Order and Learning in Sequential Neural Structured Prediction

by

Sean Welleck

A dissertation submitted in partial fulfillment

of the requirements for the degree of

Doctor of Philosophy

Department of Computer Science

New York University

January, 2021

Kyunghyun Cho

Acknowledgements

Thank you to Kyunghyun Cho for advising me throughout the PhD, for teaching me how to decide which ideas are worth pursuing and how to investigate them, and for the countless conversations that have shaped my approach to research and life in general. Kyunghyun's enthusiasm, creativity, and rigor are qualities that I aspire to emulate, along with his ability and willingness to provide guidance on everything from the lowest level details to the highest level ideas.

I am grateful to Zheng Zhang (ZZ) for advising me, helping me to explore a variety of research areas, and for always being flexible in advising and collaboration. I am continually inspired by ZZ's ability to find the essence of an idea, by his lessons on forming and leading research teams, and by his creative perspective that spans multiple research fields.

I thank Jason Weston for advising me at FAIR, where he continually provided interesting ideas and guidance. I also thank Arthur Szlam, Stephen Roller, Emily Dinan, Margaret Li, and Y-Lan Boureau for collaborating at FAIR.

Thank you to New York University for the opportunity to do a PhD in conjunction with NYU Shanghai. I have fond memories of my two semesters in Shanghai, and I look forward to visiting in the future. I thank Keith Ross for his guidance at NYU Shanghai and for serving on the thesis committee. I learned invaluable lessons from Keith's ability to carefully examine an idea from first principles, and from discussing ideas in our reading group that would often last far beyond the allotted hour. The logistics of moving to and from Shanghai would have been intractable without the help of Dean Eric Mao, Vivien Du, and Chloe Ma. Thank you to Yiming Zhang for many great discussions, and to Gus Xia, Mufei Li, Che Wang, Jialin Mao, Jinjing Zhou, Zixin Yao, Yu Gai, and Yanqiu Wu for making NYUSH an intellectual environment that feels like home.

Thank you to the NYU ML^2 and CILVR lab members for creating a great research environment. I thank Ilia Kulikov, with whom I co-authored the work on unlikelihood and consistency and had many conversations, programming sessions, and side projects that were invaluable to this thesis. Thank you to Kianté Brantley and Hal Daumé III for their collaboration on non-monotonic text generation, and to He He for serving on the thesis committee. I thank Jaedeok Kim for his lessons on mathematical precision and rigor. Thank you to Isaac Henrion, Jake Zhao, Jason Lee, Elman Mansimov, Alex Wang, Richard Pang, Minjie Wang, Conrad Christensen, Jinyang Li, Sebastian Ruder, Phil Blunsom, Chris Dyer, Yoon Kim, Raphael Shu, David Ha, and many others for impactful conversations during the PhD.

Finally, thank you to my family for their constant support, whether at home, across the country, or across the globe.

Abstract

Structured objects such as sets, trees, and sequences appear in a variety of scientific and industrial domains. Developing machine learning methods that *generate* these objects is of interest for both scientific understanding and practical applications. One approach to generating structured objects, called *sequential neural structured prediction*, decomposes generation into a sequence of predictions, with each prediction made by a deep neural network. Choosing an appropriate sequential representation of each structured object and selecting an effective learning objective are key to adopting this approach. The standard method for learning specifies a canonical ordering of elements in the sequential representation and maximizes the likelihood of the resulting sequences. This thesis develops two streams of research that explore alternatives to this fixed-order, maximum likelihood approach for sequentially generating sets, trees, and sequences, with a focus on natural language processing applications.

In the first part of the thesis, we focus on text generation and study degenerate properties of fixed-order maximum-likelihood learning that are surfaced in practice, motivating new learning methods. We characterize the degeneracy using three properties that are observed in generated text: non-termination, logical incoherence, and repetition. To study non-termination, we develop theory that allows us to formally prove that conventional text generation methods can generate infinite-length sequences with high probability. To study logical incoherence, we create a dataset for investigating the degree to which a model logically contradicts its preceding statements. For reducing the three types of degeneration, we develop *unlikelihood training*, a new learning method which penalizes task-specific textual properties.

In the second part of the thesis, we remove the requirement of a fixed generation order by developing a learning framework, called *non-monotonic generation*, that yields models capable of selecting input-dependent generation orders. This flexibility is natural for set-structured objects, which lack an inherent order. For ordered objects, such as text, the selected orders induce an interpretable latent structure and allow us to study whether canonical orders such as left-to-right are optimal for learning. We use non-monotonic generation for generating multisets, parse trees, and text.

The investigations and techniques presented in this thesis lead to promising directions for future work.

Table of Contents

List of Figures

List of Tables

List of Appendices

1 Introduction

Structured objects such as sets, trees, and sequences appear in a variety of scientific and industrial domains. Developing machine learning methods that *generate* these objects is of interest for both scientific understanding and practical applications. This thesis focuses on analyzing and developing methods for sequential neural structured prediction, a machine learning approach to generating structured objects. *Structured prediction* involves learning a mapping between an input space and an output space whose elements have structure; that is, complex dependencies between variables. *Neural* refers to parameterizing the mapping with a neural network, which allows for end-to-end learning of expressive models. *Sequential* means that each structured object is predicted in a sequence of dependent steps, allowing the model to adjust its predictions based on the past.

The conventional approach to sequential neural structured prediction involves executing the sequence of predictions in a canonical order. Learning consists of maximizing the likelihood of these ordered sequences. For instance, we may learn a sentence generation model by maximizing the likelihood of word sequences that are ordered from left to right. This thesis presents two streams of research that explore alternatives to this fixed-order, maximum likelihood approach for sequentially generating sets, trees, and sequences, with a focus on natural language processing applications.

In the first stream of research, we focus on text generation and study degenerate properties of fixed-order maximum-likelihood learning that are surfaced in practice, motivating new learning methods. We characterize the degeneracy using three properties that are observed in generated text: non-termination, logical incoherence, and repetition. To study non-termination, we develop theory that allows us to formally prove that conventional text generation methods can generate infinite-length sequences with high probability. To study logical incoherence, we create a dataset for investigating the degree to which a model logically contradicts its preceding statements. For reducing the three types of degeneration, we develop *unlikelihood training*, a new learning method which penalizes task-specific textual properties.

In the second stream of research, we remove the requirement of a fixed generation order by developing a learning framework, called *non-monotonic generation*, that yields models with input-dependent generation orders. This flexibility is natural for set-structured objects, which lack an inherent order. For ordered objects, such as text, the selected orders induce an interpretable latent structure and allow us to study whether canonical orders such as left-to-right are optimal for learning. We use non-monotonic generation for generating multisets, parse trees, and text.

The investigations and techniques presented in this thesis lead to promising directions for future work.

1.1 Thesis Outline

Part I provides background about sequential neural structured prediction that is necessary for understanding the contributions in the thesis, including conventional model, learning, and inference methods, as well as applications. Parts II and III contain the novel contributions in this thesis.

Part II analyzes degenerate properties of the conventional approach to sequential neural text generation and develops alternative methods. Section 5 characterizes degeneration in terms of repetition, logical incoherence, and non-termination. Section 6 formally analyzes non-termination, showing that a model paired with a conventional decoding algorithm can decode infinite-length sequences with high probability, known as *inconsistency*. Section 7 develops *unlikelihood training*, which is used to penalize degeneration that is surfaced in model-generated text, such as repetition. Section 8 develops *dialogue natural language inference*, a dataset for measuring a model's logical incoherence, then uses the dataset to improve coherence through unlikelihood training. Part II concludes with a discussion and outlines directions for future work (§9).

Part III develops and applies the *non-monotonic generation* learning framework. Following a background section (§11), Section 12 presents the learning framework. Each subsequent section uses non-monotonic generation for a different class of outputs. Section 13 develops *multiset prediction* for generating multisets, and applies it to multiple object classification. Section 14 develops the *sequential graph dependency parser* for generating parse trees. For sequences, Section 15 develops a *binary tree generation policy* that learns input-dependent text generation orders defined by binary tree traversals. Part III concludes with a discussion and outlines directions for future work (§16).

Tables 1 and 2 classify the methods that are developed in Parts II and III of this thesis.

Class	**Method**	
Theory	Inconsistency	§6
Learning	Unlikelihood Training	§7
Data	Dialogue NLI	§8

Table 1: **Part II** Neural Text Generation

Class	**Method**	
Multisets	Multiset Prediction	§13
Trees	Sequential Graph Dependency Parser	§14
Sequences	Binary Tree Generation Policy	§15

Table 2: **Part III** Non-Monotonic Generation

Part I

Sequential Neural Structured Prediction

2 Introduction

In many scientific and industrial application domains, the data is *structured*, meaning that each data item is composed of multiple parts that are connected by complex underlying relations. For instance, a sentence is composed of words that are connected by relations that convey meaning to a reader. Rather than formally defining *structure* in general, we will assume each structured object can be decomposed into discrete elements, $\mathbf{y} = \{y_1, \ldots, y_{|y|}\}$, with underlying relations between the elements that constitute notions of structure that are present in examples such as objects in a visual scene, edges which form a parse tree, or words that comprise a document. A collection of these objects constitutes a *structured space*, such as the set of all sentences. In this thesis we will consider structured spaces $\mathcal{Y}$ whose objects $\mathbf{y} \in \mathcal{Y}$ are sets, trees, or sequences.

This thesis is concerned with *structured prediction*, which consists of finding a mapping $f : \mathcal{X} \to \mathcal{Y}$ between an input space $\mathcal{X}$ and a structured output space $\mathcal{Y}$. An example is machine translation, which finds a mapping between sentences in a source language, such as Japanese, to sentences in a target language, such as English. Structured prediction approaches can be partitioned into *global* and *local* methods. Global methods assign a score to each item $\mathbf{y}$ in the output space, then search for the highest scoring output. Local methods assume that each structured object can be decomposed into discrete elements, $\mathbf{y} = \{y_1, \ldots, y_{|\mathbf{y}|}\}$, and predict the object sequentially. Each intermediate prediction is over the space of elements, rather than the full output space, and is hence 'local'. We will use local methods in this thesis, which we call *sequential structured prediction*.

Sequential structured prediction relies on decomposing an object into elements, and choosing a sequence of predictions. The conventional approach is to choose a *canonical ordering* of elements, making the decomposition into a sequence $\mathbf{y} = (y_1, \ldots, y_{|\mathbf{y}|})$, and predicting the elements according to this order. For text, it may seem obvious that we should decompose an (English) sentence into a left-to-right sequence of words, and hence predict from left to right. But how should we order the object's elements when the object is a set, or a tree? For text, is it optimal to choose the intuitive left-to-right ordering?

Once we have selected a canonical ordering, we must choose a *model* for the local predictions. The conventional approach is to learn a probabilistic model over sequences that decomposes according to the chain rule of probability, known as an *autoregressive model*. In this case, each local prediction is made based on a conditional distribution $p_\theta(y_t | y_{<t}, \mathbf{x})$. Using a *neural network* to parameterize each conditional distribution allows for learning complex, non-linear mappings. Although several of the techniques in this

thesis are agnostic to the parameterization, neural networks are an expressive model family that have led to significant advancements in sequential structured prediction, and are thus worthy of investigation as a separate subclass, which we call *sequential neural structured prediction*.

After specifying a family of parameterized models, we *learn* the parameters using a collection of data. The conventional approach to learning the parameters of a neural autoregressive model is to find parameters that maximize the probability of the data, known as *maximum likelihood estimation* (MLE). Finally, given a learned model, we perform *inference* to generate a sequence. In our setting, inference is also known as *decoding*, which is performed using a *decoding algorithm*.

This background part of the thesis (§I) introduces notation and provides an overview of the conventional models, learning algorithms, and inference methods for sequential neural structured prediction, as well as representative applications. This will provide a background for subsequent parts of the thesis (§II,III) that analyze, and depart from, the conventional approach.

2.1 Preliminaries and Notation

We denote an output space as $\mathcal{Y}$ with elements $\mathbf{y} \in \mathcal{Y}$. We often use bold letters to emphasize the fact that we are working with objects composed of multiple elements, such as vectors, sequences, or sets. Often, $\mathbf{y}$ is a sequence, $\mathbf{y} = (y_1, \ldots, y_T)$, where we write the sequence length as T instead of $|\mathbf{y}|$, with the understanding that sequence lengths can vary. Each element of a sequence, $y_i \in \mathbf{y}$, is called a *token*. The finite set of all tokens $\mathcal{V} = \{1, \ldots, V\}$ is called a *vocabulary*, where $y_i \in \mathcal{V}$ for all i.

We denote an input space as $\mathcal{X}$, where each element $\mathbf{x} \in \mathcal{X}$ is called an *input* or a *context*. The context $\mathbf{x}$ can be an arbitrary object, such as an image or set, and when $\mathbf{x}$ is a sequence it is denoted in a similar manner as $\mathbf{y}$, i.e. $\mathbf{x} = (x_1, \ldots, x_T)$ with $x_i \in \mathcal{V}$.

We use $p_\theta(\mathbf{y})$ to denote the mass or density function of a random variable $\mathbf{y}$ with support over a subset of $\mathcal{Y}$, parameterized by a vector θ. We overload notation by using $p_\theta(\mathbf{y})$ to both denote the function $p_\theta : \mathcal{Y} \to [0, 1]$, as well as the function evaluated at a particular $\mathbf{y} \in \mathcal{Y}$. To emphasize the former case we may use $p_\theta(\cdot)$. We denote conditional distributions in a similar manner, using $p_\theta(\mathbf{y}|\mathbf{x})$ or $p_\theta(\cdot|\mathbf{x})$. We use $p_\theta(y_t|y_{<t}, \mathbf{x})$ to denote the conditional distribution of a token y_t given 'previous' tokens $y_{<t} = (y_1, \ldots, y_{t-1})$, meaning the tokens are part of a sequence $\mathbf{y} = (y_1, \ldots, y_{t-1}, y_t, \ldots, y_T)$.

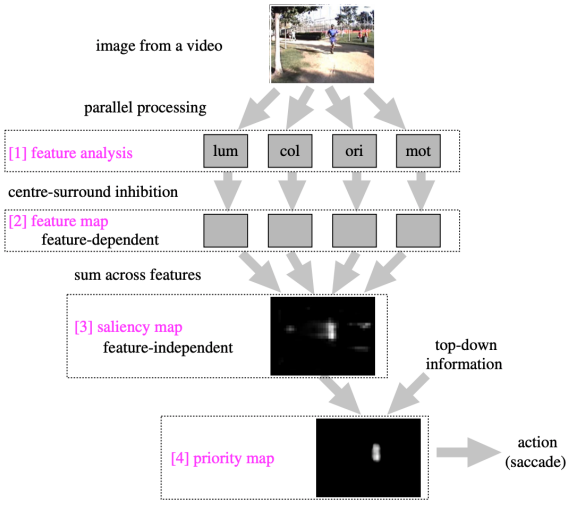


Fig. 1: An illustration of the saliency and priority maps influencing an eye movement (saccade). From (Lamme and Roelfsema 2000).

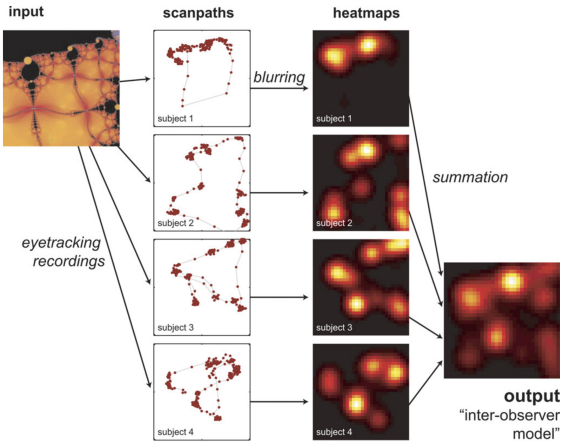


Fig. 2: Human scanpaths (saccade sequences) for the image on the left. Each heatmap indicates fixation frequency. From (Peters et al. 2005).

3 Applications

In this section we overview three applications of sequential neural structured prediction that will be visited throughout the thesis. These concrete applications can be kept in mind as we overview the conventional model, learning, and inference methods at an abstract level in the subsequent sections.

3.1 Multisets: Multiple Object Classification

Picture yourself in a park, and consider the task of listing the objects that you see. One aspect to notice is that this process is *sequential*. More specifically, in the human visual system, processing a scene with multiple attentional shifts may be interpreted as a feed-forward process followed by sequential, recurrent stages (Lamme and Roelfsema 2000). Computational models and various forms of psychophysical and neuro-biological evidence suggest that the human visual system compiles visual input into a *saliency map* that encodes the conspicuity of locations in the visual field in a parallel, feed-forward process (Koch and Ullman 1985; Itti et al. 1998; Veale et al. 2017). Goal-specific relevance of locations are then incorporated to form a *priority map*, which is used to select the next target of attention (Fecteau and Munoz 2006; Veale et al. 2017). Viewing a scene consists of a sequence of these shifts in attention, with the salience of previous attentional targets decreased by a process referred to as *inhibition of return* (Raymond M. Klein 2000; Fecteau and Munoz 2006). Figure 1 illustrates one step of this procedure, and Figure 2 shows sequential viewings of an image. These findings in the human visual system motivate investigating machine learning systems that sequentially process a scene after computing a representation from low-level information. Strong performance of neural networks in both image classification and sequential

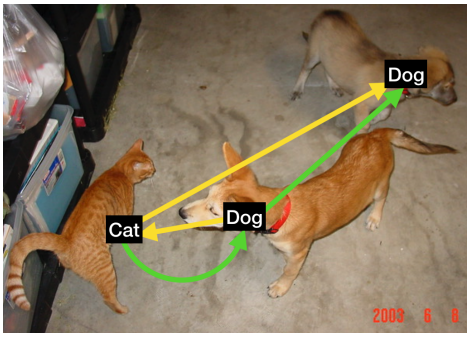


Fig. 3: A scene with a multiset of animals $\{\texttt{dog}, \texttt{dog}, \texttt{cat}\}$. The green and yellow arrows sequentially represent the multiset as $(\texttt{cat}, \texttt{dog}, \texttt{dog})$ and $(\texttt{dog}, \texttt{cat}, \texttt{dog})$, respectively.

Fig. 4: A multiset of digits. Any ordering of the digits $\{0, 0, 0, 0, 0, 1, 1, 2, 2, 3, 5, 5, 5, 6, 6, 6, 7, 7, 9, 9\}$ is a valid labeling. Is there a 'natural' ordering that you use when listing the digits?

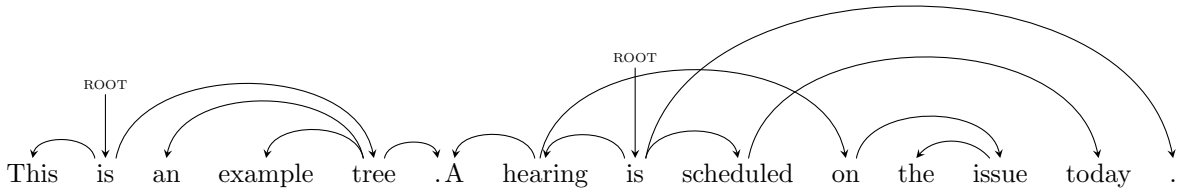


Fig. 5: Two example dependency trees.

tasks suggests that sequential neural structured prediction is a worthwhile approach to investigate for this object listing task.

Returning to the imaginary park, a second aspect to notice is that the order in which you list the objects does not matter. Moreover, certain objects can occur more than once. These two properties suggest that the objects form a *multiset*, which is an unordered collection (set) that allows for duplicate elements. Thus we can view this listing task, called *multiple object classification*, as mapping from an input image $\mathbf{x}$ to a multiset $\mathbf{y} = \{y_1, \ldots, y_T\}$, which is an instance of *multiset prediction*. Figures 3 and 4 show example images that each contain a multiset of objects. A naïve approach to sequential structured prediction for multisets is to ignore the unordered nature of the multiset, and treat it as a sequence. However, it is of scientific interest to develop methods which respect the unordered structure, in part because the choice of a canonical ordering can impact performance. We will quantify this impact and explore multiset prediction in more depth using a learning method that does not require selecting a canonical ordering, and instead allows the model to choose any generation order (§13).

3.2 Trees: Dependency Parsing

The next application relates to tree-structured representations of natural language sentences, known as *dependency trees*. A dependency tree is a directed, acyclic graph whose nodes are words in a sentence,

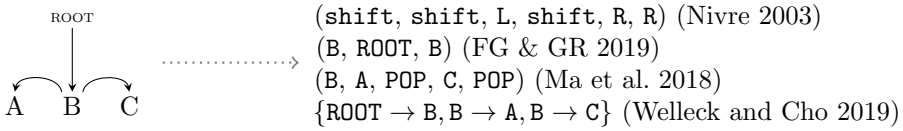


Fig. 6: Sequence and set representations of the dependency tree on the left.

along with an extra root node. Each word is the target of exactly one directed edge, as seen in the examples in Figure 5. The problem of mapping an input sentence $\mathbf{x}$ to a dependency tree $\mathbf{y}$ is called *dependency parsing*.

There is a rich history of framing dependency parsing as sequential structured prediction, with creative methods for representing a dependency tree $\mathbf{y}$ as a sequence. Figure 6 shows three example sequential representations, along with a set-based representation. The traditional arc-standard method (Yamada and Matsumoto 2003; Nivre 2003) processes text from left to right, representing a parse as a sequence of operations {`shift, L, R`} involving an imaginary stack data structure. Recent variations build the parse tree in left-to-right (Fernández-González and Gómez-Rodríguez 2019) or depth-first (Ma et al. 2018) orders. A natural alternative is to treat the dependency tree as a set of edges, with the constraint that only certain sets are valid dependency trees. As in multiset prediction, it is unclear which ordering of the edges is best to use for training a parsing model. We will develop a learning method that allows the model to choose an edge ordering, rather than imposing a fixed ordering such as left-to-right or depth-first (§14).

3.3 Sequences: Conditional Language Modeling and Generation

The next application deals with modeling and generating sequences, specifically text. *Language modeling* is the task of learning a distribution $p_\theta(\mathbf{y})$ over text sequences $\mathbf{y}$. The learned distribution, called a *language model*, can be used for *scoring*, i.e. measuring the probability of a sequence, and for *generation*, i.e. producing a new sequence $\hat{\mathbf{y}}$. In subsequent sections we will formally define language modeling and discuss different way of generating sequences. For now, we will focus on applications, each of which involves *conditional language modeling*, the task of modeling distributions $p_\theta(\mathbf{y}|\mathbf{x})$.

