Generating Text from Language Models

Afra Amini[1] **Ryan Cotterell**[1] **John Hewitt**[2]
Luca Malagutti[1] **Clara Meister**[1] **Tiago Pimentel**[3]
[1]ETH Zürich [2]Stanford University [3]University of Cambridge
afra.amini@inf.ethz.ch ryan.cotterell@inf.ethz.ch
johnhew@cs.stanford.edu luca.malagutti@inf.ethz.ch
clara.meister@inf.ethz.ch tp472@cam.ac.uk

Abstract

An increasingly large percentage of natural language processing (NLP) tasks center around the generation of text from probabilistic language models. Despite this trend, there lacks a unified framing of the techniques for generating from language models, both in terms of methods that improve text quality and methods that allow more fine-grained control of generation. Without this framing, practitioners must either be experts in the generation field or choose somewhat blindly between a large range of algorithms that can lead to wildly different results depending on the specific use-case, e.g., top-p sampling and beam search. In this tutorial, we will provide a centralized and cohesive discussion of critical considerations when choosing how to generate from a language model. We will first discuss the formal definition of a probabilistic language generator and taxonomize a wide range of empirically-observed problems with systems using these models, like degradation, hallucination, and repetition. We will then discuss their corresponding proposed algorithmic solutions under a unified light; specifically as *locally adapting* the probabilities of a model to avoid failure cases. Finally, we will then cover methods in *controlled* generation, that go beyond just ensuring coherence to ensure text exhibits specific desired properties. We aim for NLP practitioners and researchers to leave our tutorial with a unified framework which they can use to evaluate and contribute to the latest research in language generation.

1 Introduction and Motivation

With their widespread public availability, large pretrained language models have become a core part of many natural language processing (NLP) pipelines. This trend is particularly evident in language generation tasks, where prompt engineering and controlled generation techniques have shown that these models can essentially be used "out-of-the-box" for various language generation needs. Yet, as has been observed repeatedly, how one chooses to generate text from these models can lead to vastly different results; make the wrong choice and a language model can fall into repetitive loops (Welleck et al., 2020), generate gibberish (Holtzman et al., 2020), or spew out random (and possibly falsifiable) declarations (Maynez et al., 2020). In the effort to circumnavigate these issues, one can make use of a variety of relatively straightforward methods: (i) sampling adapters, simple modifications to token-level distributions that help prevent the generation of incoherent text; (ii) controlled generation methods, techniques that guide these models to output strings with a set of desired attributes. While employing these methods often does not require domain expertise, many people do not have proper knowledge of the tools available—and much less how and when to apply them. Hence, without years of experience in this subfield, both NLP researchers and practitioners may have difficulty using pretrained language models for text generation, as they will likely encounter the problematic behaviors mentioned above.

In this **cutting-edge** tutorial, we aim to offer a comprehensive introduction to techniques for generating strings from language models, discussing both how to sample adeptly from and explicitly control them. This tutorial will be divided in four parts. First, we will present background knowledge on language modeling, discussing both its mathematical formulation, the empirically-observed successes and shortcomings of modern models when used to generate language, and the difficulty in evaluating these successes and failures. Second, we will give a brief overview of the basics of language generation, framing generation as the combination of a choice of a decoding algorithm and objective. The final two parts of this tutorial, which focus on alleviating the previously discussed issues with using language models out-of-the-box for generation, will be discussed within this framing: we

present heuristic modifications to the objective that have empirically-proven themselves effective at improving generation quality as well as new decoding algorithms that—when combined with the right objective—can be used to enforce constraints on the text output by models. We believe this will equip the NLP community with the knowledge of how to better employ these models for their downstream use-cases, thus making them more broadly accessible.

2 Target Audience

Our tutorial is targeted at members of the NLP community who wish to make use of language models for various language generation tasks. This includes researchers, interested in e.g., data augmentation techniques, as well as practitioners wishing to make use of pretrained language models in their language generation pipelines. We expect that participants are comfortable with probabilistic formulations of NLP tasks, as well as the structure and formulation of standard autoregressive models e.g., transformers. **While we do not require any readings, we recommend reviewing (in no particular order) the works cited in this proposal.**

3 Outline

3.1 Part 1: Background

Modern natural language processing tends to proceed by (1) framing a task in probabilistic terms, (2) estimating a model to imitate the task's generative processes (typically using finite training datasets as a proxy), and then (3) using this model as a tool to accomplish the task. This is how the task of **language modeling** is often approached. More precisely, practitioners take a corpus $\mathcal{D} = \{\mathbf{y}^{(n)}\}_{n=1}^{N}$—an N-sized set of strings consisting of tokens y from some vocabulary $\mathcal{V}$—and treat it as a set of independently and identically distributed samples from a distribution $p(\mathbf{y})$. We will use p to denote the *true* language modeling distribution, i.e., the distribution defined by the data-generating process, from which we drew our samples. In practice, the vast majority of these models, which we denote as p_{θ}, are trained to minimize the cross-entropy with the empirical distribution defined by our finite set of samples $\mathcal{D}$. In this tutorial, we'll focus largely on autoregressive models of p, meaning that we decompose the probability of a string as $p(\mathbf{y}) = \prod_{t=1}^{T} p(y_t \mid \mathbf{y}_{<t})$ and build a model of the conditional distribution $p(y_t \mid \mathbf{y}_{<t})$ instead.

