jackson Petty, Richard Yuanzhe Pang, Julien Dirani, Julian Michael, and Samuel R. Bowman. GPQA: A Graduate-Level Google-Proof Q&A Benchmark. https://arxiv.org/abs/2311.12022, 2023.

[Rob16] Christian P. Robert. The Metropolis-Hastings algorithm. https://arxiv.org/abs/1504.01896, 2016.

[Rob21] Haakon Robinson. Approximate piecewise affine decomposition of neural networks. *IFAC-PapersOnLine*, 54(7):541–546, 2021. 19th IFAC Symposium on System Identification SYSID 2021.

[Ros61] S. Rosenbaum. Moments of a truncated bivariate normal distribution. *Journal of the Royal Statistical Society. Series B (Methodological)*, 23(2):405–408, 1961.

[RS94] Michael Rubinstein and Peter Sarnak. Chebyshev's bias. *Experimental Mathematics*, 3(3):173 – 197, 1994.

[RSR⁺23] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer. https://arxiv.org/abs/1910.10683, 2023.

[SB53] Ludwig Schläfli and JJ Burckhardt. On the multiple integral $\int dxdy \dots dz$, whose limits are $p = a_1x + b_1y + \cdots + h_1z > 0, p_2 > 0, \dots, p_n > 0$, and $x^2 + y^2 + \cdots + z^2 < 1$. *Gesammelte Mathematische Abhandlungen*, pages 219–270, 1953.

[SEG⁺24] Abhay Sheshadri, Aidan Ewart, Phillip Guo, Aengus Lynch, Cindy Wu, Vivek Hebbar, Henry Sleight, Asa Cooper Stickland, Ethan Perez, Dylan Hadfield-Menell, and Stephen Casper. Latent Adversarial Training Improves Robustness to Persistent Harmful Behaviors in LLMs. https://arxiv.org/abs/2407.15549, 2024.

[SVK⁺22] Rohin Shah, Vikrant Varma, Ramana Kumar, Mary Phuong, Victoria Krakovna, Jonathan Uesato, and Zac Kenton. Goal Misgeneralization: Why Correct Specifications Aren't Enough For Correct Goals. https://arxiv.org/abs/2210.01790, 2022.

[Sze78] Endre Szemerédi. Regular partitions of graphs. In *Problèmes combinatoires et théorie des graphes (Colloq. Internat. CNRS, Univ. Orsay, 1976)*, volume 260 of *Colloques Internationaux CNRS*, pages 399–401, Paris, 1978. CNRS.

[Tao05a] Terence Tao. The dichotomy between structure and randomness, arithmetic progressions, and the primes. https://arxiv.org/abs/math/0512114, 2005.

[Tao05b] Terence Tao. Szemerédi's regularity lemma revisited. https://arxiv.org/abs/math/0504472, 2005.

[Tao07] Terence Tao. Structure and randomness in combinatorics. https://arxiv.org/abs/0707.4269, 2007.

[Tao12] Terence Tao. The probabilistic heuristic justification of the ABC conjecture, 2012. Blog post, available at https://terrytao.wordpress.com/2012/09/18/the-probabilistic-heuristic-justification-of-the-abc-conjecture/.

[Tao16] Terence Tao. Biases between consecutive primes, 2016. Blog post, available at https://terrytao.wordpress.com/2016/03/14/biases-between-consecutive-primes/.

[TvWW22] Lewis Tunstall, Leandro von Werra, and Thomas Wolf. *Natural Language Processing with Transformers: Building Language Applications with Hugging Face*. O'Reilly Media, Incorporated, 2022.

[Van13] Robert Vanderbei. Girard's Theorem: On a Sphere, Area(Triangle) = Radius2 x Angle-Excess. https://vanderbei.princeton.edu/WebGL/GirardThmProof.html, 2013.

[WCN⁺25] Marcus Williams, Micah Carroll, Adhyyan Narang, Constantin Weisser, Brendan Murphy, and Anca Dragan. On Targeted Manipulation and Deception when Optimizing LLMs for User Feedback. https://arxiv.org/abs/2411.02306, 2025.

[WH25] Gabriel Wu and Jacob Hilton. Estimating the probabilities of rare outputs in language models. https://arxiv.org/abs/2410.13211, 2025.

[WHS24] Alexander Wei, Nika Haghtalab, and Jacob Steinhardt. Jailbroken: How does LLM safety training fail? *Advances in Neural Information Processing Systems*, 36, 2024.

[WRTK19] Stefan Webb, Tom Rainforth, Yee Whye Teh, and M. Pawan Kumar. A Statistical Approach to Assessing Neural Network Robustness. https://arxiv.org/abs/1811.07209, 2019.

[Xu24] Mark Xu. Estimating Tail Risk in Neural Networks. Blog post, available at https://www.alignment.org/blog/estimating-tail-risk-in-neural-networks, 2024.

[YK21] Kevin Yang and Dan Klein. FUDGE: Controlled Text Generation With Future Discriminators. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. Association for Computational Linguistics, 2021.

[ZBMG24] Stephen Zhao, Rob Brekelmans, Alireza Makhzani, and Roger Grosse. Probabilistic Inference in Language Models via Twisted Sequential Monte Carlo. https://arxiv.org/abs/2404.17546, 2024.

[Zha14] Yitang Zhang. Bounded gaps between primes. *Annals of Mathematics*, 179(3):1121–1174, 2014.

[ZSL$^+$23] Hanqing Zhang, Haolin Song, Shaoyu Li, Ming Zhou, and Dawei Song. A Survey of Controllable Text Generation using Transformer-based Pre-trained Language Models. https://arxiv.org/abs/2201.05337, 2023.

[ZWC$^+$23] Andy Zou, Zifan Wang, Nicholas Carlini, Milad Nasr, J. Zico Kolter, and Matt Fredrikson. Universal and Transferable Adversarial Attacks on Aligned Language Models. https://arxiv.org/abs/2307.15043, 2023.

[ZZC$^+$24] Yiran Zhao, Wenyue Zheng, Tianle Cai, Xuan Long Do, Kenji Kawaguchi, Anirudh Goyal, and Michael Shieh. Accelerating Greedy Coordinate Gradient via Probe Sampling. https://arxiv.org/abs/2403.01251, 2024.

A

Principles for choosing a decomposition in QLD

In this section, we justify the claim that we rely on the following two assumptions of the decomposition $\mathbf{u} = \mathbf{a} + \mathbf{b}$ for QLD to perform well: 1) $\mathbf{a}$ and $\mathbf{b}$ are independent, and 2) the contribution towards the output behavior is split roughly equally between the two terms.

The first assumption is straightforward: if $\mathbf{a}$ and $\mathbf{b}$ are not independent, then $\mathbf{a} + \mathbf{b}'$ (where $\mathbf{b}'$ comes from an i.i.d. copy of $\mathbf{u}$) does not have the same distribution as $\mathbf{u}$. The failure of this assumption introduces bias into the estimation method. Working in whitened space ensures that $\mathbf{a}$ and $\mathbf{b}$ are uncorrelated, which is a first step towards independence.

The second assumption—that the contribution to the target behavior is roughly equally split between $\mathbf{a}$ and $\mathbf{b}$—is necessary for QLD to have an advantage over naive sampling. For purposes of illustration, say that $d = 2$ (so that both $\mathbf{a}$ and $\mathbf{b}$ can be treated as scalars), and that $\mathbf{a}, \mathbf{b} \overset{\text{iid}}{\sim} \mathcal{N}(0, 1)$. Then, consider three ways that the contribution to the target behavior could be split:

- *Scenario 1 (no split):* The target token is outputted iff $\mathbf{a} > 10$. In this case, QLD provides no advantage over naive sampling, as the proportion of $(\mathbf{a}^{(i)}, \mathbf{b}^{(j)})$ pairs in the acceptance region is exactly the same as the proportion of $(\mathbf{a}^{(i)}, \mathbf{b}^{(i)})$ pairs. If p is the probability of the behavior, then p^{-1} samples are required to consistently obtain a positive estimate.
- *Scenario 2 (even split):* The target token is outputted iff $\mathbf{a} + \mathbf{b} > 10\sqrt{2}$. In this case, QLD has a quadratic advantage over naive sampling. It only requires around $p^{-1/2}$ samples for QLD to consistently obtain a positive estimate.
- *Scenario 3 (uneven split):* The target token is outputted iff $\mathbf{a} + 2\mathbf{b} > 10\sqrt{5}$. The contribution here is split between $\mathbf{a}$ and $\mathbf{b}$, though not equally, so QLD's efficiency falls in between the previous two scenarios. We can calculate that it requires around $p^{-5/9}$ samples for QLD to consistently obtain a positive estimate.[44]

In practice, the condition for outputting the target token is more complex than a single linear constraint on $\mathbf{a}$ and $\mathbf{b}$. Nevertheless, these examples motivate the idea that the more evenly we can split the contribution between two subspaces, the lower variance our estimator will have.

