Limitations of Autoregressive Models and Their Alternatives

Chu-Cheng Lin[♯*] **Aaron Jaech**[♭] **Xin Li**[♯] **Matthew R. Gormley**[♮] **Jason Eisner**[♯]

[♯]Department of Computer Science, Johns Hopkins University

[♭]Facebook AI

[♮]Machine Learning Department, Carnegie Mellon University

{kitsing,lixints,jason}@cs.jhu.edu ajaech@fb.com mgormley@cs.cmu.edu

Abstract

Standard autoregressive language models perform only polynomial-time computation to compute the probability of the next symbol. While this is attractive, it means they cannot model distributions whose next-symbol probability is *hard* to compute. Indeed, they cannot even model them well enough to solve associated *easy* decision problems for which an engineer might want to consult a language model. These limitations apply no matter how much computation and data are used to train the model, unless the model is given access to oracle parameters that grow *superpolynomially* in sequence length.

Thus, simply training larger autoregressive language models is not a panacea for NLP. Alternatives include energy-based models (which give up efficient sampling) and latent-variable autoregressive models (which give up efficient scoring of a given string). Both are powerful enough to escape the above limitations.

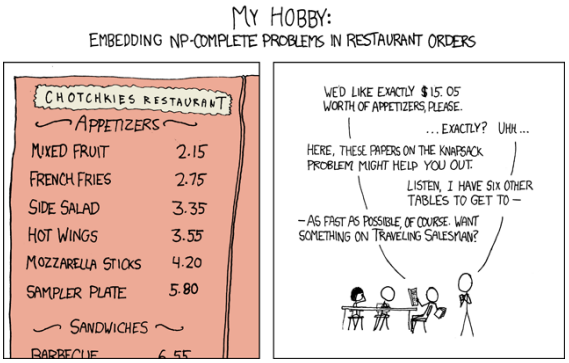

Figure 1: Valid answers to hard natural language inference problems can be hard to find (Munroe, 2009), but in many cases can be checked efficiently (*e.g.* the KNAPSACK problem in the comic). Given a *large enough* parametric autoregressive model with *correct* parameters, we can efficiently solve *all* problem instances with input length n, and efficiently verify the solutions — but the required model size can grow superpolynomially in n. (This allows the model to store precomputed results that we can look up in $O(n)$ at test time.) A main observation of this paper is that assuming NP $\not\subseteq$ P/poly, then without such a superpolynomial growth in model size, autoregressive models cannot even be used to verify answers to some problems where polynomial-time verification algorithms do exist.

1 Introduction

Sequence modeling is a core NLP problem. Many sequence models $\tilde{p}$ are efficient at *scoring strings*: given a string $\mathbf{x}$, its score $\tilde{p}(\mathbf{x})$ can be computed in $O(\text{poly}(|\mathbf{x}|))$. For example, an RNN (Mikolov et al., 2011) scores $\mathbf{x}$ in time $O(|\mathbf{x}|)$ while a Transformer (Vaswani et al., 2017) does so in time $O(|\mathbf{x}|^2)$. The score may be an unnormalized probability, and can be used to rank candidate strings.

Many sequence models also make it easy to compute marginal properties of $\tilde{p}$. They support efficient *sampling* of strings $\mathbf{x}$ (which allows unbiased approximation of marginal expectations). And they support efficient computation of the *normalizing constant* $Z = \sum_{\mathbf{x}} \tilde{p}(\mathbf{x})$ (or simply guarantee $Z = 1$) for any value of the model parameters.

How about training? Briefly: If a sequence model can efficiently compute $\tilde{p}(\mathbf{x})$ (and its derivatives with respect to model parameters), then it is efficient to compute parameter updates for noise-contrastive estimation (Gutmann and Hyvärinen, 2010; Gutmann and Hyvärinen, 2012) or score-matching (Hyvärinen, 2005). If sampling $\mathbf{x}$ or computing Z (and its derivatives) is *also* efficient, then it is efficient to compute parameter updates for ordinary MLE training.

Finally, popular sequence models are *compact*. Usually a fixed-size model is used to score strings $\mathbf{x}$ of all lengths. More generally, it might be reasonable to use an $O(\text{poly}(n))$-sized parameter vector $\boldsymbol{\theta}_n$ when $\mathbf{x}$ has length n, at least if parameter vectors can be obtained (perhaps from an oracle) for all needed lengths. In this paper, we investigate what can and cannot be achieved with models that are compact in this sense. This setup allows us to discuss the asymptotic behavior of model families.

Standard autoregressive models have the form $p(\mathbf{x}) = \prod_t p(x_t \mid \mathbf{x}_{<t})$[1] where each factor is efficient

[*]Part of this work was done at Facebook AI.

[1]In this paper we use the shorthand $\mathbf{x}_{<t} \triangleq x_1 \dots x_{t-1}$.

Model family	Compact parameters?	Efficient scoring?	Efficient sampling and normalization?	Support can be ...
ELN/ELNCP: Autoregressive models (§3.1)	✓	✓	✓	*some but not all* $L \in$ P
EC/ECCP: Energy-based models (§4.1)	✓	✓	✗	*all* $L \in$ P but *no* $L \in$ NPC
Lightly marginalized ELNCP: Latent-variable autoregressive models (§4.2)	✓	✗	✓	*all* $L \in$ NP
Lookup models (§4.3)	✗	✓	✓	*anything*

Table 1: A feature matrix of parametric model families discussed in this paper. Also see Figure 2 in the appendices.

to compute from a fixed parameter vector. These models satisfy all three of the desiderata above. By using flexible neural network architectures, standard autoregressive models have achieved stellar empirical results in many applications (Oord et al., 2016; Child et al., 2019; Zellers et al., 2019; Brown et al., 2020). However there are still tasks that they have not mastered: *e.g.*, it is reported that they struggle at deep logical structure, even when initialized to huge pretrained models (Wang et al., 2019a).

We point out that, unfortunately, there are certain sequence distributions whose unnormalized string probabilities $\tilde{p}(\mathbf{x})$ are *easy* to compute individually, yet whose autoregressive factors $p(x_t \mid \mathbf{x}_{<t})$ are NP-*hard* to compute or even approximate, or are even *uncomputable*. Thus, standard autoregressive models are *misspecified* for these distributions (cannot fit them). It does not help much to focus on strings of bounded length, or to enlarge the model: under the common complexity-theoretic assumption NP $\not\subseteq$ P/poly, the parameter size $|\boldsymbol{\theta}_n|$ must grow *superpolynomially* in n to efficiently approximate the probabilities of all strings of length up to n.

Indeed, one of our main findings is that there exist unweighted languages $L \in$ P for which *no* standard autoregressive model has L as its support, *i.e.*, assigns weight > 0 to just the strings $\mathbf{x} \in L$. This is downright depressing, considering the costs invested in training huge parametric autoregressive models (Bender et al., 2021). Since $L \in$ P, it is trivial to build an efficient scoring function $\tilde{p}(\mathbf{x})$ with fixed parameters that has L as its support — just not an autoregressive one. The problem holds for *all* standard autoregressive models, regardless of how much computation and training data are used to learn the model parameters.

That is, for an NP-hard problem, scoring a string $\mathbf{x}$ under a standard autoregressive model $p(\mathbf{x})$ cannot be used to *verify* a witness. Nor can *finding* a witness be solved by prompting such a model with a description of a problem instance and sampling a continuation $\mathbf{x}$ of that string. Such problems are abundant in NLP: for example, surface realization under Optimality Theory (Idsardi, 2006), decoding text from an AMR parse (Cai and Knight, 2013), phrase alignment between two sentences (DeNero and Klein, 2008), and in general inference for propositional logic (Cook, 1971), which underlies the NP-hardness of general natural language inference, as in Figure 1. In other words, our results imply that standard autoregressive models do not have the right structure to capture important linguistic regularities: *e.g.*, that observed sequences were *in fact* constructed to be phonologically optimal, expressive of a semantic form, or logically coherent!

Our work is also relevant to autoregressive models of fixed-dimensional vectors, such as NADE (Uria et al., 2016). These can be extended to arbitrary n-dimensional vectors by providing separate parameters $\boldsymbol{\theta}_n$ for each n. Our constructions imply that for some distributions, $|\boldsymbol{\theta}_n|$ must grow superpolynomially in n, even though this would be not be necessary if the models were not autoregressive.

In the remainder of this paper, we formalize our three desiderata for sequence models. We formalize compact autoregressive models and describe some limitations on their expressiveness. We then show that it can help to choose an alternative model family that relaxes any one of the three desiderata (Table 1).

2 Background

2.1 Weighted languages

An **unweighted language** $L \subseteq V^*$ is a set of strings $\mathbf{x}$ over a finite alphabet V. A **weighted language** $\tilde{p}$ is a function $\tilde{p} : V^* \to \mathbb{R}_{\geq 0}$. It may be regarded as specifying an unweighted language $L = \text{support}(\tilde{p}) \triangleq \{\mathbf{x} : \tilde{p}(\mathbf{x}) \neq 0\}$ along with positive weights for the strings in L. We say that a weighted language $\tilde{p}$ is **normalizable** if its **global normalizing constant** $Z \triangleq \sum_{\mathbf{x} \in V^*} \tilde{p}(\mathbf{x})$ is finite and strictly positive. When $\tilde{p}$ is normalizable, $p(\mathbf{x}) \triangleq \tilde{p}(\mathbf{x})/Z$ is a probability distribution over L. A **distribution** is any weighted language whose global normalizing constant is 1.

Let $\hat{\mathbf{x}} \preceq \mathbf{x}$ mean that $\hat{\mathbf{x}}$ is a prefix of $\mathbf{x} \in V^*$ (not necessarily a strict prefix). If $\tilde{p}$ is normalizable, then $Z(\hat{\mathbf{x}}) \triangleq \sum_{\mathbf{x} \in V^* : \hat{\mathbf{x}} \preceq \mathbf{x}} \tilde{p}(\mathbf{x})$ is $\leq Z$ for any $\hat{\mathbf{x}} \in V^*$, yielding a marginal **prefix probability** $Z(\hat{\mathbf{x}})/Z$. If the prefix $\hat{\mathbf{x}}$ has positive prefix probability, then it admits a **local conditional probability** $p(x \mid \hat{\mathbf{x}}) \triangleq Z(\hat{\mathbf{x}}x)/Z(\hat{\mathbf{x}})$ for each symbol $x \in V$, where the

denominator is interpreted as a **local normalizing constant**. This is the conditional probability that if a random string starts with the prefix $\hat{\mathbf{x}}$, the next symbol is x. There is also a probability $p(\$ \mid \hat{\mathbf{x}}) \triangleq 1 - \sum_{x \in V} p(x \mid \hat{\mathbf{x}}) = \tilde{p}(\hat{\mathbf{x}})/Z(\hat{\mathbf{x}}) \geq 0$ that the string ends immediately after $\hat{\mathbf{x}}$; the special symbol $\$ \notin V$ represents "end of string."

2.2 Computation for weighted languages

We define a weighted language $\tilde{p}$ to be **computable** if it is defined by a Turing machine (also called $\tilde{p}$) that maps any $\mathbf{x} \in V^*$ to $\tilde{p}(\mathbf{x}) \in \mathbb{Q}_{\geq 0}$ in finite time. The Turing machine does not have to compute Z.

While the computable weighted languages allow any computable function as $\tilde{p}$, most architectures for defining weighted languages (*e.g.*, RNNs or Transformers) do only a bounded or linear amount of work per input symbol. As a result, they compute $\tilde{p}(\mathbf{x})$ in time $O(\text{poly}(|\mathbf{x}|))$ (that is, $\tilde{p} \in \text{FP}$). We refer to such weighted languages as **efficiently computable** (**EC**). This does not imply that the normalized version p is efficiently computable, since finding the denominator Z requires summing over all of V^*.

If we tried to construct the same normalized distribution p as in the previous paragraph using a standard autoregressive model, we would model it as a product of local conditional probabilities, $p(\mathbf{x}) = (\prod_{t=1}^{|\mathbf{x}|} p(x_t \mid \mathbf{x}_{<t}))p(\$ \mid \mathbf{x})$. Most such architectures again do only a bounded or linear amount of work per input symbol. Yet one suspects that this may not always be enough work to do the job: the local conditional probabilities of the original $\tilde{p}$ are expensive to compute (unless $\tilde{p}$ has some special structure making $Z(\hat{\mathbf{x}})$ tractable).

Indeed, the observation of this paper is that for some efficiently computable weighted languages $\tilde{p}$, the local conditional probabilities are expensive to compute or even to approximate well. More precisely, autoregressive models cannot fit the local conditional probabilities unless they are superpolynomial in either their runtime or in their number of parameters (where the parameters may be precomputed at training time). We now explain how to formalize these notions.

2.3 Non-uniform computation

In the machine learning approach to sequence modeling, we usually do not manually design the Turing machine behind $\tilde{p}$. Rather, we design a model M with *parameters* $\boldsymbol{\theta}$. M is a Turing machine that reads $\boldsymbol{\theta}$ and outputs a specialized Turing machine $\tilde{p}_{\boldsymbol{\theta}} \triangleq M(\boldsymbol{\theta})$ that can score strings $\mathbf{x}$ and hence defines a weighted language. Without loss of generality, we will express $\boldsymbol{\theta}$ as a string in $\mathbb{B}^*$ (where $\mathbb{B} \triangleq \{0, 1\}$). For each $\boldsymbol{\theta}$, we obtain a potentially different weighted language.

Strings vary in length, and accurate modeling of longer strings may sometimes require more complex computations with more parameters. For example, when V is a natural language alphabet, a recurrent neural network may require more hidden units to model sentences of the language rather than individual words, and even more units to model whole documents. To accommodate this, we allow an *infinite sequence* of parameter vectors, $\boldsymbol{\Theta} = \{\boldsymbol{\theta}_n \in \mathbb{B}^* \mid n \in \mathbb{N}\}$, which yields an infinite sequence of Turing machines $\{\tilde{p}_n \mid n \in \mathbb{N}\}$ via $\tilde{p}_n \triangleq M(\boldsymbol{\theta}_n)$. We then define $\tilde{p}_{\boldsymbol{\Theta}}(\mathbf{x}) \triangleq \tilde{p}_{|\mathbf{x}|}(\mathbf{x})$, so a string of length n is scored by the $\tilde{p}_n$ machine. This is known as **non-uniform computation**. Of course, it is legal (and common) for all of the $\boldsymbol{\theta}_n$ to be equal, or empty, but if desired, we can obtain more power by allowing the number of parameters to grow with n if needed.

We can now consider *how rapidly* the parametric and runtime complexity may grow.

- If $|\boldsymbol{\theta}_n|$ is permitted to grow exponentially, then one can fit *any* weighted language $\tilde{p}$ (even an uncomputable one).[2] Simply use $\boldsymbol{\theta}_n$ to encode a trie with $O(|V|^{n+1})$ nodes that maps $\mathbf{x} \mapsto \tilde{p}(\mathbf{x})$ for any $|\mathbf{x}|$ of length n, and design M such that the Turing machine $\tilde{p}_n = M(\boldsymbol{\theta}_n)$ has a (large) state transition table that mirrors the structure of this trie. The resulting collection of Turing machines $\{\tilde{p}_n \mid n \in \mathbb{N}\}$ can then compute $\tilde{p}(\mathbf{x})$ exactly for any $\mathbf{x}$, with only linear runtime $O(|\mathbf{x}|)$ (which is used to traverse the trie).
- Separately, if unbounded runtime is permitted for M, then one can exactly fit *any computable* weighted language $\tilde{p}$. Simply have M, when run on $\boldsymbol{\theta}_n$, *compute* and return the large trie-structured $\tilde{p}_n$ that was mentioned above. In this case, M need not even use the parameters $\boldsymbol{\theta}_n$, except to determine n.
- Finally, if unbounded runtime is permitted for $\tilde{p}_n$, then again one can exactly fit *any computable* weighted language $\tilde{p}$. In this case, M trivially returns $\tilde{p}_n = \tilde{p}$ for all n.
- However, if the parameters $\boldsymbol{\Theta}$ are "compact" in the sense that $|\boldsymbol{\theta}_n|$ grows only as $O(\text{poly}(n))$, and also $\tilde{p}_n = M(\boldsymbol{\theta}_n)$ is constructed by M in time $O(\text{poly}(n))$, and $\tilde{p}_n$ scores any $\mathbf{x}$ of length n in time $O(\text{poly}(n))$, then we say that the resulting weighted language $\tilde{p}$ is **efficiently computable with compact parameters** (ECCP).[3] We refer

[2] See our remark on computability in Appendix A.

[3] Since we require M to run in polytime, it can only look at a polynomial-sized portion of $\boldsymbol{\theta}_n$. Hence it is not really crucial for

to M paired with a parameter space of possible compact values for $\boldsymbol{\Theta}$ as an **ECCP model**.

Neural models of weighted languages are typically ECCP models. The construction and execution of the neural network $\tilde{p}_n$ may perform a polynomial amount of total computation to score the string $\mathbf{x}$. This computation may involve parameters that were precomputed using any amount of effort (*e.g.*, training on data) or even obtained from an oracle (they need not be computable). However, the exponentially many strings of length n must share a polynomial-size parameter vector $\boldsymbol{\theta}_n$, which prevents the solution given in the first bullet point above.

In practice one takes $\boldsymbol{\theta}_n = \boldsymbol{\theta}$ for all n and obtains $\boldsymbol{\theta} \in \mathbb{R}^d$ by training. However, we do not consider whether such parameters are easy to estimate or even computable. We simply ask, for a given target language $\tilde{p}$, whether there *exists* a polynomially growing sequence $\boldsymbol{\Theta}$ of "good" parameter vectors for any parametric model M. When not, there can be no scheme for estimating arbitrarily long finite prefixes of such a sequence. So for any polynomial f, any training scheme that purports to return a trained model of size $f(n)$ that works "well" for strings of length $\leq n$ must fail for large enough n — even if unlimited data, computation, and oracles are allowed at training time.

2.4 P, P/poly, and NP/poly

The phrase "efficiently computable with compact parameters" means that without access to those parameters, the ECCP weighted language may no longer be efficiently computable. Indeed, it need not be computable at all, if the parameter vectors store the outputs of some uncomputable function.

Our definitions above of EC and ECCP weighted languages are weighted generalizations of complexity classes P and P/poly, respectively,[4] and their supports are always unweighted languages in P and P/poly, respectively. An unweighted language L is in P iff there is a deterministic Turing machine that decides in $O(\text{poly}(|\mathbf{x}|))$ time whether $\mathbf{x} \in L$. And an unweighted language L' is in P/poly iff[5] there exist Turing machines $\{M_n : n \in \mathbb{N}\}$ such that M_n decides in $O(\text{poly}(n))$ time whether $\mathbf{x}$ of length n is in L', where each M_n can be constructed in $O(\text{poly}(n))$ time as $M(\boldsymbol{\theta}_n)$, for some Turing machine M and some sequence of polynomially-sized **advice strings** $\boldsymbol{\Theta} = \{\boldsymbol{\theta}_n \mid n \in \mathbb{N}\}$ with $|\boldsymbol{\theta}_n| \in O(\text{poly}(n))$. We define the language class NP/poly similarly to P/poly: the only difference is the family $\{M_n : n \in \mathbb{N}\}$ consists of *nondeterministic Turing machines*.

Naturally, P $\subseteq$ P/poly. But P/poly is larger than P: it contains all sparse languages, regardless of their hardness — even sparse undecidable languages — as well as many dense languages. The extra power of P/poly comes from its access to compact advice strings that do not have to be recursively enumerable, let alone efficient to find. This corresponds to statistical modeling, where the trained model has a computationally efficient architecture plus access to parameters that might have taken a long time to find.