Successes and known failures. It is hard to overstate the improvements in modeling performance that have occurred in the last five years, as measured simply in terms of perplexity on held-out data. These models are used ubiquitously as the base for fine-tuning on downstream tasks, leading to SOTA performance for myriad tasks. Indeed, the recent ChatGPT is one such instance of a large language model fine-tuned to generate astoundingly fluent and realistic text.

However, when used out-of-the-box for language generation (i.e., without any fine-tuning), these models exhibit a number of failure modes. Among others:

- **Low-quality, low-probability words.** Due to the use of the cross-entropy objective, language models place non-zero probability on poor continuations.
- **Degradation of long texts.** Possibly as a result of the above, generating longer texts can present a greater challenge, as errors tend to propagate and accumulate.
- **Repetition when searching for the mode.** In cases where *highly probable* text under the training set is desired, language models' probability estimates tend to fail and overestimate the probability of highly repetitive text.
- **Inability to guide or constrain generation.** There is no builtin way to direct or shift generation towards a particular concept, meaning one may have to sample indefinitely in order to get a text with the desired attributes.

Further, there is the added difficulty of measuring the quality of generations in many settings: automatic metrics such as BLEU or ROUGE require references and reference-free metrics still do not have direct mechanisms for measuring attributes of text that may be of interest, e.g., faithfulness to a topic. A range of language generation techniques are used both to avoid known failure modes, to coax more desirable properties out of language models, and to direct generation. These methods will be the focus of our tutorial.

3.2 Part 2: Language Generation

Given a language model $p_{\theta}(\cdot \mid \mathbf{y}_{<t})$, how does ones generate text from it? In this part of the tutorial, we give an overview of **decoding strategies**: techniques for generating from probability distributions over strings. Specifically, we will frame all decoding strategies as consisting of two choice points: a scoring function (or objective) and a decoding al-

gorithm. For example, standard ancestral sampling can be recovered when standard log-probability is used as the scoring function and multinomial sampling is used as the algorithm. While this framing may seem excessive at first, it emphasizes the ability to combine the components of well-known decoding strategies. This in turn allows us to build decoding strategies—whose efficacy depend on the underlying model and the desired outcome—with specific goals in mind. For example, one could use the truncation scoring function specified by typical sampling in combination with the beam search algorithm if the user has reason to believe this would help them achieve their goals.

We will then motivate the usage of different scoring functions and decoding algorithms, providing both intuition and formal reasons as to why we might want either in different settings. A main focus of this discussion will be the scale of "open-endedness" on which a generation task falls. For example, story generation can be very **open-ended** when there are no specific desired directions for the story to follow. On the other hand, machine translation is quite **semantically-constrained**. In this tutorial, we will discuss open-endedness as a scale well-described by the **entropy** of the true distribution a task specifies, an attribute which—without explicitly added modeling biases—we expect to be reflected in models of this distribution.

These attributes of a generation task motivate different quantitative approaches during decoding. In machine translation, we often look for high probability strings, for which we can rely on deterministic decoding algorithms that "search" over the support of the distribution $p_\theta(\cdot \mid \mathbf{y}_{<t})$ for this correct answer. On the other hand, if generating from a distribution over web text documents, the notion of the "most likely" web text document is unintuitive, to say the least. This motivates the use of stochastic generation strategies, which naturally add diversity to the generated output. Yet in both of these cases, several issues arise from simply using $p(\mathbf{y})$ as the scoring function, such as the inability to steer generation in a desired direction (if not encoded directly in $p(\mathbf{y})$ itself) or the possibility to sample from low probability regions of $p(\mathbf{y})$. In the next section, we dive into different methods to mitigate these issues.