Given that $\mathbf{b}$ has $d-1$ dimensions while $\mathbf{a}$ only has 1, most choices of $\boldsymbol{d}$ will end up giving $\mathbf{b}$ much more influence over the behavior than $\mathbf{a}$. This motivates us to identify a particularly important direction with $\boldsymbol{d}$; in some informal sense, we want to find a direction that is "$d/2$ times more important" than the average direction.

When we run QLD with many more samples than its standard budget of 2^{16}, its performance improves but plateaus at a level that is still worse than the sampling methods. This shows that QLD is currently limited by a poor independence assumption, as the error arising from an unequal split of contribution should vanish with a sufficient number of samples.

[44]In general, if the condition is $\alpha\mathbf{a} + \sqrt{1-\alpha^2}\mathbf{b} > 10$, it requires roughly $p^{-1/\left(\alpha+\sqrt{1-\alpha^2}\right)^2}$ samples to consistently obtain positive estimates.

B

Computing the Shortest Accepting Vector

Recall that the acceptance region $S \subseteq \mathbb{R}^d$ is the subset of whitened pre-unembed space that results in the model outputting t:

$$S := \left\{ \boldsymbol{u} \in \mathbb{R}^d \,\middle|\, \arg\max_i((\boldsymbol{A}\boldsymbol{u} + \boldsymbol{\mu}) \cdot \boldsymbol{W}_U)_i = t \right\}.$$

We define $\boldsymbol{d}$ to point in the direction of the shortest vector in S, i.e., $\arg\min_{\boldsymbol{u} \in S} \|\boldsymbol{u}\|$. This vector can be approximated using an iterative convex projection algorithm.

$S \subseteq \mathbb{R}^d$ is an intersection of $|\mathcal{V} - 1|$ half-spaces $H_1, \ldots, H_{t-1}, H_{t+1}, \ldots H_{\mathcal{V}}$, where H_i represents the set of all activations (in whitened pre-unembed space) that result in the logit on token t being larger than the logit on token i.

Given any convex set C (such as a half-plane), the *projection* of $\boldsymbol{x} \notin C$ onto C is $\arg\min_{\boldsymbol{x}' \in C} \|\boldsymbol{x} - \boldsymbol{x}'\|_2$. Given a collection of convex sets, there exists a simple algorithm for finding a point in their intersection: start with an arbitrary point $\boldsymbol{x}$, then repeatedly project $\boldsymbol{x}$ onto a random convex set that does not already contain $\boldsymbol{x}$. Eventually, this process converges to a point in their intersection [GPR67].

We apply this method to find an element of S. To ensure that it is the shortest element, we also project $\boldsymbol{x}$ onto balls centered at 0 with smaller and smaller radii by multiplying $\boldsymbol{x}$ by 0.99. The exact procedure is described in Algorithm 5.[45] In practice, it always takes much less than $100 \cdot n_{\text{reps}}$ steps for the algorithm to return a value.

Algorithm 5 Random Constraint Projection

```
Require: Half-spaces $H_1, \ldots, H_{t-1}, H_{t+1}, \ldots, H_{|\mathcal{V}|}$, number of repetitions $n_{\text{reps}}$
 1: $\boldsymbol{x} \leftarrow \mathbf{0} \in \mathbb{R}^d$
 2: for step_cnt $\leftarrow 1$ to $100 \cdot n_{\text{reps}}$ do
 3:     Pick a random $i$ among all $i$ such that $\mathbf{x} \notin H_i$
 4:     Project $\boldsymbol{x}$ onto $H_i$
 5:     if $\boldsymbol{x}$ lies in $S$ (up to some tolerance $\epsilon$) then
 6:         if step_cnt $< n_{\text{reps}}$ then
 7:             Scale $\boldsymbol{x}$ by 0.99.
 8:         else
 9:             return $\boldsymbol{x}$
10:         end if
11:     end if
12: end for
```

[45]We found a few minor bugs in our implementation of the ϵ-tolerance in our algorithm after we ran experiments, but we don't expect them to have affected the results at all.

C

Ground Truth Token Distribution

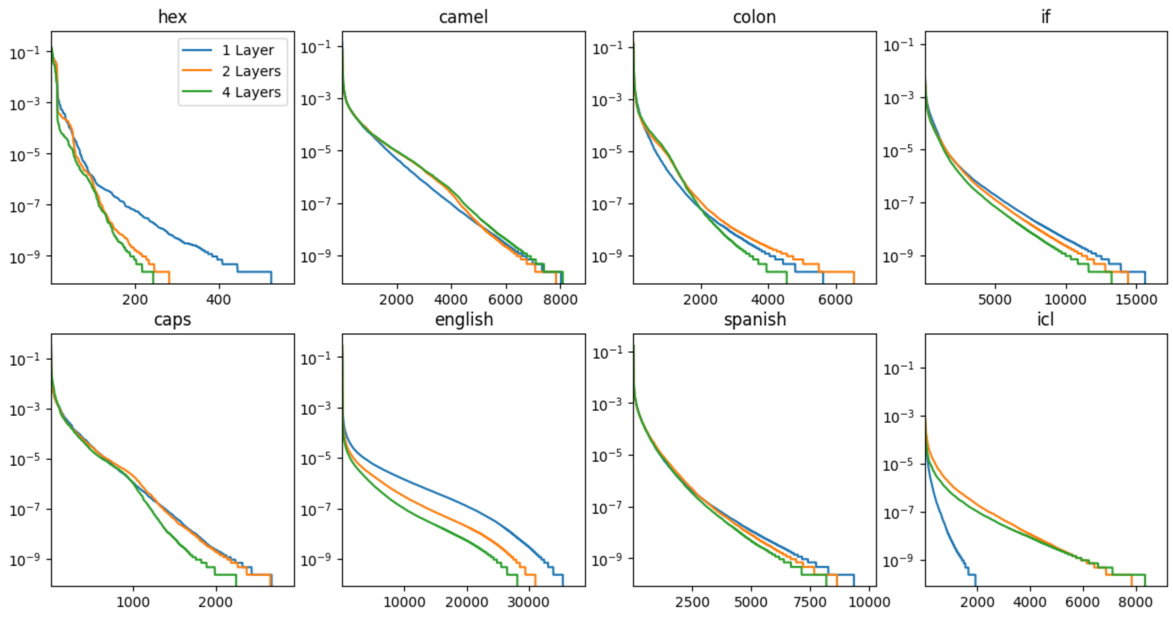


Figure 17: The ground truth probabilities of tokens for each distribution and model size, sorted from most to least probable (the height of the curve at position x is the probability of the x-th most common token). Any tokens that appeared 0 times across all 2^{32} samples are not plotted. The `hex` distribution only had 159 and 135 tokens in the range $[10^{-9}, 10^{-5}]$ for the 2- and 4-layer models, respectively, so we used every such token instead of sampling 256 of them.