3.3.1 Text completion. In the task of *text completion*, a model is given a sub-sequence and must fill in the remaining contents of the sequence. Commonly the model is given a prefix and the sequence is completed from left-to-right, called *prefix completion*. In this case, given a ground-truth sequence, $\mathbf{y} \sim p_*(\mathbf{y})$, we use its initial tokens as a context, $\mathbf{x} = (y_1, \ldots, y_k)$, and produce a continuation $(\hat{y}_{k+1}, \ldots, \hat{y}_T) \sim p_\theta(\cdot|\mathbf{x})$ with a conditional language model. In *non-monotonic text completion*, the model completes an arbitrary

Type	Completions
Prefix	my favorite food is mac and cheese !
	my favorite food is pizza .
	at the 2016 Olympics , Japan won twelve gold medals .
	at the 2016 Olympics , Japan won three bronze medals in swimming.
Non-monotonic	my favorite food is mac and cheese !
	whats your favorite food ? mine is pizza !
	at the 2016 Olympics , Japan won more gold medals than Ukraine .
	in the 2012 Olympics , Japan won eight bronze medals in swimming .

Fig. 7: Text completion examples. Each context $\mathbf{x}$ is in plain text, and each continuation $\hat{\mathbf{y}}$ is underlined.

sub-sequence. That is, given a sequence $\mathbf{y}$ and a partition of its timesteps $\{t_{x_1}, \ldots, t_{x_m}\}, \{t_{y_1}, \ldots, t_{y_n}\}$, the context is $\mathbf{x} = (y_{t_{x_1}}, \ldots, y_{t_{x_m}})$ and the model's output is $(\hat{y}_{t_{\hat{y}_1}}, \ldots, \hat{y}_{t_{\hat{y}_n}})$. Figure 7 shows examples.

Modeling. For prefix completion, the predominant modeling approach is to model *next-token distributions* using the factorization $p_\theta(\mathbf{y}|\mathbf{x}) = \prod_{t=1}^{T} p_\theta(y_t|\mathbf{y}_{<t}, \mathbf{x})$, known as an *autoregressive model*. The learned model is used to produce full continuations, for instance by choosing the most likely next-token or sampling from the conditional distribution at each step. A model is trained using a dataset $\mathcal{D} = \{(\mathbf{x}^{(i)}, \mathbf{y}^{(i)})\}_{i=1}^{N}$ formed by extracting prefixes from a collection of sequences as discussed above. We will formalize the model, generation, and training procedures further in the next section (§4.1.1).

Non-monotonic text completion requires modifying either the input and output format or the model. For instance, Donahue et al. (2020) mark each uncompleted segment with a special mask token, then predict the masked segments left-to-right, separated by another special token. Using their method, the first non-monotonic example in Figure 7 would consist of $\mathbf{x} = (\langle \mathrm{M} \rangle, \texttt{favorite}, \texttt{food}, \langle \mathrm{M} \rangle, \texttt{!}, \langle \mathrm{eos} \rangle)$ and $\mathbf{y} = (\texttt{my}, \langle \emptyset \rangle, \texttt{is}, \texttt{mac}, \texttt{and}, \texttt{cheese}, \langle \emptyset \rangle)$. One can then train an auto-regressive model on examples of this form. Instead of modifying the input and output format, we will explore an alternative that modifies the model to generate sequences in a non-left-to-right, more generally a *non-monotonic*, order (§15).

Evaluation. Evaluating a text completion model typically consists of evaluating its language modeling and generation capabilities. Intuitively, a good language model assigns high probability to sequences from the true distribution that it is modeling. Thus language modeling capability is quantified by *perplexity*, a metric that is inversely-proportional to the probability that the model assigns to a set of sequences,

$$\text{perplexity}(p_\theta, \mathcal{D}) = 2^{-\frac{1}{T_{\text{total}}} \sum_{i=1}^{N} \log_2 p_\theta(\mathbf{y}^{(i)}|\mathbf{x}^{(i)})}, \tag{1}$$

where $\mathcal{D} = \{(\mathbf{x}^{(i)}, \mathbf{y}^{(i)})\}_{i=1}^{N}$ is a dataset of N examples with total length $T_{\text{total}} = \sum_{i=1}^{N} |\mathbf{y}^{(i)}|$ that is assumed to contain samples from the true distribution. Perplexity is defined on $[1, \infty)$, with 1 meaning a perfect model (assigning probability 1 to $\mathcal{D}$), and a higher perplexity meaning a worse model. For

	A	B	$\langle eos \rangle$
A	0.4	0.1	0.5
B	0.3	0.3	0.4
$\langle bos \rangle$	0.4	0.6	0.0

$p_{\theta_1}(col|row)$

	A	B	$\langle eos \rangle$
A	0.4	0.1	0.5
B	0.6	0.1	0.3
$\langle bos \rangle$	0.4	0.3	0.3

$p_{\theta_2}(col|row)$

$\mathcal{D}$

$(\langle bos \rangle, A, A, \langle eos \rangle)$

$(\langle bos \rangle, B, A, \langle eos \rangle)$

Fig. 8: Two models with the same perplexity on the dataset $\mathcal{D}$, but different generations under greedy decoding. The conditional distributions are independent of time and only depend on the preceding token. The log probability of $\mathcal{D}$ is $\log(0.4*0.4*0.5)+\log(0.6*0.3*0.5)$ under both p_{θ_1} and p_{θ_2}, yielding the same perplexity. Greedy decoding generates a sequence by beginning with $\hat{y}_0 = \langle bos \rangle$, then repeatedly selecting the most probable token given the preceding token, $\hat{y}_t = \arg\max p_\theta(\cdot|\hat{y}_{t-1})$, until $\langle eos \rangle$ is selected. This yields $(\langle bos \rangle, B, \langle eos \rangle)$ for p_{θ_1} and $(\langle bos \rangle, A, \langle eos \rangle)$ for p_{θ_2}.

autoregressive models, perplexity intuitively measures the average number of samples needed to obtain the true next-token. Measuring perplexity does not involve generating sequences and hence does not fully evaluate a model's generation capability. For example, Figure 8 shows two models that have the same perplexity on a dataset, yet generate different sequences when the most probable next-token is repeatedly selected, called *greedy decoding*. As a result, we must additionally directly evaluate a model's generation capability.

Intuitively, a model is good at generation when it yields completions that resemble samples from the true distribution that it is modeling. Unfortunately, the notion of 'resemble' is not well-defined, and we do not have access to the true distribution. Furthermore, comparing a generation against a ground-truth completion from the dataset is unreliable since there are often many sensible completions (e.g. consider the *my favorite food* example in Figure 7). As a result, evaluating generation quality involves a mixture of property-specific metrics and human evaluation. A *property-specific metric* measures a single aspect of the generation, such as the amount of repetition, number of unique tokens or bigrams, or the length distribution (mean, standard deviation, etc.) of generated sequences. Comparing these metrics against those of the ground-truth data can identify weaknesses in a generation model.

Human evaluation is used to judge the overall quality of generations, including aspects such as 'humanness', 'coherence', and 'style' that are difficult to formally define and capture in property-specific metrics. An example human evaluation setup presents a human evaluator with a context $\mathbf{x}$ and completions $\mathbf{y}_1$ and $\mathbf{y}_2$ that come from two different sources (e.g. the ground-truth data, different models), and the evaluator picks the 'better' completion, where the notion of 'better' can be clarified through instructions or examples given to the evaluator. With a sufficient number of evaluators and trials, one can perform a hypothesis test to reject the null hypothesis that two sources result in equal evaluation scores. We will see a detailed instantiation of this approach in the section on unlikelihood training (§7).

Discussion. Text completion is useful for studying the behavior of language models due to its generality (see, e.g. Sutskever et al. (2011); Graves (2013); Radford et al. (2018); Holtzman et al. (2020); Welleck et al. (2020); Brown et al. (2020)). For instance, text completion encompasses story generation, unconditional language modeling (for $k = 0$), and dialogue modeling where $\mathbf{x}$ is a dialogue history and a continuation is the next utterance. The non-monotonic setting generalizes left-to-right completion, and includes applications such as ancient text restoration (Assael et al. 2020; Shen et al. 2020).

3.3.2 Machine translation.

Machine translation is the task of mapping a source sequence $\mathbf{x}$ to a target sequence $\mathbf{y}$, where the source and target sequences are written in different languages.

Modeling. The predominant modeling approach is to use a conditional language model $p_\theta(\mathbf{y}|\mathbf{x})$, where $\mathbf{x}$ is a sequence in the source language (e.g. Japanese) and $\mathbf{y}$ is a sequence in the target language (e.g. English). The standard approach is to use an autoregressive factorization, trained using a *parallel corpus* $\mathcal{D} = \{(\mathbf{x}^{(i)}, \mathbf{y}^{(i)})\}_{i=1}^{N}$, where $\mathbf{y}^{(i)}$ is a ground-truth translation of $\mathbf{x}^{(i)}$.

Evaluation. BLEU (Papineni et al. 2002) is a widely used metric for evaluating machine translation systems. It is based on matching n-grams[1] between the source and target sequence(s) in a dataset. BLEU uses *modified n-gram precision*, which counts the number of predicted n-grams that occur in the target sequence, clipped by the number of times each n-gram occurs in the target sequence. Specifically, letting $\mathcal{Y} = \{(\hat{\mathbf{y}}, \mathbf{y})\}$ be a set of (predicted, ground-truth) pairs,

$$p_n(\mathcal{Y}) = \frac{\sum_{(\hat{\mathbf{y}},\mathbf{y})\in\mathcal{Y}} \left[\sum_{\hat{\mathbf{y}}_n \in \text{n-grams}(\hat{\mathbf{y}})} [c_{\text{clip}}(\hat{\mathbf{y}}_n, \hat{\mathbf{y}}, \mathbf{y})] \right]}{\sum_{\hat{\mathbf{y}}\in\mathcal{Y}} \left[\sum_{\hat{\mathbf{y}}_n \in \text{n-grams}(\hat{\mathbf{y}})} [c(\hat{\mathbf{y}}_n, \hat{\mathbf{y}})] \right]}, \tag{2}$$

where n-grams$(\hat{\mathbf{y}}) = \{(y_t, \ldots, y_{t+n}) \mid t \in 1, \ldots, |\hat{\mathbf{y}}| - n\}$ returns the set of n-grams in the sequence $\hat{\mathbf{y}}$, $c(\hat{\mathbf{y}}_n, \hat{\mathbf{y}})$ counts the number of times that the n-gram $\hat{\mathbf{y}}_n$ occurs in $\hat{\mathbf{y}}$, and $c_{\text{clip}} = \min\left(c(\hat{\mathbf{y}}_n, \hat{\mathbf{y}}), c(\hat{\mathbf{y}}_n, \mathbf{y})\right)$. BLEU is then defined as,

$$\text{BLEU}(\mathcal{Y}) = \text{BP}(\mathcal{Y}) \cdot \exp\left(\sum_{n=1}^{N_g} w_n \log p_n(\mathcal{Y}) \right), \tag{3}$$

$$\text{BP}(\mathcal{Y}) = \begin{cases} 1 & \sum_{\hat{\mathbf{y}}} |\hat{\mathbf{y}}| > \sum_{\mathbf{y}} |\mathbf{y}|, \\ \exp\left(1 - \frac{\sum_{\mathbf{y}} |\mathbf{y}|}{\sum_{\hat{\mathbf{y}}} |\hat{\mathbf{y}}|}\right) & \sum_{\hat{\mathbf{y}}} |\hat{\mathbf{y}}| \leq \sum_{\mathbf{y}} |\mathbf{y}|, \end{cases} \tag{4}$$

[1] An n-gram is a length-n consecutive subsequence. For instance, (a, b, c) contains the 2-grams (a, b) and (b, c).

where BP is a brevity penalty that penalizes short predictions, and $\sum_{n=1}^{N_g} w_n = 1$ are weights. Other machine translation metrics include METEOR (Banerjee and Lavie 2005), translation error rate (Snover et al. 2006), and RIBES (Isozaki et al. 2010b).

Discussion. Machine translation has driven many advances in machine learning and natural language processing, including learning methods (Och 2003; Shen et al. 2016), models (Bahdanau et al. 2015; Vaswani et al. 2017), and evaluation (Papineni et al. 2002). Due in part to its prominence in the research community and as a practical application, machine translation has well-known benchmark datasets that are useful for evaluating the performance of new sequential structured prediction approaches.

3.3.3 Dialogue generation. Dialogue generation is the task of generating a conversation between agents. A dialogue is a sequence of utterances $\mathbf{u}_1, \ldots, \mathbf{u}_M$, where an *utterance* is a sequence of tokens representing a turn in a conversation, $\mathbf{u}_t = (u_{t,1}, \ldots, u_{t,T})$ with $u_{t,t'} \in \mathcal{V}$. Each utterance is made by an *agent*. For instance, an alternating two-agent dialogue which starts with agent A and ends with agent B is written as $\mathbf{u}_1^A, \mathbf{u}_2^B, \mathbf{u}_3^A, \mathbf{u}_4^B, \ldots, \mathbf{u}_M^B$. Often, the dialogue is accompanied by contextual information represented as sentences $\{\mathbf{s}_1, \ldots, \mathbf{s}_k\}$, which for instance can represent the 'personality' of each agent, a scenario in which the conversation takes place, or background knowledge.[2]

Modeling. The predominant modeling approach frames dialogue generation as *next utterance prediction*. In this setting, $\mathbf{x} = (\mathbf{s}_1, \ldots, \mathbf{s}_k, \mathbf{u}_1, \ldots, \mathbf{u}_{t-1})$ contains the context sentences and conversation history, and $\mathbf{y} = (y_1, \ldots, y_T)$ is the next utterance $\mathbf{u}_t$ in the conversation. A model $p_\theta(\mathbf{y}|\mathbf{x})$ is trained using a dataset $\mathcal{D} = \{(\mathbf{x}^{(i)}, \mathbf{y}^{(i)})\}_{i=1}^N$ constructed from a collection of conversations and contextual information.

Evaluation. The intended outcome of dialogue generation varies. In *goal-directed dialogue*, the model generates a dialogue to achieve a measurable outcome (e.g. the interlocutor presses a 'buy' button on a website), and evaluation is based on the outcome measurement. In *chit-chat dialogue*, the model generates a 'human-like' dialogue without a specified goal in mind, which often necessitates human evaluation. Human evaluation of dialogue is itself difficult, leading to research on evaluation protocols (Li et al. 2019b) and calibrating evaluation scores (Kulikov et al. 2019). Language modeling quality is also measured, using perplexity on a held out dataset.

Discussion. Generating a full dialogue at evaluation time requires multiple sequential predictions, resulting in three properties that are not present in tasks with a single sequence as output (e.g. machine translation). First, the input contains entire predicted sequences, i.e. $\mathbf{x} = (\hat{\mathbf{u}}_1^{\text{model}}, \mathbf{u}_2^B, \hat{\mathbf{u}}_3^{\text{model}}, \mathbf{u}_4^B, \ldots, \mathbf{u}_{t-1}^B)$, meaning the model's predictions are a function of its previous predicted utterances, which do not occur

[2] In general, contextual information can be treated as a vector $\mathbf{c} \in \mathbb{R}^d$, though for the datasets we use in this thesis it will be intuitively helpful to consider the case where contextual information is a set of sentences.

in the training data. Second, the dataset does not contain responses $\mathbf{u}^B_\cdot$ to each predicted utterance, necessitating human evaluation or a separate model for obtaining responses, known as *self-chat* (Li et al. 2019b). Third, each prediction must maintain *contextual consistency* with the preceding predictions; e.g. if $\hat{\mathbf{u}}_1^{\text{model}}$ is `i have a cat`, then $\hat{\mathbf{u}}_3^{\text{model}}$ cannot be `i don't have pets`. We will encounter dialogue generation and contextual consistency when we explore text degeneration and logical incoherence (§8).

4 Sequential Neural Structured Prediction

With the preceding applications in mind, we now formalize the model, learning, and inference methods used in conventional sequential neural structured prediction.

4.1 Model

Each application above consists of inputs $\mathbf{x} \in \mathcal{X}$ and outputs $\mathbf{y} \in \mathcal{Y}$. There may be many valid outputs $\mathbf{y}$ for an input (e.g. consider Figure 7), which suggests modeling distributions $p_\theta(\mathbf{y}|\mathbf{x})$. We will assume that each object $\mathbf{y} \in \mathcal{Y}$ can be decomposed into tokens, $\mathbf{y} = \{y_1, \dots, y_T\}$, where $y_t \in \mathcal{V}$. Furthermore, we assume a **canonical ordering** σ that lets us treat each object as a sequence, $\mathbf{y} = (y_{\sigma(1)}, y_{\sigma(2)}, \dots, y_{\sigma(T)})$ where σ is a permutation of $\{1, \dots, T\}$. For example, when $\mathbf{y}$ is a multiset of digits in an image, we may order the digits in increasing order. When $\mathbf{y}$ is a tree, we may order its nodes using a depth-first traversal. When $\mathbf{y}$ is a sentence, we may order its words from left to right, meaning $\sigma(t) = t$.

4.1.1 Autoregressive models. The decomposition and ordering let us factorize $p_\theta(\mathbf{y}|\mathbf{x})$ into a product of conditional token distributions. An **autoregressive sequence model** factorizes $p_\theta(\mathbf{y}|\mathbf{x})$ as,

$$p_\theta(\mathbf{y}|\mathbf{x}) = \prod_{t=1}^{T} p_\theta(y_{\sigma(t)}|y_{<\sigma(t)}, \mathbf{x}), \tag{5}$$

where $y_{<\sigma(t)} = (y_{\sigma(1)}, \dots, y_{\sigma(t-1)})$, and $y_{<\sigma(1)} = \emptyset$. Each conditional is a distribution over the vocabulary $\mathcal{V}$, which is much smaller than the sequence space. The factorization in (5) does not introduce any conditional independence assumptions among the variables y_t, and hence can exactly match any sequence distribution in the limit of infinite parameters and data. This is unlike earlier models which assume that a variable is independent of variables sufficiently far in the past (Chen and Goodman 1996), or conditionally independent of all other variables given its neighbors (McCallum et al. 2000; Lafferty et al. 2001b). In this thesis we will write $p_\theta(y_t|y_{<t}, \mathbf{x})$ for brevity, unless the choice of σ is relevant for the discussion.

When the canonical order is left-to-right ($\sigma(t) = t$) or right-to-left ($\sigma(t) = T - t + 1$), we call the model **monotonic**. When the order σ is fixed for all $\mathbf{x}$ and $\mathbf{y}$, we say the model is **fixed-order**. When σ varies

based on $\mathbf{x}$ or $\mathbf{y}$, the model is **adaptive-order**. The conventional approach to neural text generation uses fixed-order, monotonic autoregressive models, which we adopt when we analyze text generation in Part II. Then we explore non-monotonic, adaptive-order models in Part III.

4.1.2 Recurrent parameterization. In this thesis, we will parameterize models p_θ using neural networks, so that θ is a vector of neural network parameters. The *recurrent neural network* is a natural choice for parameterizing autoregressive models (Equation 5), since it can theoretically encode arbitrarily long contexts. Specifically, a **recurrent neural network sequence model** $p_\theta(\mathbf{y}|\mathbf{x}) = \prod_{t=1}^{T} p_\theta(y_t|y_{<t}, \mathbf{x})$ computes the following conditional distribution at each time step,

$$p_\theta(y_t = v \,|\, y_{<t}, \mathbf{x}) = \frac{\exp(u_v^\top h_t + c_v)}{\sum_{v' \in \mathcal{V}} \exp(u_{v'}^\top h_t + c_{v'})}, \tag{6}$$

where $h_t = f_\theta(y_t, h_{t-1}) \in \mathbb{R}^d$ is the *hidden state*, with $h_0 = g_\theta(\mathbf{x}) \in \mathbb{R}^d$, and u, c, θ are parameters.

Practical variants of the recurrent neural network differ by the choice of transition function f_θ (Elman 1990; Hochreiter and Schmidhuber 1997; Cho et al. 2014). The function g_θ is called an *encoder*. In the *encoder-decoder framework* (Bahdanau et al. 2015), the encoder is a separate neural network.

4.1.3 Self-attention parameterization. An alternative is to compute the hidden states using a *feed-forward* network $f_\theta(y_1, \ldots, y_T, \mathbf{x}) \to (h_1, \ldots, h_T)$ structured as a series of L layers f_θ^ℓ. Instead of recurrence, meaning $h_t^\ell = f_\theta^\ell(h_{t-1}^\ell)$, the feed-forward network computes states at each layer that are a function of all states from the previous layer, $f_\theta^\ell(h_{1:T}^{\ell-1}) \to h_{1:T}^\ell$, where $h_{1:T}^\cdot$ means $(h_1^\cdot, \ldots, h_T^\cdot)$.

The **transformer sequence model** (Vaswani et al. 2017) implements each layer f_θ^ℓ using a mechanism called **self-attention**. Here, an *attention function* maps a query vector q_t, key vectors $k_{1:T}$, and value vectors $v_{1:T}$ to an output vector, $a(q_t, k_{1:T}, v_{1:T}) \to \tilde{h}_t$. The attention function produces a vector that is a weighted sum of the values $v_{1:T}$, with the weights determined by the similarity of the query with each key. When the same vectors act as query, key, and value, i.e. $a(h_t^{\ell-1}, h_{1:T}^{\ell-1}, h_{1:T}^{\ell-1}) \to \tilde{h}_t^\ell$, the function is called *self-attention*. Intuitively, implementing each layer f_θ^ℓ with self-attention lets the network produce token representations h_t that use information from the entire input sequence.