3.3 Part 3: Sampling Adapters

In this part of the tutorial, we will discuss simple modifications to the standard log-probability scoring function that have been proposed in the effort to the mitigate the generation failures discussed in part 1 (Fan et al., 2018; Holtzman et al., 2020; Basu et al., 2021; Meister et al., 2022; Hewitt et al., 2022). For example, one issue that has received large focus is the constraint that language models must assign nonzero probability to all token in the vocabulary. Even if a model assigns inappropriate tokens very low probability, there is still the chance of sampling them when using stochastic decoding algorithms. This can lead to undesirable outputs, as a single incoherent token can render a natural language string virtually incomprehensible (Fan et al., 2018; Holtzman et al., 2020). Under the assumption that our training data consisted of coherent text, the model will subsequently not be able to predict appropriate continuations for such a text as it was not exposed to text of this nature during training. While intuitively we might expect this issue to only occur with low probability, a concrete example proves otherwise.[1]

Methods such as nucleus and top-k sampling have proposed simple modifications to the scoring function $p(\cdot \mid \mathbf{y}_{<t})$ to exclude undesirable tokens from the candidate pool. These types of transformations are widely-employed when sampling from probabilistic language generators: they are quick to implement, efficient in practice, and surprisingly effective. Indeed, nucleus sampling is often used as a baseline in various language generation tasks (Welleck et al., 2020; Pillutla et al., 2021; Basu et al., 2021).

Here we will offer a formal treatment of these transformations; we present a general framework for what we call **sampling adapters**, the class of functions $g : \mathbb{R}^{|\mathcal{V}|} \to \mathbb{R}^{|\mathcal{V}|}$ that adapts each conditional distribution $p_\theta(\cdot \mid \mathbf{y}_{<t})$ in a locally normalized language model to a new distribution. We will show results from prior works comparing these methods, describing the problems that they mitigate (such as sampling incoherent tokens) as well as the problems that they introduce (such as repetitive generations). Finally, we will discuss possible interpretations of the effectiveness of these methods, in order to provide intuition for why they lead to better language generation.

[1] Let's say we have a model that assigns a very small collective probability mass of 0.1% to all tokens in the tail (low-probability region) of the distribution at any given point. If we sample a sequence of 200 tokens from this model, there is a $1-(1-0.001)^{200} \approx 20\%$ chance it will contain at least one token from the tail of the distribution.

3.4 Part 4: Controlled Generation

Generated samples from language models often contain toxic or non-factual content (Gehman et al., 2020; Maynez et al., 2020). Further, they also often go off-topic, even after applying the sampling adapters discussed in the previous section (Yang and Klein, 2021). To ensure that the generated samples satisfy a set of desired properties—e.g. being non-toxic or talking about a certain topic—we need methods to impose controls during the sampling process. The question we will discuss in this part of the tutorial is how can we sample from a pretrained language model p_{θ}, while ensuring that samples satisfy a specific control c? This can be formalized as turning our scoring function into a different distribution $p_{\theta}(\mathbf{y} \mid c)$. We look methods for building $p_{\theta}(\mathbf{y} \mid c)$ using an arbitrary language model and the decoding algorithms that can used with this distribution under different circumstances.

Given a control c, our goal is to sample a token y_t from the distribution $p(\cdot \mid \mathbf{y}_{<t}, c)$. Following Bayes' rule, this distribution is proportional to $p_{\theta}(\cdot \mid \mathbf{y}_{<t})\, p(c \mid \mathbf{y}_{\leq t})$, where we use p_{θ} to denote an arbitrary language model. In other words, we can view our problem as reweighting the score of a candidate y_t under the language model p_{θ} according to the probability that $\mathbf{y}_{\leq t}$ satisfies the control target: $p(c \mid \mathbf{y}_{\leq t})$ (Yang and Klein, 2021). This control target can be estimated with a supervised classifier parameterized by ϕ: $p_{\phi}(c \mid \mathbf{y}_{\leq t})$ (Ghazvininejad et al., 2017; Holtzman et al., 2018). Building such a classifier, however, is arguably an easier problem than building the entire distribution over natural language strings, if due to the smaller size of the support alone. Once we obtain such estimates, we can make use of an arbitrary language model p_{θ} and standard autoregressive decoding algorithm for controlled generation.

While autoregressive methods have proven effective for controlling the topic or the sentiment of samples, they fail for more complex controls such as toxicity or syntax. Particularly, for more complex controls, estimating $p(c \mid \mathbf{y}_{\leq t})$ becomes challenging. If at any point this probability distribution diverges from the true value, the error will propagate to the next steps due to structure of most of these models. To address this issue, other controlled generation methods propose sampling the whole sequence $\mathbf{y}$ at once, using Markov-Chain methods. Specifically, these methods propose a decoding algorithm for building Markov-Chains based that have the stationary distribution $p(\mathbf{y} \mid c)$. Given that the sampling space is high dimensional, Hamiltonian Monte Carlo (HMC) algorithms, such as Langevin Dynamics, have been shown to be effective for drawing samples from those Markov-Chains (Qin et al., 2022; Kumar et al., 2022).