D
Computational Budgets

Each estimation method was given a budget of roughly 2^{16} model calls. More specifically:

- Independent Token Gradient Importance Sampling uses 2^8 batches of size 2^8, for a total of 2^{16} samples. The average gradient is updated after each batch. Note that this method requires backward passes as well as forward passes.
- Metropolis–Hastings Importance Sampling uses $2^{10} + 2^{11}$ batches of size 2^5, for a total of $1.5 \cdot 2^{16}$ samples (the batch size indicates the number of independent random walks the method simulates). The first 2^{10} batches are used as a burn-in period for the random walk and are discarded, so only 2^{16} samples are actually used to calculate the estimate.
- Quadratic Logit Decomposition uses $n = 2^{16}$ samples of the pre-unembed activation $\mathbf{v}$. The MLE direction is approximated with $n_{\text{reps}} = 200$ iterations of the Random Constraint Projection algorithm (Appendix *B*); this makes up a trivial fraction of the total compute usage of the method).
- Gaussian Logit Difference uses 2^{16} samples of the logit difference to estimate μ and σ, the mean and standard deviation of the difference between the target logit and the maximum logit. Note that in practice, the μ and σ can be accurately estimated with much fewer than 2^{16} samples.

In practice, ITGIS and MHIS take the longest to test because they require separate samples for each target token. In contrast, QLD reuses the same 2^{16} samples of v for all 256 target tokens associated with a given behavior.

E

All Method Performances

Table 3: Itakura–Saito loss $(p/q - \ln(p/q) - 1)$ comparison of all estimation methods on all input distributions and model sizes.

(a) 1-layer model

Distribution	Constant	GLD	QLD	ITGIS	MHIS
`hex`	2.5891	2.2163	2.0038	2.0484	**1.3960**
`camel`	2.5908	2.4419	2.0648	**1.1997**	2.0187
`colon`	2.7770	2.7091	1.2786	1.2209	**1.0267**
`if`	2.2424	2.1872	1.2321	**1.0916**	1.4455
`caps`	2.6619	2.6147	1.9413	**1.4788**	2.4023
`english`	1.9095	1.8120	1.2409	1.4539	**0.7017**
`spanish`	2.6079	2.4463	**1.5628**	1.6396	2.1538
`icl`	2.5467	2.4328	2.2373	**0.7992**	1.2344
Average	2.4906	2.3575	1.6952	**1.3665**	1.5474

(b) 2-layer model

Distribution	Constant	GLD	QLD	ITGIS	MHIS
`hex`	2.8839	2.8453	2.7698	2.6652	**1.4985**
`camel`	2.3242	2.2268	2.2679	**1.9221**	2.0637
`colon`	2.9893	2.8335	2.1215	2.2986	**1.9042**
`if`	2.6172	2.4377	1.8397	**1.0301**	1.2516
`caps`	2.6345	2.6820	2.4847	1.9271	**1.6981**
`english`	2.0989	2.0956	1.3462	**0.9908**	1.4480
`spanish`	2.5442	2.3662	1.5670	**1.0951**	2.3095
`icl`	2.6381	2.5419	2.3601	1.6420	**1.0502**
Average	2.5913	2.5036	2.0946	1.6964	**1.6530**

(c) 4-layer model

Distribution	Constant	GLD	QLD	ITGIS	MHIS
`hex`	2.4803	2.2934	2.4282	2.3090	**2.0830**
`camel`	2.4895	2.3271	2.4534	**2.0937**	2.2961
`colon`	2.9325	2.7319	2.4382	2.1295	**1.1710**
`if`	2.6477	2.5408	1.8313	1.8430	**0.9261**
`caps`	2.5970	2.5382	2.4142	2.0484	**1.2972**
`english`	2.7008	2.7432	1.5681	**0.9943**	1.3051
`spanish`	2.6415	2.5022	1.7716	**1.6611**	2.1936
`icl`	2.5029	2.1925	2.3866	2.2971	**0.5477**
Average	2.6240	2.4837	2.1615	1.9220	**1.4775**

F

Squared Error in Log-Space

Figure 18 and Table 4 show the method performances when measured using squared error in log-space loss (i.e., $(\log p - \log q)^2$) instead of Itakura–Saito loss. The results are qualitatively identical using either metric. Note that we use separate affine fits to minimize each loss function—in Table 4 we naturally report the results of the fit corresponding to squared error in log-space. However, the importance sampling temperatures are not changed between the two metrics (they were tuned while minimizing Itakura–Saito loss).

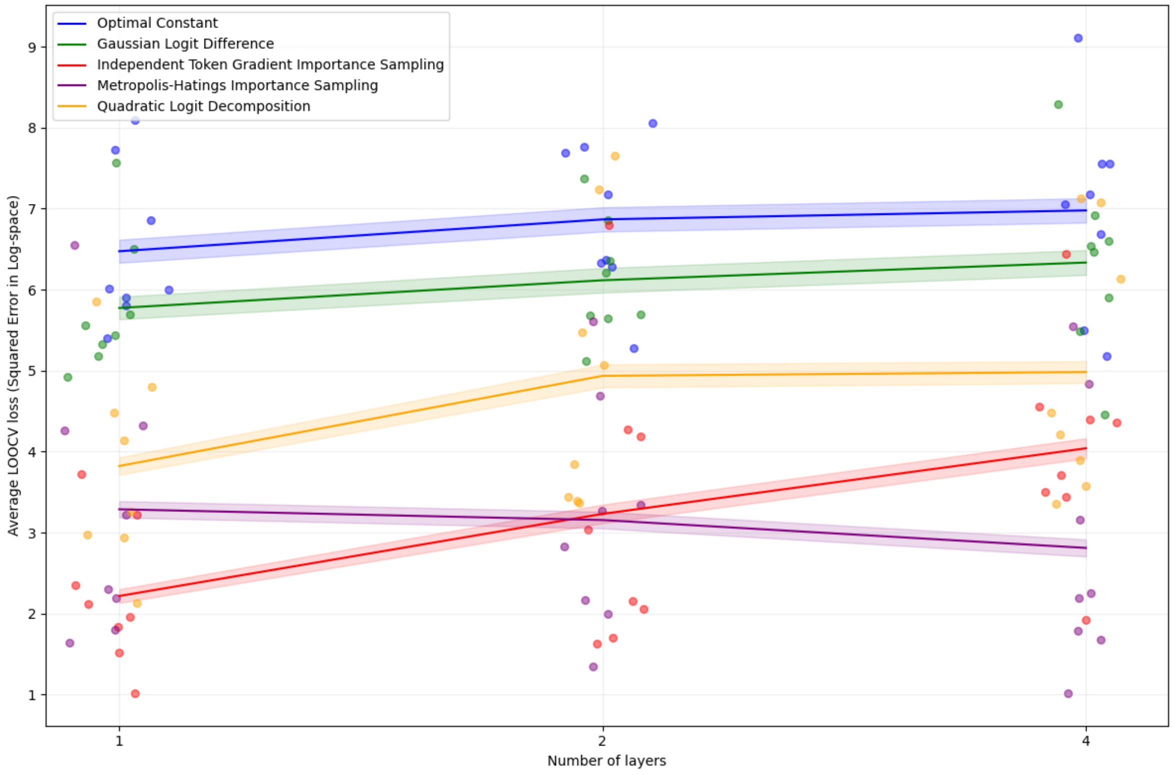


Figure 18: The squared error in log-space loss of all methods across different model sizes. The solid lines indicate the loss of each method averaged over all 8 distributions, with bands indicating standard error. The colored points indicate the loss on individual distributions, with horizontal jitter added for visibility.

Table 4: Squared error in log-space loss comparison of all methods, distributions, and model sizes.