The self-attention function from Vaswani et al. (2017) applies multiple instances of self-attention, with each instance linearly transforming each key, query, and value. The dot product is used to compute the similarity of the query and keys, and the attention is applied for all timesteps (i.e. $a(\cdot)$ is applied with q_t for all $t \in \{1, \ldots, T\}$). This overall design is called **multihead scaled dot product attention** and

can be computed efficiently with matrix operations,

$$
\begin{aligned}
a_{\text{multi-head}}(Q, K, V) &= \left[a^{(1)}(Q, K, V); \ldots; a^{(H)}(Q, K, V)\right] W_O \\
a^{(i)}(Q, K, V) &= a(QW_Q^{(i)}, KW_K^{(i)}, VW_V^{(i)}) \\
a(Q, K, V) &= \text{softmax}\left(\frac{QK^\top}{\sqrt{\tilde{d}}}\right) V,
\end{aligned}
$$

where H is the number of self-attention instances, $[\cdot;\cdot]$ is concatenation, $\tilde{d} = \frac{d}{H}$, the matrices $Q, K, V \in \mathbb{R}^{T \times d}$ respectively contain stacked query, key, and value vectors, and $W_{\{Q,K,V\}}^{(i)} \in \mathbb{R}^{d \times \tilde{d}}$, $W_O \in \mathbb{R}^{H\tilde{d} \times d}$.

In summary, the transformer sequence model consists of L layers f_θ^ℓ, each containing multihead scaled dot product self-attention. Each output h_t can be used to parameterize a conditional distribution,

$$
p_\theta(y_t = v | \mathbf{y}_\mathcal{O}, \mathbf{x}) = \frac{\exp(u_v^\top h_t + c_v)}{\sum_{v' \in \mathcal{V}} \exp(u_{v'}^\top h_t + c_{v'})}. \tag{7}
$$

For an autoregressive model, $\mathbf{y}_\mathcal{O} = y_{<t}$, the self-attention inputs from future timesteps cannot be used, which is enforced efficiently in practice by setting $h_{t+1:T}^{\ell-1} = 0$ with an upper-triangular matrix multiplication (see Vaswani et al. (2017) for details). Vaswani et al. (2017) demonstrated that transformer sequence models outperformed recurrent models on machine translation benchmarks, and subsequently transformers parameterized state-of-the-art models in other NLP tasks, e.g. Devlin et al. (2018); Brown et al. (2020); Adiwardana et al. (2020); Raffel et al. (2020). We will use the transformer in several sections of this thesis.

4.2 Learning

With a model parameterization specified, the next step is find parameters that minimize an objective function. For autoregressive models, the conventional approach for this *learning* stage is to find parameters that maximize a likelihood function $\mathcal{L}(\theta; \mathcal{D})$ defined using a training dataset.

4.2.1 Maximum likelihood estimation. Given a fixed-order autoregressive sequence model and a dataset $\mathcal{D} = \{(\mathbf{x}^{(i)}, \mathbf{y}^{(i)})\}_{i=1}^N$, **maximum likelihood estimation (MLE)** is defined as,

$$
\begin{aligned}
\theta_{\text{MLE}} &= \arg\max_{\theta \in \Theta} \mathcal{L}(\theta) + \Omega(\theta) \\
&= \arg\max_{\theta \in \Theta} \sum_{n=1}^{N} \sum_{t=1}^{|\mathbf{y}^{(n)}|} \left[\log p_\theta(y_t^{(n)} | y_{<t}^{(n)}, \mathbf{x}^{(n)})\right] + \Omega(\theta),
\end{aligned} \tag{8}
$$

where $\Omega : \Theta \to \mathbb{R}$ is a regularization function. For notational brevity we will ignore the regularization function from here on. Neural networks are trained by minimizing a differentiable loss function; the per-example MLE loss corresponding to Equation 8 is,

$$\mathcal{L}_{\text{MLE}}(\theta, \mathbf{x}, \mathbf{y}) = \sum_{t=1}^{|\mathbf{y}|} \mathcal{L}_{\text{MLE}}^{t}(\theta, x, y_{\leq t}) \tag{9}$$

$$= -\sum_{t=1}^{|\mathbf{y}|} \log p_\theta(y_t | y_{<t}, \mathbf{x}), \tag{10}$$

where $\mathcal{L}_{\text{MLE}}^{t}$ is called the per-step MLE loss. The problem in Equation 8 is solved in practice by minimizing the loss $\mathcal{L}_{\text{MLE}}$ using (a variant of) stochastic gradient descent on batches sampled from $\mathcal{D}$.

4.2.1.1 Strengths. The maximum likelihood criterion in Equation 9 has become the standard approach to training neural autoregressive sequence models in a variety of applications, such as machine translation (Bahdanau et al. 2015), dialogue modeling (Vinyals et al. 2015), and language modeling (Radford et al. 2018). The strengths of MLE include (1) scalability, e.g. the transformer model (§4.1.3) can compute the required per-step losses in parallel; (2) self-supervision, i.e. only requiring a dataset of sequences for training; and (3) consistency in the limit of infinite data and model capacity. Since the required supervision is often widely available, e.g. in the form of web documents, owing to its scalability MLE has been used to train neural sequence models with billions of parameters on gigabytes of data (Radford et al. 2018; Adiwardana et al. 2020; Brown et al. 2020).

4.2.1.2 Drawbacks. Despite this success, maximum likelihood training has several potential drawbacks. MLE has a **task cost mismatch**, in that maximizing the likelihood of a dataset only acts as a surrogate to minimizing a task cost, such as BLEU in the case of machine translation. MLE only trains using ground-truth sequences $\mathbf{y}$, and in particular does not train with sequences generated by the model. This results in a **distribution mismatch** between the distribution of sequences used for training and those encountered by decoding from the model. Finally, autoregressive sequence models trained with MLE often have **miscalibrated sequence probabilities** in practice. For instance, the model may assign higher probability to a single-token sequence than to a longer correct sequence (Sountsov and Sarawagi 2016b). We return to these drawbacks when we study neural text degeneration (§II).

4.3 Inference

After training a sequence model, we would like to use it to generate sequences, for instance representing multisets, trees, or text. This generation process is called *inference* or *decoding*; we use either term interchangeably. The decoding process relies on a *decoding algorithm*.

Definition 1 (Decoding algorithm). *A decoding algorithm $\mathcal{F}(p_\theta, \mathbf{x}, \epsilon)$ is a function that generates a sequence $\hat{\mathbf{y}}$ given an autoregressive neural sequence model p_θ, a context $\mathbf{x}$, and noise $\epsilon \sim p_\epsilon$. We write $\hat{\mathbf{y}} \sim \mathcal{F}(p_\theta, \mathbf{x})$ to denote sampling from the stochastic function $\mathcal{F}$ with an implicit noise distribution p_ϵ.*

For example, given a trained translation model p_θ and a Japanese sentence $\mathbf{x}$, we use a decoding algorithm to obtain an English translation $\hat{\mathbf{y}} \sim \mathcal{F}(p_\theta, \mathbf{x})$. The decoding algorithm induces a distribution over sequences that can differ from the model's distribution.

Definition 2 (Induced sequence distribution). *A decoding algorithm $\mathcal{F}$, paired with a neural sequence model p_θ and a context distribution $p_{\mathbf{x}}$, induces a sequence distribution $q_{\mathcal{F}}(\mathbf{y}; p_\theta, p_{\mathbf{x}})$.*

To clarify this idea, let us first introduce two basic decoding algorithms. *Ancestral sampling* samples tokens from the model's next-token distribution at each step, while *greedy decoding* places all of its probability mass on the most-likely token at each step.

Definition 3 (Ancestral sampling). *Ancestral sampling $\mathcal{F}_{anc}$ generates a sequence by recursively sampling from $p_\theta(y_t \mid \hat{y}_{<t}, \mathbf{x})$ until $\hat{y}_t = \langle eos \rangle$:*

$$\hat{y}_t \sim p_\theta(y_t \mid \hat{y}_{<t}, \mathbf{x}).$$

Definition 4 (Greedy decoding). *Greedy decoding $\mathcal{F}_{greedy}$ generates a sequence by recursively selecting the most-likely token until $\hat{y}_t = \langle eos \rangle$:*

$$\hat{y}_t \sim q_t(y_t | \hat{y}_{<t}, \mathbf{x}, p_\theta), \tag{11}$$

$$\text{where } q_t = \begin{cases} 1 & y_t = \arg\max_{v \in \mathcal{V}} p_\theta(y_t = v \mid \hat{y}_{<t}, \mathbf{x}) \\ 0 & \text{otherwise.} \end{cases} \tag{12}$$

Ancestral sampling induces a sequence distribution $q_{\mathcal{F}_{anc}}$ that is identical to the model p_θ, since ancestral sampling directly samples from the model's conditional distributions. Greedy decoding, however, modifies the model's conditional distributions so that the resulting induced sequence distribution $q_{\mathcal{F}_{greedy}}$ concentrates its mass on a single sequence per input $\mathbf{x}$. Based on these two examples, we can see that the decoding algorithm influences the induced sequence distribution, which in turn influences which sequences are generated. We will study this phenomenon when we explore inconsistency (§6).

4.3.1 A classification of decoding algorithms.

After staring at Definition 4, one sees that many decoding algorithms can be devised by varying q_t. One can also view decoding algorithms as traversing a

	Exact	Deterministic	Complete	Reference
Enumeration	✓	✓	✗	-
Depth-first search	✓	✓	✗	[146]
Minimum bayes risk	✓	✓	✗	[76]
Greedy decoding	✗	✓	✗	Def. 4
Beam search	✗	✓	✗	Def. 10
Ancestral sampling	✗	✗	✓	Def. 3
Top-k sampling	✗	✗	✗	[37]
Nucleus sampling	✗	✗	✗	[63]
Entmax sampling	✗	✗	✗	[101]
Approx. min. bayes risk	✗	✗	✓	[34]
Consistent top-k	✗	✗	✓	§6
Consistent nucleus	✗	✗	✓	§6

Table 3: Decoding methods for autoregressive neural sequence models.

lattice of tokens across time in various ways, where a path represents a decoded sequence. For instance, greedy decoding selects a single path by choosing locally optimal tokens. With these two perspectives in mind, we now survey common decoding algorithms for autoregressive sequence generation, classified according to three properties, (1) *exact* vs. *approximate*, (2) *stochastic* vs. *deterministic*, and (3) *complete* vs. *incomplete*. Table 3 summarizes the discussion which follows.

4.3.1.1 Exact vs. approximate decoding. One use of a decoding algorithm is to find a single optimal generation for a given input. For instance, given a Japanese sentence $\mathbf{x}$, in many applications we only want a single, best English translation $\mathbf{y}$. The *argmax problem* defines optimality by the model's probability.

Definition 5 (Argmax problem). *The argmax problem consists of finding an output sequence* $\hat{\mathbf{y}}$ *with maximal conditional probability given a context* $\mathbf{x}$*,*

$$\hat{\mathbf{y}} = \arg\max_{\mathbf{y} \in \mathcal{Y}} p_\theta(\mathbf{y}|\mathbf{x}). \tag{13}$$

The optimization problem in Definition 5 involves maximizing over a set $\mathcal{Y}$ whose size grows exponentially in sequence length T (i.e. $|\mathcal{V}|^T$). An **exact decoding algorithm** solves the argmax problem exactly, which is only tractable when $|\mathcal{V}|$ and the maximum sequence length are small. In this case, one option is to **enumerate** each possible sequence, measure their probabilities, and choose one that has maximum probability. Another option is to perform a **depth-first search** on the tree of possible sequences while pruning suboptimal branches. For instance, Stahlberg and Byrne (2019b) prune branches using the monotonicity of sequence probabilities to perform exact inference with a neural translation model,

albeit with a large runtime. Instead of solving the argmax problem (Definition 5), **minimum bayes risk** decoding (Kumar and Byrne 2004) finds the sequence that minimizes a task cost (e.g. BLEU) in expectation,

$$\arg\min_{\hat{\mathbf{y}} \in \mathcal{Y}} \mathbb{E}_{\mathbf{y} \sim p_\theta(\mathbf{y}|\mathbf{x})} \left[\ell(\mathbf{y}, \hat{\mathbf{y}}) \right]. \tag{14}$$

As with the argmax problem, solving this minimum bayes risk problem (Equation 14) is intractable in almost all practical applications, which have a large vocabulary and possibly unbounded sequence length. Thus it is necessary to use an **approximate decoding algorithm**. The remaining algorithms in Table 3 are approximate, and in this thesis we will only use approximate decoding algorithms in experiments.

4.3.1.2 Stochastic vs. deterministic decoding. A second dimension along which we classify decoding algorithms is whether the algorithm uses randomness. Ancestral sampling (Definition 3) uses randomness by directly sampling from the model's conditional distributions. One can then approximate the arg max problem using Monte-Carlo sampling,

$$\arg\max_{\mathbf{y} \in \mathcal{Y}} p_\theta(\mathbf{y}|\mathbf{x}) \approx \arg\max_{\mathbf{y}^{(1)}, \ldots, \mathbf{y}^{(M)}} p_\theta(\mathbf{y}^{(i)}|\mathbf{x}),$$

where $\mathbf{y}^{(m)} \sim \mathcal{F}_{anc}(p_\theta, \mathbf{x})$. In a similar manner, Eikema and Aziz (2020) approximately minimize the bayes risk (Equation 14).

A family of ancestral sampling variants sample from a proposal distribution $q_t(y_t|y_{<t}, \mathbf{x}, \theta)$ at each decoding step. When the model's conditional distribution is defined using the softmax[3] operation, q_t can be formed by rescaling $p_\theta(y_t|y_{<t}, \mathbf{x})$ using a *temperature*.

Definition 6 (Softmax Temperature Scaling (Hinton et al. 2015)). *Softmax scaling with temperature $\tau \in (0, \infty)$ recursively samples from:*

$$q_t(y_t|y_{<t}, \mathbf{x}, f_\theta) = \frac{\exp\left(f_\theta(y_t, y_{<t}, \mathbf{x})/\tau\right)}{\sum_{y'_t \in \mathcal{V}} \exp\left(f_\theta(y'_t, y_{<t}, \mathbf{x})/\tau\right)}.$$

As $\tau \to \infty$ the distribution becomes uniform, and as $\tau \to 0$, the distribution's mode receives all of the probability mass while all other tokens receive zero mass, which is equivalent to greedy decoding. Only placing positive probability on the mode can be generalized to the top-k most probable tokens.

[3] $p_\theta(y_t|y_{<t}, \mathbf{x}) = \frac{\exp f_\theta(y_t, y_{<t}, \mathbf{x})}{\sum_{y'_t \in \mathcal{V}} \exp f_\theta(y'_t, y_{<t}, \mathbf{x})}$.

Definition 7 (Top-k sampling (Fan et al. 2018)). *Top-k sampling with $k \in \mathbb{Z}, 1 \leq k \leq |\mathcal{V}|$, recursively samples from:*

$$q_t(y_t|y_{<t}, \mathbf{x}, p_\theta) \propto \begin{cases} p_\theta(y_t \mid y_{<t}, X), & \text{if } y_t \in \arg\operatorname*{top\text{-}k}_y p_\theta(y \mid y_{<t}, \mathbf{x}), \\ 0, & \text{otherwise.} \end{cases}$$

Alternatively, q_t can be defined so that mass is given only to the minimal subset of tokens whose total probability is higher than a budget μ.

Definition 8 (Nucleus sampling (Holtzman et al. 2020)). *Nucleus sampling with budget $\mu \in [0, 1)$ recursively samples from:*

$$q(v) \propto \begin{cases} p_\theta(v \mid y_{<t}, C), & \text{if } v \in \{v_1, \ldots, v_k\}, \\ 0, & \text{otherwise,} \end{cases}$$

where k is the least index of the vocabulary sorted by decreasing probability[4] *with $\sum_{i=1}^{k} p_\theta(v_i|y_{<t}, \mathbf{x}) > \mu$.*

Top-k and nucleus sampling are often used in NLP applications to generate text that tends to be subjectively better than ancestral sampling, by way of discarding low probability tokens.

In general, the sampling algorithms above produce different sequences each time they are run. In contrast, deterministic decoding algorithms such as greedy decoding map a neural sequence model and an input to a single output sequence. A deterministic decoding algorithm induces a conditional distribution with mass on a single sequence per input.

$$q_{\mathcal{F}}(\mathbf{y}|\mathbf{x}, p_\theta) = \begin{cases} 1 & \mathbf{y} = \mathcal{F}(p_\theta, \mathbf{x}) \\ 0 & \text{otherwise.} \end{cases}$$

The induced marginal probability of $\mathbf{y}$ is the total probability of inputs that map to $\mathbf{y}$,

$$q_{\mathcal{F}}(\mathbf{y}|p_\theta) = \sum_{\mathbf{x}:\mathbf{y}=\mathcal{F}(p_\theta,\mathbf{x})} p_{\mathbf{x}}(\mathbf{x}).$$

For instance, the induced probability of a translation $\mathbf{y}$ is the total probability of drawing any source sentence $\mathbf{x}$ from the dataset that translates to $\mathbf{y}$ using the model and decoding algorithm.

[4] $p_\theta(v_i \mid y_{<t}, X) \geq p_\theta(v_j \mid y_{<t}, X)$ for all $i < j$.

Above, we saw greedy decoding (Definition 4), which is a deterministic decoding algorithm. A generalization called **beam search** operates on the level of partial sequences or prefixes.

Definition 9 (Prefix). *A prefix ρ_t is an ordered collection of items from $\mathcal{V}$. The score of a prefix is*

$$s(\rho_t) = \sum_{\tau=1}^{t} \log p_\theta(y_\tau = \rho_t[\tau] \,|\, \rho_t[< \tau], X),$$

where $\rho_t[\tau]$ is a token at time τ from ρ_t.

Beam search starts from a set of empty prefixes, and forms a new prefix set at each iteration by expanding each prefix then choosing the highest scoring expanded prefixes.

Definition 10 (Beam search). *Beam search with width k, $\mathcal{F}_{beam-k}$, generates a sequence from a recurrent language model p_θ by maintaining a size-k prefix set P_t^{top}. Starting with $P_0^{top} = \varnothing$, at each iteration $t \in \{1, 2, \ldots\}$ beam search forms a new prefix set P_t^{top} by expanding the current set, $\mathrm{P}_t = \bigcup_{\rho \in \mathrm{P}_{t-1}^{top}} \{\rho \circ v \,|\, v \in V\}$ (where $\rho \circ v$ is concatenation), then choosing the k highest scoring elements,*

$$\mathrm{P}_t^{top} = \arg \underset{\rho \in \mathrm{P}_t}{top\text{-}k}\ s(\rho).$$

Any $\rho \in \mathrm{P}_t^{top}$ ending with $\langle eos \rangle$ is restricted from being expanded further, and is added to a set S. Beam search ends when S contains k sequences, and returns the highest scoring sequence in S.

Beam search with a width of $k = 1$ yields greedy decoding, and beam search with $k = \infty$ is an exact decoding algorithm. In practice, the beam width k is finite and selected as a hyperparameter. Increasing the beam width can lead to performance degradation (Koehn and Knowles 2017; Ott et al. 2018), since the model can assign high probability to the empty sequence (Stahlberg and Byrne 2019b). Murray and Chiang (2018) demonstrated that this is a consequence of the model's local normalization (§4.1.1), and can be alleviated with an approximated globally-normalized sequence score.

4.3.1.3 Complete vs. incomplete decoding. Several of the decoding algorithms above are *incomplete* in that they only consider a strict subset of the the full vocabulary $\mathcal{V}$ at each time step.

Definition 11 (Incomplete Decoding). *A decoding algorithm $\mathcal{F}$ is incomplete when for each context $\mathbf{x}$ and prefix $y_{<t}$, there is a strict subset $V_t' \subsetneq \mathcal{V}$ such that*

$$\sum_{v \in V_t'} q_{\mathcal{F}}(y_t = v \,|\, y_{<t}, \mathbf{x}, p_\theta) = 1.$$

Greedy search, beam search, softmax temperature scaling in the limit of $\tau \to \infty$, top-k sampling, and nucleus sampling are incomplete decoding algorithms.[5] Incompleteness will be important later to our investigation of a sequence model's *consistency* (§6), where we will show that modifying an incomplete algorithm so that it is complete prevents the algorithm from decoding an infinite-length sequence.

4.4 Summary

This part of the thesis contained an overview of applications, learning, models, and inference methods for sequential neural structured prediction. In this thesis we generate multisets for multiple object classification, trees for dependency parsing, and sequences for dialogue modeling, machine translation, and text completion. The predominant approach is to model conditional probabilities with an autoregressive function parameterized by a transformer or recurrent neural network. The standard training method is to sequentialize each structured object with a canonical ordering, then maximize the likelihood of the resulting sequences. After training, a decoding algorithm such as nucleus sampling or greedy search is used to generate sequences. In the next part of the thesis (§II), we will investigate abnormalities that arise when using this standard approach in text generation, leading to new approaches.

[5] Nucleus sampling is incomplete when for every context $\mathbf{x}$ and prefix $y_{<t}$, $\min_{v \in \mathcal{V}} p_\theta(v|y_{<t}, \mathbf{x}) < 1 - \mu$.

Part II

Neural Text Generation

5 Text Degeneration

In this part of the thesis, we will focus on generating sequences, specifically text. As we saw in Part I, the standard approach to sequence generation consists of training a fixed-order neural autoregressive model with maximum likelihood estimation (MLE), then using a decoding algorithm such as ancestral sampling or beam search to generate sequences. This approach has led to strong performance in a variety of natural language processing applications such as machine translation (Bahdanau et al. 2015), summarization (Rush et al. 2015), dialogue modeling (Vinyals et al. 2015), and text completion (Sutskever et al. 2011; Graves 2013; Radford et al. 2018; Brown et al. 2020). Despite this success, MLE-trained models are known to have flaws which are surfaced in generated text. We will characterize these flaws as *non-termination*, *repetition*, and *logical coherence*, collectively termed *text degeneration*.

5.1 Non-termination

Sequences that are used to train neural sequence models end with a special ⟨eos⟩ token that denotes the end of the sequence. Decoding proceeds until ⟨eos⟩ is generated, which leaves open the possibility of a *non-terminating sequence* when the decoding algorithm never generates ⟨eos⟩.

Definition 12 (Non-terminating sequence). *A sequence* $\mathbf{y} = (y_1, y_2, \ldots)$ *is non-terminating if* $y_t \neq \langle eos \rangle$ *for all* $t \in \{1, \ldots, \infty\}$.

In practice, we must halt the decoding algorithm after a finite number of steps. As a proxy for measuring true non-termination, we measure the proportion of decoded sequences that have not terminated after a predefined number of steps L, called the *non-termination rate*:

$$\mathbf{nonterm}_L(\mathcal{F}, p_\theta, \mathcal{D}) = \frac{1}{|\mathcal{D}|} \sum_{n=1}^{|\mathcal{D}|} \mathbb{1}(|\hat{\mathbf{y}}^{(n)}| \geq L), \tag{15}$$

where $\hat{\mathbf{y}}^{(n)} \sim \mathcal{F}(p_\theta, \mathbf{x}^{(n)})$ for each input $\mathbf{x}^{(n)}$ in dataset $\mathcal{D}$. A non-termination rate that is greater than zero means that some sequences did not terminate within L steps, which we interpret as evidence that those sequences are non-terminating. Although sequences in the training set always terminate, neural sequence models frequently generate non-terminating sequences when paired with incomplete decoding algorithms such as greedy decoding. Table 4 shows examples.