4 Presenters

- **Afra Amini** is a PhD student at ETH Zürich in the ETH AI Center. Her current foci include language generation and parsing.
- **Ryan Cotterell** is an assistant professor at ETH Zürich in the Institute for Machine Learning. His research focuses on a wide range of topics, including information-theoretic linguistics, parsing, computational typology and morphology, and bias and fairness in NLP systems.
- **John Hewitt** Is a PhD student at Stanford University. His research tackles basic problems in learning models from broad distributions over language, characterizing and understanding those models, and building smaller, simpler models.
- **Clara Meister** is a PhD student at ETH Zürich in the Institute for Machine Learning and a Google PhD Fellow. Her current foci include language generation, pyscholinguistics, and the general application of statistical methods to natural language processing.
- **Tiago Pimentel** is a PhD student at the University of Cambridge and a Facebook Fellow. His research focuses on information theory, and its applications to the analysis of pre-trained language models and natural languages.

Diversity Considerations

As our tutorial focuses on language generation, we will cover issues related to modeling and generating strings in languages which are typologically different from English. Further, this tutorial was developed by a group of researchers from three universities (Stanford, ETHZ and Cambridge), who are originally from 3 continents (Asia, North America, and South America). Lastly, it will discuss work produced by authors spanning many backgrounds, both in industry—where institutions have the resources to train these large language models and make them publicly available—and academia—which has given a large focus to making efficient use of pretrained models during generation.

References

Sourya Basu, Govardana Sachitanandam Ramachandran, Nitish Shirish Keskar, and Lav R. Varshney. 2021. Mirostat: A perplexity-controlled neural text decoding algorithm. In *Proceedings of the 9th International Conference on Learning Representations*.

Angela Fan, Mike Lewis, and Yann Dauphin. 2018. Hierarchical neural story generation. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 889–898, Melbourne, Australia. Association for Computational Linguistics.

Samuel Gehman, Suchin Gururangan, Maarten Sap, Yejin Choi, and Noah A. Smith. 2020. RealToxicityPrompts: Evaluating neural toxic degeneration in language models. In *Findings of the Association for Computational Linguistics*, pages 3356–3369, Online. Association for Computational Linguistics.

Marjan Ghazvininejad, Xing Shi, Jay Priyadarshi, and Kevin Knight. 2017. Hafez: an interactive poetry generation system. In *Proceedings of ACL 2017, System Demonstrations*, pages 43–48, Vancouver, Canada. Association for Computational Linguistics.

John Hewitt, Christopher D. Manning, and Percy Liang. 2022. Truncation sampling as language model desmoothing. In *Findings of the Conference on Empirical Methods in Natural Language Processing*.

Ari Holtzman, Jan Buys, Li Du, Maxwell Forbes, and Yejin Choi. 2020. The curious case of neural text degeneration. In *Proceedings of the 8th International Conference on Learning Representations*.

Ari Holtzman, Jan Buys, Maxwell Forbes, Antoine Bosselut, David Golub, and Yejin Choi. 2018. Learning to write with cooperative discriminators. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1638–1649, Melbourne, Australia. Association for Computational Linguistics.

Sachin Kumar, Biswajit Paria, and Yulia Tsvetkov. 2022. Constrained sampling from language models via langevin dynamics in embedding spaces. *CoRR*, abs/2205.12558.

Joshua Maynez, Shashi Narayan, Bernd Bohnet, and Ryan McDonald. 2020. On faithfulness and factuality in abstractive summarization. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 1906–1919, Online. Association for Computational Linguistics.

Clara Meister, Tiago Pimentel, Gian Wiher, and Ryan Cotterell. 2022. Locally typical sampling. *Transactions of the Association for Computational Linguistics*.

Krishna Pillutla, Swabha Swayamdipta, Rowan Zellers, John Thickstun, Sean Welleck, Yejin Choi, and Zaid Harchaoui. 2021. MAUVE: Measuring the gap between neural text and human text using divergence frontiers. In *Advances in Neural Information Processing Systems*, volume 34, pages 4816–4828. Curran Associates, Inc.

Lianhui Qin, Sean Welleck, Daniel Khashabi, and Yejin Choi. 2022. COLD decoding: Energy-based constrained text generation with langevin dynamics. In *Advances in Neural Information Processing Systems*.

Sean Welleck, Ilia Kulikov, Stephen Roller, Emily Dinan, Kyunghyun Cho, and Jason Weston. 2020. Neural text generation with unlikelihood training. In *Proceedings of the 8th International Conference on Learning Representations*.

Kevin Yang and Dan Klein. 2021. FUDGE: Controlled text generation with future discriminators. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 3511–3535, Online. Association for Computational Linguistics.