2.5 NP-completeness and Sat

NP-complete decision problems have solutions that are efficient to validate but inefficient to find (assuming P $\neq$ NP). One of the most well-known NP-complete problems is the boolean satisfiability problem (Sat) (Cook, 1971). Given a boolean formula ϕ, Sat accepts ϕ iff ϕ can be satisfied by some value assignment. For example, the formula $(A_1 \vee \neg A_2 \vee A_3) \wedge (A_1 \vee \neg A_4)$ is in Sat, since there is a satisfying assignment $A_{1\ldots4} = \texttt{1101}$. We denote the number of satisfying assignments to ϕ as $\#(\phi)$.

It is widely believed that no NP-complete languages are in P/poly. Otherwise we would have all of NP $\subseteq$ P/poly and the polynomial hierarchy would collapse at the second level (Karp and Lipton, 1980).

A capacity limitation of EC/ECCP weighted languages naturally follows from this belief:[6]

Lemma 1. *For any $L \in$ P, there exists an EC weighted language with support L. For any $L \in$ P/poly, there exists an ECCP language with support L. But for any $L \in$ NP-complete, there exists no ECCP language with support L (assuming NP $\not\subseteq$ P/poly).*

In addition to not capturing the support of NP-complete languages, ECCP weighted languages can-

the parameters $\boldsymbol{\theta}_n^{\text{p}}$ to be compact, but we nonetheless include this intuitive condition, without loss of generality.

[4] Namely the nonnegative functions in FP and FP/poly.

[5] Our presentation of P/poly is a variant of Arora and Barak (2009, §6), in which inputs $\mathbf{x}$ of length n are evaluated by a polytime function M that is given an advice string $\boldsymbol{\theta}_n$ as an auxiliary argument. This corresponds to a neural architecture M that can consult trained parameters $\boldsymbol{\theta}_n$ at runtime. We have replaced the standard call $M(\boldsymbol{\theta}_n, \mathbf{x})$ with the "curried" expression $M(\boldsymbol{\theta}_n)(\mathbf{x})$, which we still require to execute in polynomial total time. Here the intermediate result $M_n = M(\boldsymbol{\theta}_n)$ corresponds to a trained runtime model for inputs of length n. Our Turing machines M_n have size polynomial in n (because they are constructed by M in polynomial time). They correspond to the polynomial-sized boolean circuits M_n that are used to evaluate inputs of length n under the classical definition of P/poly (Ladner, 1975). We exposed these intermediate results M_n only to observe in §2.3 and §4.3 that if we had allowed the M_n to grow exponentially, they would have been able to encode the answers in tries.

[6] All omitted proofs are in Appendix A.

not help solve other NP-hard problems, either. For example, many structured prediction problems in NLP can be formulated as $\text{argmax}_{\mathbf{x}:\hat{\mathbf{x}}\leq\mathbf{x}} \tilde{p}(\mathbf{x})$: we are given a prefix $\hat{\mathbf{x}}$ as input and look for its optimal continuation under $\tilde{p}$. But if this problem is NP-hard for a particular $\tilde{p}$, then it is not in P/poly (assuming NP $\not\subseteq$ P/poly), so it cannot be accomplished by any polytime algorithm that queries an ECCP model.

3 Autoregressive ECCP models (ELNCP models) have reduced capacity

In this section we formally define *autoregressive* ECCP models, and prove that they have strictly less capacity than general ECCP models or even just EC models. Our proofs rely on the construction of a EC model $\tilde{p}$ where computing the local conditional probabilities $p(x \mid \hat{\mathbf{x}})$ is NP-hard, so they cannot be computed with compact parameters, if NP $\not\subseteq$ P/poly.

3.1 ELN and ELNCP models

Many parameter estimation techniques and inference methods specifically work with local conditional probabilities $p(x \mid \hat{\mathbf{x}})$. Thus, it is common to use parametric models where such quantities can be computed in time $O(\text{poly}(|\hat{\mathbf{x}}|))$ (given the parameters).[7] These are the "standard autoregressive models" we discussed in §1. We say that the resulting distributions are **efficiently locally normalizable**, or **ELN**.

We may again generalize ELNs to allow the use of compact parameters. For any weighted language $\tilde{p}$, the Turing machine M^{q} **efficiently locally normalizes** $\tilde{p}$ **with compact parameters** $\boldsymbol{\Theta}^{\mathrm{q}} = \{\boldsymbol{\theta}^{\mathrm{q}}_n \mid n \in \mathbb{N}\}$ if

- the parameter size $|\boldsymbol{\theta}^{\mathrm{q}}_n|$ grows only as $O(\text{poly}(n))$
- $M^{\mathrm{q}}(\boldsymbol{\theta}^{\mathrm{q}}_n)$ returns a Turing machine q_n (similar to $\tilde{p}_n$ in §2.3) in time $O(\text{poly}(n))$
- $\tilde{p}$ is normalizable (so p exists)
- q_n maps $\hat{\mathbf{x}}x \mapsto p(x \mid \hat{\mathbf{x}})$ for all $x \in V \cup \{\$\}$ and all prefixes $\hat{\mathbf{x}} \in V^*$ with $|\hat{\mathbf{x}}| \leq n$ and $Z(\hat{\mathbf{x}}) > 0$
- q_n runs on those inputs $\hat{\mathbf{x}}x$ in time $O(\text{poly}(n))$

If there is M^{q} that efficiently locally normalizes a weighted language $\tilde{p}$ with compact parameters $\boldsymbol{\Theta}^{\mathrm{q}}$, we say $\tilde{p}$ is **efficiently locally normalizable with compact parameters**, or **ELNCP**. Note that this is a property of the weighted language itself. In this case, it is obvious that $\tilde{p}$ is ECCP:

Lemma 2. *An ELNCP model $\tilde{p}$ is also ECCP. Likewise, an ELN model is also EC.*

If we define ELNCP *models* analogously to ECCP models, Lemma 2 means that locally normalized models do not provide any extra power. Their distributions can always be captured by globally normalized models (of an appropriate architecture that we used in the proof). But we will see in Theorem 1 that the converse is likely not true: provided that NP $\not\subseteq$ P/poly, there are efficiently computable weighted languages that cannot be efficiently locally normalized, even with the help of compact parameters. That is, they are EC (hence ECCP), yet they are not ELNCP (hence not ELN).

3.2 ELNCP models cannot exactly capture all EC (or ECCP) distributions

We reducing SAT to computing certain local conditional probabilities of $\tilde{p}$ (as defined in §2.1). Each decision $\text{SAT}(\phi)$ (where ϕ ranges over formulas) corresponds to a particular local conditional probability, implying that there is no polytime scheme for computing all of these probabilities, even with polynomially sized advice strings (*i.e.*, parameters).

Without loss of generality, we consider only formulae ϕ such that the set of variables mentioned at least once in ϕ is $\{A_1, \ldots, A_j\}$ for some $j \in \mathbb{N}$; we use $|\phi|$ to denote the number of variables j in ϕ. We say that $\mathbf{a}$ **satisfies** ϕ if $\mathbf{a} \in \mathbb{B}^{|\phi|}$ and $(A_1 = a_1, \ldots, A_{|\phi|} = a_{|\phi|})$ is a satisfying assignment. Finally, let boldface $\boldsymbol{\phi} \in \mathbb{B}^*$ denote $\text{enc}(\phi)$ where enc is a prefix-free encoding function. We can now define the unweighted language $L = \{\boldsymbol{\phi}\mathbf{a} \mid \phi$ is a formula and $\mathbf{a} \in \mathbb{B}^{|\phi|}$ and $\mathbf{a}$ satisfies $\phi\}$ over alphabet $\mathbb{B}$, which contains each possible SAT problem concatenated to each of its solutions.[8]

We now convert L to a weighted language $\tilde{p}$, defined by $\tilde{p}(\mathbf{x}) = \tilde{p}(\boldsymbol{\phi}, \mathbf{a}) = (\frac{1}{3})^{|\mathbf{x}|+1}$ for $\mathbf{x} \in L$ (otherwise $\tilde{p}(\mathbf{x}) = 0$). $\tilde{p}$ is normalizable since Z is both finite ($Z = \sum_{\mathbf{x}\in\mathbb{B}^*} \tilde{p}(\mathbf{x}) \leq \sum_{\mathbf{x}\in\mathbb{B}^*} (\frac{1}{3})^{|\mathbf{x}|+1} = 1$) and positive ($Z > 0$ because the example string in footnote 8 has weight > 0). The conditional distribution

[7] An autoregressive model architecture generally defines $p(\mathbf{x})$ as an efficiently computable (§2.2) product of local conditional probabilities. However, the parametrization usually ensures only that $\sum_{x\in V} p_{\boldsymbol{\theta}}(x \mid \hat{\mathbf{x}}) = 1$ for all prefixes $\hat{\mathbf{x}}$. Some parameter settings may give rise to **inconsistent** distributions where $Z \triangleq \sum_{\mathbf{x}\in V^*} p_{\boldsymbol{\theta}}(\mathbf{x}) < 1$ because the generative process terminates with probability < 1 (Chen et al., 2018). In this case, the factors $p_{\boldsymbol{\theta}}(x \mid \hat{\mathbf{x}})$ defined by the autoregressive model are not actually the conditional probabilities of the weighted language (as defined by §2.1). It is true that training $\boldsymbol{\theta}$ with a likelihood objective does encourage finding a weighted language whose generative process always terminates (hence $Z = 1$), since this is the behavior observed in the training corpus (Chi and Geman, 1998; Chen et al., 2018; Welleck et al., 2020). Our definitions of ELN(CP) models require the *actual* conditional probabilities to be efficiently computable. Autoregressive models that do not sum to 1, whose normalized probabilities can be uncomputable, are not ruled out by our theorems that concern ELN(CP).

[8] For example, L contains the string $\boldsymbol{\phi}\mathbf{a}$ where $\boldsymbol{\phi} = \text{enc}((A_1 \vee \neg A_2 \vee A_3) \wedge (A_1 \vee \neg A_4))$ and $\mathbf{a} = 1101$.

$p(\mathbf{a} \mid \boldsymbol{\phi})$ is uniform over the satisfying assignments $\mathbf{a}$ of $\boldsymbol{\phi}$, as they all have the same length $|\phi|$.

$\tilde{p}$ is efficiently computable, and so is $p = \tilde{p}/Z$.[9] Yet deciding whether the local conditional probabilities of $\tilde{p}$ are greater than 0 is NP-hard. In particular, we show that SAT can be reduced to deciding whether certain local probabilities are greater than 0, namely the ones that condition on prefixes $\hat{\mathbf{x}}$ that consist only of a formula: $\hat{\mathbf{x}} = \boldsymbol{\phi}$ for some ϕ. This implies, assuming NP $\not\subseteq$ P/poly, that no $(M^{\mathrm{q}}, \boldsymbol{\Theta}^{\mathrm{q}})$ can efficiently locally normalize $\tilde{p}$ with compact parameters. Granted, the restriction of $\tilde{p}$ to the finite set $\{\mathbf{x} \in \mathbb{B}^* : |\mathbf{x}| \leq n\}$ can be locally normalized by some polytime Turing machine q_n, using the same trie trick sketched in §2.3. But such tries have sizes growing exponentially in n, and it is not possible to produce a sequence of such machines, $\{q_n : n \in \mathbb{N}\}$, via a single master Turing machine M^{q} that runs in $O(\mathrm{poly}(n))$ on $\boldsymbol{\theta}^{\mathrm{q}}_n$. That is:

Theorem 1. *Assuming* NP $\not\subseteq$ P/poly*, there exists an efficiently computable normalizable weighted language $\tilde{p}$ that is not ELNCP.*

Proof sketch. Take $\tilde{p}$ to be the weighted language we defined earlier in this section. $\tilde{p}$ is clearly efficiently computable. We will show that if it is ELNCP via $(M^{\mathrm{q}}, \boldsymbol{\Theta}^{\mathrm{q}})$, then the NP-complete problem SAT is in P/poly, contradicting the assumption. We must give a method for using $(M^{\mathrm{q}}, \boldsymbol{\Theta}^{\mathrm{q}})$ to decide SAT in polytime and with compact parameters $\boldsymbol{\Theta}$. Given ϕ, our method constructs a simple related formula ϕ' such that

- ϕ' has at least one satisfying assignment (so $Z(\boldsymbol{\phi}') > 0$ and thus $p(1 \mid \boldsymbol{\phi}')$ is defined)
- ϕ' has satisfying assignments with $A_1 = 1$ (*i.e.*, $p(1 \mid \boldsymbol{\phi}') > 0$) if and only if ϕ is satisfiable

Our construction also provides a polynomial function f such that $|\boldsymbol{\phi}'|$ is guaranteed to be $\leq f(|\boldsymbol{\phi}|)$. We now define $\boldsymbol{\Theta}$ by $\boldsymbol{\theta}_n = \boldsymbol{\theta}^{\mathrm{q}}_{f(n)}$ $(\forall n)$. When our SAT algorithm with compact parameters $\boldsymbol{\Theta}$ is given $\boldsymbol{\phi}$ of length n, it can use the polynomial-size advice string $\boldsymbol{\theta}_n$ to ask $(M^{\mathrm{q}}, \boldsymbol{\Theta}^{\mathrm{q}})$ in polynomial time for $p(1 \mid \boldsymbol{\phi}')$. $\textsc{Sat}(\boldsymbol{\phi})$ returns true iff that probability is > 0.[10] □

3.3 ELNCP models cannot even capture all EC (or ECCP) supports or rankings

We can strengthen Theorem 1 as follows:

Theorem 2. *Assuming* NP $\not\subseteq$ P/poly*, there exists an efficiently computable normalizable weighted language $\tilde{p}$ where there is no ELNCP $\tilde{q}$ such that* $\mathrm{support}(\tilde{p}) = \mathrm{support}(\tilde{q})$.

Proof. Observe that for any two weighted languages $\tilde{p}$ and $\tilde{q}$ with the same support, $\forall \hat{\mathbf{x}} \in V^*, Z_{\tilde{p}}(\hat{\mathbf{x}}) > 0 \iff Z_{\tilde{q}}(\hat{\mathbf{x}}) > 0$ (where $Z_{\tilde{p}}$ and $Z_{\tilde{q}}$ return the prefix probabilities of $\tilde{p}$ and $\tilde{q}$ respectively). Thus, for any $\hat{\mathbf{x}}$ with $Z_{\tilde{p}}(\hat{\mathbf{x}}) > 0$, $p(1 \mid \hat{\mathbf{x}}) \triangleq Z_{\tilde{p}}(\hat{\mathbf{x}}1)/Z_{\tilde{p}}(\hat{\mathbf{x}})$ and $q(1 \mid \hat{\mathbf{x}}) \triangleq Z_{\tilde{q}}(\hat{\mathbf{x}}1)/Z_{\tilde{q}}(\hat{\mathbf{x}})$ are well-defined and $p(1 \mid \hat{\mathbf{x}}) > 0 \iff q(1 \mid \hat{\mathbf{x}}) > 0$. If $\tilde{q}$ is ELNCP, then all such probabilities $q(1 \mid \hat{\mathbf{x}})$ can be computed in polytime with compact parameters, so it is likewise efficient to determine whether $p(1 \mid \hat{\mathbf{x}}) > 0$. But this cannot be the case when $\tilde{p}$ is the weighted language used in the proof of Theorem 1, since that would suffice to establish that SAT $\in$ P/poly, following the proof of that theorem. □

To put this another way, there exists an unweighted language in P (namely $\mathrm{support}(\tilde{p})$) that is not the support of *any* ELNCP distribution.

If they have different support, normalizable languages also differ in their ranking of strings:

Lemma 3. *Let $\tilde{p}, \tilde{q}$ be normalizable weighted languages with* $\mathrm{support}(\tilde{p}) \neq \mathrm{support}(\tilde{q})$*. Then $\exists \mathbf{x}_1, \mathbf{x}_2 \in V^*$ such that $\tilde{p}(\mathbf{x}_1) < \tilde{p}(\mathbf{x}_2)$ but $\tilde{q}(\mathbf{x}_1) \geq \tilde{q}(\mathbf{x}_2)$.*

Therefore, no ELNCP $\tilde{q}$ captures the string ranking of $\tilde{p}$ from Theorem 2. And for some $\tilde{p}$, any ELNCP $\tilde{q}$ misranks even string pairs of "similar" lengths:

Theorem 3. *Assuming* NP $\not\subseteq$ P/poly*, there exists an efficiently computable normalizable weighted language $\tilde{p}$ such that no ELNCP $\tilde{q}$ with* $\mathrm{support}(\tilde{q}) \supseteq \mathrm{support}(\tilde{p})$ *has $\tilde{p}(\mathbf{x}_1) < \tilde{p}(\mathbf{x}_2) \Rightarrow \tilde{q}(\mathbf{x}_1) < \tilde{q}(\mathbf{x}_2)$ for all $\mathbf{x}_1, \mathbf{x}_2 \in V^*$. Indeed, any such $\tilde{q}$ has a counterexample where $\tilde{p}(\mathbf{x}_1) = 0$. Moreover, there is a polynomial $f_{\tilde{q}} : \mathbb{N} \to \mathbb{N}$ such that a counterexample exists for* every $\mathbf{x}_1$ *such that $\tilde{p}(\mathbf{x}_1) = 0$ and $\tilde{q}(\mathbf{x}_1) > 0$, where the $\mathbf{x}_2$ in this counterexample always satisfies $|\mathbf{x}_2| \leq f_{\tilde{q}}(|\mathbf{x}_1|)$.*

Theorem 3 is relevant if one wishes to train a model $\tilde{q}$ to rerank strings that are proposed by another method (*e.g.*, beam search on $\tilde{q}$, or exact k-best decoding from a more tractable distribution). If the desired rankings are given by Theorem 3's $\tilde{p}$, any smoothed[11] ELNCP model $\tilde{q}$ will misrank some sets of candidate strings, even sets all of whose strings are "close" in length, by failing to rank an impossible string ($\mathbf{x}_1$ with $\tilde{p}(\mathbf{x}_1) = 0$) below a possible one ($\mathbf{x}_2$ with $\tilde{p}(\mathbf{x}_2) > 0$).

[9] Almost. This Z could be irrational, but at least it is computable to any desired precision. For any rational $\hat{Z} \approx Z$, we can say $\hat{p} = \tilde{p}/\hat{Z} \approx p$ is EC, via a Turing machine $M^{\hat{p}}$ that stores $\hat{Z}$. Further remarks on irrationality appear in Appendix A.

[10] See also the remark on implications for seq2seq models following the proof in Appendix A.

[11] Smoothing is used to avoid ever incorrectly predicting 0 (a "false negative") by ensuring $\mathrm{support}(\tilde{q}) \supseteq \mathrm{support}(\tilde{p})$. E.g., autoregressive language models often define $q(x \mid \hat{\mathbf{x}})$ using a softmax over $V \cup \{\$\}$, ensuring that $q(\mathbf{x}) > 0$ for all $\mathbf{x} \in V^*$.

3.4 ELNCP models cannot even *approximate* EC (or ECCP) distributions

Theorem 2 implies that there exists $\tilde{p}$ whose local probabilities $p(x \mid \hat{\mathbf{x}})$ are not approximated by any ELNCP q to within any constant factor λ, since that would perfectly distinguish zeroes from non-zeroes and the resulting support sets would be equal.[12]

However, this demonstration hinges on the difficulty of multiplicative approximation of zeroes — whereas real-world distributions may lack zeroes. Below we further show that it is hard even to approximate the *non-zero* local conditional probabilities (even with the additional help of randomness).