5.2 Repetition

Generated text in open-ended applications such as text completion or dialogue generation has been observed to contain excessive repetition compared to human-generated text (represented by a training corpus), especially with decoding methods that approximately solve the argmax problem (Def. 5).

We distinguish between repetition that occurs in a model-generated sequence and repetition that occurs when making next-token predictions given a correct history. For the first case, called *sequence-level repetition*, one can get a sense of the issue by observing the samples in Table 4, which shows completions from the large-scale MLE-trained GPT-2 language model (Radford et al. 2018) using prefixes from the Wikitext-103 corpus (Merity et al. 2016). Greedy decoding as well as top-k and nucleus sampling exhibit degenerate repetition (at certain hyper-parameter settings), although greedy decoding shows the worst degradation. Repetition in a sequence $\mathbf{y}$ is quantified by the proportion of duplicate n-grams:

$$\mathbf{rep}_n(\mathbf{y}) = 1.0 - \frac{|\text{unique n-grams}(\mathbf{y})|}{|\text{n-grams}(\mathbf{y})|}. \tag{16}$$

This metric, called the *repetition rate*, allows us to measure repetition similar to that of Table 4 occurs frequently across a variety of prefixes. Indeed, across all prefixes from the Wikitext-103 validation set, GPT-2 continuations with greedy decoding have an average repetition rate of roughly 50%, which far exceeds that of the ground-truth text (0.5%), even after fine-tuning the model (46%).

Neural language models also frequently predict an incorrect and repeating next-token given a ground-truth token history,

$$\Pr\left(\hat{y}_t = \arg\max p_\theta(y|y_{<t}) \in y_{<t}\right) > \Pr\left(y_t \in y_{<t}\right), \tag{17}$$

where $y_{<t}$ and y_t are from a ground-truth continuation. For instance, GPT-2 predicted next-tokens that appeared in the preceding 128 words roughly 60% of the time, versus 40% in the ground-truth data. This *token-level repetition* is concerning since maximum-likelihood focuses on optimizing next-token conditional distributions.

5.3 Logical Incoherence

A long standing issue faced by neural text generation models is *logical coherence* (Vinyals et al. 2015; Li et al. 2016; Zhang et al. 2018). In dialogue modeling, an example from Vinyals et al. (2015) shows a two-round dialogue in which their neural sequence model first responds to *what is your job?* with *i'm a lawyer*, then responds to *what do you do?* with *i'm a doctor*. As seen in Figures 9 and 10, logical contradictions

Prefix	*Lyrically the song has excerpts of different languages including French , Spanish ,*
GPT2-greedy	German , Italian , Portuguese , Spanish , Portuguese , Portuguese , Portuguese , Portuguese , Portuguese , Portuguese , Portuguese , Portuguese , Portuguese , Portuguese , Portuguese , ...
GPT2-top3	German , Italian , Portuguese , Spanish , Portuguese , Portuguese, Italian and Spanish. It is also available in English, French and Spanish. In addition to its lyrics, the album features a number of original songs, ...
GPT2-nucleus-0.3	German , Italian , Portuguese , Spanish , Portuguese , Italian , Portuguese , Spanish , Portuguese , Spanish , Portuguese , Spanish , Portuguese , Spanish , Portuguese , Spanish , Portuguese , Spanish , Portuguese , Spanish , Portuguese , ...
Prefix	*The first pair of pereiopods is armed with a large , asymm*
GPT2-greedy	etrical, and long-range laser cannon. The second pair is armed with a large , asymmetrical, and long-range laser cannon. The third pair is armed with a large , asymmetrical, and long-range laser cannon. The fourth pair is armed with a large ...
GPT2-top3	etrical, and highly mobile head, and the second pair has a small , asymmetrical, and highly mobile head. The second pair has a large and highly mobile head, and the third pair is armed with a large and highly mobile head. The first pair ...
GPT2-nucleus-0.3	etrical head and a large body. The first pair of pereiopods is armed with a large , asymmetrical head and a large body. The first pair of pereiopods is armed with a large , asymmetrical head and a large body. The first pair of pereiopods is armed ...

Table 4: Non-termination and repetition in completions from GPT-2, a large-scale language model trained with MLE (Radford et al. 2018). The examples contain single-word repetitions, phrase-level repetitions, and structural repetitions where some tokens within a repeating phrase vary. Recently proposed stochastic samplers (top-k, nucleus) exhibit degeneration based on hyper-parameter settings.

still occur in recent dialogue models and large-scale language models. Even when inconsistencies are relatively rare and semantically plausible, they are jarring, and because semantic plausibility is not enough to root them out, preventing them is challenging.

Logical incoherence is difficult to measure and define in general. As a first step, we will construct a dataset that allows us to measure simple forms of logical incoherence in a specific dialogue setting (§8).

5.4 Other

Maximum-likelihood trained models have been shown to generate text with a *mismatched unigram distribution* compared to human text. That is, for each token y in a nontrivial subset $V \subseteq \mathcal{V}$,

$$|p_\theta(y) - p_*(y)| > \delta, \tag{18}$$

where δ allows for small differences. Given a dataset $\mathcal{D} = \{\mathbf{x}^{(i)}, \mathbf{y}^{(i)}\}_{i=1}^N$ and generations $\{\hat{\mathbf{y}}^{(i)}\}$, the unigram distributions are approximated as (notice $\hat{\mathbf{y}}^{(i)}$ versus $\mathbf{y}^{(i)}$),

$$p_\theta(y) \approx \frac{\sum_{i=1}^N \sum_{t=1}^{|\hat{\mathbf{y}}^{(i)}|} \mathbb{I}\left[\hat{y}_t^{(i)} = y\right]}{\sum_{i=1}^N |\hat{\mathbf{y}}^{(i)}|}, \qquad p_*(y) \approx \frac{\sum_{i=1}^N \sum_{t=1}^{|\mathbf{y}^{(i)}|} \mathbb{I}\left[y_t^{(i)} = y\right]}{\sum_{i=1}^N |\mathbf{y}^{(i)}|}. \tag{19}$$

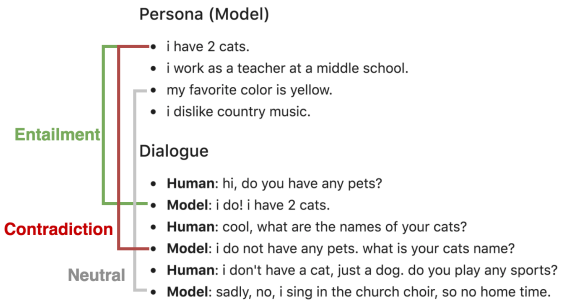


Fig. 9: Persona-based dialogue with a key-value memory network (Zhang et al. 2018)

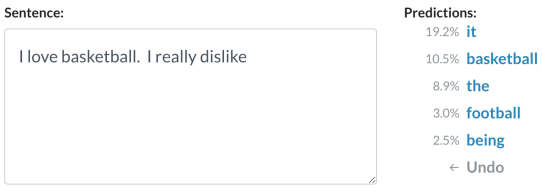


Fig. 10: Next-token probabilities from GPT-2. Obtained with `https://demo.allennlp.org/next-token-lm`.

In dialogue, models tend to overproduce common tokens (e.g. *the*, *a*, *of*) and underproduce rare tokens compared to the ground-truth distribution For example, this was observed across all generative models submitted to the ConvAI2 NeurIPS 2018 competition (Dinan et al. 2019a). In language modeling, the work of Holtzman et al. (2020) highlighted problems with the unigram distribution and level of repetition in model generations compared to human text. We will show that altering the learning algorithm can address unigram distribution mismatch (§7).

Several studies have investigated other drawbacks of maximum-likelihood training, including *label bias* (Lafferty et al. 2001a; Andor et al. 2016), *exposure bias* (Daumé III et al. 2009; Ross et al. 2011; Bengio et al. 2015), and *loss mismatch* (Lee et al. 2020). Neural machine translation models trained with maximum likelihood have been shown to exhibit decreased performance with increased beam size (Koehn and Knowles 2017; Ott et al. 2018) and a *short-sequence bias* (Sountsov and Sarawagi 2016a; Stahlberg and Byrne 2019a), which have been attributed to label bias due to local normalization (Murray and Chiang 2018). Investigating possible connections between these issues and the text degeneration issues considered in this thesis is interesting future work.

5.5 Summary

In this section, we introduced *text degeneration* in terms of *non-termination*, *repetition*, and *logical incoherence*. In the subsequent sections we will develop theory, data, and algorithms to study and address these forms of degeneration.

6 Theory: Inconsistency

In this section we investigate the issue of *non-termination* (§5.1) in neural text generation. To do so, we formalize and study the discrepancy between a learned model's distribution and the distribution induced by a decoding algorithm, in terms of a notion called *consistency*.

In Part I, we defined the notion of a *decoding algorithm* (Definition 1). We saw that decoding algorithms are used to draw samples (Definition 3) or to approximate the argmax problem (Definition 5). In this section we show that the distribution induced by a decoding algorithm can contradict these intended uses, instead resulting in non-terminating (i.e. infinite-length) sequences that are assigned zero probability by the underlying model. We will use the *recurrent language model* (Equation 6) in our investigation, but we will see that in practice our findings also transfer to the transformer language model (Equation 7).

Our main finding is that a sequence which receives zero probability under a recurrent language model's distribution can receive nonzero probability under the distribution induced by a decoding algorithm. This occurs when the recurrent language model always ranks the sequence termination token outside of the set of tokens considered at each decoding step, yielding an infinite-length, zero probability sequence. This holds whenever the decoding algorithm is *incomplete* (Definition 11), in the sense that the algorithm excludes tokens from consideration at each step of decoding, which is the case for common methods such as greedy search, beam search, top-k sampling, and nucleus sampling (recall §4.3). We formalize our main finding using the notion of *consistency* (Chen et al. 2017b) – whether a distribution assigns probability mass only to finite sequences – and prove that a consistent recurrent language model paired with an incomplete decoding algorithm can induce an inconsistent sequence distribution.

Based on the insight that inconsistency occurs due to the behavior of the termination token under incomplete decoding, we develop two methods for addressing inconsistency. First, we propose *consistent sampling methods* which guarantee that the termination token is not excluded from selection during decoding. Second, we introduce a *self-terminating language model* which ensures that the termination token is eventually top-ranked, guaranteeing consistency under incomplete decoding.

To empirically measure inconsistency, we decode sequences from trained recurrent language models and measure the proportion of sequences with lengths far exceeding the maximum training sequence length. Our experiments on the Wikitext2 dataset (Merity et al. 2016) suggest that inconsistency occurs in practice when using incomplete decoding methods, while the proposed consistent sampling methods and self-terminating model parameterization prevent inconsistency and maintain language modeling quality. Moreover, we empirically demonstrate inconsistency using the GPT-2 language model paired with greedy

decoding, suggesting that our analysis of recurrent models has implications for large-scale transformer models that are widely used in practice.

The theoretical analysis reveals defects of existing decoding algorithms, providing a way to develop new methods. Here, we present methods related to sampling and model parameterization. A later investigation in Welleck and Cho (2020) addresses inconsistency through learning.

6.1 Method

Our investigation of non-terminating sequences uses *consistency*, which is a property of a sequence distribution. A sequence distribution is consistent if it assigns probability mass only to finite sequences.

Definition 13 (Consistency of a sequence distribution). *A sequence distribution $p(\mathbf{y})$ is consistent under a context distribution $p(\mathbf{x})$ if $p(|\mathbf{y}| = \infty) = 0$. Otherwise, the distribution is said to be inconsistent.*

We can use Definition 13 to study the consistency of a *model* $p_\theta(\mathbf{y})$, or study the consistency of a *decoding algorithm*'s induced distribution $q_{\mathcal{F}}(\mathbf{y}; p_\theta)$ (Definition 2). One useful property is that for a given context, any sequence sampled from a consistent distribution is guaranteed to terminate.

Lemma 1. *If a sequence distribution p is consistent, $p(|\mathbf{y}| = \infty \,|\, \mathbf{x}) = 0$ for any probable context $\mathbf{x}$.*[6]

Figure 11 shows four scenarios involving the consistency of p_θ and $q_{\mathcal{F}}$. Our goal is to understand which of these occurs when we see a non-terminating generation in practice. First, we show that under practical conditions, the model p_θ is consistent, which rules out cases (b) and (c).

Lemma 2. *A recurrent LM p_θ is consistent if $\|h_t\|_p$ is uniformly bounded for some $p \geq 1$.*

Proof (Proof sketch). If $\|h_t\|_p$ is bounded, then each $u_v^\top h_t$ is bounded, hence $p_\theta(\langle \text{eos} \rangle \,|y_{<t}, \mathbf{x}) > \xi > 0$ for a constant ξ. Thus $p_\theta(|\mathbf{y}| = \infty) \leq \lim_{t\to\infty}(1-\xi)^t = 0$, meaning that p_θ is consistent.

This condition is practical because layer normalization or bounded activation functions (Elman 1990; Cho et al. 2014; Vaswani et al. 2017) result in bounded h_t. Lemma 2 tells us that either case (a) or case (d) from Figure 11 must describe the situation that underlies non-terminating sequences observed in practice. Our next theorem tells us that non-terminating sequences in practice correspond to case (d). Specifically, we prove that any incomplete decoding algorithm (Definition 11) can induce an inconsistent distribution, because there is a recurrent language model that places $\langle \text{eos} \rangle$ outside of the decoding algorithm's set of probable tokens (V_t') at every step of decoding.

[6] Proofs of Lemmas 1-2 are in Appendix A.1.1.

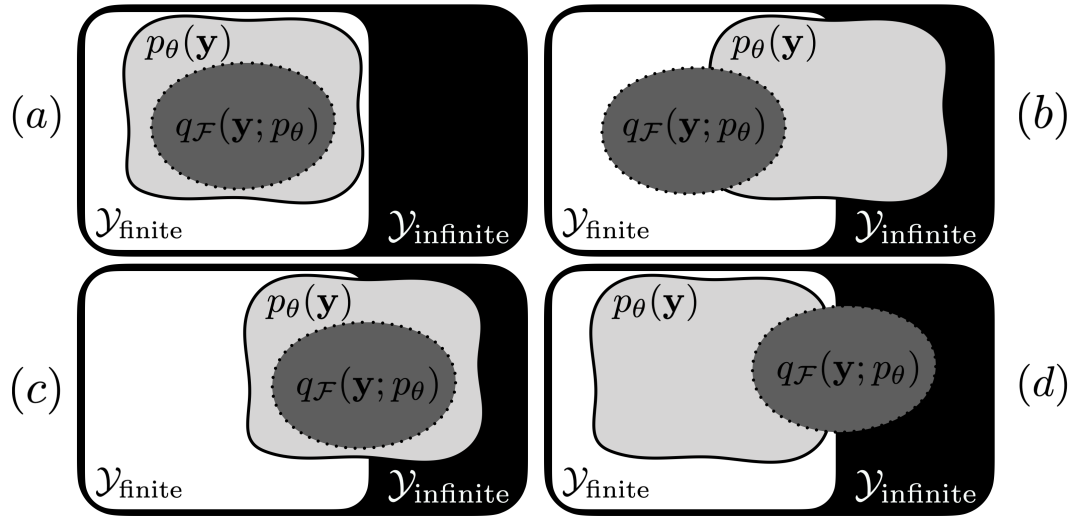


Fig. 11: Four possibilities involving consistency of the model's sequence distribution (light grey, solid border) and the decoder's induced sequence distribution (dark grey, dotted border). The white and black rectangles depict the set of all finite and infinite sequences, respectively. In (a) the model and the induced distribution are consistent. In (b) only the model is inconsistent, while in (c) both distributions are inconsistent. Our analysis shows that non-terminating generations in practice correspond to case (d).

Theorem 1 (Inconsistency of an incomplete decoding algorithm). *There exists a consistent recurrent language model p_θ from which an incomplete decoding algorithm $\mathcal{F}$, that considers only up to $(|V|-1)$-most likely tokens according to $p_\theta(y_t \mid y_{<t}, \mathbf{x})$ at each step t, finds an infinite-length sequence $\tilde{\mathbf{y}}$ with probability 1, i.e., $q_\mathcal{F}(|\mathbf{y}| = \infty) = 1$.*

Proof. We prove this theorem by constructing a tanh RNN. We define the recurrent function f_θ as

$$\begin{aligned} h_t &= f_\theta(y_t, h_{t-1}) \\ &= \tanh\left(\left[\begin{array}{c|c} W_h & \mathbf{0} \\ \hline \mathbf{0} & I \end{array}\right] h_{t-1} + \left[\begin{array}{c} \mathbf{0} \\ \hline e(y_t) \end{array}\right]\right), \end{aligned}$$

where $e(y_t) \in \mathbb{R}^{|V|}$ is a one-hot representation of y_t, $W_h \in \mathbb{R}^{d \times d}$ where every entry is positive, and I is an identity matrix of size $|V| \times |V|$. $h_0 = g_\theta(\mathbf{x})$ is constructed to consist of positive values only. Because each element of $|h_t|$ is bounded by 1, the constructed recurrent language model p_θ is consistent by Lemma 2.

We set u_v (see Equation 6) to

$$u_v = \left[\begin{array}{c} \bar{u}_v \\ \hline e(v) \end{array}\right], \quad u_{\langle \text{eos} \rangle} = \left[\begin{array}{c} \bar{u}_{\langle \text{eos} \rangle} \\ \hline e(\langle \text{eos} \rangle) \end{array}\right],$$

where $v \neq \langle\text{eos}\rangle$, all elements of $\bar{u}_v$ are positive, all elements of $\bar{u}_{\langle\text{eos}\rangle}$ are negative, and $e(v)$ is a one-hot representation of v. c_v is set to zero.

This defines a valid recurrent language model (Equation 6), since the conditional distribution at each time t is influenced by all the previous tokens. More specifically, the logit of a token v depends on $\sum_{t'=1}^{t} \mathbb{1}(y_{t'} = v)$, where $\mathbb{1}$ is an indicator function.

This recurrent language model always outputs positive logits for non-$\langle\text{eos}\rangle$ tokens, and outputs negative logits for the $\langle\text{eos}\rangle$ token. This implies $p(\langle\text{eos}\rangle \mid y_{<t}, \mathbf{x}) < p(v \mid y_{<t}, \mathbf{x})$ for all $v \in V \setminus \{\langle\text{eos}\rangle\}$. This means that $\langle\text{eos}\rangle$ is always ranked last at each time step, so an incomplete decoding algorithm that considers at most $(|V| - 1)$ most probable tokens at each time step from $p_\theta(y_t \mid y_{<t}, \mathbf{x})$ cannot decode $\langle\text{eos}\rangle$ and thus always decodes an infinitely long sequence $\hat{\mathbf{y}}$, i.e., $q_{\mathcal{F}}(|\mathbf{y}| = \infty \mid \mathbf{x}) = 1$ for any context $\mathbf{x}$. It yields $q_{\mathcal{F}}(|\mathbf{y}| = \infty) = 1$, while $p_\theta(|\mathbf{y}| = \infty) = 0$ due to consistency of the model p_θ. □

By this theorem, greedy decoding, beam search, top-k, and nucleus sampling are inconsistent.

6.1.1 Fixing the inconsistency. Our analysis shows that inconsistency in practice is depicted by case (d) in Figure 11. To prevent inconsistency, we must ensure that $q_{\mathcal{F}}$ places all of its mass on finite sequences. This can be done by modifying the decoding algorithm $\mathcal{F}$ or by modifying the model p_θ in a way that guarantees that $q_{\mathcal{F}}(\mathbf{y}; p_\theta)$ in consistent. We now propose a decoding algorithm and a model that provably do so. Subsequent work (Welleck and Cho 2020) suggests that changing the *learning algorithm* can yield a model p_θ with an *empirically* consistent distribution $q_{\mathcal{F}}$ under greedy decoding.

Consistent sampling. The proof of Theorem 1 suggests that inconsistency of incomplete decoding algorithms arises from the fact that $\langle\text{eos}\rangle$ may be excluded indefinitely from the set of top-ranked tokens. We propose a simple modification to top-k (Definition 7) and nucleus sampling (Definition 8) that forces $\langle\text{eos}\rangle$ to be included at each step of decoding. First, we give a condition for when a particular model p_θ paired with a decoding algorithm $\mathcal{F}$ is consistent.

Theorem 2. *Suppose a recurrent LM p_θ has uniformly bounded $\|h_t\|_p$ for some $p \geq 1$. If a decoding algorithm $\mathcal{F}$ satisfies $q_{\mathcal{F}}(\langle eos\rangle \mid y_{<t}, \mathbf{x}) \geq p_\theta(\langle eos\rangle \mid y_{<t}, \mathbf{x})$ for every prefix $y_{<t}$ and context $\mathbf{x}$, then the decoding algorithm $\mathcal{F}$ is consistent with respect to the model p_θ.*[7]

We define consistent variants of top-k and nucleus sampling which satisfy this condition.

[7] See Appendix A.1.1 for the proof.

Definition 14 (Consistent top-k sampling). *Consistent top-k sampling is top-k sampling with the following modified proposal distribution:*

$$q(v) \propto \begin{cases} p_\theta(v|y_{<t}, \mathbf{x}), & \text{if } v \in V', \\ 0, & \text{otherwise,} \end{cases}$$

where $V' = \{\langle eos \rangle\} \cup \arg\operatorname*{top\text{-}k}_{v'} p_\theta(v' \mid y_{<t}, \mathbf{x})$.

Definition 15 (Consistent nucleus sampling). *Consistent nucleus sampling is nucleus sampling with the following modified proposal distribution:*

$$q(v) \propto \begin{cases} p_\theta(v \mid y_{<t}, \mathbf{x}), & \text{if } v \in V_\mu \cup \{\langle eos \rangle\}, \\ 0, & \text{otherwise.} \end{cases}$$

The induced probability of $\langle \text{eos} \rangle$ under these two algorithms is always equal to or larger than the model's probability. By Theorem 2, they are consistent with respect to any consistent recurrent LM.