(a) 1-layer model

Distribution	Constant	GLD	QLD	ITGIS	MHIS
`hex`	6.0093	5.1815	4.4863	3.7179	**2.1963**
`camel`	7.7295	6.4975	5.8488	**1.5156**	4.3248
`colon`	6.8554	5.4361	2.1379	2.3549	**1.8061**
`if`	5.3977	4.9287	2.9356	**1.9555**	3.2223
`caps`	8.0927	7.5675	4.1349	**3.2173**	6.5550
`english`	6.0051	5.3220	3.2639	1.8357	**1.6395**
`spanish`	5.7990	5.5649	2.9740	**2.1231**	4.2612
`icl`	5.9035	5.6915	4.8023	**1.0222**	2.3030
Average	6.4740	5.7737	3.8230	**2.2178**	3.2885

(b) 2-layer model

Distribution	Constant	GLD	QLD	ITGIS	MHIS
`hex`	8.0536	7.3674	7.6533	6.8004	**1.9913**
`camel`	7.6849	6.3518	7.2325	**4.2695**	5.6046
`colon`	7.1728	5.6416	3.3924	4.1851	**2.8242**
`if`	6.3346	5.6923	3.3695	**1.7059**	2.1639
`caps`	7.7682	6.8605	5.4771	**3.0411**	3.3432
`english`	5.2742	5.1185	3.4466	**1.6335**	3.2728
`spanish`	6.3718	5.6877	3.8404	**2.1582**	4.6937
`icl`	6.2785	6.2061	5.0740	2.0615	**1.3515**
Average	6.8673	6.1158	4.9357	3.2319	**3.1556**

(c) 4-layer model

Distribution	Constant	GLD	QLD	ITGIS	MHIS
`hex`	7.5559	6.9208	7.0750	6.4364	**3.1605**
`camel`	7.5597	6.5413	7.1263	**3.5067**	5.5501
`colon`	7.1791	6.6035	3.5817	4.3916	**1.7897**
`if`	6.6900	5.9072	3.8987	3.7153	**1.6824**
`caps`	9.1105	8.2957	6.1383	4.3615	**2.2514**
`english`	5.5003	5.4866	3.3561	**1.9180**	2.1938
`spanish`	7.0490	6.4641	4.2083	**3.4425**	4.8425
`icl`	5.1793	4.4569	4.4772	4.5584	**1.0216**
Average	6.9780	6.3345	4.9827	4.0413	**2.8115**

G
Temperature Tuning

Both importance sampling methods require choosing a temperature parameter T. To tune T, we sweep over 9 different temperatures from 0.2 to 5, uniformly spaced in log-space. We choose the value of T that achieves the lowest loss on 100 randomly chosen tokens with ground-truth probabilities in the range $[10^{-5}, 10^{-3}]$ to prevent over-fitting. We tune separate temperatures for each distribution, model size, and importance sampling method, shown in Table 5. It is likely that spending more effort to tune these temperatures (e.g., by tuning on more and rarer tokens) would moderately improve the final performances of the importance sampling methods.

Table 5: Temperatures T used for the different methods.

	1 layer		2 layers		4 layers	
Distribution	ITGIS	MHIS	ITGIS	MHIS	ITGIS	MHIS
`hex`	1.00	0.67	1.50	0.67	5.00	0.67
`camel`	1.00	2.24	1.50	2.24	1.00	2.24
`colon`	1.00	1.00	1.00	1.50	0.67	1.00
`if`	1.00	2.24	0.45	1.50	1.00	1.00
`caps`	1.50	3.34	0.45	1.50	0.67	1.00
`english`	0.45	1.50	0.67	2.24	0.45	1.50
`spanish`	0.67	2.24	0.67	2.24	1.00	2.24
`icl`	0.45	1.00	0.30	0.67	3.34	0.67

H

Plots of Method Outputs

Figures 19, 20, and 21 show the outputs of the four methods on all three model sizes, using log-log plots of ground-truth probability vs method output. All graphs show the outputs after the Itakura–Saito fit has been applied. The horizontal lines of points reveal the value of the additive constant in the fit; any outputs of 0 will all lie on this line after the fit is applied.

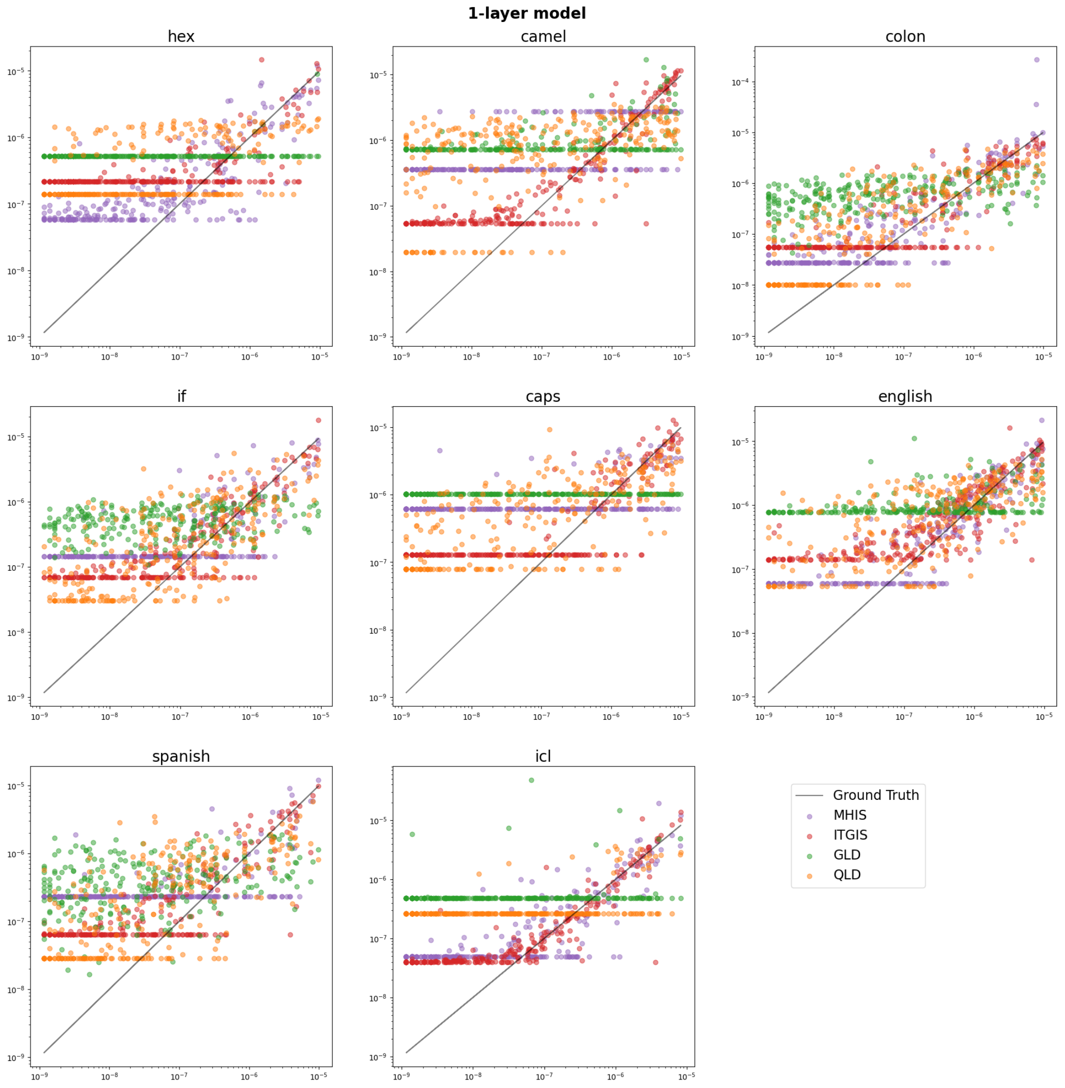


Figure 19: The outputs of methods, after a fit is applied, on all 256 tokens for each distribution on the 1-layer model. The horizontal axis represents the ground-truth token probability, while the vertical axis is the output of the model after a fit.