Theorem 4. *Assuming* NP $\not\subseteq$ P/poly*, there exists an efficiently computable weighted language* $\tilde{p} : V^* \to \mathbb{R}_{\geq 0}$ *such that there is no* $(M^{\mathrm{q}}, \boldsymbol{\Theta}^{\mathrm{q}})$ *where* $\boldsymbol{\Theta}^{\mathrm{q}} = \{\boldsymbol{\theta}^{\mathrm{q}}_n \mid n \in \mathbb{N}\}$ *that satisfies all of the following properties (similar to §3.1):*

- *the parameter size* $|\boldsymbol{\theta}^{\mathrm{q}}_n|$ *grows only as* $O(\mathrm{poly}(n))$
- $M^{\mathrm{q}}(\boldsymbol{\theta}^{\mathrm{q}}_n)$ *returns a* probabilistic *Turing machine* q_n *in time* $O(\mathrm{poly}(n))$
- *there exists* $\lambda \geq 1$ *such that for each* $x \in V \cup \{\$\}$ *and* $\hat{\mathbf{x}} \in V^*$ *with* $|\hat{\mathbf{x}}| \leq n$ *and* $p(x \mid \hat{\mathbf{x}}) > 0$*, the probabilistic computation* $q_n(\hat{\mathbf{x}}x)$ *has probability* $> 2/3$ *of approximating* $p(x \mid \hat{\mathbf{x}})$ *to within a factor of* λ *(that is,* $q_n(\hat{\mathbf{x}}x)/p(x \mid \hat{\mathbf{x}}) \in [1/\lambda, \lambda]$*)*
- q_n *runs on those inputs* $\hat{\mathbf{x}}x$ *in time* $O(\mathrm{poly}(n))$

Moreover, the statement above remains true

(a) when the approximation guarantee is only required to hold for prefixes $\hat{\mathbf{x}}$ *where* $\{\mathbf{x} : \hat{\mathbf{x}} \preceq \mathbf{x}\}$ *is finite (so* $p(x \mid \hat{\mathbf{x}})$ *is computable by brute force)*

(b) or, when $\mathrm{support}(\tilde{p}) = V^*$

3.5 ELN models are *unconditionally* weak

Our above results rely on the NP-*hardness* of computing or approximating an EC distribution's autoregressive factors $p(\cdot \mid \mathbf{x}_{<t})$. In Appendix A, we show that these factors can even be *uncomputable*. In such cases, the distribution cannot be ELN (Theorem 5), though sometimes it is still ELNCP (Theorem 6). This result does *not* assume P $\neq$ NP or NP $\not\subseteq$ P/poly.

4 Alternative model families

We now discuss alternative families of sequence distributions that trade away efficiency or compactness in exchange for greater capacity, as shown in Table 1.

[12] Dropping the normalization requirement on the approximated local probabilities (so that possibly $\sum_{x \in V} q(x \mid \hat{\mathbf{x}}) \neq 1$) does not help. Otherwise, again, SAT could be solved in polynomial time (with the help of polysize advice strings) by using $q(1 \mid \boldsymbol{\phi}')$ to determine in the proof of Theorem 1 whether $p(1 \mid \boldsymbol{\phi}') > 0$.

4.1 Energy-based models (EBMs)

Energy-based models (LeCun et al., 2006) of discrete sequences (Rosenfeld et al., 2001; Sandbank, 2008; Huang et al., 2018) traditionally refer to the EC models of §2.2. Only the unnormalized probabilities $\tilde{p}_{\boldsymbol{\theta}}(\mathbf{x})$ are required to be efficiently computable. Lemmas 1 and 2 showed that this model family contains all ELN languages and can achieve any support in P. Theorem 1 shows that it also contains languages that are not ELN or even ELNCP: intuitively, the reason is that the sums $Z(\hat{\mathbf{x}})$ needed to compute the local normalizing constants (see §2.1) can be intractable.

If we generalize energy-based sequence models to include all ECCP models — that is, we allow nonuniform computation with compact parameters — then Lemmas 1 and 2 guarantee that they can capture all ELNCP languages and furthermore all languages in P/poly (though still not NP-complete languages).

Experiments on different parameterizations. Maximum-likelihood parameter estimation (MLE) can be expensive in an EBM because the likelihood formula involves the expensive summation $Z = \sum_{\mathbf{x} \in V^*} \tilde{p}_{\boldsymbol{\theta}}(\mathbf{x})$. This forces us in practice to use alternative estimators that do not require computing normalized probabilities, such as noise-contrastive estimation (NCE) or score matching (§1), which are less statistically efficient. In pilot experiments we found that both RNN- and Transformer-based EBMs trained with NCE achieved *worse* held-out perplexity than comparable locally normalized models trained with MLE.[13]

Fortunately, it is possible to infuse a globally normalized architecture with the inductive bias of a locally normalized one, which empirically yields good results. **Residual energy-based models** (**REBMs**) (Bakhtin et al., 2021) are a simple hybrid architecture:

$$p_{\boldsymbol{\theta}}(\mathbf{x}) \propto \tilde{p}_{\boldsymbol{\theta}}(\mathbf{x}) \triangleq p_0(\mathbf{x}) \cdot \exp g_{\boldsymbol{\theta}}(\mathbf{x})$$

This simply multiplies our previous weight by a new factor $p_0(\mathbf{x})$. The *base model* $p_0 : L \to (0, 1]$ is a locally normalized neural sequence model (ELN model) that was pretrained on the same distribution. $g_{\boldsymbol{\theta}} : V^* \to \mathbb{R}$ is a learnable function (with parameters $\boldsymbol{\theta}$) that is used to adjust p_0, yielding a weighted language $\tilde{p}_{\boldsymbol{\theta}}$ with the same support L. We implemented

[13] This might be due to a capacity limitation of the specific globally normalized architectures (*i.e.*, no parameters work well), or excess capacity (*i.e.*, too many parameters work well on the finite sample), or statistical inefficiency of the estimator (the NCE objective on the finite sample, with the noise distribution we chose, does not distinguish among parameters as well as MLE does), or an optimization difficulty caused by local optima in the NCE optimization landscape.

REBMs, again with NCE training, and evaluated them on two different neural architectures (GRU- and Transformer-based) and 3 datasets (WikiText (Merity et al., 2017), Yelp (Yelp), and RealNews (Zellers et al., 2019)). In each setting we tried, the REBM slightly but significantly improved the perplexity of the base model p_0 ($p < 0.05$).[14]

4.2 Latent-variable models

Autoregressive models have $Z = 1$ for any setting of the parameters (or at least any setting that guarantees consistency: see footnote 7). Clearly $Z = 1$ ensures that Z is both finite and tractable. Can we find a model family that retains this convenience (unlike EBMs), while still being expressive enough to have any non-empty language in P as support?

Autoregressive *latent-variable* models form such a family. As in directed graphical models, the use of latent variables provides a natural way to model partial observations of an underlying stochastic sequence of events. We will model an observed sequence $\mathbf{x}$ of length n as a function of a latent string $\mathbf{z}$ of length $O(\text{poly}(n))$. As in EBMs, the probability $p(\mathbf{x})$ can be computationally intractable, allowing these models to break the expressivity bottleneck of ordinary autoregressive models. However, the intractability no longer comes from exponentially many summands in the denominator Z, but rather from exponentially many summands in the numerator — namely, the summation over all latent $\mathbf{z}$ that could have produced $\mathbf{x}$. Notice that as a result, even unnormalized string weights are now hard to compute, although once computed they are already normalized.

Formally, we define **marginalized** weighted languages. We say that $\tilde{p}$ is a **marginalization** of the weighted language $\tilde{r}$ if it can be expressed as $\tilde{p}(\mathbf{x}) = \sum_{\mathbf{z}:\mu(\mathbf{z})=\mathbf{x}} \tilde{r}(\mathbf{z})$, where $\mu : S \to V^*$ is some function (the **marginalization operator**). We say it is a **light marginalization** if $|\mathbf{z}| \in O(\text{poly}(|\mu(\mathbf{z})|))$ and μ runs in time $O(\text{poly}(|\mathbf{z}|))$.[15] Typically $\mu(\mathbf{z})$ extracts a subsequence of $\mathbf{z}$; it can be regarded as keeping the observed symbols while throwing away a polynomially bounded number of latent symbols.

Light marginalizations of ELN distributions are a reasonable formalization of latent-variable autoregressive models. They are more powerful than ELN distributions, and even include some distributions that (by Lemma 1) are not even ELNCP or ECCP:

Theorem 7. *There exists a light marginalization p of an ELN distribution, such that* support(p) *is an* NP*-complete language.*

Our proof of Theorem 7 relies on special structure of a certain NP-complete language (SAT) and does not evidently generalize to all languages in NP.

However, light marginalizations of *ELNCP* distributions are more powerful still,[16] and can have *any* language $\in$ NP or even NP/poly (§2.4) as support:

Theorem 8. *The following statements are equivalent for any nonempty $L \subseteq V^*$:*

(a) $L \in$ NP/poly.

(b) *L is the support of a light marginalization of an ELNCP distribution.*

(c) *L is the support of a light marginalization of an ECCP weighted language.*

Theorems 7 and 8 make use of unrestricted latent-variable autoregressive models. There exist more practical restricted families of such models that admit tractable computation of $p(\mathbf{x})$ (Lafferty et al., 2001; Rastogi et al., 2016; Wu et al., 2018; Buys and Blunsom, 2018). Such models are EC (and indeed, typically ELN) — but this limits their expressivity, by Theorem 1. Both Lin et al. (2019) and Buys and Blunsom (2018) observed that such models yield worse empirical results than models that do not have tractable exact inference methods. The tractability requirement is dropped in "self-talk" (blixt, 2020; Gontier et al., 2020; Shwartz et al., 2020), where a neural autoregressive language model generates an analysis of the prefix $\hat{\mathbf{x}}$ via latent intermediate symbols before predicting the next output symbol.[17]

We remark that for autoregressive models, the *position* of the latent variables is significant. Marginalizing out latent variables at the *end* of the string adds no power. More precisely, if an ELNCP distribution is over strings $\mathbf{z}$ of the form $\mathbf{x}\#\mathbf{y}$, then its marginalization

[14] We independently conceived of and implemented the REBM idea proposed in Bakhtin et al. (2021). Details of neural architecture choice, model parameter sizes, training regimen, and evaluation (Appendices B–D) differ between our work and theirs, which also reported positive empirical results (on different datasets). We regard the two independent positive findings as a strong indication that the REBM design is effective.

[15] WLOG, μ can be required to run in linear time $O(|\mathbf{z}|)$, as it does in our constructions below.

[16] The capacity established by Theorem 8 does not need the full power of marginalization. We could similarly define light *maximizations* of ELNCP distributions, $\tilde{p}(\mathbf{x}) = \max_{\mathbf{z}:\mu(\mathbf{z})=\mathbf{x}} \tilde{r}(\mathbf{z})$. Replacing sum by max does not change the support.

[17] Here the marginal distribution of the next observed symbol can require superpolynomial time to compute (if #P $\neq$ FP, which follows from NP $\not\subseteq$ P/poly). Theorem 1 could likewise be evaded by *other* autoregressive approaches that invest superpolynomial computation in predicting the next symbol (Graves, 2016). Each autoregressive step might explicitly invoke lookahead or reasoning algorithms, just as feed-forward network layers can invoke optimizers or solvers (Amos and Kolter, 2017; Wang et al., 2019b).

via $\mu(\mathbf{x}\#\mathbf{y}) = \mathbf{x}$ can be expressed more simply as an ELNCP language. Thus, by Theorem 2, marginalizations of such distributions cannot have arbitrary NP languages as support. Our proofs of Theorems 7 and 8 instead use latent strings of the form $\mathbf{y}\#\mathbf{x}$, where all latent variables precede all observed ones (as in Kingma and Welling, 2014). (This simple design can always be used without loss of generality.) Trying to reorder those latent strings as $\mathbf{x}\#\mathbf{y}$ while preserving their weights would have yielded a non-ELNCP distribution $p(\mathbf{x}\#\mathbf{y})$ (because if it were ELNCP, then $p(\mathbf{x})$ would be ELNCP also, and we know from Lemma 1 that it cannot be for any distribution whose support is an NP-complete language).

How about lightly marginalizing ECCP languages instead of ELNCP ones? This cannot model any additional *unweighted* languages, by Theorem 8. But it may be able to model more probability distributions. One can easily construct a light marginalization p of an ECCP distribution such that $\#(\phi) = c_n \cdot p(\boldsymbol{\phi})$, where $\#(\phi)$ is the number of satisfying assignments of ϕ and the constant c_n depends only on $n = |\boldsymbol{\phi}|$. We conjecture that this is not possible with lightly marginalized ELNCP distributions.

4.3 Lookup models

§2.3 noted that with exponential growth in stored parameters, it is possible to fit any weighted language up to length n, with local probabilities computed in only $O(n)$ time by lookup. Of course this rapidly becomes impractical as n increases, even if the amount of training data increases accordingly. However, there has been some recent movement toward storage-heavy models. Such models are typically semiparametric: they use a parametric neural model, such as an autoregressive model, together with an external knowledge base of text strings or factoids that are not memorized in the layer weights. The neural model generates queries against the knowledge base and combines their results. Examples include kNNLMs (Khandelwal et al., 2020) and semiparametric LMs (Yogatama et al., 2021). The knowledge base grows linearly with the training data rather than compressing the data into a smaller parameter vector. It is in fact a copy of the training data, indexed to allow fast lookup (Indyk and Motwani, 1998). (Preparing the index is much cheaper than neural network training.) Access to the large knowledge base may reduce the amount of computation needed to find the local conditional probabilities, much as in the trie construction of §2.3.

5 Related work

Chen et al. (2018) show that it is hard to map *RNN parameters* to properties of the resulting autoregressive weighted language, such as consistency ($Z = 1$). We focus on cases where the RNN parameters are already known to be consistent, so the RNN efficiently maps a *string* $\hat{\mathbf{x}}$ to its local conditional distribution $p(\cdot \mid \hat{\mathbf{x}})$. Our point is that for some weighted languages, this is not possible (even allowing polynomially larger RNNs for longer strings), so consistent RNNs and their ilk cannot be used to describe such languages.

In a Bayes network — which is really just an autoregressive model of fixed-length strings — approximate marginal inference is NP-hard (Roth, 1996). Assuming NP $\not\subseteq$ P/poly and the grid-minor hypothesis, Chandrasekaran et al. (2008, Theorem 5.6) further showed that for any infinite sequence of graphs $G_1, G_2, \ldots$ where G_n has treewidth n, there is no sequence of algorithms $M_1, M_2, \ldots$ such that M_n performs approximate marginal inference in time $O(\text{poly}(n))$ on graphical models of structure G_n. This remarkable negative result says that in *any* graph sequence of unbounded treewidth, approximating the normalizing constant for G_n given arbitrary parameters is hard (not $O(\text{poly}(n))$), even with advice strings. Our negative result (Theorem 4) focuses on *one particular* infinite weighted language, showing that approximating local conditional probabilities given an arbitrary length-n prefix is hard in the same way. (So this language cannot be captured by an RNN, even with advice strings.)

6 Conclusion and future work

Autoregressive models are suited to those probability distributions whose prefix probabilities are efficiently computable. This efficiency is convenient for training and sampling. But unless we sacrifice it and allow runtime or parameter size to grow superpolynomially in input length, autoregressive models are less expressive than models whose prefix probabilities expensively marginalize over suffixes or latent variables.

All model families we have discussed in this paper can be seen as making compromises between different desiderata (Table 1). Natural follow-up questions include *'Are there model families that win on all fronts?' 'What are other modeling desiderata?'*

While *some* languages $\in$ P cannot be supports of ELNCPs, we do not know if the same can be said for *most* languages $\in$ P. This problem seems to be closely related to the average complexity of NP-complete languages, where most questions remain open (Levin, 1986; Bogdanov and Trevisan, 2006).

Acknowledgements

We thank the anonymous reviewers for their comments. We also thank our colleagues at Johns Hopkins University, Facebook, and Carnegie Mellon University for their comments on earlier versions of the manuscript. This material is based upon work at Johns Hopkins University supported by the National Science Foundation under Grant No. 1718846. It does not represent the views of Microsoft (where Dr. Eisner is also a paid employee, in an arrangement that has been reviewed and approved by the Johns Hopkins University in accordance with its conflict of interest policies).

References

Brandon Amos and J. Zico Kolter. 2017. OptNet: Differentiable optimization as a layer in neural networks. In *ICML*.

Sanjeev Arora and Boaz Barak. 2009. *Computational Complexity: a Modern Approach*. Cambridge University Press.

Anton Bakhtin, Yuntian Deng, Sam Gross, Myle Ott, Marc'Aurelio Ranzato, and Arthur Szlam. 2021. Residual energy-based models for text generation. *JMLR*, 22(40):1–41.

Emily M. Bender, Timnit Gebru, Angelina McMillan-Major, and Shmargaret Shmitchell. 2021. On the dangers of stochastic parrots: Can language models be too big? 🦜. In *FAccT*.

blixt. 2020. Re: Teaching gpt-3 to identify nonsense. `https://news.ycombinator.com/item?id=23990902`. Online (accessed Oct 23, 2020).

Andrej Bogdanov and Luca Trevisan. 2006. Average-case complexity. *Foundations and Trends in Theoretical Computer Science*, 2(1):1–106.

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language models are few-shot learners.

Jan Buys and Phil Blunsom. 2018. Neural syntactic generative models with exact marginalization. In *NAACL*.

Shu Cai and Kevin Knight. 2013. Smatch: an evaluation metric for semantic feature structures. In *ACL*.

Venkat Chandrasekaran, Nathan Srebro, and Prahladh Harsha. 2008. Complexity of inference in graphical models. In *UAI*.

Yining Chen, Sorcha Gilroy, Andreas Maletti, Jonathan May, and Kevin Knight. 2018. Recurrent neural networks as weighted language recognizers. In *NAACL*.

Zhiyi Chi and Stuart Geman. 1998. Estimation of probabilistic context-free grammars. *Computational Linguistics*, 24(2):299–305.

Rewon Child, Scott Gray, Alec Radford, and Ilya Sutskever. 2019. Generating long sequences with sparse transformers. *ArXiv*, abs/1904.10509.

Stephen A. Cook. 1971. The complexity of theorem-proving procedures. In *STOC*.

John DeNero and D. Klein. 2008. The complexity of phrase alignment problems. In *ACL*.

Nicolas Gontier, Koustuv Sinha, Siva Reddy, and C. Pal. 2020. Measuring systematic generalization in neural proof generation with transformers. In *NeurIPS*.

A. Graves. 2016. Adaptive computation time for recurrent neural networks. *ArXiv*, abs/1603.08983.

Michael U. Gutmann and Aapo Hyvärinen. 2010. Noise-contrastive estimation: A new estimation principle for unnormalized statistical models. In *AISTATS*.

Michael U. Gutmann and Aapo Hyvärinen. 2012. Noise-contrastive estimation of unnormalized statistical models, with applications to natural image statistics. *JMLR*, 13(11):307–361.

Y. Huang, A. Sethy, K. Audhkhasi, and B. Ramabhadran. 2018. Whole sentence neural language models. In *ICASSP*.

Aapo Hyvärinen. 2005. Estimation of non-normalized statistical models by score matching. *JMLR*, 6(24):695–709.

W. Idsardi. 2006. A simple proof that optimality theory is computationally intractable. *Linguistic Inquiry*, 37:271–275.

Piotr Indyk and Rajeev Motwani. 1998. Approximate nearest neighbors: Towards removing the curse of dimensionality. In *STOC*.

Richard M. Karp and Richard J. Lipton. 1980. Some connections between nonuniform and uniform complexity classes. In *STOC*.

Urvashi Khandelwal, Omer Levy, Dan Jurafsky, Luke Zettlemoyer, and Mike Lewis. 2020. Generalization through memorization: Nearest neighbor language models. In *ICLR*.

Diederik P. Kingma and Max Welling. 2014. Auto-encoding variational Bayes. In *ICLR*.