Consistent model. Although these consistent sampling algorithms can be used with any recurrent language model, their stochastic nature may not be suitable for finding a single, highly probable sequence (see Definition 5). To avoid this limitation, we will modify p_θ to guarantee that $q_{\mathcal{F}}$ is consistent, even when $\mathcal{F}$ is an incomplete decoding algorithm. To do so, we propose the *self-terminating recurrent language model* (STRLM) which guarantees that the probability of $\langle \text{eos} \rangle$ monotonically increases over time.

Definition 16 (Self-terminating recurrent language model). *A self-terminating recurrent language model computes the following conditional probability at each time step:*

$$p_\theta(v \mid y_{<t}, \mathbf{x}) = \begin{cases} 1 - \alpha(h_t), & v = \langle eos \rangle, \\ \frac{\alpha(h_t) \exp(u_v^\top h_t + c_v)}{\sum_{v' \in V'} \exp(u_{v'}^\top h_t + c_{v'})}, \end{cases} \tag{20}$$

$$\alpha(h_0) = \sigma(u_{\langle eos \rangle}^\top h_0), \tag{21}$$

$$\alpha(h_t) = \sigma(u_{\langle eos \rangle}^\top h_t) \left[1 - p_\theta(\langle eos \rangle \mid y_{<t-1}, \mathbf{x})\right], \tag{22}$$

with $\sigma : \mathbb{R} \to [0, 1 - \varepsilon]$ *and* $\varepsilon \in (0, 1)$. h_t *is computed as in the original recurrent LM.*

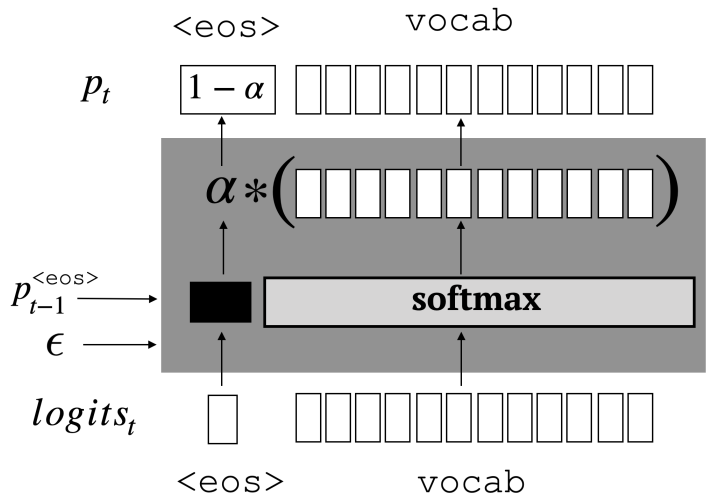


Fig. 12: The self-terminating recurrent language model uses the layer shown in grey instead of the standard softmax layer. The layer takes the logits ($u_{\cdot}^{\top} h_t$), the previous step's ⟨eos⟩ probability ($p_{t-1}^{\langle \text{eos} \rangle}$), and a hyper-parameter $\epsilon \in (0, 1)$. The layer computes α using Equation 22, which determines $p_t^{\langle \text{eos} \rangle} \in (\epsilon, 1)$ and guarantees that $p_t^{\langle \text{eos} \rangle} > p_{t-1}^{\langle \text{eos} \rangle}$. The remaining probability mass is allocated to the non-⟨eos⟩tokens.

The underlying idea is that the probability of ⟨eos⟩ increases monotonically, since

$$p_t^{\langle \text{eos} \rangle} = 1 - \prod_{t'=0}^{t} \sigma(u_{\langle \text{eos} \rangle}^{\top} h_{t'}).$$

Consequently, the STRLM is consistent when paired with greedy decoding or beam search; see Appendix A.1.1 for formal statements and proofs. The self-terminating recurrent language model can be understood as replacing the standard softmax layer with a self-terminating variant, depicted in Figure 12, which guarantees $p_t^{\langle \text{eos} \rangle} \in (\epsilon, 1)$ and $p_t^{\langle \text{eos} \rangle} > p_{t-1}^{\langle \text{eos} \rangle}$.

6.2 Empirical Evaluation

The theoretical results rely on the existence of a model that results in inconsistency; it remains to be shown that inconsistency with respect to incomplete decoding occurs with recurrent language models encountered in practice. Moreover, while the proposed methods carry theoretical guarantees in terms of consistency, we must check whether they retain language modeling quality. To do so, we perform experiments using a text completion task. As we saw in (§3.3.1), text completion consists of decoding a continuation $\hat{\mathbf{y}} \sim \mathcal{F}(p_\theta, \mathbf{x})$ given a length-k prefix $\mathbf{x} = (x_1, \ldots, x_k)$, resulting in a completion $(x_1, \ldots, x_k, \hat{y}_1, \ldots, \hat{y}_T)$. In each experiment, we measure the proportion of non-terminated continuations in order to approximately measure inconsistency. The first experiment shows that inconsistency occurs in practice, and the second experiment shows the effectiveness of the proposed approaches. The third

experiment shows that inconsistency also occurs frequently in GPT-2, a large-scale transformer language model.

Experiment setup. For the first two experiments we use Wikitext2 (Merity et al. 2016), which consists of paragraphs from English Wikipedia, since it has frequently been used to evaluate language models (Grave et al. 2017; Melis et al. 2018; Merity et al. 2018). We use BPE[8] tokenization, but we verified that word tokenization gives analogous results. We split each paragraph into sentences using Spacy[9]. We split each sentence, using the first k tokens as a context and the remaining tokens as a continuation. To ensure that each sequence contains a prefix, we prepend padding tokens to make it length k. Special $\langle \text{bos} \rangle$ and $\langle \text{eos} \rangle$ tokens are inserted at the beginning and end of each sequence. We use $k = 10$. We define empirical context distributions with prefixes from the train, valid, and test sets: $p(\mathbf{x}; \mathcal{D}) = \frac{1}{|\mathcal{D}|} \sum_{n=1}^{|\mathcal{D}|} \mathbb{1}(\mathbf{x} = \mathbf{x}^{(n)})$, where $\mathcal{D} = \{(\mathbf{x}^{(n)}, \mathbf{y}^{(n)})\}_{n=1}^{N}$ is a dataset split.

To approximately measure inconsistency, we use $\mathbf{nonterm}_L$ (Equation 15). Recall from (§5.1) that a value of $\mathbf{nonterm}_L$ greater than zero means that some sequences did not terminate within L steps. When L is infinity, this implies that the model paired with the decoding algorithm is inconsistent. In practice, we use a finite L and we interpret a non-zero $\mathbf{nonterm}_L$ as evidence that the model paired with the decoding algorithm is inconsistent. We use $L = 1500$, which is more than 10 times the maximum training sequence length.

We train recurrent language models for sequence completion with maximum likelihood, using the loss $\mathcal{L}(p_\theta, \mathbf{y}, \mathbf{x}) = -\sum_{t=1}^{T} \log p_\theta(y_t \mid y_{<t}, \mathbf{x})$, In practice we run the full completion $(\mathbf{x}, \mathbf{y})$ through a recurrent model and zero the loss for the first k tokens, so that the first k steps correspond to learning a g_θ that encodes $\mathbf{x}$. In each experiment, we report the mean and standard deviation of metrics across 10 independent initializations. Unless specified otherwise, we report metrics using the test context distribution, since the train, valid, and randomly generated context distributions had similar results.

We consider recurrent neural networks with hyperbolic tangent activations (tanh-RNN; Elman 1990) and LSTM units (LSTM-RNN; Hochreiter and Schmidhuber 1997). We perform an initial hyper-parameter sweep and select the best set of hyper-parameters for each of tanh-RNN and LSTM-RNN based on the validation perplexities.[10] With this best set of hyperparameters, we train each of these models with 10 different initializations. The choice of tanh and LSTM RNNs implies that all of the recurrent language models that we train are consistent according to Lemma 2. Our LSTM models achieve similar test perplexity (91.86 ± 0.4, word tokenization) to those reported in previous work (Merity et al. 2018).

[8] `https://github.com/huggingface/tokenizers`
[9] `https://spacy.io/`
[10] Refer to Appendix B.1.1 for the hyper-parameter ranges.

	tanh-**RNN**	**LSTM-RNN**
ancestral	0.00 ± 0.00	0.00 ± 0.00
greedy	12.35 ± 5.18	1.53 ± 1.41
beam-2	1.38 ± 0.95	0.07 ± 0.06
beam-4	0.25 ± 0.19	0.00 ± 0.01
topk-2	0.01 ± 0.01	0.01 ± 0.01
topk-4	0.00 ± 0.00	0.00 ± 0.01
nucleus-0.2	0.06 ± 0.02	0.13 ± 0.15
nucleus-0.4	0.04 ± 0.02	0.02 ± 0.01
consistent topk-2	0.00 ± 0.00	0.00 ± 0.01
consistent topk-4	0.00 ± 0.00	0.00 ± 0.00
consistent nucleus-0.2	0.04 ± 0.02	0.01 ± 0.01
consistent nucleus-0.4	0.02 ± 0.02	0.01 ± 0.01

Table 5: Non-termination (**nonterm**$_L$ (%)) of decoded sequences using ancestral sampling, incomplete, and consistent decoding methods.

	ST	ϵ	**nonterm**$_L$(%)	**perplexity**
tanh-RNN	✓	10^{-2}	00.00 ± 0.00	229.09 ± 9.2
	✓	10^{-3}	00.00 ± 0.00	191.63 ± 1.4
	✓	10^{-4}	00.02 ± 0.02	188.36 ± 2.2
	✗	–	12.35 ± 5.20	186.44 ± 1.4
LSTM	✓	10^{-2}	0.00 ± 0.00	219.71 ± 9.2
	✓	10^{-3}	0.00 ± 0.00	186.04 ± 1.6
	✓	10^{-4}	0.18 ± 0.35	183.57 ± 2.3
	✗	–	1.48 ± 1.43	178.19 ± 1.3

Table 6: Non-termination of greedy-decoded sequences and test perplexity for STRLMs.

Additionally, we train self-terminating tanh-RNN and LSTM-RNN variants (Definition 16) at various values of ε, which controls a lower bound on the termination probability at each step. We use $\sigma(x) = (1-\varepsilon)\cdot\text{sigmoid}(x)$. We use the hyper-parameters selected in the preceding grid search.

Experiment 1: Inconsistency occurs in practice. In this experiment, we demonstrate evidence of inconsistency with incomplete decoding methods. Table 5 shows non-termination for the recurrent language models using various decoding algorithms. Decoding with ancestral sampling always resulted in sequences that terminated within L steps, since the induced distribution is the same as that of the consistent model. On the other hand, the non-zero non-termination for the incomplete decoding algorithms suggests inconsistency, providing evidence for Theorem 1.

Using greedy decoding, roughly 12% of all contexts resulted in a non-terminating continuation with the tanh-RNN, and roughly 1% with the LSTM-RNN. Nucleus sampling also produced non-terminating sequences with the tanh-RNN (0.06%, nuc-0.2) and LSTM-RNN (0.13%, nuc-0.2). Top-k sampling yielded a small number of non-terminating samples. In general, non-termination approaches zero as k and μ increase, since $\langle\text{eos}\rangle$ has a lower chance of being excluded. Our finding that non-termination occurs at low settings of μ and k is relevant in practice, since low settings (e.g. $\mu \in \{0.1, 0.3\}$) have been shown to receive substantially better human evaluation scores in open-ended generation compared to high settings (e.g. $\mu \in \{0.7, 0.9\}$) (Zhang et al. 2020).

Beam search produced non-terminating sequences with both the tanh-RNN and LSTM-RNN models. This means that $\langle\text{eos}\rangle$ was outside of the top tokens (determined by the beam width) considered at each step, since in our experiments we terminated the beam search when a single beam prefix contained $\langle\text{eos}\rangle$. Larger beam widths reduce or eliminate non-termination, similar to increasing k or μ.

Experiment 2: Fixing the inconsistency. Table 5 shows that consistent nucleus and top-k sampling resulted in only terminating sequences, except for a few cases that we attribute to the finite limit L used to measure the non-termination ratio. Consistent nucleus paired with tanh-RNN did not reduce $\mathbf{nonterm}_L$ as much as when it was paired with LSTM-RNN. Example continuations are shown in Table 7. On prefixes that led to non-termination with the baseline method, the quality tends to improve with the consistent variant since the continuation now terminates. Note that since the model's non-⟨eos⟩ token probabilities at each step are only modified by a multiplicative constant, the sampling process can still enter a repetitive cycle (e.g., when the constant is close to 1), though it is guaranteed to terminate.

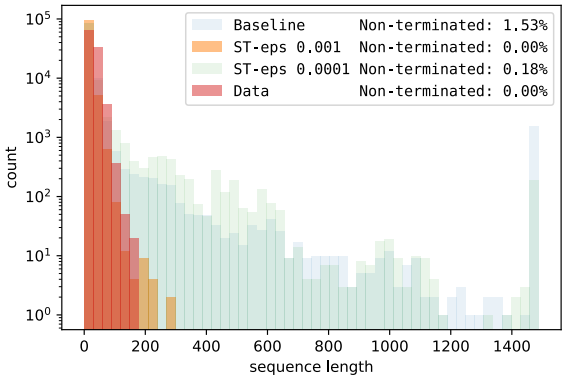


Fig. 13: Ground-truth and greedy-decoded continuation lengths from vanilla and self-terminating LSTMs (Wikitext-2).

As seen in Table 6, the self-terminating recurrent language models are consistent with respect to greedy decoding, at the expense of perplexity compared to the vanilla model. The value of ε from Definition 16, which controls a lower-bound on termination probability at each step, influences both $\mathbf{nonterm}_L$ and perplexity. When ε is too large ($\varepsilon = 10^{-2}$), perplexity degrades. When ε is too small ($\varepsilon = 10^{-4}$), the lower-bound grows slowly, so ⟨eos⟩ is not guaranteed to be top-ranked within L steps, resulting in a positive $\mathbf{nonterm}_L$. An ε of 10^{-3} balanced consistency and language modeling quality, with a zero non-termination ratio and perplexity within 8 points of the baseline.

As shown in Figure 13, the self-terminating model matches the data length distribution better than the baseline. Example decoded sequences are shown in Table 7. For prefixes that led to non-termination with the baseline, the self-terminating models yields finite sequences with reasonable quality. The examples suggest that some cases of degenerate repetition (§5.2) are attributed to inconsistency.

Experiment 3: Inconsistency in a large-scale transformer. We perform a final experiment with GPT-2 117M (Radford et al. 2018), a transformer language model with a byte-level BPE vocabulary of 50k tokens, pre-trained with maximum likelihood on WebText, a dataset of scraped web pages (refer to Radford et al. (2018) for details). We fine-tune GPT-2 on the Wikitext-103 dataset (Merity et al. 2016), a large-scale collection of Wikipedia articles with over 100 million words and 260 thousand unique tokens. We split the dataset into sequences

	$\mathbf{nonterm}_L$ (%)	**perplexity**
GPT2-117M	37.91	20.92
GPT2-117M ST	00.00	27.25

Table 8: Greedy non-termination for GPT2 the self-terminating variant (ST).

Prefix	*One Direction delivered a performance of " Kiss You*
nucleus	" , and the album 's second album , " The X @-@ Files " , " The A. " , " The Preder " , " We 've Have You " , " I 've You Wanna Stay " , " The Dream " , " The Bide " , " My Achievement ", " The B. B. " , " A Life " ...
c-nucleus	" , and " My Boo " was released on September 29 , 2010 . ⟨eos⟩
Prefix	*Boulter starred in two films in 2008 ,*
nucleus	and the band 's music , and " The Rise of Monkey " , " The One With the Way " , " The " Always " , " " Always Your " , " The Wift " , " The Baste " , " The Special With " , " The Way " , " The Special With You " ...
c-nucleus	and the latter was released in the United States . ⟨eos⟩
Prefix	*This period of unhappiness was the making of*
Baseline	the " most important " of the " mad " , and the " " most important " of the " " " , " the most important " , and the " devil " , " The " , " The One " , " The One " , " The One " , " The One " , " The One " , " The One " , " The One " , " The One " , " The One " , " The One " , " The One " , " The One " , " The One " , " The One " ...
STRLM	the first commandment of the poem . ⟨eos⟩
Prefix	*Du Fu 's mother died shortly after he was*
Baseline	a member of the Order of the Order of the Order of the Order of the Order of the Order of the Order of the Order of the Order of the Republic of the Republic of the Republic of the Republic of the Republic of ...
STRLM	a member of the Order of the British Empire . ⟨eos⟩

Table 7: Continuations with consistent nucleus sampling ($\mu = 0.2$) and self-terminating LSTM ($\epsilon = 10^{-3}$).

according to the dataset's newline boundaries, then split each sequence into a context $\mathbf{x}$ and continuation $\mathbf{y}$, resulting in a dataset of $(\mathbf{x}, \mathbf{y})$ pairs. Each continuation ends in an ⟨eos⟩ token. We use a context size of $k = 10$ tokens, and discard sequences that are length k or shorter. The resulting dataset contains 874,556 training, 1,896 validation, and 2,162 test pairs.

We fine-tune the pre-trained GPT-2 model using maximum likelihood for 400k steps, and select the model state with the lowest validation perplexity (evaluated every 5k steps). Each training batch contains a maximum of 1024 total tokens. We use the implementation and default hyper-parameters from the `transformers` library (Wolf et al. 2019). We fine-tune the self-terminating GPT-2 models in a similar manner, starting from the pre-trained GPT-2 model and using the same hyper-parameters.

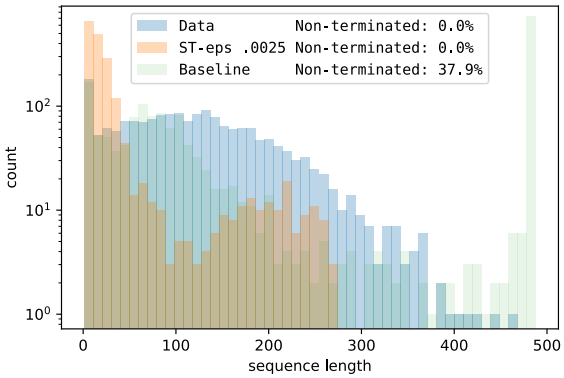


Fig. 14: Ground-truth and greedy-decoded continuation lengths from baseline and self-terminating GPT-2 117M (Wikitext-103).

Each model is evaluated using greedy decoding with a maximum sequence length of 500, which was selected so that each decoded validation batch could fit in GPU memory. The non-termination rate ($\mathbf{nonterm}_L$) also uses $L = 500$, which is more strict than the limit used in the preceding experiments, yet still yields large differences between generations and the ground truth (e.g. Figure 14).

Table 8 shows non-termination and perplexity of the baseline and self-terminating GPT-2 models. The self-terminating variant prevents non-termination, at the cost of perplexity. The model here uses $\epsilon = 2.5 \times 10^{-3}$, which we selected after observing that at higher values of ϵ, e.g. 1.0×10^{-3}, the self-terminating model generated sequences longer than the limit used to determine termination (500). Figure 14 shows the length distributions of the baseline GPT-2 continuations and those of the self-terminating GPT-2. The GPT-2 117M model generates many sequences at or near the maximum sequence length (500), unlike the ground-truth data. Introducing self-termination shifts the mass towards shorter sequences, whose lengths are also present in the ground-truth data, though it does not match the ground-truth length distribution as well as the self-terminating LSTM. The self-terminating GPT-2 would also place $\langle \text{eos} \rangle$ midway through a sentence, which we did not observe with the LSTM models.

6.3 Discussion

In this section, we analyzed non-termination through the notion of sequence distribution consistency. We discovered a discrepancy between the model's sequence distribution and the distribution induced by a decoding algorithm: in practice, the model is consistent, yet the induced distribution is inconsistent. We demonstrated how inconsistency can be prevented by changing either the decoding algorithm or the model, and confirmed that empirical inconsistency occurs in practice.

A drawback of our proposed methods is that they are specifically designed for the inconsistency problem. Moreover, when applied to GPT-2, the self-terminating layer led to a mismatched length distribution and odd behavior such as placing $\langle \text{eos} \rangle$ in the middle of a sentence, suggesting that there is room for further improving decoding algorithms or model parameterizations. Another approach is to address inconsistency in the *learning* phase, as the lack of decoding during maximum-likelihood training may be a cause of inconsistency.

7 Learning: Unlikelihood

In this section, we investigate how varying the *learning algorithm* can impact text degeneration. Reducing *repetition* will be our motivation, but we develop a generic learning method that can also improve *unigram distribution mismatch* and will prove useful later for reducing *logical incoherence* (§8).

As we discussed in (§5), model-generated text in open-ended applications such as language modeling or dialogue has been observed to contain token, phrase, and sentence level repetition that does not resemble the model's training corpus. Moreover, generations are often dull, meaning that high frequency tokens (e.g. *the*, *of*, *how*) are used too often and interesting content words are used too rarely (Dinan et al. 2019a; Holtzman et al. 2020), which we formalized as a unigram distribution mismatch (§5.4). Repetition and dullness are often addressed by using nucleus sampling (Definition 8, Holtzman et al. (2020)) rather than using a decoding algorithm that approximates the argmax sequence (Definition 5), or by blocking repeating tokens during beam search. However, avoiding particular decoding algorithms or resorting to heuristics ignores the core issue: some of the model's underlying sequence probabilities are incorrect.

Several hypotheses for why neural text is degenerate have been posited, including degeneration being (i) a by-product of the model architecture, e.g. the transformer architecture preferring repeats (Holtzman et al. 2020), (ii) an intrinsic property of human language (Holtzman et al. 2020) rather than a modeling deficiency, or that (iii) a training objective relying on fixed corpora cannot take into account the real goal of using the language (Choi 2018). In this section we show that a primary factor is the use of the likelihood objective itself, as we demonstrate that degeneration is alleviated if we replace the likelihood objective with our proposal.

While low perplexity in the limit should lead to predicting the correct next target word, we hypothesize that in practice there are at least two flaws of the likelihood objective which contribute to text degeneration: (i) it uses only positive supervision, relying solely on normalization to decrease probability; (ii) it only uses ground-truth sequences, and in particular does not train on decoded sequences. The first issue means that maximum-likelihood does not distinguish between errors – such as predicting a synonym versus a repeating token – and does not allow for explictly penalizing known biases. The second issue means that properties that are present in sequences sampled from the distribution induced by the model and decoding algorithm but not present in the training data, such as excessive repetition, are not encountered during likelihood training.