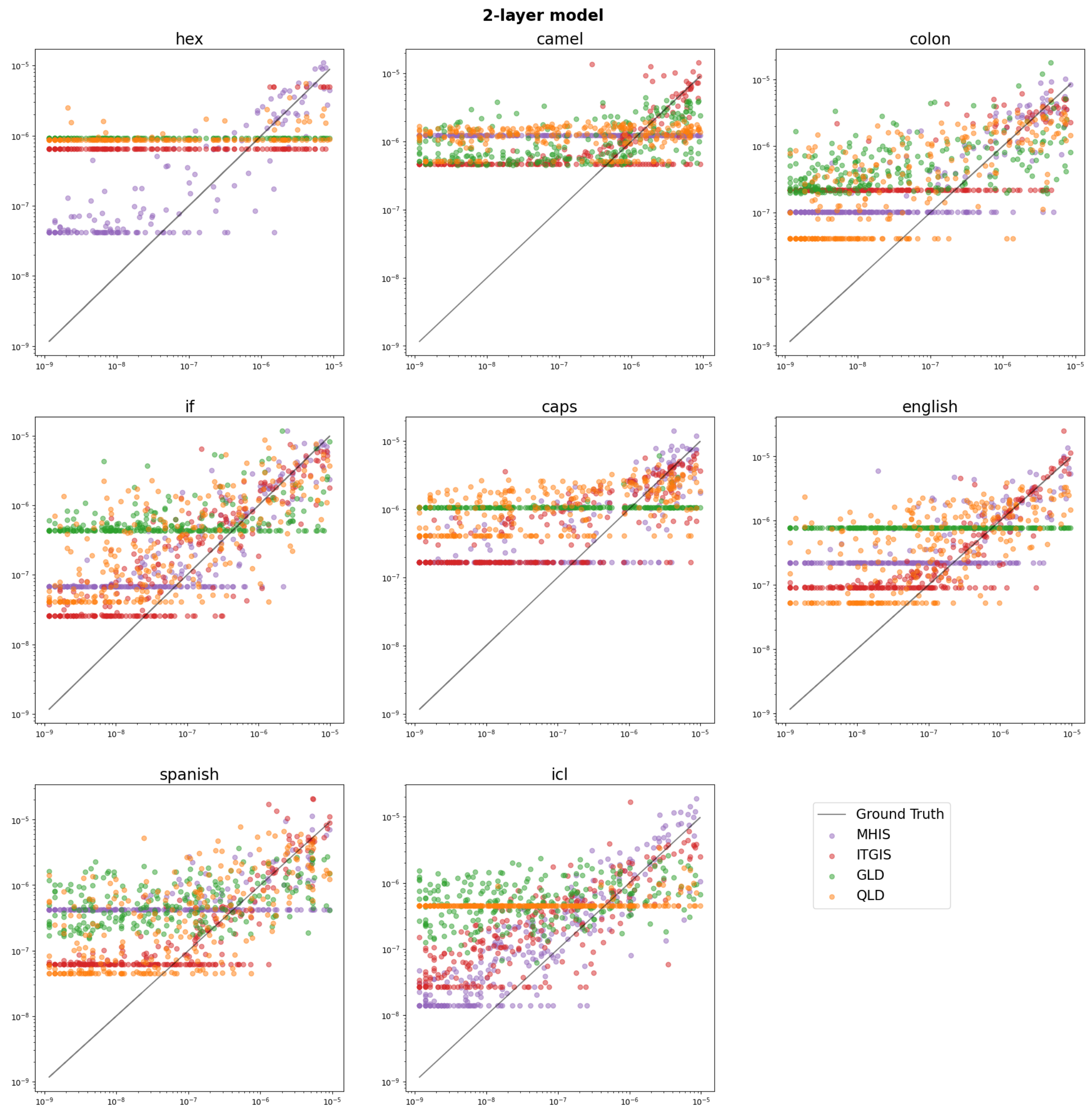


Figure 20: The outputs of methods, after a fit is applied, on all 256 tokens for each distribution on the 2-layer model. The horizontal axis represents the ground-truth token probability, while the vertical axis is the output of the model after a fit.

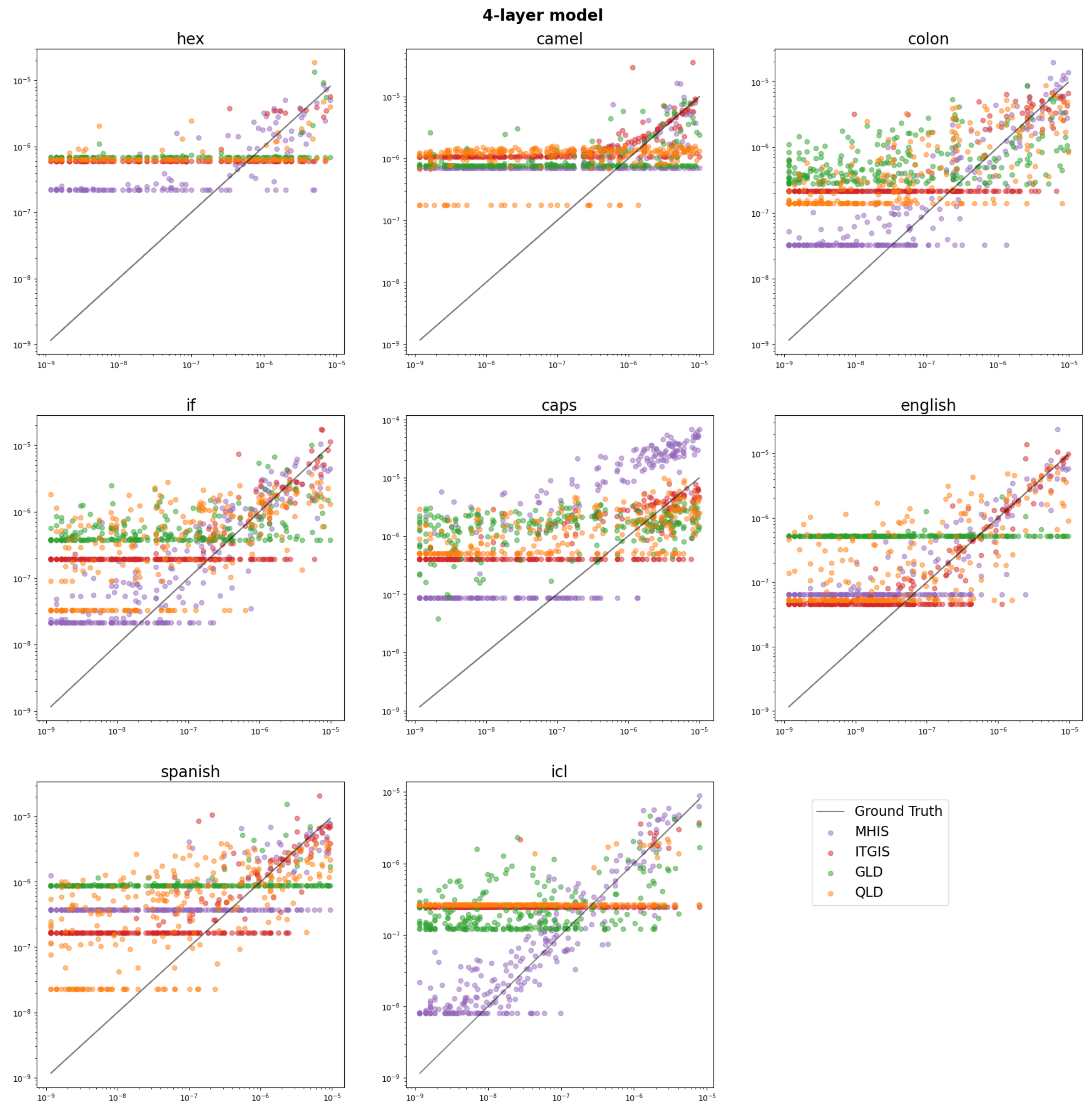


Figure 21: The outputs of methods, after a fit is applied, on all 256 tokens for each distribution on the 4-layer model. The horizontal axis represents the ground-truth token probability, while the vertical axis is the output of the model after a fit.

I

Girard Accuracy for $n = 3, d = 2$

Training directly against Girard Accuracy does not immediately yield impressive results for $d = 2$: the accuracy improves by a mere 0.06%, from 42.52% to 42.58%.

However, we observe an interesting result if we slightly modify the definition of $\text{Acc}(\theta)$. The CE-trained $n = 3, d = 2$ model very frequently outputs 0 for both logits. Whenever this happens, we usually treat this as a "half-accurate" prediction (see footnote 36). We could instead treat this as an *incorrect* prediction; call this new notion of accuracy $\text{Acc}'(\theta)$. Note that there is no particular reason why we should prefer one definition of accuracy over the other; the choice made in this thesis was somewhat arbitrary. The Acc' of the original $n = 3, d = 2$ model is 22.5%, which is significantly lower than its Acc of 42.5%.