Richard E. Ladner. 1975. The circuit value problem is log space complete for P. *SIGACT News*, 7(1):18–20.

John D. Lafferty, Andrew McCallum, and Fernando C. N. Pereira. 2001. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *ICML*.

Yann LeCun, Sumit Chopra, Raia Hadsell, Marc'Aurelio Ranzato, and Fu-Jie Huang. 2006. A tutorial on energy-based learning. In *Predicting Structured Data*. MIT Press.

Leonid A. Levin. 1986. Average case complete problems. *SIAM Journal on Computing*, 15:285–286.

Chu-Cheng Lin, Hao Zhu, Matthew R. Gormley, and Jason Eisner. 2019. Neural finite-state transducers: Beyond rational relations. In *NAACL*.

Zhuang Ma and Michael Collins. 2018. Noise contrastive estimation and negative sampling for conditional models: Consistency and statistical efficiency. In *EMNLP*.

Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. 2017. Pointer sentinel mixture models. *ArXiv*, abs/1609.07843.

Tomas Mikolov, Stefan Kombrink, Anoop Deoras, Lukas Burget, and Jan Honza Cernocky. 2011. RNNLM—Recurrent neural network language modeling toolkit. In *IEEE Automatic Speech Recognition and Understanding Workshop*.

Randall Munroe. 2009. *My Hobby: Embedding* NP-*Complete Problems in Restaurant Orders*. Online (accessed May 29, 2020).

Alexei G. Myasnikov and Alexander N. Rybalov. 2008. Generic complexity of undecidable problems. *The Journal of Symbolic Logic*, 73(2):656–673.

Aaron van den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew Senior, and Koray Kavukcuoglu. 2016. WaveNet: A generative model for raw audio. *ArXiv*, abs/1609.03499.

Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. 2019. Language models are unsupervised multitask learners.

Pushpendre Rastogi, Ryan Cotterell, and Jason Eisner. 2016. Weighting finite-state transductions with neural context. In *NAACL*.

Ronald Rosenfeld, Stanley Chen, and Xiaojin Zhu. 2001. Whole-sentence exponential language models: A vehicle for linguistic-statistical integration. *Computer Speech & Language*, 15(1):55–73.

Dan Roth. 1996. On the hardness of approximate reasoning. *Artificial Intelligence*, 82(1–2):273–302.

Ben Sandbank. 2008. Refining generative language models using discriminative learning. In *EMNLP*.

Vered Shwartz, Peter West, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. 2020. Unsupervised commonsense question answering with self-talk. In *EMNLP*.

Hava T. Siegelmann and Eduardo D. Sontag. 1992. On the computational power of neural nets. In *COLT*.

Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. 2014. Sequence to sequence learning with neural networks. In *NeurIPS*.

Benigno Uria, Marc-Alexandre Côté, Karol Gregor, Iain Murray, and Hugo Larochelle. 2016. Neural autoregressive distribution estimation. *JMLR*, 17(1):7184–7220.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *NeurIPS*.

Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R. Bowman. 2019a. GLUE: A multi-task benchmark and analysis platform for natural language understanding. In *ICLR*.

Po-Wei Wang, P. Donti, B. Wilder, and J. Z. Kolter. 2019b. SATNet: Bridging deep learning and logical reasoning using a differentiable satisfiability solver. In *ICML*.

Sean Welleck, Ilia Kulikov, Jaedeok Kim, Richard Yuanzhe Pang, and Kyunghyun Cho. 2020. Consistency of a recurrent language model with respect to incomplete decoding. In *EMNLP*.

Shijie Wu, Pamela Shapiro, and Ryan Cotterell. 2018. Hard non-monotonic attention for character-level transduction. In *EMNLP*.

Yelp. Yelp open dataset. `https://www.yelp.com/dataset`.

Dani Yogatama, Cyprien de Masson d'Autume, and Lingpeng Kong. 2021. Adaptive semiparametric language models. *ArXiv*, abs/2102.02557.

Rowan Zellers, Ari Holtzman, Hannah Rashkin, Yonatan Bisk, Ali Farhadi, Franziska Roesner, and Yejin Choi. 2019. Defending against neural fake news. In *NeurIPS*.

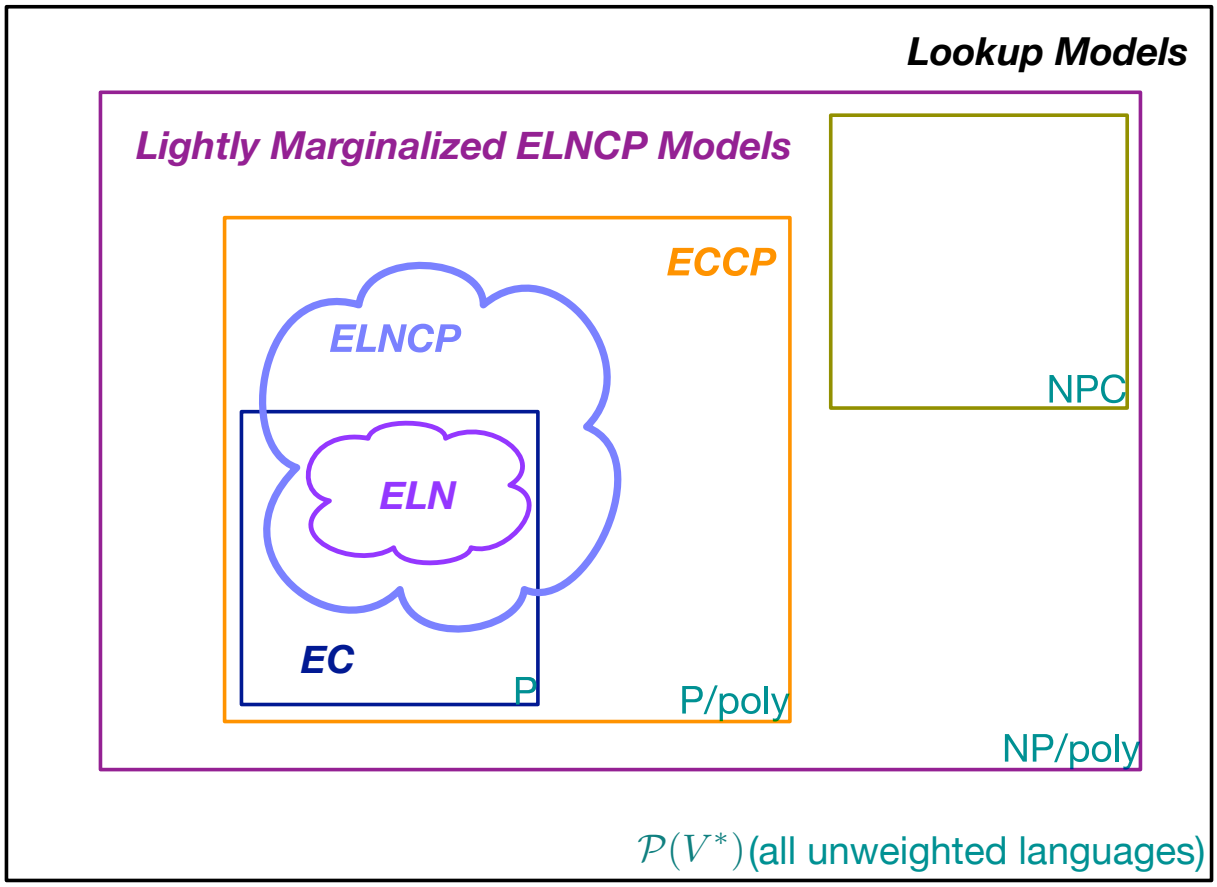

Figure 2: The space of unweighted languages. We assume in this diagram that NP $\not\subseteq$ P/poly. Each rectangular outline corresponds to a complexity class (named in its lower right corner) and encloses the languages whose decision problems fall into that class. Each ***bold-italic label*** (colored to match its shape outline) names a model family and encloses the languages that can be expressed as the *support* of some *weighted* language in that family. All induced partitions in the figure are non-empty sets: shape A properly encloses shape B if and only if language class A is a strict superset of language class B. As mentioned in Table 1, standard autoregressive models (ELN models) have support languages that form a strict subset of P (Lemmas 1 and 2, Theorem 5, and §2.4). ELNCP models (§3.1) extend ELN models by allowing the parameter size to grow polynomially in string length, allowing them to capture both more languages inside P (Theorem 6) and languages outside P (including undecidable but sparse languages) that can be characterized autoregressively with the help of these compact parameters. All of those languages belong in the class P/poly. Theorem 2 establishes that energy-based (EC) and ECCP models go strictly further than ELN and ELNCP models, respectively (Theorem 2): they correspond to the *entire* classes P and P/poly (Lemma 1). However, even ECCP does not capture any NP-complete languages under our assumption NP $\not\subseteq$ P/poly. Allowing a polynomial number of latent symbols extends the power further still: lightly marginalized ELNCP or ECCP distributions cover exactly the languages $\in$ NP/poly (Theorem 8). Finally, if we were to drop the requirement that the parameters $\boldsymbol{\Theta}$ must be compact, we could store lookup tries to model any weighted language (§4.3).

A Proofs

Lemma 1. *For any $L \in$ P, there exists an EC weighted language with support L. For any $L \in$ P/poly, there exists an ECCP language with support L. But for any $L \in$ NP-complete, there exists no ECCP language with support L (assuming NP $\not\subseteq$ P/poly).*

This simple lemma relates our classes EC and ECCP of *weighted* languages to the complexity classes P and P/poly of their supports, which are *unweighted* formal languages (§2). It holds because computing a string's weight can be made as easy as determining whether that weight is nonzero (if we set the weights in a simple way), but is certainly no easier. We spell out the trivial proof to help the reader gain familiarity with the formalism.

Proof. Given L, define a weighted language $\tilde{p}$ with support L by $\tilde{p}(\mathbf{x}) = 1$ if $\mathbf{x} \in L$ and $\tilde{p}(\mathbf{x}) = 0$ otherwise.

If $L \in P$, then clearly $\tilde{p}$ is EC since the return value of 1 or 0 can be determined in polytime.

If $L \in$ P/poly, L can be described as a tuple $(M, \boldsymbol{\Theta})$ following our characterization in §2.4. It is easy to show that $\tilde{p}$ is ECCP, using the same polynomially-sized advice strings $\boldsymbol{\Theta}$. We simply construct $M^{\tilde{\mathrm{p}}}$ such that $M^{\tilde{\mathrm{p}}}(\boldsymbol{\theta}_n)$ returns 1 or 0 on input $\mathbf{x}$ according to whether $M(\boldsymbol{\theta}_n)$ accepts or rejects $\mathbf{x}$. Both $M^{\tilde{\mathrm{p}}}(\boldsymbol{\theta}_n)$ and $M^{\tilde{\mathrm{p}}}(\boldsymbol{\theta}_n)(\mathbf{x})$ are computed in time $O(\mathrm{poly}(n))$ if $|\mathbf{x}| = n$. (The technical construction is that $M^{\tilde{\mathrm{p}}}$ simulates the operation of M on the input $\boldsymbol{\theta}_n$ to obtain the description of the Turing machine $M_n = M(\boldsymbol{\theta}_n)$, and then outputs a slightly modified version of this description that will write 1 or 0 on an output tape.)

For the second half of the lemma, we use the reverse construction. Suppose $\tilde{p}$ is an ECCP weighted language with support L. $\tilde{p}$ can be characterized by a tuple $(M^{\tilde{\mathrm{p}}}, \boldsymbol{\Theta})$. It is easy to show that $L \in$ P/poly, using the same polynomially-sized advice strings $\boldsymbol{\Theta}$. We simply construct M such that $M(\boldsymbol{\theta}_n)$ accepts $\mathbf{x}$ iff $M^{\tilde{\mathrm{p}}}(\boldsymbol{\theta}_n)(\mathbf{x}) > 0$. Then by the assumption, $L \notin$ NP-complete. □

Lemma 2. *An ELNCP model $\tilde{p}$ is also ECCP. Likewise, an ELN model is also EC.*

Proof. Let $\tilde{p}$ be an ELNCP language. This implies that $\tilde{p}$ is normalizable, so let $p(\mathbf{x}) \triangleq \tilde{p}(\mathbf{x}) \,/\, Z$ as usual. Specifically, let M^{q} efficiently locally normalize $\tilde{p}$ with compact parameters $\boldsymbol{\Theta}^{\mathrm{q}} = \{\boldsymbol{\theta}^{\mathrm{q}}_n \mid n \in \mathbb{N}\}$. It is simple to define a Turing machine M^{r} that maps each parameter string $\boldsymbol{\theta}^{\mathrm{q}}_n$ to a Turing machine r_n, where $r_n(\mathbf{x})$ simply computes $\left(\prod_{t=1}^{n} q_n(x_t \mid \mathbf{x}_{<t})\right) \cdot q_n(\$ \mid \mathbf{x})$. Then for all $\mathbf{x}$ of length n, $r_n(\mathbf{x}) = \left(\prod_{t=1}^{n} p(x_t \mid \mathbf{x}_{<t})\right) \cdot p(\$ \mid \mathbf{x})$, by the definition of local normalization, and thus $r_n(\mathbf{x}) = p(\mathbf{x})$.

M^{r} can be constructed by incorporating the definition of M^{q}, so that $r_n = M^{\mathrm{r}}(\boldsymbol{\theta}^{\mathrm{q}}_n)$ can include $q_n = M^{\mathrm{q}}(\boldsymbol{\theta}^{\mathrm{q}}_n)$ as a subroutine. This allows r_n to query q_n for local conditional probabilities and multiply them together.

- Since M^{q} runs in polytime, it is straightforward for this construction to ensure that M^{r} runs in polytime as well.
- Since $q_n(\cdot \mid \hat{\mathbf{x}}) \in O(\mathrm{poly}(n))$, this construction can ensure that r_n runs in polytime as well.
- We were given that $|\boldsymbol{\theta}^{\mathrm{q}}_n| \in O(\mathrm{poly}(n))$ (compact parameters).

Since p is the weighted language defined by $(M^{\mathrm{r}}, \boldsymbol{\Theta}^{\mathrm{q}})$, and M^{r} and $\boldsymbol{\Theta}^{\mathrm{q}}$ have the properties just discussed, we see that p is efficiently computable with compact parameters (ECCP). Therefore $\tilde{p}(\mathbf{x}) = Zp(\mathbf{x})$ is also ECCP.

In the case where $\tilde{p}$ is more strongly known to be ELN (the parameters $\boldsymbol{\Theta}^{\mathrm{q}}$ are not needed), a simplification of this argument shows that it is EC. □

Theorem 1. *Assuming NP $\not\subseteq$ P/poly, there exists an efficiently computable normalizable weighted language $\tilde{p}$ that is not ELNCP.*

Proof. The proof was sketched in §3.2. Here we fill in the details.

The unweighted language $\tilde{p}$ defined in that section is efficiently computable via the following simple algorithm that outputs $\tilde{p}(\mathbf{x})$ given $\mathbf{x} \in \mathbb{B}^*$. If $\mathbf{x}$ has a prefix that encodes a formula ϕ, and the remainder of $\mathbf{x}$ is a satisfying assignment $\mathbf{a}$ to the variables of ϕ, then return $(\frac{1}{3})^{|\mathbf{x}|+1}$. Otherwise return 0. This algorithm can be made to run in polynomial time because whether an assignment satisfies a formula can be determined in polynomial time (a fact that is standardly used to establish that SAT $\in$ NP).

Given a formula ϕ with variables $A_1, \ldots, A_j$, we define $\phi' = (\neg A_1 \wedge \neg A_2 \wedge \ldots \wedge \neg A_j \wedge \neg A_{j+1}) \vee (A_1 \wedge \mathrm{Shift}(\phi))$, where $\mathrm{Shift}(\phi)$ is a version of ϕ in which A_i has been renamed to A_{i+1} for all $1 \le i \le j$. It is obvious that ϕ' and p have the properties stated in the proof sketch. The strings in L that begin with $\boldsymbol{\phi}'$ are precisely the strings of the form $\boldsymbol{\phi}'\mathbf{a}'$ where $\mathbf{a}'$ is a satisfying assignment of ϕ' — which happen just when $\mathbf{a}' = \mathbf{0}^{j+1}$ or $\mathbf{a}' = \mathbf{1}\mathbf{a}$ where $\mathbf{a}$ is a satisfying

assignment of ϕ. At least one string in L begins with $\boldsymbol{\phi}'$, namely $\boldsymbol{\phi}'\mathbb{0}^{j+1}$, so $Z(\boldsymbol{\phi}') > 0$. Moreover, $Z(\boldsymbol{\phi}'1) > 0$ iff ϕ has any satisfying assignments. Therefore the local probability $p(1 \mid \boldsymbol{\phi}') = Z(\boldsymbol{\phi}'1) \,/\, Z(\boldsymbol{\phi}')$ is defined (see §2.1), and is > 0 iff $\textsc{Sat}(\phi)$.

Notice that the formal problem used in the proof is a version of Sat whose inputs are encoded using the same prefix-free encoding function enc that was used by our definition of L in §3.2. We must choose this encoding function to be concise in the sense that $\boldsymbol{\phi} \triangleq \text{enc}(\phi)$ can be converted to and from the conventional encoding of ϕ in polynomial time. This ensures that our version of Sat is $\leq_m^P$-interreducible with the conventional version and hence NP-complete. It also ensures that there is a polynomial function f such that $|\boldsymbol{\phi}'| \leq f(|\boldsymbol{\phi}|)$, as required by the proof sketch, since there is a polynomial-time function that maps $\boldsymbol{\phi} \to \phi \to \phi' \to \boldsymbol{\phi}'$ and the output length of this function is bounded by its runtime. This is needed to show that our version of Sat is in P/poly.

Specifically, to show that the existence of $(M^{\text{q}}, \boldsymbol{\Theta}^{\text{q}})$ implies $\textsc{Sat} \in \text{P/poly}$, we use it to construct an appropriate pair $(M, \boldsymbol{\Theta})$ such that $(M(\boldsymbol{\theta}_n))(\boldsymbol{\phi}) = \textsc{Sat}(\phi)$ if $|\boldsymbol{\phi}| = n$. As mentioned in the proof sketch, we define $\boldsymbol{\Theta}$ by $\boldsymbol{\theta}_n = \boldsymbol{\theta}^{\text{q}}_{f(n)}$, and observe that $|\boldsymbol{\theta}_n| \in O(\text{poly}(n))$ (thanks to compactness of the parameters $\boldsymbol{\Theta}^{\text{q}}$ and the fact that f is polynomially bounded). Finally, define $M(\boldsymbol{\theta}_n)$ to be a Turing machine that maps its input $\boldsymbol{\phi}$ of length n to $\boldsymbol{\phi}'$ of length $\leq f(n)$, then calls $M^{\text{q}}(\boldsymbol{\theta}_n) = M^{\text{q}}(\boldsymbol{\theta}^{\text{q}}_{f(n)})$ on $\boldsymbol{\phi}'1$ to obtain $p(1 \mid \boldsymbol{\phi}')$, and returns true or false according to whether $p(1 \mid \boldsymbol{\phi}') > 0$. Computing $\boldsymbol{\phi}'$ takes time polynomial in n (thanks to the properties of enc). Constructing $M^{\text{q}}(\boldsymbol{\theta}_{f(n)})$ and calling it on $\boldsymbol{\phi}'$ each take time polynomial in n (thanks to the properties of f and M^{q}). □

Remark on conditional models. While we focus on modeling *joint* sequence probabilities in this work, we note that in many applications it often suffices to just model *conditional* probabilities (Sutskever et al., 2014). Unfortunately, our proof of Theorem 1 above implies that ELNCPs do not make good conditional models either: specifically, there exists $\boldsymbol{\phi}$ such that deciding whether $p(1 \mid \boldsymbol{\phi}) > 0$ is NP-hard, and thus beyond ELNCP's capability.

