Generating Text from Language Models

Afra Amini[1] **Ryan Cotterell**[1] **John Hewitt**[2]
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 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, techniques for improving or specifying preferences in these generated texts rely mostly on intuition-based heuristics. Further, there lacks a unified presentation of their motivations, practical implementation, successes and pitfalls. Practitioners must, therefore, choose somewhat blindly between generation algorithms—like top-p sampling or beam search—which can lead to wildly different results. At the same time, language generation research continues to criticize and improve the standard toolboxes, further adding entropy to the state of the field. 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 cover a wide range of empirically-observed problems (like degradation, hallucination, repetition) and their corresponding proposed algorithmic solutions from recent research (like top-p sampling and its successors). We will then discuss a subset of these algorithms under a unified light; most stochastic generation strategies can be framed 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 make up random facts (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, and the empirically-observed shortcomings of modern models. Second, we will cover the basics of language generation, presenting both deterministic and stochastic decoding strategies. Third, we present a unifying framework for sampling adapters, the family of methods often used for stochastic decoding that transform the output of a model according to qualitatively motivated rules. Finally, we will discuss several methods for controlled text generation, i.e., methods that allow

Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics
Volume 6: Tutorial Abstracts, pages 27–31

users 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 and Preferred Venue

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. Given the rising popularity of tasks involving language generation, we estimate an audience of approximately 100 people. We would be willing to present this tutorial at both ACL and EMNLP.

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 generative model as a tool to accomplish the task. More precisely, practitioners take a textual dataset $\mathcal{D} = \{\mathbf{y}_n\}_{n=1}^{N}$—an N-sized set of strings over some vocabulary $\mathcal{V}$—and treat it as a set of independently and identically distributed samples from a distribution $p(\mathbf{y})$, where $\mathbf{y} \in \mathcal{V}^*$. We will use p to denote the *true* distribution—the distribution defined by the task's hypothetical generative process, from which we drew our samples.

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. In practice, the vast majority of these models, which we denote as $p_{\boldsymbol{\theta}}$, are trained to minimize the empirical KL-divergence with the finite set of samples $\mathcal{D}$.

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 cross entropy. Still, language generation techniques are used both to avoid known failure modes and to coax more desirable properties out of language models. In our tutorial, we will discuss the following failure modes of language models, among others:

- **Low-quality low-probability words.** Due to their use of the softmax to compute $p_{\boldsymbol{\theta}}(y_t \mid \mathbf{y}_{<t})$, 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.

High- and low-entropy generation. In some discussions around language generation, tasks are often discussed as "open-ended" (for example, story generation) or not (for example, machine translation). The techniques and histories of the corresponding literatures are often somewhat separate. We will discuss open-endedness as a scale well-described by the **entropy** of the true distribution a task specifies, as well as the entropy of the desired output behavior of the model. So, for example, in machine translation, the true distribution over correct translations has a relatively low entropy, even though texts (especially long ones) have a number of roughly equivalent translations; further, it is common to look for only the "most likely" translation. Story generation typically has more entropy (the set of nice stories is large) and the generation of arbitrary web text has more entropy still; further, the notion of the "most likely" web text document is unintuitive, to say the least. We will thus discuss models and the methods used to generate from them with the concept of entropy in mind, rather than using the more traditional (albeit qualitative) notion of open-endedness.

3.2 Part 2: Language Generation

Given a pretrained language model $p_{\boldsymbol{\theta}}(\cdot \mid \mathbf{y}_{<t})$, how does ones generate text from it? There is a plethora of options available. We split these into two subgroups: deterministic and stochastic decoding strategies (Wiher et al., 2022).

Deterministic decoding. In tasks with one (or only a small number of) correct answers, researchers typically rely on deterministic strategies, which "search" over the support of the distribution $p_\theta(\cdot \mid \mathbf{y}_{<t})$ for this correct answer. In short, these strategies rely on some quantification of a string $\mathbf{y}$'s quality, e.g., its probability under p_θ, and they try to find the string which maximizes it. Finding this string, however, is an NP-hard problem (Chen et al., 2018). These decoding strategies thus propose heuristic methods for performing this search. Beam search, for instance, searches for this maximizing string by iteratively expanding all substrings $\mathbf{y}_{<t}$, albeit at any given point, keeping only the k best substrings found so far.