We can easily modify our Girard Accuracy algorithm to calculate Acc'. When we train the model against this new accuracy metric, its Acc' **more than doubles from 22.5% to 46.2%**! Such gains are achievable because it is possible to significantly drive down the probability that the model outputs $\mathbf{0}$. See Figure 22 for the training curve.

Notably, the final Acc after training is also 46.2%. This means that training for Acc' achieves a higher Acc than training against Acc itself, by a significant margin (3.6 percentage points, which also handily beats SS and Hook loss). This points to the difficulty of optimization via gradient descent in tiny models — local minima are very hard to avoid when there are only 12 parameters.

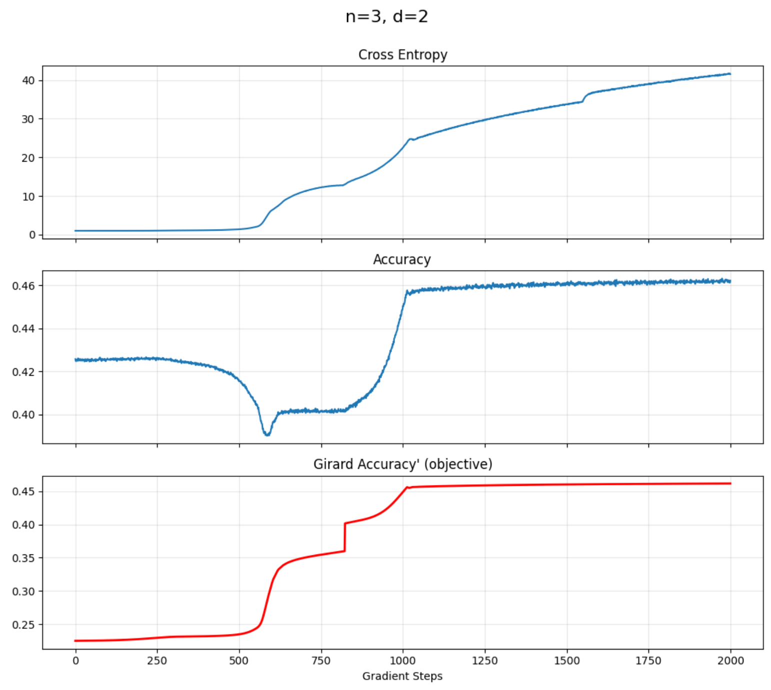


Figure 22: The training trajectory of the $n = 3, d = 2$ model on Acc'. Note that we only train against Acc' (bottom), but we track the CE (top) and a Acc (middle, estimated via samples) along the way. We use a constant learning rate of 5×10^{-3} with a warm-up.

J

The Effect of C on GMHP

Figure 23 shows what happens when we optimize against the Gaussian Mixture Half-space Pruning objective for different values of the pruning parameter C. When we decrease C, it becomes a worse proxy for $\text{Acc}(\theta)$. Intuitively, optimizing solely for the accuracy of the largest regions comes at the cost of hurting the smaller regions.

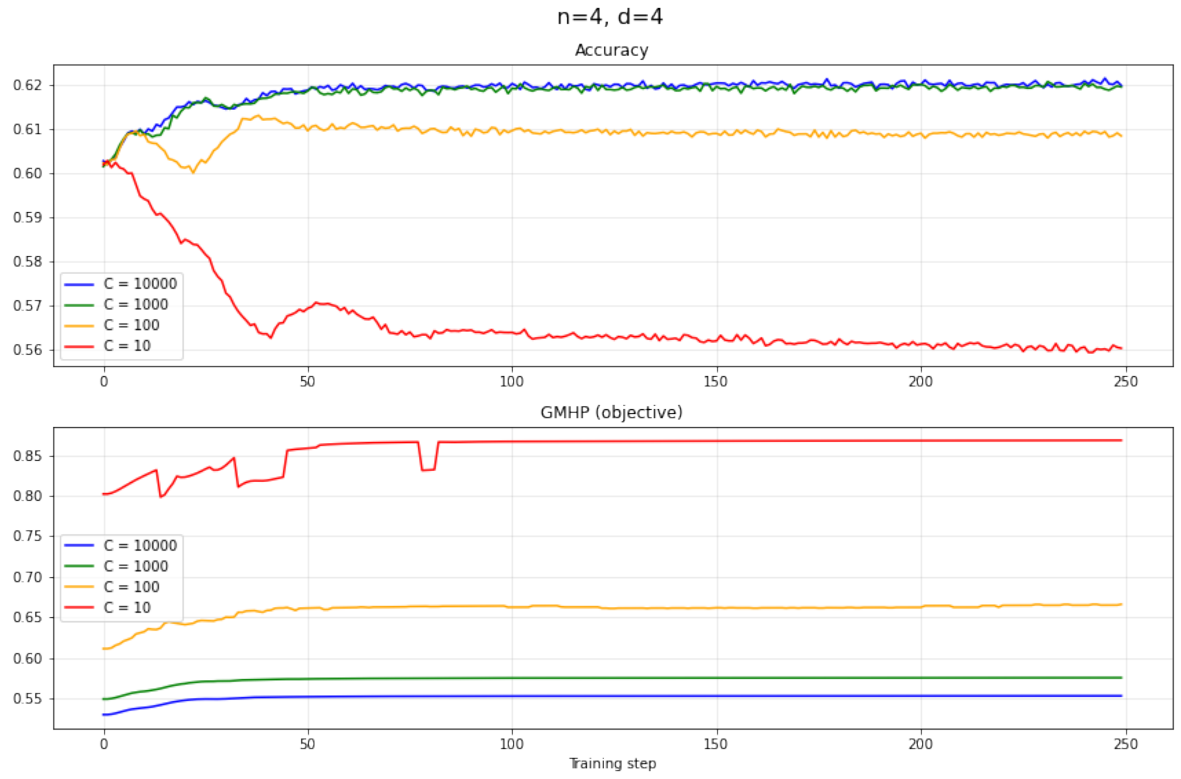


Figure 23: Training against GMHP works better when C is large.

K

Cross-Entropy Loss has Converged

To make sure our comparisons against the accuracy of the "pre-trained" cross-entropy baseline models are fair, we check that continued training on CE loss makes no improvements in either CE or accuracy. Figure 24 shows the results on two of the models. Indeed, even with a batch size of 2^{20} and training for 2500 gradient steps, variation in the sample accuracy between steps is almost completely explained by sampling error. The average sample accuracy over the last 1000 steps compared to the first 1000 steps only increases by 3×10^{-4} (for the $n = 4, d = 3$ model) and 5×10^{-5} (for the $n = 8, d = 5$ model).