Remark on irrationality. In our definitions of ECCP and ELNCP languages, we implicitly assumed that the Turing machines that return weights or probabilities would write them in full on the output tape, presumably as the ratio of two integers. Such a Turing machine can only return rational numbers.

But then our formulation of Theorem 1 allows another proof. We could construct $\tilde{p}$ such that the local conditional probabilities $p(x \mid \hat{\mathbf{x}}) \triangleq Z(\hat{\mathbf{x}}x)/Z(\hat{\mathbf{x}})$ are sometimes irrational. In this case, they cannot be output exactly by a Turing machine, implying that $\tilde{p}$ is not ELNCP. However, this proof exposes only a trivial weakness of ELNCPs, namely the fact that they can only define distributions whose local marginal probabilities are rational.

We can correct this weakness by formulating ELNCP languages slightly differently. A real number is said to be **computable** if it can be output by a Turing machine to any desired precision. That Turing machine takes an extra input b which specifies the number of bits of precision of the output. Similarly, our definitions of ECCP and ELNCP can be modified so that their respective Turing machines $\tilde{p}_n$ and q_n take this form, are allowed to run in time $O(\text{poly}(n + b))$, and have access to the respective parameter vectors $\boldsymbol{\Theta}^{\text{p}}_{n+b}$ and $\boldsymbol{\Theta}^{\text{q}}_{n+b}$. Since some of our results concern the ability to distinguish zero from small values (arbitrarily small in the case of Theorem 6), our modified definitions also require $\tilde{p}_n$ and q_n to output a bit indicating whether the output is *exactly* zero. For simplicity, we suppressed these technical details from our exposition.

Relatedly, in §4.3, we claimed that lookup models can fit any weighted language up to length n. This is not strictly true if the weights can be irrational. A more precise statement is that for any weighted language $\tilde{p}$, there is a lookup model that maps $(\mathbf{x}, b)$ to the first b bits of $\tilde{p}(\mathbf{x})$. Indeed, this holds even when $\tilde{p}(\mathbf{x})$ is uncomputable.

Remark on computability. In §2.1 we claimed that any weighted language $\tilde{p}$ that has a finite and strictly positive Z can be normalized as $p(\mathbf{x}) = \tilde{p}(\mathbf{x})/Z$. However, Z may be uncomputable: that is, there is no algorithm that takes number of bits of precision b as input, and outputs an approximation of Z within b bits of precision. Therefore, even if $\tilde{p}$ is computable, p may have weights that are not merely irrational but even *uncomputable*. An example appears in the proof of Theorem 6 below. Weighted language classes (*e.g.* ELNCP) that only model normalized languages will not be able to model such languages, simply because the partition function is uncomputable.

However, our proof of Theorem 1 does not rely on this issue, because the $\tilde{p}$ that it exhibits happens to have a computable Z. For any b, Z may be computed to b bits of precision as the explicit sum $\sum_{\mathbf{x}:|\mathbf{x}|\leq N} \tilde{p}(\mathbf{x})$

for a certain large N that depends on b.

Remark on RNNs. Our proof of Theorem 1 showed that our problematic language $\tilde{p}$ is efficiently computable (though not by any locally normalized architecture with compact parameters). Because this paper is in part a response to popular neural architectures, we now show that $\tilde{p}$ can in fact be computed efficiently *by a recurrent neural network (RNN)* with compact parameters. Thus, this is an example where a simple globally normalized RNN parameterization is fundamentally more efficient (in runtime or parameters) than any locally normalized parameterization of any architecture (RNN, Transformer, etc.).

Since we showed that $\tilde{p}$ is efficiently computable, the existence of an RNN implementation is established in some sense by the ability of finite rational-weighted RNNs to simulate Turing machines (Siegelmann and Sontag, 1992), as well as an extension to Chen et al. (2018, Thm. 11) to a family of RNNs, where each RNN instance also takes some formula encoding as input. However, it is straightforward to give a concrete construction, for each $n \in \mathbb{N}$, for a simple RNN that maps each string $\mathbf{x} \in \mathbb{B}^n$ to $\tilde{p}(\mathbf{x})$. Here $\tilde{p}(\mathbf{x})$ will be either $(\frac{1}{3})^{n+1}$ or 0, according to whether $\mathbf{x}$ has the form $\boldsymbol{\phi}\mathbf{a}$ where $\boldsymbol{\phi}$ encodes a 3-CNF-SAT formula ϕ that is satisfied by $\mathbf{a}$.[18] The basic idea is that $\boldsymbol{\phi}$ has $j \leq n$ variables, so there are only $O(n^3)$ possible 3-CNF clauses. The RNN allocates one hidden unit to each of these. When reading $\boldsymbol{\phi}\mathbf{a}$, each clause encountered in $\boldsymbol{\phi}$ causes the corresponding hidden unit to turn on, and then each literal encountered in $\mathbf{a}$ turns off the hidden units for all clauses that would be satisfied by that literal. If any hidden units remain on after $\mathbf{x}$ has been fully read, then $\boldsymbol{\phi}$ was not satisfied by $\mathbf{a}$, and the RNN's final output unit should return 0. Otherwise it should return $(\frac{1}{3})^{n+1}$, which is constant for this RNN. To obtain digital behaviors such as turning hidden units on and off, it is most convenient to use ramp activation functions for the hidden units and the final output unit, rather than sigmoid activation functions. Note that our use of a separate RNN M_n^{RNN} for each input length n is an example of using more hidden units for larger problems, a key idea that we introduced in §2.3 in order to look at asymptotic behavior. The RNN's parameter sequence $\boldsymbol{\Theta}^{\text{RNN}} = \{\boldsymbol{\theta}_n^{\text{RNN}} \mid n \in \mathbb{N}\}$ is obviously compact, as $\boldsymbol{\theta}_n^{\text{RNN}}$ only has to store the input length n. With our alphabet $\mathbb{B}$ for $\tilde{p}$, $|\boldsymbol{\theta}_n^{\text{RNN}}| \in O(\log n)$.

[18] The restriction to 3-CNF-SAT formulas is convenient, but makes this a slightly different definition of L and $\tilde{p}$ than we used in the proofs above. Those proofs can be adjusted to show that this $\tilde{p}$, too, cannot be efficiently locally normalized with compact parameters. The only change is that in the construction of Theorem 1, ϕ' must be converted to 3-CNF. The proof then obtains its contradiction by showing that 3-CNF-SAT $\in$ P/poly (which suffices since 3-CNF-SAT is also NP-complete).

Lemma 3. *Let $\tilde{p}, \tilde{q}$ be normalizable weighted languages with* $\text{support}(\tilde{p}) \neq \text{support}(\tilde{q})$. *Then* $\exists \mathbf{x}_1, \mathbf{x}_2 \in V^*$ *such that* $\tilde{p}(\mathbf{x}_1) < \tilde{p}(\mathbf{x}_2)$ *but* $\tilde{q}(\mathbf{x}_1) \geq \tilde{q}(\mathbf{x}_2)$.

Proof. Suppose that the claim is false, *i.e.*, $\tilde{p}$ and $\tilde{q}$ have the same ranking of strings. Then the minimum-weight strings under $\tilde{p}$ must also be minimum-weight under $\tilde{q}$. WLOG, there exists $\mathbf{x} \in V^*$ with $\tilde{p}(\mathbf{x}) = 0$ and $\tilde{q}(\mathbf{x}) = c > 0$. Then $c > 0$ is the minimum weight of strings in $\tilde{q}$. But this is not possible for a normalizable language $\tilde{q}$, since it means that $Z_{\tilde{q}} \triangleq \sum_{\mathbf{x}' \in V^*} q(\mathbf{x}') \geq \sum_{\mathbf{x}' \in V^*} c$ diverges. □

Theorem 3. *Assuming* NP $\not\subseteq$ P/poly, *there exists an efficiently computable normalizable weighted language $\tilde{p}$ such that no ELNCP $\tilde{q}$ with* $\text{support}(\tilde{q}) \supseteq \text{support}(\tilde{p})$ *has* $\tilde{p}(\mathbf{x}_1) < \tilde{p}(\mathbf{x}_2) \Rightarrow \tilde{q}(\mathbf{x}_1) < \tilde{q}(\mathbf{x}_2)$ *for all* $\mathbf{x}_1, \mathbf{x}_2 \in V^*$. *Indeed, any such $\tilde{q}$ has a counterexample where* $\tilde{p}(\mathbf{x}_1) = 0$. *Moreover, there is a polynomial $f_{\tilde{q}} : \mathbb{N} \to \mathbb{N}$ such that a counterexample exists for* every $\mathbf{x}_1$ *such that* $\tilde{p}(\mathbf{x}_1) = 0$ *and* $\tilde{q}(\mathbf{x}_1) > 0$, *where the $\mathbf{x}_2$ in this counterexample always satisfies* $|\mathbf{x}_2| \leq f_{\tilde{q}}(|\mathbf{x}_1|)$.

Proof. Let $\tilde{p}$ be the weighted language from Theorem 2. Given an ELNCP $\tilde{q}$. By Theorem 2, $\text{support}(\tilde{q}) \neq \text{support}(\tilde{p})$, so there must exist a string $\mathbf{x}_1$ that is in one support language but not the other. With the additional assumption that $\text{support}(\tilde{q}) \supseteq \text{support}(\tilde{p})$, it must be that $\mathbf{x}_1 \in \text{support}(\tilde{q})$, so $\tilde{p}(\mathbf{x}_1) = 0$ but $\tilde{q}(\mathbf{x}_1) > 0$.

Given *any* such $\mathbf{x}_1$ with $\tilde{p}(\mathbf{x}_1) = 0$ but $\tilde{q}(\mathbf{x}_1) > 0$, we must find a $\mathbf{x}_2$ of length $O(\text{poly}(|\mathbf{x}_1|))$ with $\tilde{p}(\mathbf{x}_2) > 0$ but $\tilde{q}(\mathbf{x}_2) \leq \tilde{q}(\mathbf{x}_1)$.

To ensure that $\tilde{p}(\mathbf{x}_2) > 0$, let us use the structure of $\tilde{p}$. For any j, we can construct a tautological formula ϕ over variables $A_1, \ldots A_j$, as $\phi = (A_1 \vee \neg A_1) \wedge \cdots \wedge (A_j \vee \neg A_j)$. It follows that $\tilde{p}(\boldsymbol{\phi}\mathbf{a}) > 0$ for any $\mathbf{a} \in \mathbb{B}^j$. We will take $\mathbf{x}_2 = \boldsymbol{\phi}\mathbf{a}$ for a particular choice of j and $\mathbf{a}$.

Specifically, we choose them to ensure that $\tilde{q}(\mathbf{x}_2) \leq \tilde{q}(\mathbf{x}_1)$. Since $\tilde{q}$ is ELNCP, it is normalizable and hence has a finite Z. Thus, $\sum_{\mathbf{a} \in \mathbb{B}^j} \tilde{q}(\boldsymbol{\phi}\mathbf{a}) \leq Z$. So there must exist some $\mathbf{a} \in \mathbb{B}^j$ such that $\tilde{q}(\boldsymbol{\phi}\mathbf{a}) \leq Z/2^j$. We choose that $\mathbf{a}$, after choosing j large enough such that $Z/2^j \leq \tilde{q}(\mathbf{x}_1)$. Then $\tilde{q}(\mathbf{x}_2) = \tilde{q}(\boldsymbol{\phi}\mathbf{a}) \leq Z/2^j \leq \tilde{q}(\mathbf{x}_1)$.

To achieve the last claim of the theorem, we must also ensure that $|\mathbf{x}_2| \in O(\text{poly}(|\mathbf{x}_1|))$. Observe that $\tilde{q}(\mathbf{x}_1)$ can be computed in polytime (with access to

compact parameters), by Lemma 2. But this means that the representation of $\tilde{q}(\mathbf{x}_1) > 0$ as a rational number must have $\leq g(|\mathbf{x}_1|)$ bits for some polynomial g. Then $\tilde{q}(\mathbf{x}_1) \geq 2^{-g(|\mathbf{x}_1|)}$, and it suffices to choose $j = \lceil g(|\mathbf{x}_1|) + \log_2 Z \rceil$ to ensure that $Z/2^j \leq 2^{-g|\mathbf{x}_1|} \leq \tilde{q}(\mathbf{x}_1)$ as required above.

But then $j \in O(\text{poly}(|\mathbf{x}_1|))$. Also, recall that the encoding function enc used in the construction of $\tilde{p}$ is guaranteed to have only polynomial blowup (see the proof of Theorem 2). Thus, $|\mathbf{x}_2| = |\boldsymbol{\phi}| + |\mathbf{a}| = |\text{enc}(\phi)| + j \in O(\text{poly}(j)) \subseteq O(\text{poly}(|\mathbf{x}_1|))$ as required by the theorem. □

Lemma A.1. *The first part of Theorem 4 (without the modifications (a) and (b)).*

We first prove the first part of Theorem 4 (which is restated in full below). In this case we will use a distribution $\tilde{p}$ that does not have support V^* (so it does not prove modification (b)).

Proof. We take $\tilde{p}$ to be the weighted language that was defined in §3.2, which was already shown to be efficiently computable. Suppose $(M^{\mathrm{q}}, \boldsymbol{\Theta}^{\mathrm{q}}, \lambda)$ is a counterexample to Lemma A.1. Choose integer $k \geq 1$ in a manner (dependent only on λ) to be described at the end of the proof.

Suppose we would like to answer Sat where ϕ is a formula with variables $A_1, \ldots, A_j$. Define $\phi' = (\neg A_1 \wedge \neg A_2 \wedge \ldots \wedge \neg A_j \wedge \neg A_{j+1} \wedge \neg A_{j+k}) \vee (A_1 \wedge \text{Shift}(\phi))$. Note that ϕ' augments ϕ with k additional variables, namely A_1 and $A_{j+2,\ldots,j+k}$. For $k = 1$, this is the same construction as in the proof of Theorem 1. Let $n = |\boldsymbol{\phi}'|$ and note that n is polynomial in the size of ϕ (holding k constant).

The strings in $L = \text{support}(\tilde{p})$ that begin with $\boldsymbol{\phi}'$ are precisely the strings of the form $\boldsymbol{\phi}'\mathbf{a}'$ where $\mathbf{a}'$ is a satisfying assignment of ϕ'. This is achieved precisely when $\mathbf{a}' = \mathtt{0}^{j+k}$ or $\mathbf{a}' = \mathtt{1}\mathbf{a}\vec{b}$ where $\mathbf{a}$ is a satisfying assignment of ϕ and $\vec{b} \in \mathbb{B}^{k-1}$.

By our definition of $\tilde{p}$, all strings in L that begin with $\boldsymbol{\phi}'$ have equal weight under $\tilde{p}$. Call this weight w.[19] Clearly $Z(\boldsymbol{\phi}'\mathtt{0}) = w$, and $Z(\boldsymbol{\phi}'\mathtt{1}) = w \cdot 2^{k-1} \cdot$ (number of satisfying assignments of ϕ).

Recall that $p(\mathtt{0} \mid \boldsymbol{\phi}') = Z(\boldsymbol{\phi}'\mathtt{0})/(Z(\boldsymbol{\phi}'\mathtt{0}) + Z(\boldsymbol{\phi}'\mathtt{1}))$. Let us abbreviate this quantity by p. It follows from the previous paragraph that if ϕ is unsatisfiable, then $p = 1$, but if ϕ is satisfiable, then $p \leq 1/(1+2^{k-1})$. By hypothesis, p is approximated (with error probability $< 1/3$) by the possibly random quantity $(M^{\mathrm{q}}(\boldsymbol{\theta}^{\mathrm{q}}_{|\boldsymbol{\phi}'|}))(\boldsymbol{\phi}'\mathtt{0})$, which we abbreviate by q, to within a factor of λ. That is, $p \in [q/\lambda, \lambda q]$. By choosing k large enough[20] such that $[q/\lambda, \lambda q]$ cannot contain both 1 and $1/(1+2^{k-1})$, we can use q to determine whether $p = 1$ or $p \leq 1/(1+2^{k-1})$. This allows us to determine $\text{Sat}(\phi)$ in polynomial time with error probability $< 1/3$, since by hypothesis q is computable in polynomial time with compact parameters. This shows that $\text{Sat} \in \text{BPP/poly} = \text{P/poly}$, implying $\text{NP} \subseteq \text{P/poly}$, contrary to our assumption. (BPP/poly is similar to P/poly but allows M^{q} to be a bounded-error probabilistic Turing machine.) □

Theorem 4. *Assuming* $\text{NP} \not\subseteq \text{P/poly}$*, there exists an efficiently computable weighted language* $\tilde{p} : V^* \to \mathbb{R}_{\geq 0}$ *such that there is no* $(M^{\mathrm{q}}, \boldsymbol{\Theta}^{\mathrm{q}})$ *where* $\boldsymbol{\Theta}^{\mathrm{q}} = \{\boldsymbol{\theta}^{\mathrm{q}}_n \mid n \in \mathbb{N}\}$ *that satisfies all of the following properties (similar to §3.1):*

- *the parameter size* $|\boldsymbol{\theta}^{\mathrm{q}}_n|$ *grows only as* $O(\text{poly}(n))$
- $M^{\mathrm{q}}(\boldsymbol{\theta}^{\mathrm{q}}_n)$ *returns a* probabilistic *Turing machine* q_n *in time* $O(\text{poly}(n))$
- *there exists* $\lambda \geq 1$ *such that for each* $x \in V \cup \{\$\}$ *and* $\hat{\mathbf{x}} \in V^*$ *with* $|\hat{\mathbf{x}}| \leq n$ *and* $p(x \mid \hat{\mathbf{x}}) > 0$*, the probabilistic computation* $q_n(\hat{\mathbf{x}}x)$ *has probability* $> 2/3$ *of approximating* $p(x \mid \hat{\mathbf{x}})$ *to within a factor of* λ *(that is,* $q_n(\hat{\mathbf{x}}x)/p(x \mid \hat{\mathbf{x}}) \in [1/\lambda, \lambda]$*)*
- q_n *runs on those inputs* $\hat{\mathbf{x}}x$ *in time* $O(\text{poly}(n))$

Moreover, the statement above remains true

(a) when the approximation guarantee is only required to hold for prefixes $\hat{\mathbf{x}}$ *where* $\{\mathbf{x} : \hat{\mathbf{x}} \preceq \mathbf{x}\}$ *is finite (so* $p(x \mid \hat{\mathbf{x}})$ *is computable by brute force)*

(b) or, when $\text{support}(\tilde{p}) = V^*$

Proof. It remains to show that the statement remains true with modification (a) and with modification (b). For (a), the proof of Lemma A.1 suffices, since it reduces Sat to approximate local probability queries of the stated form. That is, the true local probabilities $p(x \mid \hat{\mathbf{x}})$ that can be computed with finite summations, thanks to the structure of our example language $\tilde{p}$, which guarantees that the prefix $\hat{\mathbf{x}}$ can only continue with suffixes of a fixed length that is easily determined from $\hat{\mathbf{x}}$.

For modification (b), again let $V = \mathbb{B} = \{\mathtt{0}, \mathtt{1}\}$. Choose some $\epsilon > 0$ (any choice will do), and let

$$\tilde{p}_1(\mathbf{x}) = \begin{cases} (\frac{1}{3})^{|\mathbf{x}+1|} & \text{if } \mathbf{x} = \boldsymbol{\phi}\mathbf{a} \text{ where } \boldsymbol{\phi} = \text{enc}(\phi) \\ & \quad \text{and } \mathbf{a} \text{ satisfies } \phi \\ 0 & \text{otherwise} \end{cases}$$

$$\tilde{p}_2(\mathbf{x}) = (\tfrac{1}{9})^{|\mathbf{x}+1|} > 0$$

$$\tilde{p}(\mathbf{x}) = \tilde{p}_1(\mathbf{x}) + \epsilon \cdot \tilde{p}_2(\mathbf{x})$$

[19] Specifically, each such string has length $n + j + k$, so $\tilde{p}$ gives it a weight of $w = (\frac{1}{3})^{n+j+k+1}$.