In this section, we introduce *unlikelihood training*, a general framework for incorporating negative supervision into sequence model training, which alleviates the two aforementioned issues. It combines two

types of updates: a likelihood update on the true target tokens so that they are assigned high probability, and an *unlikelihood* update on *negative candidate* tokens that are assigned too high of a probability. We show how to construct negative candidates for next-token distributions – called *token-level unlikelihood*, for decoded sequences – called *sequence-level unlikelihood*, and for sequences from an external dataset – called *external unlikelihood*.

We evaluate unlikelihood on text completion and on dialogue modeling. On text completion, both token-level and sequence-level unlikelihood training are shown to improve metrics that measure repetition and dullness in generated text, while maintaining performance in other metrics such as perplexity compared to the maximum likelihood baseline. According to human evaluations, the generations have vastly improved quality compared to likelihood-trained models when both models use beam search decoding. Moreover, our approach with beam search also significantly improves over likelihood-trained models using either beam blocking or nucleus sampling, thus outperforming the current state-of-the-art.

On dialogue modeling, we confirm that unlikelihood can also reduce repetition in the dialogue domain, and show that unlikelihood can be used to address unigram distribution mismatch. Unlikelihood opens up many possibilities for incorporating negative supervision into sequence model training; later in the thesis (§8), we show how it can be used to reduce logical incoherence.

7.1 Method

We will introduce the general unlikelihood objective, then consider special cases that correspond to different training settings and applications.

The key idea behind unlikelihood training is decreasing the model's probability of certain tokens at each timestep, called *negative candidates* $\mathcal{C}_t$, using the following *unlikelihood loss*,

$$\mathcal{L}_{\text{UL}}(p_\theta, \mathcal{C}_{1:T}, \mathbf{x}, \mathbf{y}) = -\sum_{t=1}^{T} \sum_{y_c \in \mathcal{C}_t} \beta(y_c) \log\left(1 - p_\theta(y_c | y_{<t}, \mathbf{x})\right), \tag{23}$$

where $\mathcal{C}_t \subseteq \mathcal{V}$ is a subset of the vocabulary, $\beta(y_c)$ is a candidate-dependent scale that controls how much the candidate token should be penalized, $\mathcal{C}_{1:T} = \{\mathcal{C}_1, \ldots, \mathcal{C}_T\}$, and $T = |\mathbf{y}|$. The unlikelihood loss decreases as the model assigns smaller probability to each negative candidate.

Unlikelihood training then consists of mixing the likelihood and unlikelihood losses,

$$\mathcal{L}_{\text{ULE}}(p_\theta, \mathcal{C}_{1:T}, \mathbf{x}, \mathbf{y}, \mathbf{y}_*) = \mathcal{L}_{\text{MLE}}(p_\theta, \mathbf{x}, \mathbf{y}_*) + \alpha \mathcal{L}_{\text{UL}}(p_\theta, \mathbf{x}, \mathbf{y}), \tag{24}$$

where $\alpha \in \mathbb{R}_{\geq 0}$ is a scaling hyper-parameter. The likelihood term $\mathcal{L}_{\text{MLE}}$ tries to model the overall sequence distribution through *increasing* the probability of the ground truth continuation $\mathbf{y}_*$. With an autoregressive model, this corresponds to increasing the probability of the ground-truth next-token in each conditional distribution $p_\theta(y|y^*_{<t}, \mathbf{x})$. The unlikelihood term $\mathcal{L}_{\text{UL}}$ corrects for known biases by *decreasing* the probability of negative candidate tokens.

We now introduce three training settings for unlikelihood, which amount to different choices of the sequence $\mathbf{y}$ to which we apply the unlikelihood loss (Equation 24). After, we introduce different candidate choices ($\mathcal{C}_t$) and token-dependent weights (β) that yield applications of unlikelihood training.

Token-level unlikelihood. When $\mathbf{y}$ is the ground-truth sequence $\mathbf{y}_*$, we call the resulting objective *token-level unlikelihood* since it operates on the same next-token distributions as MLE:

$$\mathcal{L}_{\text{ULE-token}}(p_\theta, \mathcal{C}_{1:T}, \mathbf{x}, \mathbf{y}_*, \mathbf{y}_*), \tag{25}$$

where each time-step's loss is,

$$\mathcal{L}^t_{\text{ULE-token}} = -\log p_\theta(y^*_t|y^*_{<t}, \mathbf{x}) - \alpha \cdot \sum_{y_c \in \mathcal{C}_t} \beta(y_c) \log\left(1 - p_\theta(y_c|y^*_{<t}, \mathbf{x})\right). \tag{26}$$

Sequence-level unlikelihood. When $\mathbf{y}$ is a decoded sequence, $\hat{\mathbf{y}} \sim \mathcal{F}(p_\theta, \mathbf{x})$, we call the resulting objective *sequence-level unlikelihood*,

$$\mathcal{L}_{\text{ULE-seq}}(p_\theta, \mathcal{C}_{1:T}, \mathbf{x}, \hat{\mathbf{y}}, \mathbf{y}_*). \tag{27}$$

The candidates are designed to penalize unwanted properties of the decoded sequences (e.g. repetition), as we will discuss below (§7.1.1).

External unlikelihood. Unlikelihood can also be applied to sequences from an external dataset. We assume access to two data collections,

$$\mathcal{D}^+ = \{(\mathbf{x}, \mathbf{y}^+)\}, \quad \mathcal{D}^- = \{(\mathbf{x}, \mathbf{y}^-)\}. \tag{28}$$

Intuitively, $\mathcal{D}^+$ represents behavior that we want to encourage, while $\mathcal{D}^-$ contains behavior that we want to discourage. Applying likelihood to $\mathbf{y}^+$ and unlikelihood to $\mathbf{y}^-$ yields *external unlikelihood*,

$$\mathcal{L}_{\text{ULE-ext}}(p_\theta, \mathcal{C}_{1:T}, \mathbf{x}, \mathbf{y}^-, \mathbf{y}^+). \tag{29}$$

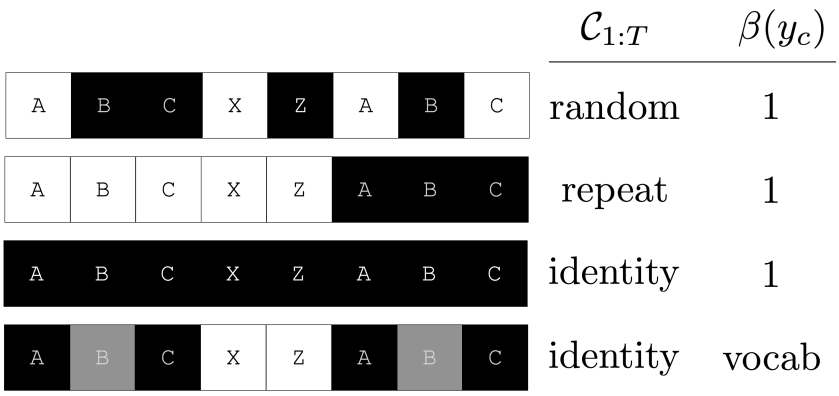


Fig. 15: Per-step unlikelihood losses induced by candidate choices ($\mathcal{C}_{1:T}$) and token-dependent weights ($\beta(y_c)$) using sequence-level unlikelihood ($\mathcal{L}_{\text{ULE-seq}}$). Each row shows a decoded continuation $\hat{\mathbf{y}}$. Sequence-level candidates determine whether to apply an unlikelihood loss (i.e. "penalize") at each step by setting $\mathcal{C}_t$ to either $\emptyset$ (no loss) or $\{y_t\}$ (loss). Darker values indicate larger unlikelihood losses for a fixed value of $(1 - p_\theta(y_t|y_{<t}, \mathbf{x}))$. $\mathcal{C}_{1:T}^{\text{random}}$ candidates penalize a random subset of timesteps. $\mathcal{C}_{1:T}^{\text{repeat}}$ candidates penalize timesteps which are part of a repeating n-gram (here $n = 3$). $\mathcal{C}_{1:T}^{\text{identity}}$ penalizes all timesteps, and $\beta(y_c)$ scales the penalty. The final row shows a scaling based on unigram frequency (Equation 34); the dark cells imply that A, B, C are over-produced by the model, especially A and C.

There is a raft of recent large-scale, high quality data that can be massaged into this form, from natural language inference (NLI) tasks (Bowman et al. 2015; Williams et al. 2018) to commonsense reasoning tasks (Zellers et al. 2019; Qin et al. 2019). Later (§8), we develop a dataset of this form and use it for reducing logical incoherence with unlikelihood training.

7.1.1 Applications. The choice of negative candidates ($\mathcal{C}_t$) and token-dependent weights ($\beta(y_c)$) influence which behaviors are penalized. Figure 15 summarizes the following discussion.

Unspecified behavior. The simplest form of unlikelihood does not specify a behavior to penalize. In this case, we use sequence-level unlikelihood to randomly penalize a subset of each decoded sequence,

$$\mathcal{C}_t^{\text{random}} = \begin{cases} \{x_t\} & \text{if } z_t = 1 \\ \emptyset & \text{if } z_t = 0, \end{cases} \tag{30}$$

where $z_t \sim \text{Bernoulli}(p_{\text{penalize}})$, and $p_{\text{penalize}} \in [0, 1]$ is a hyperparameter. Intuitively, these candidates treat the model's generations as problematic, thus penalizing random timesteps in each generation.

Repetition. For reducing degenerate repetition, we propose candidates for token-level and sequence-level unlikelihood. For token-level unlikelihood, we use the preceding context tokens,

$$\mathcal{C}_t^{\text{context}} = \{y_1, \ldots, y_{t-1}\} \setminus \{y_t\}. \tag{31}$$

Intuitively, these candidates make incorrect repeating tokens less likely, since repetition amounts to generating a token from the preceding context. Crucially, we never include the ground-truth token y_t.

For sequence-level unlikelihood, we set the decoded token y_t to be the single negative candidate for step t if it is part of an n-gram that already appeared in $y_{<t}$,

$$\mathcal{C}_t^{\text{repeat}} = \begin{cases} \{y_t\} & y_t \in \text{repeat n-gram} \\ \emptyset & \text{otherwise.} \end{cases} \tag{32}$$

In summary, to reduce repetition we propose $\mathcal{L}_{\text{UL-token}}(p_\theta, \mathcal{C}_{1:T}^{\text{context}}, \mathbf{x}, \mathbf{y}_*, \mathbf{y}_*)$ and $\mathcal{L}_{\text{UL-seq}}(\mathcal{C}_{1:T}^{\text{repeat}}, \mathbf{x}, \hat{\mathbf{y}}, \mathbf{y}_*)$. The former can be used for training from scratch, while the latter is used for fine-tuning a model since decoding $\hat{\mathbf{y}}$ is computationally expensive.

Unigram distribution mismatch. Neural sequence models trained with maximum-likelihood suffer from unigram distribution mismatch, meaning they tend to produce high frequency tokens too often and low frequency tokens too rarely, where frequency is defined by the human token distribution (see §5.4). We address this with unlikelihood by penalizing tokens according to the mismatch between the model and ground-truth unigram distributions. Specifically, we first maintain an empirical estimate of the model's unigram distribution $p_\theta(y_t)$ and the ground-truth distribution $p_*(y_t)$,

$$p_\theta(y) \approx \frac{\sum_{i=1}^N \sum_{t=1}^{|\hat{\mathbf{y}}^{(i)}|} \mathbb{I}\left[\hat{y}_t^{(i)} = y\right]}{\sum_{i=1}^N |\hat{\mathbf{y}}^{(i)}|}, \qquad p_*(y) \approx \frac{\sum_{i=1}^N \sum_{t=1}^{|\mathbf{y}^{(i)}|} \mathbb{I}\left[y_t^{(i)} = y\right]}{\sum_{i=1}^N |\mathbf{y}^{(i)}|}, \tag{33}$$

where $\{\mathbf{x}^{(i)}, \mathbf{y}^{(i)}\}_{i=1}^N$ is a subset of training data (e.g. the preceding 256 batches), and $\hat{\mathbf{y}}^{(i)} \sim \mathcal{F}(p_\theta, \mathbf{x}^{(i)})$.

We wish to *push down* the probability of over-produced tokens, i.e. when $p_\theta(y_t) > p_*(y_t)$. Each step's candidate is thus the current token, $\mathcal{C}_t^{\text{identity}} = \{y_t\}$, and each token's unlikelihood loss is scaled according to the mismatch between the approximated model and human distributions,

$$\beta(y_c) = p_\theta(y_c) \log\left(\frac{p_\theta(y_c)}{p_*(y_c)}\right). \tag{34}$$

The unlikelihood loss for a token y_c is non-zero when the token occurs more often in the model's estimated unigram distribution. In summary, the resulting loss is $\mathcal{L}_{\text{ULE-seq}}(p_\theta, \mathcal{C}_{1:T}^{\text{identity}}, \mathbf{x}, \hat{\mathbf{y}}, \mathbf{y}_*)$, where $\hat{\mathbf{y}} \sim \mathcal{F}(p_\theta, \mathbf{x})$ is a model-generated sequence.

7.1.2 Gradient analysis. To understand the behavior of unlikelihood compared to standard likelihood training, we analyze the gradient of the per-step token-level unlikelihood objective ($\mathcal{L}_{\text{ULE-token}}^t$,

		$\mathbf{rep}_4$
Prefix	*... Lyrically the song has excerpts of different languages including French , Spanish*	
$\mathcal{L}_{\text{MLE}}$	, Italian , Spanish , Italian , Spanish , Italian , Spanish , Spanish , Portuguese , Portuguese , Portuguese , Portuguese , Portuguese , Portuguese , Portuguese , Portuguese , Portuguese , Portuguese , Portuguese , Portuguese , Spanish , Portuguese , Portuguese , Portuguese , Portuguese , Portuguese ...	0.744
$\mathcal{L}_{\text{ULE-token+seq}}$	, and German . In the first verse , the protagonist sings about being a " girl who 's been in love with someone else " , while the second verse describes the relationship between the protagonist and her lover . In the third verse , the protagonist sings ...	0.063
Prefix	*... starboard engines and was going to crash . " We 're going in ,"*	
$\mathcal{L}_{\text{MLE}}$	he said . " We 're going to crash . We 're going to crash . We 're going to crash . We 're going to crash . We 're going to crash . We 're going to crash . We 're going to crash . We 're going to crash . We 're going to ...	0.787
$\mathcal{L}_{\text{ULE-token+seq}}$	Hood said . " I 'm going to make sure we 're going to get back to the water . " The order to abandon ship was given by Admiral Beatty , who ordered the remaining two battlecruisers to turn away . At 18 : 25 , Hood turned his ...	0.000
Prefix	*... career - high 27 points on 8 - for - 11 shooting with three rebounds*	
$\mathcal{L}_{\text{MLE}}$	and two assists . On January 3 , 2012 , he was named to the 2012 – 13 All - Atlantic 10 first team . On February 3 , 2012 , he was named to the Atlantic 10 first team . On February 5 , 2012 , he was named ...	0.277
$\mathcal{L}_{\text{ULE-token+seq}}$	and a career - high 7 assists against the Minnesota Timberwolves . On February 3 , 2012 , he was named to the 2012 All - NBA First Team . On March 7 , 2012 , he was named one of five finalists for the Naismith Award , which is ...	0.064

Table 9: Example greedy completions showing representative examples of the MLE model's degenerate single-token repetition (top), phrase-level repetition (middle), and 'structural' repetition (bottom), as well as the proposed method's ability to fix these degenerate behaviors.

Equation 26). We assume $p_\theta(y_t|y^*_{<t}, \mathbf{x}) = \text{softmax}(\mathbf{a})$, and $\beta(y_c) = 1$. With a single negative candidate, the (negative) gradient with respect to $a \in \mathbb{R}^{\mathcal{V}}$ is:

$$\nabla_a \mathcal{L} = y^* - m \odot p, \quad m_i = \begin{cases} (1 - \alpha \frac{p_{\text{neg}}}{1 - p_{\text{neg}}}) & \text{if } i \neq i_{\text{neg}} \\ (1 + \alpha) & \text{if } i = i_{\text{neg}}, \end{cases} \tag{35}$$

where $y^* \in \{0,1\}^{\mathcal{V}}$ is a one-hot ground-truth vector, $m \in \mathbb{R}^{\mathcal{V}}$, $p = p_\theta(\cdot|y_{<t}, \mathbf{x})$, and p_{neg} is the probability of the negative candidate at index i_{neg}.[11]

This unlikelihood gradient (Equation 35) differs from the likelihood gradient, $(y^* - p)$, due to the term m which varies based on the hyper-parameter α and the model's negative candidate probability, p_{neg}. At the ground-truth token index i^*, the unlikelihood gradient is positive, increasing the ground-truth token's probability with a magnitude that grows with p_{neg}. Conversely, at the negative candidate index i_{neg} the gradient is negative. At all other token indices $i \notin \{i^*, i_{\text{neg}}\}$, the gradient moves from negative to positive as p_{neg} increases. For instance, with $\alpha = 1.0$ the gradient increases the probability of each token x_i when the model assigns high probability to the negative candidate ($p_{\text{neg}} > 0.5$).

7.2 Empirical Evaluation

We evaluate unlikelihood training on text completion and dialogue modeling tasks. For text completion, we focus on addressing degenerate repetition as it is a major issue in this task (Holtzman et al. 2020). To this end, we evaluate the effects of using token-level and sequence-level unlikelihood with either repetition

[11] See Appendix A.1.2 for the derivation and a generalization to multiple candidates.

candidates or random candidates. For dialogue, we evaluate unlikelihood for repetition as well as for unigram distribution mismatch, which is a key issue in dialogue modeling (Dinan et al. 2019a).

7.2.1 Text completion. We follow a standard language modeling setup from Baevski and Auli (2019), using a 16-layer transformer language model. We use the Wikitext-103 dataset (Merity et al. 2016), a large-scale collection of Wikipedia articles containing over 100 million words.[12]

We train a token-level baseline with $\mathcal{L}_{\text{MLE}}$ and a token-level unlikelihood model with $\mathcal{L}_{\text{ULE-token}}$. We then fine-tune each with sequence-level unlikelihood ($\mathcal{L}_{\text{ULE-seq}}$) and call these fine-tuned models $\mathcal{L}_{\text{ULE-seq}}$ and $\mathcal{L}_{\text{ULE-token+seq}}$, respectively. For token-level unlikelihood we use $\mathcal{C}^{\text{context}}$ candidates and for sequence-level unlikelihood we use 4-gram $\mathcal{C}^{\text{repeat}}$ candidates unless otherwise noted. Beam-search uses width 10.

Metrics. We use the portion of duplicate n-grams to quantify repetition ($\mathbf{rep}_n(\mathbf{y})$, see Equation 16), averaged over continuations. $\mathbf{rep}_n$ is zero when the continuation has no repeating n-grams and increases towards 1.0 as the model repeats. As a token-level metric for repetition, we use the fraction of next-token predictions that occur in the previous ℓ tokens (**rep-tok/ℓ**), defined as:

$$\textbf{rep-tok}/\ell = \frac{1}{|\mathcal{D}|T} \sum_{\mathbf{x},\mathbf{y} \in \mathcal{D}} \sum_{t=1}^{T} \mathbb{I}\left[\arg\max p_\theta(y|y_{<t}, \mathbf{x}) \in \mathbf{y}_{t-\ell-1:t-1}\right]. \tag{36}$$

A predicted token is called a "single-token repeat" when $\mathbb{I}[\cdot]$ is 1. Some of these single-token repeats also occur in the ground-truth sequences, and we thus report a variant which only counts single-token repeats that are additionally not equal to the ground-truth next-token (**wrep-tok/ℓ**).

As a rough proxy of a model's token distribution diversity, we use the number of unique tokens in decoded continuations (**uniq**) and in next-token predictions (**uniq-tok**). For language modeling quality we use perplexity (**ppl**), and next-token prediction accuracy (**acc**).

7.2.1.1 Results. Token-level and sequence-level results on the test set are in Table 10.

Baseline. The baseline model trained with maximum likelihood ($\mathcal{L}_{\text{MLE}}$) achieved 25.64 test perplexity, comparable to a current state-of-the-art system (Baevski and Auli 2019) (24.92). However, the greedy baseline's sequence-level repetition (rep_4 .442) and single-token repeats (rep-tok .627) far exceed those in ground-truth text (.006, .487 respectively). The baseline continuations have far fewer unique tokens than ground-truth text (uniq 11.8k vs 19.8k), with a high rate of frequent tokens (Figure 28).

[12] Code and trained models are available at `https://github.com/facebookresearch/unlikelihood_training`; implemented with Fairseq (Ott et al. 2019).

Model	decoding	rep_4	uniq	ppl	acc	rep-tok	wrep-tok	uniq-tok
$\mathcal{L}_{\text{MLE}}$	greedy	.442	10.8k	**25.64**	**.395**	.627	.352	11.8k
	beam	.523	9.5k					
$\mathcal{L}_{\text{UL-token}}$	greedy	**.283**	**13.2k**	26.91	.390	**.577**	**.311**	**12.7k**
	beam	**.336**	**11.7k**					
$\mathcal{L}_{\text{UL-seq}}$	greedy	.137	13.1k	**25.42**	**.399**	.609	.335	12.8k
	beam	.019	18.3k					
$\mathcal{L}_{\text{UL-token+seq}}$	greedy	**.058**	**15.4k**	26.72	.395	**.559**	**.293**	**13.8k**
	beam	**.013**	**19.1k**					
Human	-	.006	19.8k	-	-	.487	-	19.8k

Table 10: Token-level objectives and sequence-level fine-tuning (wikitext-103 test set).

Token-level objective. The token-level unlikelihood objective ($\mathcal{L}_{\text{UL-token}}$) reduced next-token wrong repetition (wrep-tok .311 vs. .352) while increasing the number of unique next-tokens (uniq-tok 12.7k vs. 11.8k) compared to the baseline ($\mathcal{L}_{\text{MLE}}$). Perplexity and accuracy were similar.

Importantly, the token-level unlikelihood objective yielded substantial improvements in the quality of decoded sequences. With greedy search, token-level unlikelihood training improved the 4-gram repetition in continuations by 36% (rep_4 .283 vs. .442) while generating roughly 22% more unique tokens than the baseline (uniq 13.2k vs. 10.8k), and a more favorable rate of infrequent tokens (Figure 28). With beam search, unlikelihood training showed similar improvements over the baseline.