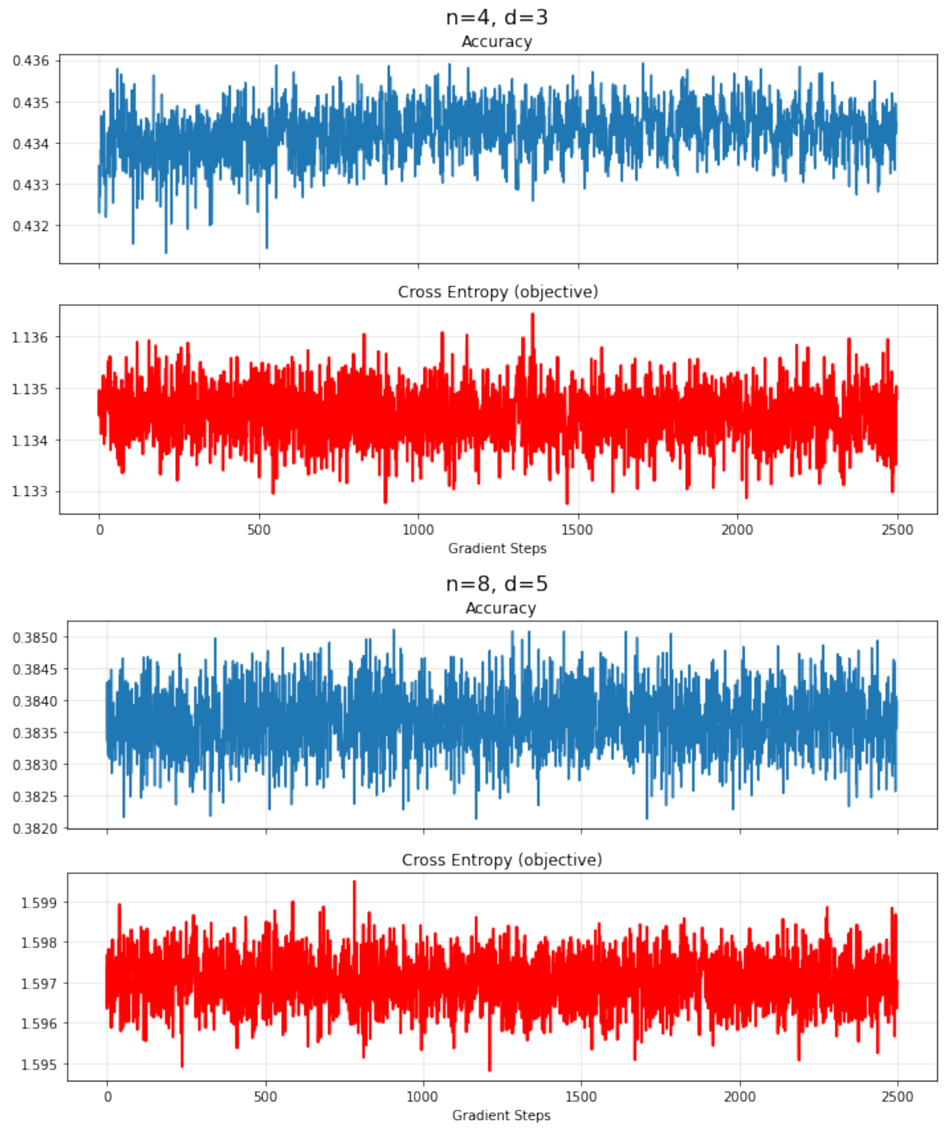


Figure 24: Continued training on cross-entropy loss for the $n = 4, d = 3$ and $n = 8, d = 5$ models.

L
Distribution of Region Sizes

Equation 4 expresses the accuracy of an RNN in terms of the sizes of $2^{nd}n(n-1)$ regions. A natural question to ask is: what does the distribution of the sizes of these regions look like? Figure 25 shows that it tends to be extremely heavy-tailed. For example, when $n = d = 3$, the largest 10 regions account for almost all of the accuracy of the model. In fact, in this model, 97.7% of the regions have size exactly 0 due to having conflicting constraints. The fraction of degenerate regions is even higher in the $n = 3, d = 4$ model (99.6%).

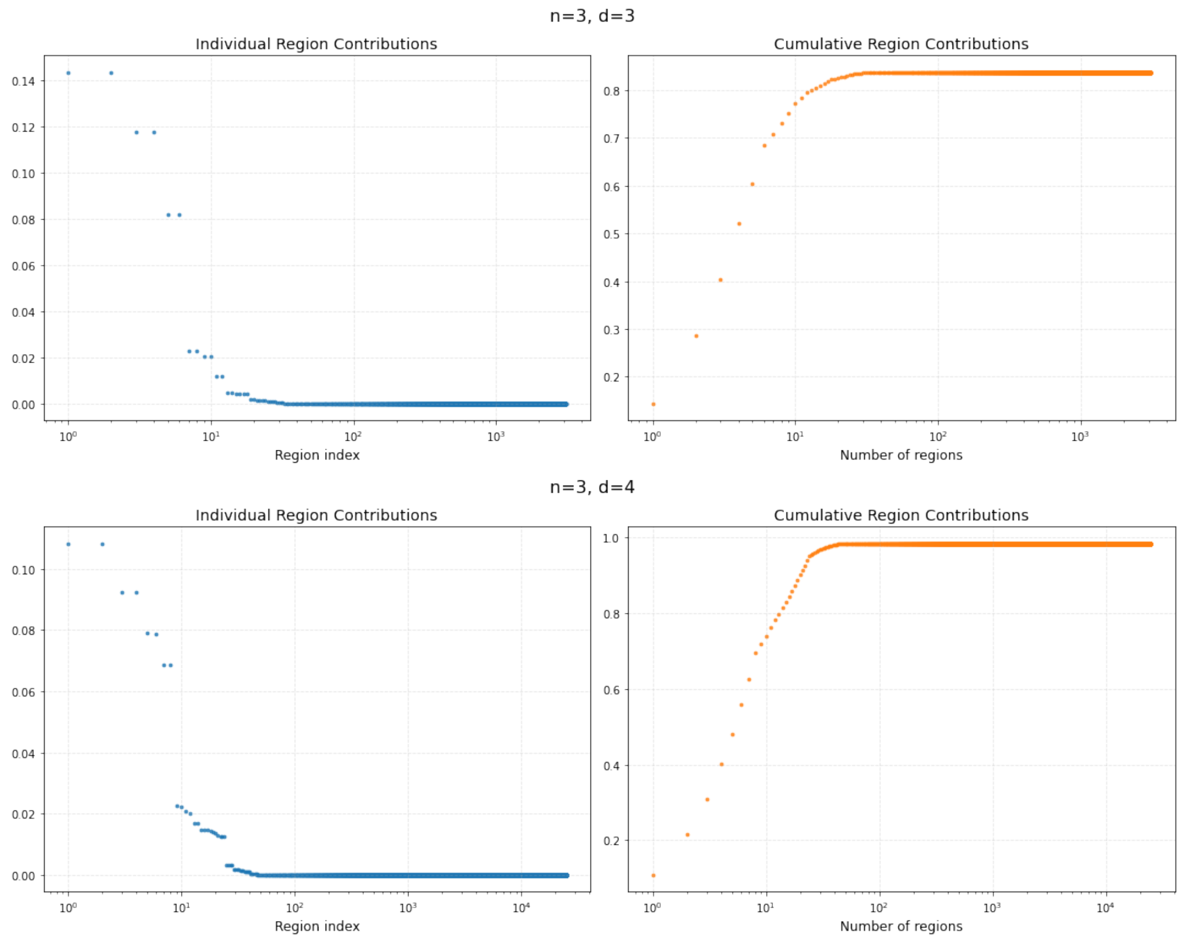


Figure 25: For the $n = 3, d = 3$ and $n = 3, d = 4$ models, we calculate the exact sizes of all of the regions in Equation 4. Left: the sizes, sorted in descending order (note the log scale on the x-axis). Right: the cumulative sum of the sizes of the largest x regions. For example, the $n = 3, d = 4$ model has an accuracy of 83.8%, which corresponds to the horizontal asymptote on the upper-right plot.

M

All Training Curves

Table 6 shows numerical data from Figure 15.

Figures 26 and 27 on the following two pages show the training curves of all 16 models trained on the GMHP, SS, and Hook objectives. The runs almost always appear to converge to a local minimum. Note that Hook loss exhibits a peculiar training dynamic — it often causes the accuracy to regularly oscillate up and down.