[20] It suffices to ensure that $1 + 2^{k-1} > \lambda^2$, so take any $k > 1 + \log_2(\lambda^2 - 1)$.

We use Z_1, Z_2, and Z respectively to denote normalizing constants of these three weighted languages. Note that $\tilde{p}_1$ is the weighted language that was previously used in the proofs of Theorem 1 and Lemma A.1. Our new $\tilde{p}$ is intended to be very similar while satisfying the additional condition (b). It is easy to show that $\tilde{p}$ is efficiently computable, much as we showed for $\tilde{p}_1$ in Theorem 1. Also, $\tilde{p}$ is normalizable, since $Z = Z_1 + \epsilon \cdot Z_2$, where $Z_1 \leq (\frac{1}{3})/(1 - \frac{2}{3}) = 1$ and $Z_2 = (\frac{1}{9})/(1 - \frac{2}{9}) = \frac{1}{7}$ are both finite.

The proof proceeds as in Lemma A.1, with ϕ' constructed from ϕ as before. Recall that ϕ has j variables, ϕ' has $j + k$ variables, and $|\boldsymbol{\phi}'| = n$. We may assume WLOG that the encoding function enc is such that an encoded formula always has at least as many bits as the number of variables in the formula, so $n \geq j + k$.

Notice that $Z_1(\boldsymbol{\phi}')$ sums over the satisfying assignments of ϕ', and there may be as few as one of these (if ϕ is unsatisfiable). By contrast, $Z_2(\boldsymbol{\phi}')$ sums over an infinite number of continuations with positive probability. The faster decay rate of $\frac{1}{9}$ in $\tilde{p}_2$ was chosen to keep $Z_2(\boldsymbol{\phi}')$ small relative to $Z_1(\boldsymbol{\phi}')$ despite this. Specifically,

$$
\begin{aligned}
Z_1(\boldsymbol{\phi}'\mathtt{0}) &= (\tfrac{1}{3})^{n+j+k+1} \\
Z_1(\boldsymbol{\phi}'\mathtt{1}) &= (\tfrac{1}{3})^{n+j+k+1} \cdot 2^{k-1} \\
&\qquad \cdot (\#\text{ of satisfying assignments of } \phi) \\
Z_2(\boldsymbol{\phi}'\mathtt{0}) &= (\tfrac{1}{9})^{n} \cdot \tfrac{1}{9} \cdot (\tfrac{1}{9}/(1 - \tfrac{2}{9})) \\
&= \tfrac{1}{7} \cdot (\tfrac{1}{3})^{2(n+1)} \\
&< \tfrac{1}{7} \cdot Z_1(\boldsymbol{\phi}'\mathtt{0}) \\
&\quad (\text{because } 2(n+1) > n + j + k + 1) \\
Z_2(\boldsymbol{\phi}'\mathtt{1}) &= Z_2(\boldsymbol{\phi}'\mathtt{0})
\end{aligned}
$$

As in the proof of Lemma A.1, we will show that $p(\mathtt{0} \mid \boldsymbol{\phi}')$ is much larger when ϕ is unsatisfiable. Recall that $Z(\hat{\mathbf{x}}) = Z_1(\hat{\mathbf{x}}) + \epsilon \cdot Z_2(\hat{\mathbf{x}})$. When ϕ has zero satisfying assignments,

$$
\begin{aligned}
p(\mathtt{0} \mid \boldsymbol{\phi}') &= \frac{Z(\boldsymbol{\phi}'\mathtt{0})}{Z(\boldsymbol{\phi}'\mathtt{0}) + Z(\boldsymbol{\phi}'\mathtt{1})} \\
&= \frac{Z(\boldsymbol{\phi}'\mathtt{0})}{Z_1(\boldsymbol{\phi}'\mathtt{0}) + \epsilon \cdot Z_2(\boldsymbol{\phi}'\mathtt{0}) + \epsilon \cdot Z_2(\boldsymbol{\phi}'\mathtt{1})} \\
&> \frac{Z(\boldsymbol{\phi}'\mathtt{0})}{Z_1(\boldsymbol{\phi}'\mathtt{0}) + 2 \cdot \frac{\epsilon}{7} \cdot Z_1(\boldsymbol{\phi}'\mathtt{0})}
\end{aligned}
$$

whereas if ϕ has at least one satisfying assignment, then

$$
\begin{aligned}
p(\mathtt{0} \mid \boldsymbol{\phi}') &= \frac{Z(\boldsymbol{\phi}'\mathtt{0})}{Z(\boldsymbol{\phi}'\mathtt{0}) + Z(\boldsymbol{\phi}'\mathtt{1})} \\
&< \frac{Z(\boldsymbol{\phi}'\mathtt{0})}{Z_1(\boldsymbol{\phi}'\mathtt{0}) + Z_1(\boldsymbol{\phi}'\mathtt{1})} \\
&\leq \frac{Z(\boldsymbol{\phi}'\mathtt{0})}{Z_1(\boldsymbol{\phi}'\mathtt{0}) + 2^{k-1} Z_1(\boldsymbol{\phi}'\mathtt{0})}
\end{aligned}
$$

This rewrites both probabilities in terms of $Z.(\boldsymbol{\phi}'0)$ quantities, which do not depend on the number of satisfying assignments. So now we can see that the first probability is at least $(1 + 2^{k-1})$ / $(1 + \frac{2\epsilon}{7})$ times as large as the second probability. Choose k large enough[21] such that $[q/\lambda, \lambda q]$ cannot contain both probabilities, and complete the proof as in Lemma A.1. □

Theorem 5. *The set* $\{\tilde{p} : \tilde{p}$ *is normalizable,* $\tilde{p} \in$ EC, $\tilde{p} \notin$ ELN$\}$ *is not empty.*

Theorem 5 states that some normalizable EC distributions cannot be expressed as ELN distributions. The proof is based on the undecidability of the halting problem, rather than the assumed inefficiency of the Boolean satisfiability problem. Thus, unlike Theorem 1, it does *not* rely on the assumption that NP $\not\subseteq$ P/poly, or even on the weaker assumption that P $\neq$ NP.

Proof. Given any unweighted language $L \subseteq \mathbb{B}^*$, we can define a normalizable weighted language $\tilde{p}$ with support L by $\tilde{p}(\mathbf{x}) = 1/3^{|\mathbf{x}|+1}$ for $\mathbf{x} \in L$ and $\tilde{p}(\mathbf{x}) = 0$ otherwise. Moreover, if $L \in$ P, then $\tilde{p} \in$ EC.

For our purposes, we take L to consist of all strings of the form $\mathbf{x}^{(1)}\mathbf{x}^{(2)}$, for which there exists a deterministic Turing machine M such that $\mathbf{x}^{(1)} = \text{enc}(M)$ (where enc is a prefix-free encoding function) and $\mathbf{x}^{(2)}$ encodes an accepting execution path of M on an empty input. (Such a path may be represented as a sequence of transitions of M that begins with an initial state and ends at an accepting state.) Note that any deterministic TM $\mathbf{x}^{(1)}$ can be paired with at most one accepting execution path $\mathbf{x}^{(2)}$, and cannot be paired with any $\mathbf{x}^{(2)}$ if it does not halt.

Clearly $L \in$ P: given $\mathbf{x} \in \mathbb{B}^*$, we can decide whether $\mathbf{x} \in L$ by first checking if $\mathbf{x}$ can be expressed as a concatenation of strings $\mathbf{x}^{(1)}$ and $\mathbf{x}^{(2)}$ of the required form. Then we build M from $\mathbf{x}^{(1)}$ and simulate it to check the transitions in $\mathbf{x}^{(2)}$ on M step-by-step. This can be done in $O(\text{poly}(|\mathbf{x}|))$ total time. We conclude that the $\tilde{p}$ derived from L is EC.

[21] It suffices to ensure that $(1 + 2^{k-1})/(1 + \frac{2\epsilon}{7}) > \lambda^2$, so take any $k > 1 + \log_2(\lambda^2 \cdot (1 + \frac{2\epsilon}{7}) - 1)$.

Now, $Z(\mathbf{x}^{(1)}) > 0$ iff M halts on the empty input. But this undecidable problem could be decided if there were an ELN weighted language that had support L, since then $Z(\mathbf{x}^{(1)}) \,/\, Z$ could be found as a product of local conditional probabilities, $\prod_{t=1}^{|\mathbf{x}^{(1)}|} p(x_t^{(1)} \mid \mathbf{x}_{<t}^{(1)})$, that could each be computed by a Turing machine. Therefore $\tilde{p}$ is not ELN. □

We have shown above that a certain unweighted language L is not the support of any ELN distribution. We conjecture that it is also not the support of any ELNCP distribution;[22] a proof of this would strengthen Theorem 5 to become an unconditional version of Theorem 1. However, ELNCP weighted languages do have more power than ELN weighted languages, as we now show.

Theorem 6. *The set* $\{\tilde{p} : \tilde{p}$ *is normalizable,* $\tilde{p} \in$ EC, $\tilde{p} \in$ ELNCP, $\tilde{p} \notin$ ELN$\}$ *is not empty.*

Theorem 6 justifies why this region is drawn as non-empty in Figure 2. Again, it does not rely on the assumption NP $\not\subseteq$ P/poly or P $\neq$ NP. Note that Theorem 5 can be regarded as a corollary of Theorem 6.

Proof. The weighted language $\tilde{p}$ constructed in Theorem 5 is not necessarily ELNCP. To fix this, we modify the construction to obtain a weighted language $\tilde{p}'$ with *sparse* support L'. We will again be able to show that $\tilde{p}'$ is EC and not ELN. To show that $\tilde{p}'$ is also ELNCP, we will rely on the sparsity of L', meaning that $\text{prefixes}(L') \triangleq \{\hat{\mathbf{x}}' : (\exists \mathbf{x}' \in L')\ \hat{\mathbf{x}}' \preceq \mathbf{x}'\}$ contains at most $O(\text{poly}(n))$ strings $\hat{\mathbf{x}}'$ of length $\leq n+1$. Thus, we can use $\boldsymbol{\Theta}_n^{\text{q}}$ to store all of those strings $\hat{\mathbf{x}}'$ in polynomial space, along with their $Z(\hat{\mathbf{x}}')$ values.[23] Notice that all strings $\hat{\mathbf{x}}' \notin \text{prefixes}(L')$ have $Z(\hat{\mathbf{x}}') = 0$, so they need not be stored. Now for any $\hat{\mathbf{x}}'$ of length $\leq n$, a Turing machine that consults $\boldsymbol{\theta}_n^{\text{q}}$ can compute $q(x \mid \hat{\mathbf{x}}') = Z_{\tilde{p}'}(\hat{\mathbf{x}}'x) \,/\, Z_{\tilde{p}'}(\hat{\mathbf{x}}')$ in time $O(\text{poly}(n))$ as desired, establishing that $\tilde{p}'$ is ELNCP.

We may define $\tilde{p}'$ as follows. Let $\text{sparsify}(\mathbf{x})$ be a version of $\mathbf{x}$ with many extra $\mathtt{0}$ symbols inserted: specifically, it inserts 2^t copies of $\mathtt{0}$ immediately before the t^{th} bit of $\mathbf{x}$, for all $1 \leq t \leq |\mathbf{x}|$. We construct $\tilde{p}'$ so that $\tilde{p}'(\text{sparsify}(\mathbf{x})) = \tilde{p}(\mathbf{x})$. Specifically, let $L' \triangleq \text{sparsify}(L)$. The inverse function $\text{sparsify}^{-1}(\mathbf{x}')$ is defined on exactly $\mathbf{x}' \in L'$, and is unique when defined. For all $\mathbf{x}' \in \mathbb{B}^*$, let $\tilde{p}'(\mathbf{x}') \triangleq \tilde{p}(\text{sparsify}^{-1}(\mathbf{x}'))$ if $\text{sparsify}^{-1}(\mathbf{x}')$ is defined, and $\tilde{p}'(\mathbf{x}') \triangleq 0$ otherwise. This can be computed in polytime, so $\tilde{p}'$ is EC. Also, its support L' is sparse as claimed, so $\tilde{p}'$ is ELNCP.

Finally, we claim $\tilde{p}'$ is not ELN. A given deterministic Turing machine M halts on the empty input iff $\text{enc}(M) \in \text{prefixes}(L)$ iff $\text{sparsify}(\text{enc}(M)) \in \text{prefixes}(L')$ iff $Z'(\text{sparsify}(\text{enc}(M))) > 0$. But as in the proof of Theorem 5, this would be decidable if $\tilde{p}'$ were ELN as defined in §3.1, since then we would have a Turing machine to compute the local conditional probabilities $p'(\hat{x}_t \mid \hat{\mathbf{x}}_{<t})$ for $\hat{\mathbf{x}} = \text{sparsify}(\text{enc}(M))$. □

Theorem 7. *There exists a light marginalization* p *of an ELN distribution, such that* $\text{support}(p)$ *is an* NP*-complete language.*

Proof. We will construct p such that $\text{support}(p)$ is the NP-complete language Sat of all satisfiable boolean formulas. The idea is to construct an ELN distribution r that can autoregressively generate any assignment $\mathbf{a}$ followed by any formula ϕ that is satisfied by $\mathbf{a}$. Thus, if we delete the $\mathbf{a}$ prefixes, the support consists of exactly the satisfiable formulas ϕ (or more precisely, their encodings $\boldsymbol{\phi}$).

To be more precise, we will have $\text{support}(r)$ be the language $L = \{\mathbf{a}\#\boldsymbol{\phi} \mid \mathbf{a} \in \mathbb{B}^*$ and ϕ is a formula satisfied by $\mathbf{a}\}$. This is defined similarly to the support language L in §3.2, but with the order of $\boldsymbol{\phi}$ and $\mathbf{a}$ crucially swapped: r will now generate the "solution" $\mathbf{a}$ *before* the "problem" $\boldsymbol{\phi}$. The alphabet V of this language contains at least the symbols $\{\mathtt{0}, \mathtt{1}, \#\}$, where # is a separator symbol, and any other symbols needed to encode ϕ as $\boldsymbol{\phi}$. The marginalization operator μ maps $\mathbf{a}\#\boldsymbol{\phi}$ to $\boldsymbol{\phi}$.

Let $j = |\mathbf{a}|$. As in §3.2, we will require ϕ to use all of the variables $A_1, \ldots, A_j$ (and only those variables), implying that $|\boldsymbol{\phi}| \geq j$. This ensures that marginalizing over the $j+1$ latent symbols is only light marginalization since $j+1+|\boldsymbol{\phi}| \in O(\text{poly}(|\boldsymbol{\phi}|))$. For convenience, we will also require ϕ to be a CNF formula. These requirements shrink $\text{support}(p)$ but do not affect its NP-completeness.

The remaining challenge is to construct an autoregressive distribution r whose support is L. We can think of this distribution as describing an efficient procedure for randomly generating a string from left to right so that the procedure generates the t^{th} symbol

[22] We have not attempted to prove this. Our loose intuition is that the compact parameters of an ELNCP language may help it to memorize some small part of L, but the halting problem would still be undecidable when restricted to the rest of L (Myasnikov and Rybalov, 2008).

[23] More precisely, the first b bits of $Z(\hat{\mathbf{x}}') \leq 1$ may be stored in $\boldsymbol{\Theta}_{n+b}^{\text{q}}$, when ELNCP is defined as explained in our "Remark on irrationality" above.

in time $O(\text{poly}(t))$, terminates with probability 1,[24] has positive probability of producing any string in L, and has zero probability of producing any string not in L. Below we give such a procedure.[25]

1. First, the procedure generates $\mathbf{a}\#$ as a sequence of random symbols from $\{\mathtt{0}, \mathtt{1}, \#\}$, making a uniform draw at each step. It stops immediately after generating # for the first time. The string generated before # is called $\mathbf{a}$ and we let $j = |\mathbf{a}|$. For example, $\mathbf{a} = \mathtt{010}$ and $j = 3$.
2. Second, the procedure must generate the encoding $\boldsymbol{\phi}$ of a random CNF formula ϕ that is satisfied by $\mathbf{a}$, such as $(A_2 \vee \neg A_3 \vee \neg A_2 \vee A_2) \wedge (\neg A_1)$ in our example. This involves generating a random sequence of 0 or more satisfied clauses connected by $\wedge$. At each step, the procedure decides whether to generate a new clause or end the formula. The probability of generating a new clause is ordinarily $1/2$. However, this probability is 1 if the previous clauses do not yet mention all the variables $A_1, \ldots, A_j$.

   How does it generate each satisfied clause? This involves generating a sequence of literals connected by $\vee$, at least one of which must be true. At each step of this subroutine, it uniformly chooses an integer $i \in [1, j]$, and then flips a fair coin to decide whether to add the literal A_i or $\neg A_i$ to the current clause. If the clause is now satisfied by $\mathbf{a}$ (*i.e.*, at least one of the literals is true), it then flips another fair coin to decide whether to end the clause.

r is ELN because there exists a Turing machine that computes from input $\hat{\mathbf{x}}x$ — in time $O(\text{poly}(|\hat{\mathbf{x}}|))$ — the probability that the next symbol generated after the prefix $\hat{\mathbf{x}}$ would be x, under the above procedure. As discussed in footnote 7, that probability equals $r(x \mid \hat{\mathbf{x}})$ — which is what our Turing machine is required to return — because the above procedure almost surely terminates (footnote 24), ensuring that r is a consistent probability distribution over V^* (that is, $\sum_{\mathbf{x} \in V^*} r(\mathbf{x}) = 1$). □

Theorem 8. *The following statements are equivalent for any nonempty $L \subseteq V^*$:*

(a) $L \in$ NP/poly.

(b) L is the support of a light marginalization of an ELNCP distribution.

(c) L is the support of a light marginalization of an ECCP weighted language.

Proof. (b) implies (c) since any ELNCP distribution is an ECCP weighted language (Lemma 2). (c) implies (a) by Lemma A.2 below. Finally, (a) implies (b) by Lemma A.3 below. □

Lemma A.2. *For any ECCP weighted language $\tilde{r}$, if $\tilde{p}$ is a light marginalization of $\tilde{r}$, then* $\text{support}(\tilde{p}) \in$ NP/poly.

Notice that this lemma concerns the class NP/poly, not P/poly (see §2.4). The proof is straightforward.

Proof. Suppose $\tilde{r}$ is ECCP via $(M^{\tilde{\mathbf{r}}}, \boldsymbol{\theta}^{\tilde{\mathbf{r}}})$, and μ is the marginalization operator such that $\tilde{p}(\mathbf{x}) = \sum_{\mathbf{z}:\mu(\mathbf{z})=\mathbf{x}} \tilde{r}(\mathbf{z})$. By the light marginalization assumption, there is a polynomial f such that $|\mathbf{z}| \leq f(|\mu(\mathbf{z})|)$.

To prove $\text{support}(\tilde{p}) \in$ NP/poly, we must show that there exists $(M, \boldsymbol{\Theta})$ such that for all $n \geq 0$, a *nondeterministic* Turing machine M_n can be constructed as $M(\boldsymbol{\theta}_n)$ in time $O(\text{poly}(n))$, which can in turn decide in time $O(\text{poly}(n))$ whether $\tilde{p}(\mathbf{x}) > 0$ for any $\mathbf{x}$ with $|\mathbf{x}| = n$.