Sequence-level objective. The sequence-level fine-tuning ($\mathcal{L}_{\text{ULE-token+seq}}$) yielded further improvements, with a 97% reduction in 4-gram repetitions rep_4 .013 vs. .442) from the baseline level (greedy $\mathcal{L}_{\text{MLE}}$), and 77% more unique tokens (uniq 19.1k vs. 10.8k) with beam search.

Compared to the token-level unlikelihood model ($\mathcal{L}_{\text{UL-token}}$) which was the starting point of fine-tuning, the fine-tuned model's repetition substantially improved (rep_4 .058 vs. .283), unique tokens increased (uniq 15.4k vs. 13.2k), and token-level metrics such as perplexity improved (ppl 26.72 vs. 26.91), despite using *only 1,500 updates*. The token distribution also improved, with infrequent tokens produced more often than the baseline, and frequent tokens approaching the ground-truth level (Figure 28). Finally, after sequence-level fine-tuning, beam search out-performed greedy search.

Candidates	rep_4	uniq	ppl
$\mathcal{L}_{\text{MLE}}$	.495	9.4k	24.59
Random (0.1)	.274	13.1k	24.33
Random (0.5)	.234	13.9k	25.57
Random (0.9)	.231	13.5k	26.52
Repeat	.018	16.8k	24.28
Human	.005	18.9k	–

Table 11: Fine-tuning $\mathcal{L}_{\text{MLE}}$ using sequence-level unlikelihood with the specified candidates. Metrics computed using beam search on the wikitext-103 valid set.

			Crowdworkers	Experts
Winner		Loser	Win rate	Win rate
$\mathcal{L}_{\text{ULE-token}}$		$\mathcal{L}_{\text{MLE}}$	57%	
$\mathcal{L}_{\text{ULE-seq}}$		$\mathcal{L}_{\text{MLE}}$	*71%	
$\mathcal{L}_{\text{ULE-token+seq}}$	*beats*	$\mathcal{L}_{\text{MLE}}$	*82%	
$\mathcal{L}_{\text{ULE-token+seq}}$		$\mathcal{L}_{\text{ULE-token}}$	*75%	
$\mathcal{L}_{\text{ULE-token+seq}}$		$\mathcal{L}_{\text{ULE-seq}}$	59%	
$\mathcal{L}_{\text{ULE-token+seq}}$	*beats*	$\mathcal{L}_{\text{MLE}}$ nucleus ($p = 0.9$)	59%	*83%
$\mathcal{L}_{\text{ULE-token+seq}}$		$\mathcal{L}_{\text{MLE}}$ beam block (4-gram)	60%	*74%

Table 12: **Human evaluation results**. * denotes statistical significance (2-sided binomial test, $p < .05$).

To visualize how these improvements in metrics translate to generation quality, Table 9 shows greedy completions that characterize the baseline's degeneration and $\mathcal{L}_{\text{ULE-token+seq}}$'s improved behavior.

Random candidates. Table 11 shows results for using sequence-level unlikelihood with random candidates ($\mathcal{C}_t^{\text{random}}$, Equation 30) for fine-tuning the MLE model. Penalizing 10% of the generated tokens at random led to substantial improvements in repetition compared to the baseline (.274 vs. .495), and improved perplexity (24.33 vs. 24.59). Penalizing more tokens further reduces repetition, at the cost of perplexity. As expected, explicitly penalizing repeats yields the lowest repetition (.018). In summary, unlikelihood improves the MLE model, even without explicitly using repetition to define the negative candidates.

Human evaluation. We perform a crowdworker evaluation to judge the quality of the generations of our proposed models compared to each other, the baseline, and two other generation methods. We employ a pairwise setup: an evaluator is presented with a prefix and shown continuations from two different models and asked to select which continuation they found more natural. Following Li et al. (2019a), we filter workers using quality controls and limit the number of annotations that they may complete (detailed in Welleck et al. (2020)). We also collected limited annotations from other NLP researchers. These *expert* annotators were given the same UI as the crowdworkers, and not told about models they were evaluating, but all annotators were familiar with language models. Prompts are from the Wikitext-103 test set, and all models use beam search unless otherwise noted. We report the win rates for each pairwise comparison.

Results are presented in Table 12. We find that all proposed models are preferred over the $\mathcal{L}_{\text{MLE}}$ baseline, and that congruent with automatic metrics, win rates improve after adding the sequence-level objective. The best unlikelihood model also outperforms the baseline used with either nucleus sampling or beam blocking, with statistical significance according to the expert annotators.

7.2.2 Dialogue modeling. In our dialogue experiments, we evaluate unlikelihood for reducing repetition and controlling vocabulary usage (i.e. unigram distribution mismatch), and their downstream effects on human judgments of dialogue quality.

Our experiments employ a large pre-trained seq2seq transformer, which we then fine-tune for particular tasks as specified.[13] Recall from (§3.3.3) that dialogue generation involves learning a model $p_\theta(\mathbf{y}|\mathbf{x})$, where the *context* $\mathbf{x} = (\mathbf{s}_1, \ldots, \mathbf{s}_k, \mathbf{u}_1, \ldots, \mathbf{u}_{t-1})$ contains contextual sentences (e.g. scenarios, personas, etc.) and the conversation history, and $\mathbf{y} = (y_1, \ldots, y_T)$ is the next utterance $\mathbf{u}_t$.

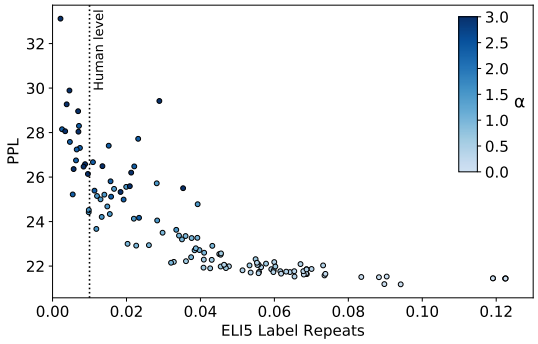


Fig. 16: ELI5: Perplexity vs. label repeats as α varies in the unlikelihood objective.

7.2.2.1 Repetition and copying. We use sequence-level unlikelihood and experiment with two candidate sets. As in text completion, we first consider penalizing an output token if it is part of an n-gram which already occurred in the model's output $y_{<t}$, called *label repetition*. Generative dialogue models are also known to rely too much on *copying* the given contextual information or dialogue history contained in $\mathbf{x}$, called *context repetition*. These two types of repetition motivate two candidate sets,

$$\mathcal{C}_t^{\text{label-repeat}} = \begin{cases} \{y_t\} & y_t \in \text{repeat label n-gram} \\ \emptyset & \text{otherwise} \end{cases}, \quad \mathcal{C}_t^{\text{context-repeat}} = \begin{cases} \{y_t\} & y_t \in \text{repeat context n-gram} \\ \emptyset & \text{otherwise,} \end{cases}$$

where y_t is a token in a repeating context n-gram when y_t is part of an n-gram that appeared in $\mathbf{x}$. We measure label repetition with the same $\mathbf{rep}_n(\mathbf{y})$ metric used in text completion, while we measure context repetition with the fraction of generated n-grams that appear in the context $\mathbf{x}$:

$$\mathbf{rep}_n^{\text{context}}(\mathbf{x}, \mathbf{y}) = \frac{|\text{n-grams}(\mathbf{y}) \cap \text{n-grams}(\mathbf{x})|}{|\text{n-grams}(\mathbf{y})|}.$$

Results. Table 13 shows results on ConvAI2 persona-based dialogue (Zhang et al. 2018), Wizard of Wikipedia knowledge-grounded dialogue (Dinan et al. 2019b) and ELI5 long-form question answering (Fan et al. 2019). The maximum-likelihood baseline exhibits repetition that far exceeds the human (i.e. ground-truth data) rate, with particularly high context repetition on ConvAI2 (.113) and Wizard of Wikipedia (.441), and high label repetition on ELI5 (.617).

[13] See Appendix B.1.2.

	ConvAI2				Wizard				ELI5			
Model	PPL	F1	Context	Label	PPL	F1	Context	Label	PPL	F1	Context	Label
Human	-	-	.0223	.0004	-	-	.160	.001	-	-	.009	.010
MLE Baseline	11.4	.199	.1131	.0210	8.3	.368	.441	.014	21.0	.130	.033	.617
UL (Context only)	11.8	.194	.0330	.0069	8.8	.346	.229	.037	21.4	.163	.008	.322
UL (Label only)	11.4	.203	.0984	.0005	8.3	.371	.426	.001	21.4	.183	.015	.055
UL (Context + Label)	11.9	.193	.0352	.0023	8.5	.358	.313	.009	21.8	.184	.009	.078

Table 13: Evaluation on the ConvAI2, Wizard of Wikipedia, and ELI5 tasks, comparing standard likelihood (MLE) with context and label repetition unlikelihood training. MLE exhibits a high level of context or label repetition compared to humans, with severity and type of repetition depending on the task. Each repetition type can be decreased depending on which form of unlikelihood is used.

			Token frequency classes			
Model	PPL	F1	Freq	Med	Rare	Rarest
Human	-	-	.400	.300	.200	.100
MLE Baseline	11.4	.199	.491	.282	.157	.068
UL, $\alpha = 10^0$	11.4	.200	.483	.289	.163	.063
UL, $\alpha = 10^1$	11.9	.201	.459	.328	.154	.058
UL, $\alpha = 10^2$	12.5	.190	.430	.335	.163	.071
UL, $\alpha = 10^3$	14.4	.174	.399	.339	.188	.073

Table 14: Vocabulary control with unlikelihood.

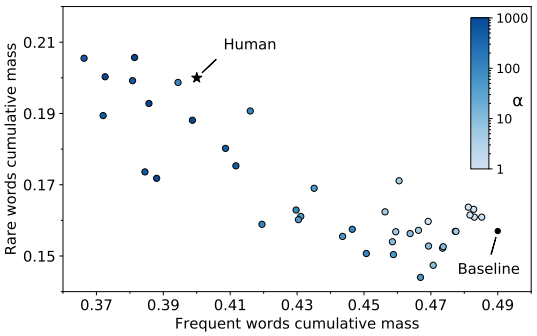


Fig. 17: Vocabulary control as α varies.

In all cases, unlikelihood using context-only candidates dramatically reduces context repetition (and analogously for label-only candidates), and training with context and label candidates (i.e. $\mathcal{C}_t^{\text{label-repeat}} \cup \mathcal{C}_t^{\text{context-repeat}}$) reduces both types of repetition without a large cost to perplexity. In Figure 16 we study how the hyperparameter α, which determines the unlikelihood loss's weight, influences repetition and perplexity. The hyperparameter α controls repeats smoothly, with only very high values resulting in increased perplexity.

7.2.2.2 Vocabulary control. We evaluate unlikelihood for reducing the mismatch between model and human token distributions (Equation 34), which we refer to as *vocabulary control*. We use the ConvAI2 dataset. Starting with a baseline MLE model, we then fine-tune several models using unlikelihood (Equation 34) at logarithmically interpolated values of $\alpha \in [1, 1000]$.

We partition the vocabulary into 'frequent', 'medium', 'rare', and 'rarest' using the human unigram distribution computed with the ConvAI2 training set, corresponding to the sorted token sets whose cumulative mass accounts for the top 40%, the next 30%, the next 20% and the final 10% of usage, respectively. We evaluate a model by generating utterances given contexts from the ConvAI2 validation set, and compute the fraction of tokens within each class.

Results. Figure 17 shows how the unigram distribution obtained after unlikelihood training is affected by the choice of mixing hyperparameter α: it can smoothly transition between the ground-truth distribution ('Human') and the MLE-trained distribution ('Baseline'). Table 14 compares the MLE baseline with unlikelihood. The unlikelihood fine-tuning shifts probability mass from the over-represented frequent words towards under-represented medium and rare words, with the effect strengthening as α increases. At a small cost to perplexity and F1, unlikelihood reduced the overuse of common tokens by 9 points, matching the human rate, while improving the production of rare tokens by 3 percentage points.

Finally, we evaluate both repetition and vocabulary unlikelihood with human judgments. Refer to (Li et al. 2020) for details on the evaluation setup. Results are shown in Figure 18, showing statistically significant improvements in human quality judgments over the baseline (two-tailed binomial test, $p < 0.01$) with both repetition (left) and vocabulary control (right) unlikelihood.

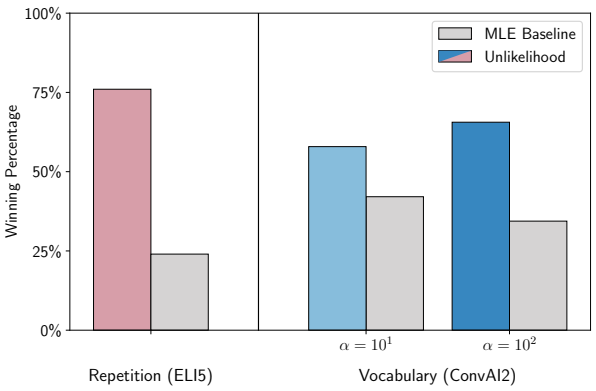


Fig. 18: Human evaluation experiments for label unlikelihood on ELI5 (left), and vocabulary unlikelihood on ConvAI2 for two values of α (right).

7.3 Discussion

In this section, we investigated neural text degeneration through the lens of the learning algorithm. In text completion and dialogue modeling tasks, we observed that state-of-the art models trained to maximize likelihood produce degenerate text characterized by excessive repetition and a mismatched token distribution. We described *unlikelihood training*, an approach to training neural language models that incorporates negative penalties. Unlikelihood training allows for controlled reduction of unwanted properties in generated sequences by specifying negative candidates and token-dependent weights.

8 Data: Dialogue Natural Language Inference

In this section we investigate the issue of *logical incoherence* (§5.3) in neural text generation, specifically dialogue modeling. As we saw in Figures 9 and 10, a model may produce statements that logically contradict its preceding statements. This phenomenon is often known as *contextual inconsistency* or *logical inconsistency*, but we will use the term *logical incoherence* to avoid confusion with the separate notion of consistency that we investigated in (§6). Studying logical incoherence requires a way to detect and quantify logical contradictions, which is difficult to do with automatic metrics, since it requires understanding logic and commonsense rather than surface-level statistics. In this section we will take a first step towards doing so, by considering a restricted domain, reducing logical incoherence to a well-studied problem, and using human evaluation in a scalable way.

We choose the domain of *persona-based chit-chat dialogue* (Zhang et al. 2018), in which each agent is given a set of facts $\{\mathbf{s}_1, \ldots, \mathbf{s}_k\}$ that describe the agent's personality (e.g. $\{$*i have a dog*, *i like sushi*$\}$), then produces utterances that reflect its personality (e.g. *i've always wanted to try the sushi at Tsukiji market*). This domain is suitable for studying logical incoherence, since (a) the personality sentences give a well-defined set of facts that should not be contradicted (e.g. the agent should not generate *i hate sushi*), and (b) the facts are typically simple statements, easing the process of checking contradictions.

We frame logical incoherence in dialogue in terms of natural language inference (NLI) (Dagan et al. 2006; Maccartney and Manning 2009; Bowman et al. 2015), a well-studied problem that amounts to learning a mapping between a sentence pair and an entailment category. This framing allows us to improve dialogue performance and study logical incoherence using methods developed for NLI. Specifically, we create a dataset, *Dialogue NLI*, which contains sentence pairs that are drawn from a dialogue domain and labeled as entailment, neutral, or contradiction. We demonstrate that Dialogue NLI can be used to evaluate a dialogue model's incoherence, improve retrieval-based dialogue performance via a separate model, and improve generative dialogue modeling through external unlikelihood training.

8.1 Method

Our setting is persona-based dialogue generation. Following the notation introduced in (§3.3.3), persona-based dialogue generation is framed as next-utterance prediction, where a model is given a context $\mathbf{x} = (\mathbf{p}_1, \ldots, \mathbf{p}_m, \mathbf{u}_1, \ldots, \mathbf{u}_{t-1})$ containing sentences $\mathbf{p}_{1:m}$ that represent an agent's persona and the conversation history $\mathbf{u}_{1:t-1}$, and the output $\mathbf{y} = (y_1, \ldots, y_T)$ is the next utterance $\mathbf{u}_t$.

A *coherence error*, or contradiction, occurs when an agent produces an utterance that contradicts one of its previous utterances. Similarly, a *persona coherence error*, or persona contradiction, occurs when

an agent produces an utterance that contradicts a subset of its persona. A contradiction may be a clear logical contradiction, e.g. *I have a dog* vs. *I do not have a dog*, but in general is less clearly defined. In addition to logical contradictions, we interpret a coherence error as being two utterances not likely to be said by the same persona. For instance, "i'm looking forward to going to the basketball game this weekend!" vs. "i don't like attending sporting events", as well as "i'm a lawyer" vs. "i'm a doctor" would be viewed here as contradictions, although they are not strict logical inconsistencies. Similarly, a persona coherence error is interpreted here as an utterance which is not likely to be said given a persona described by a given set of persona sentences, in addition to logical contradictions. Despite these distinctions, we will informally refer to the issue of producing coherence errors as *logical incoherence*.

Natural Language Inference (NLI) assumes a dataset $\mathcal{D} = \{(\mathbf{s}_1, \mathbf{s}_2)_i, y_i\}_{i=1}^N$ which associates an input pair $(\mathbf{s}_1, \mathbf{s}_2)$ to one of three classes $y \in \{\text{entailment}, \text{neutral}, \text{contradiction}\}$. Each input item $\mathbf{s}_j$ comes from an input space $\mathcal{S}_j$, which in typical NLI tasks is the space of natural language sentences, i.e. s_j is a sequence of words $(w_1, \ldots, w_T)$ where each word w_k is from a vocabulary $\mathcal{V}$.

The input $(\mathbf{s}_1, \mathbf{s}_2)$ are referred to as the *premise* and *hypothesis*, respectively, and each label is interpreted as meaning the premise *entails* the hypothesis, the premise is *neutral* with respect to the hypothesis, or the premise *contradicts* the hypothesis. The problem is to learn a function $f_{\text{NLI}}(\mathbf{s}_1, \mathbf{s}_2) \to \{E, N, C\}$ which generalizes to new input pairs.

Identifying utterances which contradict previous utterances or an agent's persona can be reduced to natural language inference by assuming that contradictions are contained in a sentence pair. That is, given a persona $P_A = \{\mathbf{p}_1^A, \ldots, \mathbf{p}_m^A\}$ for agent A and a length-T dialogue $\mathbf{u}_1^A, \mathbf{u}_2^B, \ldots \mathbf{u}_{T-1}^A, \mathbf{u}_T^B$, it is assumed that a dialogue contradiction for agent A is contained in an utterance pair $(\mathbf{u}_i^A, \mathbf{u}_j^A)$, and a persona contradiction is contained in a pair $(\mathbf{u}_i^A, \mathbf{p}_k^A)$. Similarly, we assume that entailments and neutral interactions are contained in sentence pairs. We do not consider relationships which require more than two sentences to express. As we will detail below, under these assumptions we can use a natural language inference model f_{NLI} to identify entailing, neutral, or contradicting utterances.

8.1.1 Dialogue NLI dataset. The Dialogue NLI dataset consists of sentence pairs labeled as entailment (E), neutral (N), or contradiction (C). The Dialogue NLI dataset consists of $(\mathbf{u}_i, \mathbf{p}_j)$ and $(\mathbf{p}_i, \mathbf{p}_j)$ pairs from the Persona-Chat dataset (Zhang et al. 2018), but the dataset collection process is applicable to other persona-based dialogue datasets.

In order to determine labels for our dataset, we require human annotation of the utterances and persona sentences in Persona-Chat, as the original dataset does not contain this information. We perform such annotation by first associating a human-labeled *triple* (e_1, r, e_2) with each persona sentence, and a subset

Triple	Premise	Hypothesis	Triple	Label
(i, like_activity, chess)	i listen to a bit of everything . it helps me focus for my chess tournaments .	i like to play chess .	(i, like_activity, chess)	E
-	how are you today?	i drink espresso .	(i, like_drink, espresso)	N
(i, like_goto, spain)	i love spain so much , i been there 6 times .	i think i will retire in a few years .	(i, want_do, retire)	N
(i, have_vehicle, car)	my vehicle is older model car .	i have pets .	(i, have_pet, pets)	N
(i, dislike, cooking)	i really do not enjoy preparing food for myself .	i like to cook with food i grow in my garden .	(i, like_activity, cooking)	C
(i, physical_attribute, short)	height is missing from my stature .	i am 7 foot tall .	(i, physical_attribute, tall)	C
(i, have_family, 3 sister)	i have a brother and 3 sisters .	i have a brother and four sisters .	(i, have_family, 4 sister)	C

Table 15: Examples from the Dialogue NLI validation set.

of all the utterances, detailed detailed below. Each triple contains the main fact conveyed by a persona sentence, such as `(i, have_pet, dog)` for the persona sentence *I have a pet dog*, or a fact mentioned in an utterance, such as *No, but my dog sometimes does*.

Persona sentences and utterances are grouped by their triple (e.g. see Figure 19), and pairs $(\mathbf{u}, \mathbf{p})$ and $(\mathbf{p}, \mathbf{p})$ are defined as entailment, neutral, or contradiction based on their triple according to the criteria below. For examples and a summary, see Tables 15–16.

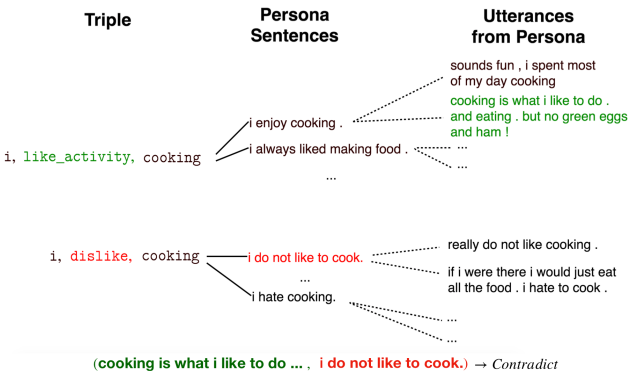


Fig. 19: Relating triples, persona sentences, and utterances to derive annotated sentence pairs. Shown here is a "relation swap" contradiction.

Entailment. Each unique pair of sentences that share the same triple are labeled as entailment.