Stochastic decoding. In tasks for which text diversity is a desired attribute, stochastic strategies are usually employed. Typically, these strategies work incrementally: first, one word is sampled from $p_\theta(\cdot \mid \mathbf{y}_{<t})$; this word is then appended to the context, producing $\mathbf{y}_t$; the next word is then sampled from $p_\theta(\cdot \mid \mathbf{y}_{<t+1})$. Sampling stops at some pre-determined length, or once the end-of-string token is sampled. Following this iterative process, we sample strings according to distribution $p(\mathbf{y})$. Several issues arise from simply sampling from $p(\mathbf{y})$, though. In the next section, we dive into different methods to mitigate these issues.

3.3 Part 3: Sampling Adapters

As discussed in part 1, due to the structure of most probabilistic language generators, no token in the vocabulary can be assigned a probability of zero under $p_\theta(\cdot \mid \mathbf{y}_{<t})$. Even if a model assigns inappropriate tokens very low probability, there is still the chance of sampling them when using stochastic decoding strategies. 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). While intuitively we might expect this issue to only occur with low probability, a concrete example proves otherwise. 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.

In an attempt to prevent this issue, several works have proposed simple modifications to the sampling distribution to exclude undesirable tokens from the candidate pool. Two prominent examples are nucleus and top-k sampling, both of which truncate the distribution to some subset of its most probable items (and then renormalize it). 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).

In this part of the tutorial, 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 discuss the motivation and formulation of several popular sampling adapters (Fan et al., 2018; Holtzman et al., 2020; Basu et al., 2021; Meister et al., 2022; Hewitt et al., 2022), describing the problems that they mitigate (such as sampling incoherent tokens) as well as the problems that they introduce (such as repetitive generations). Further, we will show results from prior works comparing these methods. 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.

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 sampling from a conditional distribution $p_\theta(\mathbf{y} \mid c)$ instead. We split prior work on sampling from this distribution into two groups: autoregressive and non-autoregressive controlled generation methods.

Autoregressive generation. Similar to the decoding strategies discussed earlier, these methods

incrementally generate text one token at a time, in a sequential manner. At each step of the generation, a token y_t is sampled with probability $p(y_t \mid \mathbf{y}_{<t}, c)$—which, following Bayes' rule, is proportional to $p_\theta(y_t \mid \mathbf{y}_{<t})\, p(c \mid \mathbf{y}_{\leq t})$ (Yang and Klein, 2021). In other words, at each timestep, the score of a candidate y_t under the language model $p_\theta(y_t \mid \mathbf{y}_{<t})$ is reweighted according to the probability that $\mathbf{y}_{\leq t}$ satisfies the control target: $p(c \mid \mathbf{y}_{\leq t})$. This control target is usually estimated with a supervised classifier parameterized by ϕ: $p_\phi(c \mid \mathbf{y}_{\leq t})$ (Ghazvininejad et al., 2017; Holtzman et al., 2018). The implication of this approach is that we need to have reliable estimates of how much a prefix satisfies the desired control. However, this 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 for controlled generation.

Non-autoregressive 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, non-autoregressive strategies propose to sample the whole sequence $\mathbf{y}$ at once. This is usually done by designing Markov-Chains based off of some (autoregressive) language model $p_\theta(\mathbf{y})$ 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).

3.5 Breadth of Research Covered

This tutorial is intended as a primer for recent language generation techniques. To this end, it will need to pull on research from a large number of authors, spanning several institutions. Explicitly, the background section on language modeling will cover, for example, works from OpenAI, Google, AI2, and DeepMind, as institutions with the resources to train these large language models and make them publicly available. The introduction to generation will touch on prominent methods, such as beam search (Graves, 2012), nucleus sampling (Holtzman et al., 2020), Mirostat (Basu et al., 2021), top-k sampling (Fan et al., 2018), typical decoding (Meister et al., 2022) and top-η sampling (Hewitt et al., 2022). The controlled generation section will summarize work on weighted decoding (Ghazvininejad et al., 2017; Holtzman et al., 2018), FUDGE (Yang and Klein, 2021), and recently proposed HMC-based methods (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, ETH and Cambridge), who are originally from 3 continents (Asia, North America, and South America). Lastly, it will discuss work produced by authors spanning many institutions and backgrounds (see § 3.5).

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*.

Yining Chen, Sorcha Gilroy, Andreas Maletti, Jonathan May, and Kevin Knight. 2018. Recurrent neural networks as weighted language recognizers. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 2261–2271, New Orleans, Louisiana. Association for Computational Linguistics.

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.

Alex Graves. 2012. Sequence transduction with recurrent neural networks. *CoRR*, abs/1211.3711.

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*.

Gian Wiher, Clara Meister, and Ryan Cotterell. 2022. On Decoding Strategies for Neural Text Generators. *Transactions of the Association for Computational Linguistics*, 10:997–1012.

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.