Deciding $\tilde{p}(\mathbf{x}) > 0$ means deciding whether $(\exists \mathbf{z} \in V^*)\ \mu(\mathbf{z}) = \mathbf{x}$ and $\tilde{r}(\mathbf{z}) > 0$. But if $|\mathbf{x}| = n$, the first condition $\mu(\mathbf{z}) = \mathbf{x}$ implies $|\mathbf{z}| \leq f(|\mu(\mathbf{z})|) = f(|\mathbf{x}|) = f(n)$. Thus, we need M_n to nondeterministically check only the $\mathbf{z}$ of length up to $f(n)$ to see whether $\mu(\mathbf{z}) = \mathbf{x}$ and $\tilde{r}(\mathbf{z}) > 0$.

How can M_n check a string $\mathbf{z}$ of length m? It can decide the first condition $\mu(\mathbf{z}) = \mathbf{x}$ in time $O(\text{poly}(m))$, since the marginalization operator μ is a polytime function. To decide the second condition $\tilde{r}(\mathbf{z}) > 0$, it must construct the (deterministic) Turing machine $M^{\tilde{\mathbf{r}}}(\boldsymbol{\theta}^{\tilde{\mathbf{r}}}_m)$ and then apply it to $\mathbf{z}$ to obtain $\tilde{r}(\mathbf{z})$: since $\tilde{r}$ is ECCP, both steps take time $O(\text{poly}(m)) = O(\text{poly}(f(n))) \subseteq O(\text{poly}(n))$ as required.

However, this means that $M_n = M(\boldsymbol{\theta}_n)$ must have access to the parameter vectors $\boldsymbol{\theta}^{\tilde{\mathbf{r}}}_m$ for all $m \leq f(n)$. We therefore make $\boldsymbol{\theta}_n$ include this collection of parameter vectors. Each $|\boldsymbol{\theta}^{\tilde{\mathbf{r}}}_m| \in O(\text{poly}(m)) \subseteq O(\text{poly}(n))$ since $\tilde{r}$ is ECCP. So $|\boldsymbol{\theta}_n| \in O(\text{poly}(n))$ as required. □

Lemma A.3. *For any $L \in$ NP/poly, there exists a light marginalization p of an ELNCP distribution, such that* $\text{support}(p) = L$.

Lemma A.3 resembles Theorem 7, but it constructs distributions for *all* $L \in$ NP/poly, not just for *one*

[24] Phase 1 almost surely terminates after a finite number of bits. Phase 2 almost surely terminates after a finite number of clauses, and each clause almost surely terminates after a finite number of literals. "Almost surely" means "with probability 1."

[25] Our presentation here makes use of an infinite alphabet that includes symbols such as A_i and $\neg A_i$ for all $i \in \mathbb{N}_{>0}$, as well as symbols such as $\mathtt{0}$, $\mathtt{1}$, $\wedge$, $\vee$. We implicitly invoke some prefix-free encoding scheme to translate each symbol into a fixed string over the finite alphabet V.

particular $L \in$ NPC. The proof is similar but more complicated. In both cases, the goal is to demonstrate how an ELNCP distribution r can define a left-to-right stochastic string generation process such that the suffix of the generated string must be in L and can be any element of L.

Our string generation process in this case is inspired by rejection sampling, a widely used method for sampling from an energy-based model with support L. The standard scheme is to first sample a string $\mathbf{x}$ from a tractable distribution q such that $\text{support}(q) \supseteq L$, then accept the sample with an appropriate probability, which is 0 if $\mathbf{x} \notin L$. The process is repeated until a sample is finally accepted. There is no guarantee that this standard scheme will terminate in polynomial time, however. Fortunately, in our setting, we are not trying to match our sampling distribution p to a given energy-based model, but simply match its support to a given language L. We make use of the polysize parameter vectors of ELNCP languages to store certain 'fallback strings' that are guaranteed to be in the desired language L. Wherever ordinary rejection sampling would reject a string and try generating another, we switch to generating a stored fallback string of an appropriate length. This scheme places all of the rejected probability mass on the small set of fallback strings (in contrast to rejection sampling, which in effect throws away this mass and renormalizes). The advantage is that it does not iterate indefinitely. At a high level, r is a distribution over strings $\mathbf{z}$ that record traces of this generative story we describe above.

Proof. WLOG we assume L uses the alphabet $V = \{\mathtt{0}, 1, \#\}$. In the case where L is finite, the result is trivial. We simply define $r(\mathbf{x}) = 1/|L|$ for $\mathbf{x} \in L$ and $r(\mathbf{x}) = 0$ otherwise. We then take $p = r$ (a trivial marginalization). It is easy to show that r is ELN, and therefore ELNCP as desired, by constructing an appropriate Turing machine that maps $\hat{\mathbf{x}}x$ to $r(x \mid \hat{\mathbf{x}})$ in time $O(|\hat{\mathbf{x}}x|)$, for any $\hat{\mathbf{x}}$ that is a prefix of some string in L and any $x \in V \cup \{\$\}$. The finite state table of the Turing machine includes states that correspond to all possible strings $\hat{\mathbf{x}}x$, with transitions arranged in a trie. It reads the input string $\hat{\mathbf{x}}x$ from left to right to reach the state corresponding to $\hat{\mathbf{x}}x$. If it detects the end of the input while in that state, it writes $r(x \mid \hat{\mathbf{x}})$ on the output tape.

Now we consider the case where L is infinite. For each $j \in \mathbb{N}_{\geq 0}$, let the 'fallback string' $\mathbf{x}^{(j)}$ be some string in L of length $\geq j$. For definiteness, let us take it to be the shortest such string, breaking ties lexicographically. At least one such string does exist because L is infinite, so $\mathbf{x}^{(j)}$ is well-defined.

Also, since $L \in$ NP/poly (§2.4), let $(M, \boldsymbol{\Theta})$ be an ordered pair and f be a polynomial such that $M_j = M(\theta_j)$ nondeterministically accepts $\mathbf{a}$ within $\leq f(j)$ steps iff $\mathbf{a} \in L$.

As in the proof of Theorem 7, we now describe a procedure for randomly generating a string $\mathbf{z}$ from left to right. $\mathbf{z}$ will have the form $\mathbf{a}\#\mathbf{b}\#c\mathbf{d}$, where $\mathbf{d} \in L$ and the latent substring $\mathbf{a}\#\mathbf{b}\#c$ will be removed by the marginalization operator μ.

1. First we generate a random string $\mathbf{a} \in \mathbb{B}^*$ followed by #, just as in the proof of Theorem 7. Again let $j = |\mathbf{a}|$.
2. Next, we must consider whether $\mathbf{a} \in L$. We generate a random computation path $\mathbf{b}$ of M_j on input $\mathbf{a}$ until it either accepts (in which case we then generate `#1` to record acceptance of $\mathbf{a}$) or has run for $f(j)$ steps without accepting (in which case we then generate `#0` to record rejection).
3. In the former case ($c = 1$) we finish by deterministically generating $\mathbf{d} \triangleq \mathbf{a} \in L$. In the latter case ($c = \mathtt{0}$), $\mathbf{a} \notin L$, so we fall back and finish by deterministically generating $\mathbf{d} \triangleq \mathbf{x}^{(j)} \in L$.

Let $r(\mathbf{z})$ be the probability that the above procedure generates $\mathbf{z}$. $\text{support}(r)$ is then the set of strings that can be generated by the above procedure. The marginalized language $\mu(\text{support}(r))$ keeps just the $\mathbf{d}$ parts of those strings. It consists of all strings $\mathbf{a}$ that are accepted by at least one path $\mathbf{b}$ of $M_{|\mathbf{a}|}$ (which are exactly the strings in L) together with the fallback strings (which form a subset of L). Thus, $\mu(\text{support}(r)) = L$ as desired.

We wish to show that r is ELNCP. In other words, some Turing machine M^{q} efficiently locally normalizes r with compact parameters $\boldsymbol{\Theta}^{\mathrm{q}}$, as defined in §3.1. The parameters will be used to store information about the infinite set of fallback strings.

In particular, for each n, $\boldsymbol{\theta}^{\mathrm{q}}_n$ must have enough information to construct a Turing machine $q_n = M^{\mathrm{q}}(\boldsymbol{\theta}^{\mathrm{q}}_n)$ such that $q_n(\hat{\mathbf{z}}z)$ returns $r(z \mid \hat{\mathbf{z}})$ for all $z \in V \cup \{\$\}$ and all $\hat{\mathbf{z}}$ with $|\hat{\mathbf{z}}| \leq n$ and $Z(\hat{\mathbf{z}}) > 0$. Here $Z(\hat{\mathbf{z}}) > 0$ means that $\hat{\mathbf{z}}$ is a prefix of a string $\mathbf{z} = \mathbf{a}\#\mathbf{b}\#c\mathbf{d}$ that could be generated by the above procedure. The computation $q_n(\hat{\mathbf{z}}z)$ proceeds by simulating the sequence of choices in the above procedure that would be required to generate $\hat{\mathbf{z}}$, and then returning the probability that the procedure would generate symbol z next. That probability equals $r(z \mid \hat{\mathbf{z}})$ as desired because the above procedure

almost surely terminates (as explained at the end of the proof of Theorem 7).

In general, the computation $q_n(\hat{\mathbf{z}}z)$ may have to construct $M_j = M(\theta_j)$ and simulate it on $\mathbf{a}$ (for $j = |\mathbf{a}|$) if z falls in the $\mathbf{b}\#c$ portion of $\hat{\mathbf{z}}$, and it may have to look up a character of the fallback string $\mathbf{x}^{(j)}\$$ if z falls in the $\mathbf{d}$ portion of $\hat{\mathbf{z}}$ or terminates that portion with $z = \$$. Fortunately $j < n$, and fortunately if the computation looks up the t^{th} character of $\mathbf{x}^{(j)}\$$ then $t < n$. Thus, constructing and simulating M_j can be done in time $O(\text{poly}(j)) \subseteq O(\text{poly}(n))$, and looking up the t^{th} character of $\mathbf{x}^{(j)}\$$ can be achieved with access to the first n characters of each of $\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(n)}$, which can be stored by $\boldsymbol{\theta}^{\text{q}}_n$ in space $O(n^2)$. It follows that M^{q} can construct and apply q_n in polynomial time with access to compact parameters $\boldsymbol{\Theta}^{\text{q}}$, so r is ELNCP.

□

B Implementation details of REBMs

B.1 Modeling finite subsets of infinite languages

The experiments of this paper are conducted on datasets where we only observe strings that are finitely long. Given a possibly infinite language L, we use the notation $L_{\leq T} = \{\mathbf{x} \mid \mathbf{x} \in L, |\mathbf{x}| \leq T\}$ for the subset of strings that are most T symbols long. Specific values of T for datasets used in our experiments are listed in Appendix D.1.

B.2 Design of base models p_0

p_0 can be any distribution over $L_{\leq T}$ [26] provided that we can sample from it, and evaluate $p_0(\mathbf{x}), \forall \mathbf{x} \in L_{\leq T}$, both in $O(\text{poly}(|\mathbf{x}|))$. In this work, we experiment with two designs of p_0: GRU- and Transformer-based locally normalized language models. GRU-based models are used in WikiText and Yelp experiments. The GRU-based p_0's are parametrized with 2-layer GRUs with 500 hidden units, and word embeddings of dimension size 500.

As for Transformer-based p_0's, we make use of Grover models (Zellers et al., 2019), which effectively are GPT-2 models trained on the aforementioned RealNews dataset. In this work, we experiment with the 'base' variant of public available weights, which are 12-layered Transformers, with 12 heads, and 768 hidden units.

[26] Note that since p_0 does not have support over L, it has to assign $p(\$ \mid \mathbf{x}_{1\ldots T}) = 1$, which is generally not an issue.

B.3 Design of discriminators g_θ

We formulate $g_{\boldsymbol{\theta}}(\mathbf{x})$ as a summation of scores at positions $1 \ldots |\mathbf{x}|$, passed through an activation function f:

$$g_{\boldsymbol{\theta}}(\mathbf{x}) = f\left(\sum_{i=1}^{|\mathbf{x}|} g_t(\mathbf{x}; \boldsymbol{\theta})\right). \tag{1}$$

To verify whether lower-bounding $g_{\boldsymbol{\theta}}$ would help with learning, as we discuss in §4.1, we experiment with two variants of f:

- **tanh**: $f(x) = 2 \cdot \tanh(x)$
- **softplus**: $f(x) = -\log(1 + \exp(x + s))$

The former one is bounded between $(-2, 2)$, while the second one has range $(-\infty, 0)$. The offset term s in the softplus activation function determines initial values of $Z_{\boldsymbol{\theta}}$. In this paper we set $s = 20$.

The design of $g_t(\mathbf{x}; \boldsymbol{\theta})$ follows their base model counterparts: we use Bi-GRU discriminators for GRU base models; and bi-directional Transformer discriminators for Transformer ones. For GRUs $g_t(\mathbf{x}; \boldsymbol{\theta}) = \mathbf{h}_t \cdot x_t$, For Transformers $g_t(\mathbf{x}; \boldsymbol{\theta}) = \sum \mathbf{h}_t$ where $\mathbf{h}_t$ are the hidden states at time step t. In both cases, the discriminators have access to information of the whole sequence $\mathbf{x}$ at any timestep: the Bi-GRU discriminators achieve this through the bi-directional RNNs, and the Transformers through the attention mechanism without directional masking.

B.4 Training procedure

As we note in §4.1, MLE-based training methods are generally not feasible for globally normalized models. We therefore opt to train our model using the ranking variant of noise contrastive estimation (NCE) (Ma and Collins, 2018), which does not require samples from p_0 and has a simple form for residual LMs. Using p_0 as a *noise distribution*, NCE training requires minimizing the following single-sequence loss, in expectation over the true distribution p:

$$\mathcal{L}_{\text{NCE}}(\boldsymbol{\theta}, \mathbf{x}, p_0, K) = -\log \frac{\frac{\tilde{p}_{\boldsymbol{\theta}}}{p_0}(\mathbf{x})}{\sum_{k=0}^{K} \frac{\tilde{p}_{\boldsymbol{\theta}}}{p_0}(\mathbf{x}^{(k)})}, \tag{2}$$

where $\mathbf{x}^{(0)} \triangleq \mathbf{x}$, $\frac{\tilde{p}_{\boldsymbol{\theta}}}{p_0}(\mathbf{x}) \triangleq \frac{\tilde{p}_{\boldsymbol{\theta}}(\mathbf{x})}{p_0(\mathbf{x})}$, and $\mathbf{x}^{(1)} \ldots \mathbf{x}^{(K)} \sim p_0$. Since $\tilde{p}_{\boldsymbol{\theta}}(\mathbf{x}) = p_0(\mathbf{x}) \cdot \exp g_{\boldsymbol{\theta}}(\mathbf{x})$, we have $\frac{\tilde{p}_{\boldsymbol{\theta}}}{p_0}(\mathbf{x}) = \exp g_{\boldsymbol{\theta}}(\mathbf{x})$. The NCE minimization objective (2) now reduces to the simple form

$$\begin{aligned}&\mathcal{L}_{\text{NCE}}(\boldsymbol{\theta}, \mathbf{x}, p_0, K)\\ &\quad = -g_{\boldsymbol{\theta}}(\mathbf{x})\\ &\qquad + \log(\exp g_{\boldsymbol{\theta}}(\mathbf{x}) + \sum_{k=1}^{K} \exp g_{\boldsymbol{\theta}}(\mathbf{x}^{(k)})).\end{aligned} \tag{3}$$

Notice that minimizing the expected loss with stochastic gradient descent methods $\mathcal{L}_{\text{NCE}}$ defined in equation (3) requires only evaluating sequence probabilities under $g_{\boldsymbol{\theta}}$, and tuning its parameters, but not the base model p_0. We only need to generate the noise samples $\{\mathbf{x}^{(k)} \sim q \mid k \in [K]\}$ from p_0. This way we do not need to backpropagate through parameters of the base model p_0, which can speed up training considerably when p_0 is backed by a huge network. In fact, the training of $g_{\boldsymbol{\theta}}$ can be completely agnostic to the design of p_0, allowing for the application of finetuning any locally normalized p_0.

Given the same discriminator $g_{\boldsymbol{\theta}}$, the difference of KL-divergence between the true model p and residual language models $\tilde{p}'_{\boldsymbol{\theta}}(\mathbf{x}) = p'_0(\mathbf{x}) \cdot \exp g_{\boldsymbol{\theta}}(\mathbf{x})$, and the KL-divergence between the true model and $\tilde{p}''_{\boldsymbol{\theta}}(\mathbf{x}) = p''_0(\mathbf{x}) \cdot \exp g_{\boldsymbol{\theta}}(\mathbf{x})$, defined with base models p'_0 and p''_0 respectively, can be written as

$$\begin{aligned}&\mathrm{KL}[p||p'_{\boldsymbol{\theta}}] - \mathrm{KL}[p||p''_{\boldsymbol{\theta}}] \\ &\quad = \mathrm{KL}[p||p'_0] - \mathrm{KL}[p||p''_0] + \log \frac{Z'}{Z''},\end{aligned} \tag{4}$$

where $Z' = \mathbb{E}_{\mathbf{x}\sim p'_0}[\exp g_{\boldsymbol{\theta}}(\mathbf{x})]$, and Z'' is similarly defined with p''_0. As a direct result of equation (4), we can see that finding p''_0 where $\mathrm{KL}[p||p''_0] < \mathrm{KL}[p||p'_0]$ implies improvement in $\mathrm{KL}[p||p''_{\boldsymbol{\theta}}]$ over $\mathrm{KL}[p||p'_{\boldsymbol{\theta}}]$, under mild conditions:

Theorem B.1. *If* $\exists k > 0$ *such that* $\frac{\mathbb{E}_{\mathbf{x}\sim p'_0}[\exp g_{\boldsymbol{\theta}}(\mathbf{x})]}{\mathbb{E}_{\mathbf{x}\sim p''_0}[\exp g_{\boldsymbol{\theta}}(\mathbf{x})]} > \exp(-k)$ *and* $\mathrm{KL}[p||p'_0] - \mathrm{KL}[p||p''_0] > k$ *then* $\mathrm{KL}[p||p'_{\boldsymbol{\theta}}] > \mathrm{KL}[p||p''_{\boldsymbol{\theta}}]$.