Table 6: Change in accuracy incurred by training against the GMHP, SS, and Hook objectives, compared to the cross-entropy baseline.

n	d	Initial Acc (%)	GMHP Diff (%)	SS Diff (%)	Hook Diff (%)
4	3	43.27	**1.98**	0.74	−0.07
4	4	60.21	1.86	**2.76**	1.86
4	5	74.56	−0.08	**1.63**	−0.27
4	6	86.48	−0.68	**0.80**	0.36
5	3	37.57	0.85	**1.07**	−2.84
5	4	42.83	3.34	**4.06**	−1.52
5	5	54.21	−3.07	**0.17**	−1.79
5	6	70.29	−1.91	**3.61**	1.20
6	3	34.06	**0.29**	−0.40	−3.77
6	4	39.40	0.67	**1.70**	−4.23
6	5	45.03	2.45	**3.71**	0.26
6	6	48.44	1.27	**3.68**	−1.23
8	3	**28.55**	−1.06	−1.90	−5.29
8	4	34.61	1.06	**1.53**	−6.73
8	5	38.36	1.14	**1.55**	−5.27
8	6	40.47	0.17	**1.55**	−3.30

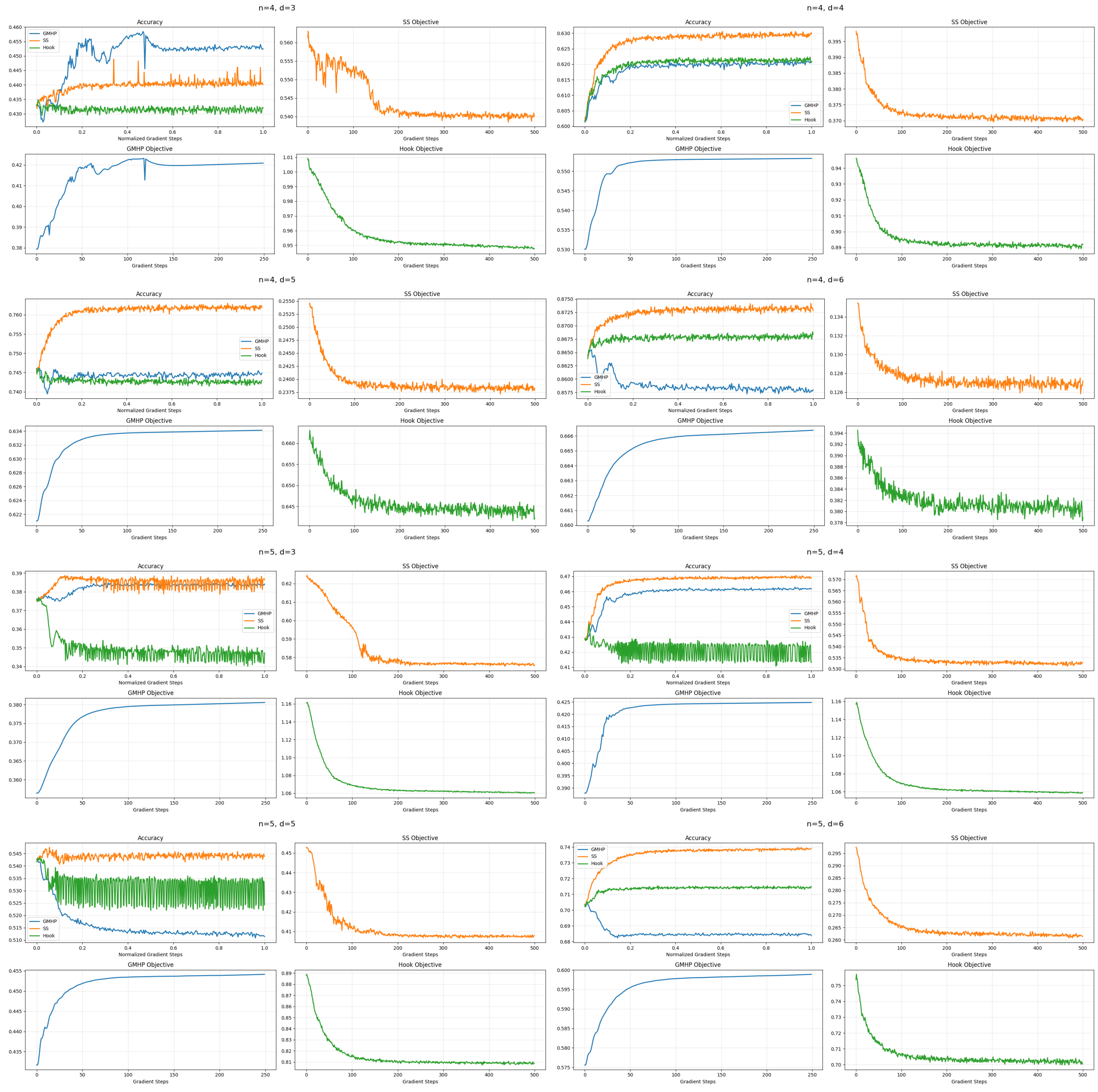


Figure 26: Training curves for $n = 4$ and $n = 5$

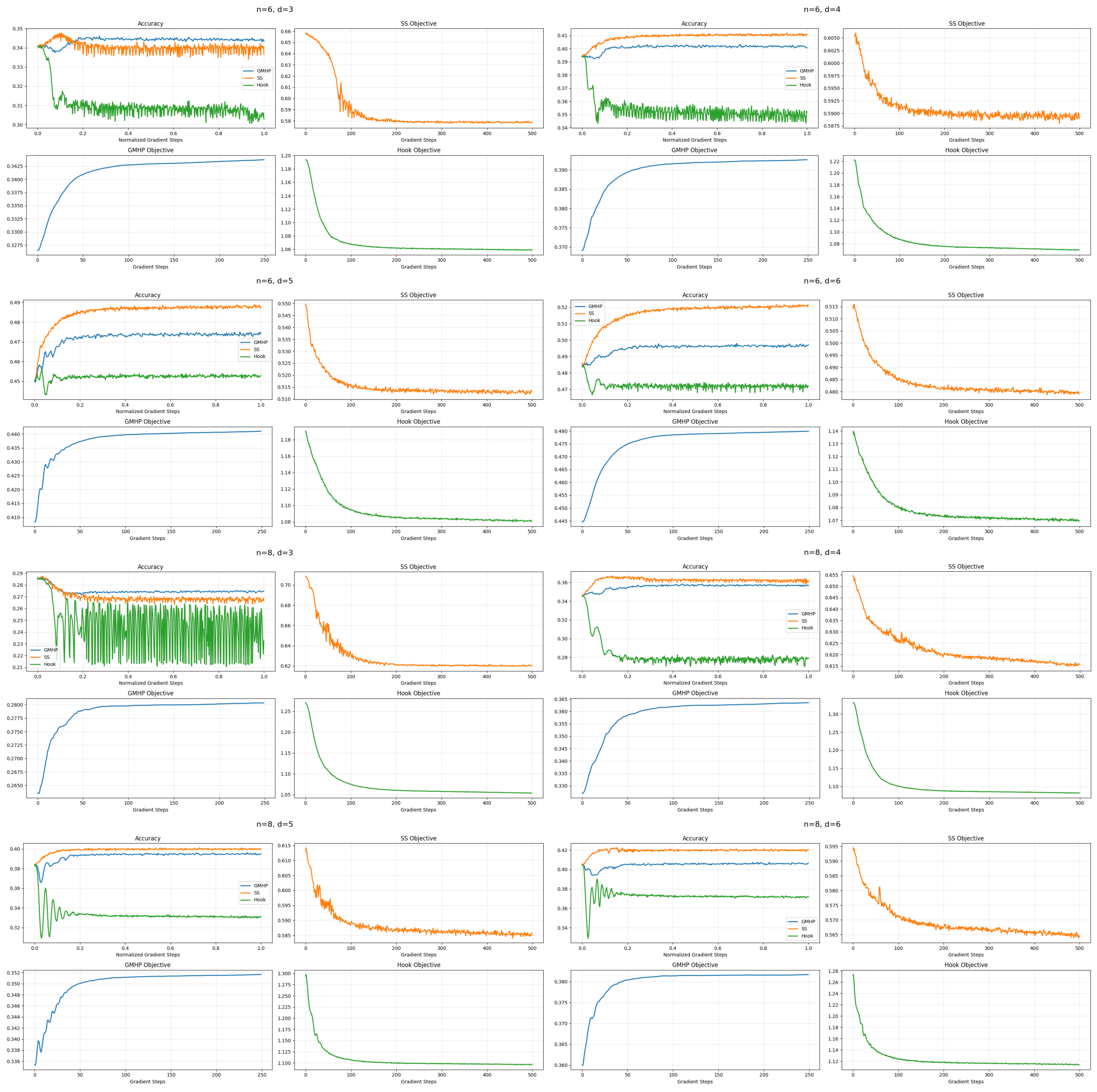


Figure 27: Training curves for $n = 6$ and $n = 8$