Proof.

$$\begin{aligned}&\mathrm{KL}[p||p'_{\boldsymbol{\theta}}] - \mathrm{KL}[p||p''_{\boldsymbol{\theta}}] \\ &= \mathop{\mathbb{E}}_{\mathbf{x}\sim p}[\log p''_{\boldsymbol{\theta}}(\mathbf{x}) - \log p'_{\boldsymbol{\theta}}(\mathbf{x})] \\ &= \mathop{\mathbb{E}}_{\mathbf{x}\sim p}[\log \frac{p''_0(\mathbf{x}) \exp g_{\boldsymbol{\theta}}(\mathbf{x})}{\sum_{\mathbf{x}'\in L_{\le T}} p''_0(\mathbf{x}) \exp g_{\boldsymbol{\theta}}(\mathbf{x})} \\ &\quad - \log \frac{p'_0(\mathbf{x}) \exp g_{\boldsymbol{\theta}}(\mathbf{x})}{\sum_{\mathbf{x}'\in L_{\le T}} p'_0(\mathbf{x}) \exp g_{\boldsymbol{\theta}}(\mathbf{x})}] \\ &= \mathop{\mathbb{E}}_{\mathbf{x}\sim p}[\log \frac{p''_0(\mathbf{x}) \exp g_{\boldsymbol{\theta}}(\mathbf{x})}{\mathbb{E}_{\mathbf{x}'\sim p''_0}[\exp g_{\boldsymbol{\theta}}(\mathbf{x})]} \\ &\quad - \log \frac{p'_0(\mathbf{x}) \exp g_{\boldsymbol{\theta}}(\mathbf{x})}{\mathbb{E}_{\mathbf{x}'\sim p'_0}[\exp g_{\boldsymbol{\theta}}(\mathbf{x})]}] \\ &= \mathop{\mathbb{E}}_{\mathbf{x}\sim p}[\log p''_0(\mathbf{x}) - \log p'_0(\mathbf{x})] \\ &\quad + \mathop{\mathbb{E}}_{\mathbf{x}\sim p}[\log \mathop{\mathbb{E}}_{\mathbf{x}'\sim p'_0}[\exp g_{\boldsymbol{\theta}}(\mathbf{x})] - \log \mathop{\mathbb{E}}_{\mathbf{x}'\sim p''_0}[\exp g_{\boldsymbol{\theta}}(\mathbf{x})]] \\ &= \mathrm{KL}[p||p'_0] - \mathrm{KL}[p||p''_0] \\ &\quad + \log \frac{\mathbb{E}_{\mathbf{x}'\sim p'_0}[\exp g_{\boldsymbol{\theta}}(\mathbf{x})]}{\mathbb{E}_{\mathbf{x}'\sim p''_0}[\exp g_{\boldsymbol{\theta}}(\mathbf{x})]}.\end{aligned} \tag{5}$$

Plugging assumptions $\frac{\mathbb{E}_{\mathbf{x}\sim p'_0}[\exp g_{\boldsymbol{\theta}}(\mathbf{x})]}{\mathbb{E}_{\mathbf{x}\sim p''_0}[\exp g_{\boldsymbol{\theta}}(\mathbf{x})]} > \exp(-k)$ and $\mathrm{KL}[p||p'_0] - \mathrm{KL}[p||p''_0] > k$ into equation (5), $\mathrm{KL}[p||p'_{\boldsymbol{\theta}}] - \mathrm{KL}[p||p''_{\boldsymbol{\theta}}] > 0$. □

Theorem B.1 suggests a training strategy that we first train the base model p_0, then finetune $g_{\boldsymbol{\theta}}$: under a roughly uniform $g_{\boldsymbol{\theta}}$ (*e.g.* when $\boldsymbol{\theta}$ is newly initialized), $\mathbb{E}_{\mathbf{x}\sim p'_0}[\exp g_{\boldsymbol{\theta}}]/\mathbb{E}_{\mathbf{x}\sim p''_0}[\exp g_{\boldsymbol{\theta}}] \approx \exp(0)$; so improvements on the inclusive KL-divergence of base model $\mathrm{KL}[p||p_0]$ will mostly translate to improvement in $\mathrm{KL}[p||\tilde{p}_{\boldsymbol{\theta}}]$. Optimizing the base model (*i.e.* finding p''_0 such that $\mathrm{KL}[p||p''_0] < \mathrm{KL}[p||p''_0]$) is much easier than directly minimizing $\mathrm{KL}[p||p'_{\boldsymbol{\theta}}]$: the former can be done by minimizing empirical cross entropy, which is computationally efficient, while the latter involves an intractable partition function $\sum_{\mathbf{x}\in L_{\le T}} \tilde{p}'_{\boldsymbol{\theta}}(\mathbf{x})$.

Pseudocode for fine-tuning $g_{\boldsymbol{\theta}}$ is listed in Algorithm 1.

B.5 Computing normalized probabilities

The unnormalized probability $\tilde{p}_{\boldsymbol{\theta}}(\mathbf{x})$ (in equation (1)) can be evaluated easily, and should suffice for (re)ranking purposes (*e.g.* for ASR and MT applications). However, the *normalized* probability $q_{\boldsymbol{\theta}}(\mathbf{x}) \triangleq \frac{\tilde{p}_{\boldsymbol{\theta}}(\mathbf{x})}{\sum_{\mathbf{x}} \tilde{p}_{\boldsymbol{\theta}}(\mathbf{x})}$ does require computing the partition function $Z_{\boldsymbol{\theta}}$. An unbiased importance sampling

Algorithm 1: Pseudocode for training g_θ

Input:
- Training/validation corpora $\mathcal{D}_{\{\text{train,dev}\}}$
- base model $p_0 : L_{\leq T} \to [0, 1]$
- initial parameter vector $\boldsymbol{\theta}_0 \in \mathbb{B}^d$
- noise sample size $K \in \mathbb{N}$

Output: unnormalized residual language model $\tilde{q}_{\boldsymbol{\theta}} : L_{\leq T} \to [0, 1]$

$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta}_0$;

/* $\mathcal{L}_{\text{NCE}}$ is defined in equation (3) */

while $\sum_{\mathbf{x} \in \mathcal{D}_{dev}} \mathcal{L}_{NCE}(\boldsymbol{\theta}, \mathbf{x}, p_0, K)$ *is still decreasing* **do**

  foreach $\mathbf{x} \in$ *shuffle*$(\mathcal{D}_{train})$ **do**

    $\nabla_{\boldsymbol{\theta}} \mathcal{L}_{\text{NCE}} = \nabla_{\boldsymbol{\theta}} \mathcal{L}_{\text{NCE}}(\boldsymbol{\theta}, \mathbf{x}, p_0, K)$;

    $\boldsymbol{\theta} \leftarrow$ update-gradient$(\boldsymbol{\theta}, \nabla_{\boldsymbol{\theta}} \mathcal{L}_{\text{NCE}})$;

  end

end

return $\mathbf{x} \mapsto p_0(\mathbf{x}) + \exp g_{\boldsymbol{\theta}}(\mathbf{x})$;

estimate of $\sum_{\mathbf{x} \in L_{\leq T}} \tilde{p}_{\boldsymbol{\theta}}(\mathbf{x})$ is

$$\begin{aligned} Z_{\boldsymbol{\theta}} &= \sum_{\mathbf{x} \in L_{\leq T}} \tilde{p}_{\boldsymbol{\theta}}(\mathbf{x}) \\ &= \sum_{\mathbf{x} \in L_{\leq T}} p_0(\mathbf{x}) \exp g_{\boldsymbol{\theta}}(\mathbf{x}) \\ &= \mathop{\mathbb{E}}_{\mathbf{x} \sim p_0} [\exp g_{\boldsymbol{\theta}}(\mathbf{x})] \\ &\approx \sum_{m=1}^{M} \frac{\exp g_{\boldsymbol{\theta}}(\mathbf{x}^{(m)})}{M} = \hat{Z}_{\boldsymbol{\theta} M}, \end{aligned} \tag{6}$$

where $\mathbf{x}^{(1)} \ldots \mathbf{x}^{(M)} \sim q_0$.

C Comparison between REBMs and autoregressive models

We evaluate the effectiveness of REBMs on two different neural architectures (GRU- and Transformer-based) and 3 datasets: WikiText (Merity et al., 2017), Yelp (Yelp), and RealNews (Zellers et al., 2019), on the task of modeling sequence probabilities. An REBM $\tilde{p}_{\boldsymbol{\theta}}$ has two components, $g_{\boldsymbol{\theta}}$ and p_0, and we would like to see how $\tilde{p}_{\boldsymbol{\theta}}$ competes against p_0 itself. We do not further tune p_0 while training $p_{\boldsymbol{\theta}}$. As a fair comparison, we also see how p'_0 compares against p_0, where p'_0 is simply a version of p_0 that has been trained as many additional epochs as were used to train $p_{\boldsymbol{\theta}}$.

p_0 models are pretrained on moderately large corpora (in GRU cases) or a very large corpus (in the Transformer case).[27] We compare residual energy-based models $\tilde{p}_{\boldsymbol{\theta}}$ to further-fine-tuned base models p'_0, on conservatively estimated (at the low end of 95% confidence interval) token perplexity and bootstrap-sampled log likelihood improvements. The results are in Table 2. Residual energy-based models show consistent perplexity improvement compared to p'_0 that are trained on the same data using the same maximum numbers of iterations. Although the improvement in log-likelihood of $p_{\boldsymbol{\theta}}$ over p_0 is modest (especially for RealNews experiments, where p_0 is a very strong baseline), we verify that these improvements are all statistically significant ($p < 0.05$) using bootstrapped test datasets.

We experiment with different designs of the discriminator $g_{\boldsymbol{\theta}}$, evaluating the effectiveness of bounding $g_{\boldsymbol{\theta}}$ and varying its number of parameters. We find that in Transformer-based experiments, bounding $g_{\boldsymbol{\theta}}$ considerably helps with performance; but the opposite happens for GRU-based models. We speculate that this is due to the base models' performance: the Transformer base models have high parameter count and were trained on a lot of data; and the true distribution p likely is relatively similar to p_0, and benefits from a small hypothesis space — even though we don't know if the at-most-ϵ error assumption in §4.1 holds. On the other hand our GRU-based p_0 has neither the capacity, nor the huge amount of training data. As a result, the unbounded variant $g_{\boldsymbol{\theta}}$ (and $q_{\boldsymbol{\theta}}$) may end up learning a better approximation of p.

D Experimental details

D.1 Datasets

Residual language model experiments are conducted on these datasets:

- **Segmented WikiText**: we take the standard WikiText-2 corpus (Merity et al., 2017), and segment it into sequences at new line breaks. We discard all empty lines, and any line that starts with the '=' token. In effect, we obtain sequences that are mostly entire paragraphs. We also only keep lines that are shorter than 800 tokens after BPE tokenization. Because of our preprocessing, Segmented WikiText loses much interparagraph context information, and doesn't have the 'simple' header sequences that were in the original WikiText corpus, and is much harder to language-model.
- **Yelp**: the Yelp dataset (Yelp) contains business reviews. As in Segmented WikiText, We keep

[27] In the Transformer case we simply take p_0 to be the Grover (Zellers et al., 2019) pretrained language model, which is based on the GPT-2 (Radford et al., 2019) architecture and performs competitively on news article generation.

Experiment (Architecture)	Model	Best configuration	log likelihood improvement (95% CI)	perplexity improvement
RealNews (Transformer)	$p_{\boldsymbol{\theta}}$	4-layer, tanh	$(-0.18, -0.13), \mu = -0.15$	.03%
RealNews (Transformer)	p_0'	N/A	N/A	.00%
WikiText (GRU)	$p_{\boldsymbol{\theta}}$	1-layer/500, softplus	$(-1.85, -1.54), \mu = -1.69$	1.44%
WikiText (GRU)	p_0'	N/A	N/A	.50%
Yelp (GRU)	$p_{\boldsymbol{\theta}}$	2-layer/500, softplus	$(-1.89, -1.67), \mu = -1.80$	1.82%
Yelp (GRU)	p_0'	N/A	N/A	.49%

Table 2: Residual energy-based model $\tilde{p}_{\boldsymbol{\theta}}$ improvements over autoregressive base models p_0. The perplexity numbers are per-token, and log likelihood improvements are per sequence (in nats). We only report each dataset's best model (according to validation data) in this table. See Appendix D for experimental details.

reviews shorter than 800 tokens.

- **RealNews**: we make use of the standard RealNews corpus comes from (Zellers et al., 2019), which contains news articles that are up to 1,024 tokens long.

In all experiments we tokenize with BPE tokenizers derived from the GPT-2 language models: the GRU models use Huggingface's implementation[28] and the Transformers use Grover's[29]. Number of sequences in preprocessed datasets are listed in Table 3.

	Train	Dev	Test
RealNews	3,855	1,533	6,158
WikiText	18,519	878	2,183
Yelp	10,951	9,964	994

Table 3: Number of sequences in preprocessed datasets (for training and tuning the discriminators $g_{\boldsymbol{\theta}}$, and evaluation).

D.2 Pretraining base models p_0

We use a pretrained Grover model as the base model in RealNews experiments. For GRU-based experiments, we train base models on WikiText and Yelp datasets using separate training and validation splits than those of the discriminator $g_{\boldsymbol{\theta}}$ (Table 4). The base models are periodically (every 1,000 iterations) evaluated on the validation split for early stopping, where we stop if there is no improvement on validation perplexity for 10 consecutive evaluations. The base models $q_{\boldsymbol{\theta}}$ achieve 113.98 for Segmented WikiText, and 110.89 in test set perplexity, respectively. Note that these base models are further fine-tuned on additional datasets in our comparison against residual language models.

D.3 Metrics

We evaluate the relative performance of residual language models against autoregressive models (*i.e.*

[28] https://github.com/huggingface/transformers

[29] https://github.com/rowanz/grover

	Train	Dev
WikiText	17,556	1,841
Yelp	9,954	1,000

Table 4: Number of sequences in preprocessed datasets (for training and tuning the base model q). Note that we do not train our own base models for RealNews, but use one of the pretrained models provided by (Zellers et al., 2019).

fine-tuned base models) on two metrics, log likelihood and perplexity improvement, which are approximated as follows:

- **Log likelihood improvement**: since p, $p_{\boldsymbol{\theta}}$ and q_0 are all distributions over $L_{\leq T}$, we can quantitatively evaluate their difference in log likelihood. We measure the difference between $\mathrm{KL}[p||p_{\boldsymbol{\theta}}]$ and $\mathrm{KL}[p||p_0]$:[30]

$$\begin{aligned}
&\mathrm{KL}[p||p_{\boldsymbol{\theta}}] - \mathrm{KL}[p||p_0] \\
&= \mathop{\mathbb{E}}_{\mathbf{x}\sim p}[\log p_{\boldsymbol{\theta}}(\mathbf{x}) - \log p_0(\mathbf{x})] \\
&= \mathop{\mathbb{E}}_{\mathbf{x}\sim p}[\log \tilde{p}_{\boldsymbol{\theta}}(\mathbf{x}) - \log p_0(\mathbf{x})] - \log Z_{\boldsymbol{\theta}} \\
&= \mathop{\mathbb{E}}_{\mathbf{x}\sim p}[g_{\boldsymbol{\theta}}(\mathbf{x})] - \log Z_{\boldsymbol{\theta}} \\
&\approx \frac{\sum_{\mathbf{x}\in\mathcal{D}_{\text{test}}} g_{\boldsymbol{\theta}}(\mathbf{x})}{|\mathcal{D}_{\text{test}}|} - \log \hat{Z}_{\boldsymbol{\theta}M}, \qquad (7)
\end{aligned}$$

  where $\hat{Z}_{\boldsymbol{\theta}M}$ is estimated using equation (6). A negative value of log likelihood difference indicates that $\tilde{q}_{\boldsymbol{\theta}}$ approximates p better than p_0 in terms of KL-divergence.
- **Perplexity improvement**: perplexity is a common language modeling metric. Following (Rosenfeld et al., 2001), we compute

$$\begin{aligned}
&\text{perplexity improvement of } p_{\boldsymbol{\theta}} \\
&= \frac{\exp \frac{|\mathcal{D}| \log \hat{Z}_{\boldsymbol{\theta}M}}{w(\mathcal{D}_{\text{test}})}}{\exp \frac{\sum_{\mathbf{x}\in\mathcal{D}_{\text{test}}} g_{\boldsymbol{\theta}}(\mathbf{x})}{w(\mathcal{D}_{\text{test}})}}, \qquad (8)
\end{aligned}$$

[30] Note that p_0 here is the base model component of $\tilde{p}_{\boldsymbol{\theta}}$. While comparing between residual language models and autoregressive models, we also finetune p_0 on additional data to get a new model q_0', which has different parameters than p_0.

where $w(\mathcal{D})$ is the total token count of dataset $\mathcal{D}$, and $|\mathcal{D}|$ is the number of sequences of $\mathcal{D}$. $\hat{Z}_{\theta M}$ is ecomputed Appendix B.5

Both evaluation metrics involve estimating the partition function with $\hat{Z}_{\theta M}$. For the perplexity improvement metric, we obtain 32 estimates of $\hat{Z}_{\theta M}$ [31], which are normally distributed, and compute equation (8) using $\hat{Z}_{\theta M}$ the conservative end of a 95% confidence level. To account for variance in our test datasets, we further make use of bootstrapping estimation for log likelihood improvement: we bootstrap-sample $1,000$ subsamples for each test dataset, and compute equation (7) for each datapoint in the Cartesian product ($1,000 \times 32$ in total). We then report results at the 2.5% and 97.5% percentiles.

D.4 Hyperparameters

Transformer experiments. We train our models on 64 GPUs across 8 nodes, with a total batch size of $64 \times 8 \times 2 = 1,024$, and with 1 noise sequence ($K = 1$ in Appendix B.4) per batch. We use an initial learning rate of $5e-5$. The rest of the hyperparameters largely follow settings in (Zellers et al., 2019). Optimization is done with the Grover implementation of AdaFactor.

GRU experiments. We train our models on 8 GPUs on a single node, with a total batch size of $8 \times 2 = 16$, and with 25 noise sequences ($K = 25$ in Appendix B.4) per batch. We have an initial learning rate of $1e-4$. Upon no improvement on validation data, we half the learning rate, with patience = 1. The model parameters are l_2 regularized with a coefficient of $1e-5$. We also apply dropout regularization with $p = 0.5$. Optimization is done with PyTorch-supplied Adam.

D.5 Configurations

We study the effects of these configurations:

- **Bounding** g_θ: we note in §4.1 that with the strong hypothesis that the base model p_0 has bounded error, g_θ will have a bounded range, and leads to a much smaller hypothesis space. In this work we experiment with both bounded and unbounded g_θ's, with ranges $(-\infty, 0)$ and $(-2, 2)$ respectively. More details can be found in Appendix B.3.
- **Model capability of** g_θ: we hypothesize that the expressiveness of g_θ does not need to be as rich as the parametrization of p_0, since g_θ essentially only has to tell whether the sequence $\mathbf{x}$ comes from p or p_0. For the GRU + WikiText experiments, we experiment with $\{1, 2\}$-layer GRU models of g_θ. For 1-layer models, we additionally experiment with a setup that has only 250 hidden units. For the Transformers/RealNews dataset, we experiment with $\{12, 4\}$-layer Transformer models.

D.6 Log likelihood improvements under different configurations

We also see in Table 5 that using tanh as the activation function f does better than softplus for Transformers; but performs very poorly for GRUs. We also observe degeneracy problems. We speculate that our Transformer-based base models q_θ have already learned a good approximation of the true distribution; and limiting the model capacity of g_θ in exchange of smaller variance results in a favorable trade-off, and vice versa for GRUs. Regarding discriminator capability: we see that performance is not sensitive to model size. Our best Transformers run actually is from the smaller-model runs. And the 1-layer 500-unit GRU models achieve best performance. Overall, results in Table 5 suggests that performance is sensitive to the choice of model configuration.

[31] We set $M = 512$ in this paper.

Model Size	Activation	log likelihood improvement	
		95% CI	μ
RealNews (Transformers)			
12-layer	softplus	$(-0.13, 0.08)$	-0.09
12-layer	tanh	$(-0.14, -0.10)$	-0.12
4-layer	softplus	$(-0.15, 2.62)$	-0.02
4-layer	tanh	$(-0.18, -0.13)$	-0.16
WikiText (GRUs)			
2-layer / 500	tanh	$(-0.00, 0.00)$	-0.00
2-layer / 500	softplus	$(-1.32, -0.85)$	-1.18
1-layer / 500	tanh	$(-0.79, -0.64)$	-0.71
1-layer / 500	softplus	$(-1.85, -1.54)$	-1.69
1-layer / 250	tanh	$(-0.02, 0.02)$	-0.00
1-layer / 250	softplus	$(-1.85, -1.46)$	-1.67
Yelp (GRUs)			
2-layer / 500	tanh	$(-0.03, 0.01)$	-0.02
2-layer / 500	softplus	$(-1.89, -1.67)$	-1.80
1-layer / 500	tanh	$(-0.65, -0.57)$	-0.61
1-layer / 500	softplus	$(-2.62, -2.03)$	-2.43
1-layer / 250	tanh	$(-0.00, 0.00)$	-0.00
1-layer / 250	softplus	$(-2.25, -1.99)$	-2.13

Table 5: Comparison of different configurations.