Neutral. Neutral pairs are obtained with three different methods. First, a *miscellaneous utterance* is a $(\mathbf{u}, \mathbf{p})$ pair of which $\mathbf{u}$ is not associated with any triple. This includes greetings (*how are you today?*) and sentences unrelated to a persona sentence (*the weather is ok today*), so such utterances are assumed to be neutral with respect to persona sentences.

The second method, *persona pairing*, takes advantage of the fact that each ground-truth persona is typically neither redundant nor contradictory. A persona sentence pair $(\mathbf{p}, \mathbf{p}')$ is first selected from a

		Train		Valid		Test		Test-gold	
Data-type	**Label**	(u, p)	(p, p)	(u, p)	(p, p)	(u, p)	(p, p)	(u, p)	(p, p)
Matching Triple	E	43,000	57,000	5,000	500	4,500	900	3,712	615
Misc. Utterance	N	50,000	-	3,350	-	3,000	-	2,282	-
Persona Pairing	N	20,000	10,000	2,000	-	2,000	-	1,466	-
Relation Swap	N	20,000	-	150	-	400	-	260	-
Relation Swap	C	19,116	2,600	85	14	422	50	279	44
Entity Swap	C	47,194	31,200	4,069	832	3,400	828	2,246	591
Numerics	C	10,000	-	500	-	1,000	-	881	-
Dialogue NLI Overall		**310,110**		**16,500**		**16,500**		**12,376**	

Table 16: Dialogue NLI dataset statistics. (u, p) and (p, p) refer to (utterance, persona sentence) and (persona sentence, persona sentence) pairs, respectively. Numerics consist of (u, u) (u, p) and (p, p) pairs.

persona if $\mathbf{p}$ and $\mathbf{p}'$ do not share the same triple. Then each sentence associated with the same triple as $\mathbf{p}$ is paired with each sentence associated with the same triple as $\mathbf{p}'$.

Lastly, we specify *relation swaps* (r, r') for certain relations (see Appendix B.1.3.3) whose triples are assumed to represent independent facts, such as `have_vehicle` and `have_pet`. A sentence pair, whose first sentence is associated with a triple $(\cdot, r, \cdot)$ and whose second sentence has triple $(\cdot, r', \cdot)$, is labeled as neutral. See Table 15 for an example.

Contradiction. We obtain contradictions using three methods. See Figure 19 for an example. First, the *relation swap* method is used by specifying contradicting relation pairs (r, r') (see Appendix B.1.3.3), such as (`like_activity`, `dislike`), then pairing each sentence associated with the triple (e_1, r, e_2) with each sentence associated with (e_1, r', e_2).

Similarly, an *entity swap* consists of specifying relations, e.g., `physical_attribute`, that would yield a contradiction when the value of e_2 is changed to a different value e_2', e.g., `short` $\rightarrow$ `tall` (see Appendix B.1.3.4). Sentences associated with (e_1, r, e_2) are paired with sentences associated with (e_1, r, e_2').

Finally, a *numeric* contradiction is obtained by first selecting a sentence which contains a number that appears in the associated triple (see Table 15). A contradicting sentence is generated by replacing the sentence's numeric surface form with a different randomly sampled integer in its numeric or text form.

Triple annotation. Each persona sentence is annotated with a triple (e_1, r, e_2) using an Amazon Mechanical Turk task. We first define a schema consisting of $\langle category \rangle \langle relation \rangle \langle category \rangle$ rules, such as $\langle person \rangle have\_pet \langle animal \rangle$, where the relation comes from a fixed set of relation types $\mathcal{R}$, listed in Appendix B.1.3.2. Given a sentence, the annotator selects a relation r from a drop-down populated with the values in $\mathcal{R}$. The annotator then selects the categories and values of the entities e_1 and e_2 using drop-downs that are populated based on the schema rules. An optional drop-down contains numeric values for annotating entity quantities (e.g., 3 brothers). If selected, the numeric value is concatenated

to the front of the entity value. The annotator can alternatively input an out-of-schema entity value in a text-box. Using this method, each of the 10,832 persona sentences is annotated with a triple (e_1, r, e_2), where $r \in \mathcal{R}$, $e_1 \in \mathcal{E}_1$, and $e_2 \in \mathcal{E}_2$. Here $\mathcal{E}_1$ is the set of all annotated e_1 from the drop-downs or the text-box, and $\mathcal{E}_2$ is similarly defined.

Finally, *utterances* are associated with a triple as follows. Let $\mathbf{p}$ be a persona sentence with triple (e_1, r, e_2). We start with all utterances, U, from agents that have $\mathbf{p}$ in their persona. An utterance $\mathbf{u} \in U$ is then associated with the triple (e_1, r, e_2) and persona sentence $\mathbf{p}$ when e_2 is a sub-string of $\mathbf{u}$, or word similarity[14] $\text{sim}(\mathbf{u}, \mathbf{p}) \geq \tau$ is suitably large.

Statistics. Table 16 summarizes the dataset and its underlying data types. The label, triple, and data type are supplied as annotations for each sentence pair. We additionally create a gold-standard test set (*test-gold*) by crowdsourcing three label annotations for each example in the test set. We keep each test example for which two or more annotators agreed with its dataset label. All sentences in Dialogue NLI were generated by humans during the crowdsourced dialogue collection process of the Persona-Chat dataset (Zhang et al. 2018). The resulting sentence pairs are thus drawn from a natural dialogue domain that differs from existing NLI datasets, which are either drawn from different domains such as image captions or created using synthetic templates (Bowman et al. 2015; Demszky et al. 2018; Marelli et al. 2014; Poliak et al. 2018a; Wang et al. 2018; Williams et al. 2018).

8.1.2 Logical incoherence in dialogue.

We now return to our primary focus: using the dialogue task to study logical incoherence and how we can alleviate it. To this end, we propose methods that use Dialogue NLI to reduce dialogue incoherence in two different model classes: (a) retrieval-based models, and (b) autoregressive models, which are the focus of this thesis. In the experiments, we demonstrate how to use Dialogue NLI to quantify logical incoherence, and empirically evaluate the proposed methods.

Retrieval-based models. Given a persona $P = \{\mathbf{p}_1, \ldots, \mathbf{p}_m\}$, previous utterances $\mathbf{u}_{\leq t}$, and a set of candidate next-utterances U, a retrieval-based dialogue model outputs a ranked list of scores $s_1, s_2, \ldots, s_{|U|}$ corresponding to next-utterance candidates $\mathbf{u}_1, \mathbf{u}_2, ..., \mathbf{u}_{|U|}$. Thus a retrieval-based dialogue model is a function $f^{\text{dialogue}}(\mathbf{u}_{\leq t}, P, U) \rightarrow (s_1, s_2, \ldots, s_{|U|})$. We propose to use an NLI model to re-rank candidate utterances based on whether the candidate is predicted to contradict a persona sentence. If the NLI model predicts that a candidate contradicts a persona sentence, the candidate's score is penalized, with the penalty weighted by the NLI model's confidence[15] scaled by a constant.

[14] We use cosine similarity between the mean of TF-IDF weighted GloVe (Pennington et al. 2014) word vectors and set $\tau = 0.9$.

[15] In our experiments, the softmax output corresponding to the contradiction class from Dialogue NLI.

Specifically, a Dialogue NLI model $f^{\text{NLI}}(\mathbf{u}, \mathbf{p}) \rightarrow \{E, N, C\}$ is run on each $(\mathbf{u}_i, \mathbf{p}_j)$ pair, predicting a label $y_{i,j} \in \{E, N, C\}$ with confidence $c_{i,j}$. Each candidate receives a contradiction score:

$$s_i^{\text{contradict}} = \begin{cases} 0, & \text{if } y_{i,j} \neq C \ \forall \ \mathbf{p}_j \in P \\ \max\limits_{j:y_{i,j}=C} c_{i,j}, & \text{otherwise.} \end{cases}$$

That is, if the candidate $\mathbf{u}_i$ does not contradict any persona sentence $\mathbf{p}_j$ according to the NLI model, $s_i^{\text{contradict}}$ is zero. If $\mathbf{u}_i$ contradicts one or more persona sentences, $s_i^{\text{contradict}}$ is the highest confidence, $c_{i,j}$, out of the contradicting $(\mathbf{u}_i, \mathbf{p}_j)$.[16] New candidate scores are then computed as

$$s_i^{\text{re-rank}} = s_i - \lambda(s_1 - s_k)s_i^{\text{contradict}} \tag{37}$$

and the candidates are sorted according to $s^{\text{re-rank}}$. Hyper-parameters λ and k control the NLI model's influence in re-ranking. For example, if the top candidate has a contradiction score of 1.0, then with $\lambda = 1$, it will be moved to the k'th position in the ranking. $\lambda = 0$ corresponds to no re-ranking.

Autoregressive generative models. We propose to use external unlikelihood training (§7.1) to 'push down' the probability of generating incoherent responses. We use Dialogue NLI to form collections

$$\mathcal{D}^+ = \{(\mathbf{x}^{(i)}, \mathbf{y}^{(i)+})\} \quad \mathcal{D}^- = \{(\mathbf{x}^{(i)}, \mathbf{y}^{(i)-})\},$$

where $\mathcal{D}^+$ is coherent behavior, i.e. neutral or entailing data, and $\mathcal{D}^-$ is incoherent behavior, i.e. contradictions. Standard likelihood training can then be performed on coherent data $\mathcal{D}^+$, while the unlikelihood loss is applied to $\mathcal{D}^-$. Recalling (Equation 29), this yields the objective,

$$\mathcal{L}_{\text{ULE-ext}}(p_\theta, \mathcal{C}_{1:T}^{\text{identity}}, \mathbf{x}, \mathbf{y}^-, \mathbf{y}^+) = \mathcal{L}_{\text{MLE}}(p_\theta, \mathbf{x}, \mathbf{y}^+) + \alpha\mathcal{L}_{\text{ULE}}(\mathcal{C}_{1:T}^{\text{identity}}, \mathbf{x}, \mathbf{y}^-), \tag{38}$$

where we penalize each token in the target ($\mathcal{C}_t^{\text{identity}} = \{y_t^-\}$). Hence, the loss makes generating the contradicting sentences less likely. We consider two setups for $\mathcal{D}^+$ and $\mathcal{D}^-$ which we describe in the empirical evaluation (§8.2.3): a two utterance generation task, and a full dialogue generation task.

8.2 Empirical Evaluation

We first evaluate Dialogue NLI as a natural language inference task (8.2.1). Then we show how Dialogue NLI is used to measure incoherence in retrieval-based dialogue models and evaluate the proposed NLI re-ranking with automatic metrics and human judgements (8.2.2). Finally, we measure incoherence in a

[16] Future work could consider filtering previous-utterance contradictions $(\mathbf{u}_i, \mathbf{u}_j)$ as well.

Model	Valid	Test	Test-gold
ESIM	**86.31**	**88.20**	**92.45**
InferSent	85.82	85.68	89.96
InferSent SNLI	47.86	46.36	47.03
InferSent Hyp-only	55.98	57.19	51.52
Most Common Class	33.33	34.54	34.96
ESIM Gold Triples	99.52	99.46	99.69

Table 17: Dialogue NLI results.

Data Type	N	Accuracy
Matching Triple (p, p)	615	83.58
Matching Triple (u, p)	3,712	91.25
Misc. Utterance	2,282	96.85
Persona Pairing	1,466	94.48
Relation Swap (p, p)	44	79.55
Relation Swap (u, p)	539	80.71
Entity Swap (p, p)	591	93.40
Entity Swap (u, p)	2,246	92.43
Numerics	881	96.25

Table 18: ESIM accuracy by data type (test-gold).

state-of-the-art generative model, and demonstrate how using external unlikelihood along with Dialogue NLI can substantially reduce incoherence while maintaining language-modeling quality (8.2.3).

8.2.1 Experiment 1: NLI. Many NLI models can be categorized into sentence encoding methods $f_{\text{MLP}}(g_{\text{enc}}(\mathbf{s}_1), g_{\text{enc}}(\mathbf{s}_2))$, and attention-based methods of the form $f_{\text{MLP}}(g_{\text{attn}}(\mathbf{s}_1, \mathbf{s}_2))$ (Lan and Xu 2018). We thus choose and train representative models of each type which have achieved competitive performance on existing NLI benchmark datasets. For the sentence encoding method, we use InferSent (Conneau et al. 2017), which encodes a sentence using a bidirectional LSTM followed by max-pooling over the output states. As the attention-based method we use the enhanced sequential inference model (ESIM, (Chen et al. 2017a)), which computes an attention score for each word pair. We also report results from a model trained and evaluated using only the hypothesis sentence (InferSent Hyp-only) (Gururangan et al. 2018), a model trained on SNLI (Bowman et al. 2015) but evaluated on Dialogue NLI (InferSent SNLI), and a model that returns the most common class from the Dialogue NLI training set.

Table 17 shows the performance of the two NLI models and three baselines on the Dialogue NLI validation and test sets. The test performance of ESIM (88.2%) and InferSent (85.68%) is similar to the performance reported on the existing SNLI dataset (88.0% (Chen et al. 2017a) and 85.5% (Conneau et al. 2017), respectively), while the results on the Dialogue NLI gold test set (92.45%, 89.96%) are higher. As seen in Table 17, however, an InferSent model trained on SNLI performs poorly when evaluated on the proposed Dialogue NLI (47.03%). This is likely due to a mismatch in sentence distributions between SNLI, which is derived from image captions, and Dialogue NLI, whose sentences more closely resemble downstream dialogue applications. The hypothesis-only performance (51.52%) is lower than the hypothesis-only baseline for SNLI (69.00% (Poliak et al. 2018b)), indicating that information from both the utterance and persona sentence is necessary to achieve good performance on Dialogue NLI.

ESIM's reasonably strong performance on Dialogue NLI suggests that the model may be useful in a downstream task - a claim which we verify in the next experiment. However, there is also room for

Data-type	Example	Predicted	Actual
Matching Triple (p, p)	i am a hopeless bookworm. when i have some spare time i read.	Neutral	Entail
Matching Triple (u, p)	i am from italy. i love the early mornings. i like getting up bright and early.	Neutral	Entail
Misc. Utterance	i do not understand football or baseball. i am employed as an engineer.	Contradict	Neutral
Persona Pairing	i lift weights every chance i get. i work in a warehouse driving a forklift.	Entail	Neutral
Relation Swap (p, p)	canines make me shake with fear. i love dogs but hate cats.	Entail	Contradict
Relation Swap (u, p)	i am heavy into fitness although i am rather large. i do not like exercise or physical activity.	Entail	Contradict
Entity Swap (p, p)	hawaii is where i reside. i do not drive because i live in new york.	Neutral	Contradict
Entity Swap (u, p)	tell me it was vegan food please , that is all i eat. i eat ham.	Neutral	Contradict
Numerics	i have two part time jobs. i have 7 part time jobs.	Neutral	Contradict

Table 19: Example ESIM mispredictions by data type on Test Gold.

improvement. In particular, we report the performance of a model which takes the ground-truth triples as input instead of sentences. As shown in the last row of Table 17, each sentence's underlying triple contains sufficient information to achieve near-perfect accuracy (99.69%). We also show ESIM's accuracy by data type on test-gold in Table 18, along with example mispredictions in Table 19. The results suggest that the NLI model could be improved further.

8.2.2 Experiment 2: Logical incoherence in retrieval-based dialogue models. In this and the following experiments, we will show that logical incoherence happens often in state-of-the-art dialogue models. First, we focus on retrieval-based dialogue models, demonstrating how to use the Dialogue NLI dataset to measure incoherence and to evaluate the proposed NLI re-ranking.

Setup. As the retrieval-based dialogue model we train a key-value memory network (Zhang et al. 2018) on the Persona-Chat dataset, which uses persona sentences and the conversation prefix as context. This model achieved the best performance on Persona-Chat in Zhang et al. (2018). We train the model using ParlAI (Miller et al. 2017) on the `personachat:self_original` task, using the hyper-parameters given for the `KVMemnnAgent` in the ConvAI2 competition.

Evaluation sets. To measure incoherence, we form evaluation sets which contain next-utterances that are likely to yield contradictions or entailments between a generated utterance $\mathbf{u}$ with one or more of the agent's persona sentences $\{\mathbf{p}_1, \ldots, \mathbf{p}_m\}$. Each evaluation example is formed by first finding a next-

utterance $\mathbf{u}_t$ in the Persona-Chat validation set which has an associated triple (e_1, r, e_2) of interest, for instance $(\texttt{i}, \texttt{like\_music}, \texttt{country})$. If a sentence in the agent's persona P has triple (e_1, r, e_2), we form the validation example $(P, \mathbf{u}_{<t}, \mathbf{u}_t)$. Each example is associated with candidates U, consisting of the ground-truth utterance $\mathbf{u}_t$, 10 entailment candidates with the same triple as $\mathbf{u}_t$, 10 contradicting candidates with a different triple than that of $\mathbf{u}_t$, and 10 random candidates. The dialogue model must avoid ranking a contradicting candidate highly.

Specifically, suppose the ground-truth next-utterance $\mathbf{u}_t$ is associated with triple (e_1, r, e_2), for instance $(\texttt{i}, \texttt{have\_pet}, \texttt{dog})$. Entailment candidates are utterances $\mathbf{u}$ from the validation or training sets such that $\mathbf{u}$ is associated with triple (e_1, r, e_2). Since by construction a sentence in the profile also has triple (e_1, r, e_2), these candidates entail a profile sentence. A contradicting candidate is an utterance associated with a specified contradicting triple (e_1', r', e_2'), e.g., $(\texttt{i}, \texttt{not\_have}, \texttt{dog})$. We construct three evaluation sets, **Haves**, **Likes**, and **Attributes** using this process.

Metrics. We introduce variants of the ranking metric hits@k, called contradict@k and entail@k. **Contradict@k** measures the proportion of top-k candidates returned by the model which contradict candidates, averaged over examples. This measures the propensity of a model to highly rank contradictions. Contradiction@1 is the proportion of coherence errors made by the model. For this metric lower values are better, in contrast to hits@k. **Entail@k** measures the proportion of top-k candidates returned by the model which are entailment candidates, averaged over examples. Entailment candidates share the same triple as the ground-truth next utterance, so this metric rewards highly ranked candidates that convey similar meaning and logic to the ground-truth utterance. It is a more permissive version of hits@k.

Results. Table 20 shows results on the three evaluation sets before and after NLI re-ranking ($\lambda = 1.0, k = 10$). With the baseline model, logical incoherence appears frequently; for instance a contradicting next-utterance is top-ranked in **more than 30% of the examples** in the Haves evaluation set. The NLI re-ranking improves all three metrics on all the evaluation sets. Dialogue performance improves, as measured by hits@1. The NLI re-ranking substantially reduces the number of contradicting utterances, and increases the number of utterances which entail a profile sentence, as seen in the contradict@1 and entail@1 scores.

To confirm that these improvements align with human judgments of quality, we perform human evaluation, where consistency is judged by human annotators in an interactive persona-based dialogue setting.[17] Table 21 shows the human evaluation results. According to human judgments, the baseline model contradicts itself roughly as often as it is logically consistent (0.25 vs. 0.27), further confirming that logical

[17] See Appendix B.1.3 for details on the evaluation setup.

	Haves			Likes			Attributes		
	Original	Rerank	Δ	Original	Rerank	Δ	Original	Rerank	Δ
Hits@1 ↑	30.2	**37.3**	**+26%**	16.9	**18.7**	**+11%**	35.2	**36.4**	**+3%**
Contra@1 ↓	32.5	**8.96**	**−72%**	17.6	**4.1**	**−77%**	8.0	**5.7**	**−29%**
Entail@1 ↑	55.2	**74.6**	**+35%**	77.9	**90.6**	**+16%**	87.5	**88.6**	**+1%**

Table 20: Incoherence and effects of NLI re-ranking in a retrieval-based dialogue model.

	Overall Score ↑		% Consistent ↑		% Contradiction ↓	
	Raw	Calibrated	Raw	Calibrated	Raw	Calibrated
KV-Mem	2.11 (1.12)	2.21 (0.26)	0.24	0.27 (0.07)	0.23	0.25 (0.08)
KV-Mem + NLI	**2.34 (1.21)**	**2.38 (0.26)**	**0.28**	**0.35 (0.08)**	**0.19**	**0.16 (0.06)**

Table 21: Human evaluation results, reported as mean (std. dev).

incoherence is an issue in neural dialogue models. The natural language inference re-ranking improves all of the metrics, notably the fine-grained consistency score (0.27 vs. 0.35) and contradiction score (0.25 vs. 0.16). The results are consistent with the conclusions drawn using the evaluation sets.

8.2.3 Experiment 3: Logical incoherence in generative dialogue models. We now return to the model class which is the focus of this thesis, autoregressive generative models. We quantify a state-of-the-art model's incoherence on two tasks, and show that unlikelihood training can improve coherence.

For the first task, called **two utterance generation**, we adapt the Dialogue NLI dataset by using entailing and neutral training sentence pairs as plausible positive utterances, and contradicting pairs as negatives. That is, if a pair $(\mathbf{s}_1, \mathbf{s}_2)$ from Dialogue NLI has label E or N, the example $(\mathbf{x}, \mathbf{y}) = (\mathbf{s}_1, \mathbf{s}_2)$ is added to $\mathcal{D}^+$, otherwise (label C) it is added to $\mathcal{D}^-$. We consider two types of entailment: entailing sentence pairs that appear together in a dialogue in the original Persona-Chat dataset and are therefore natural ('entailment'), and those that only entail via their triple relations ('triple-entailment'). The latter are more challenging, noisier targets. Evaluation is performed by measuring the test set perplexity over the four target label types, where contradictions should have relatively higher perplexity. We also evaluate a selection accuracy task, where for each test example there are two candidate responses: positive and negative (contradicting). The candidate response with the lowest perplexity is considered to be the model's selection, and we measure the selection success rate.

For the second, **full dialogue** task, we align Dialogue NLI examples with Persona-Chat dialogues to provide positive and negative continuations of the dialogue. An example $(\mathbf{x}, \mathbf{y})$ consists of a dialogue history $\mathbf{x} = \{\mathbf{p}_1, \ldots, \mathbf{p}_k, \mathbf{u}_1, \ldots, \mathbf{u}_{t-1}\}$ and utterance $\mathbf{y} = \mathbf{s}_2$, where $(\mathbf{s}_1, \mathbf{s}_2)$ is a sentence pair from Dialogue NLI, and at least one sentence in $\mathbf{x}$ has the same relation triple as $\mathbf{s}_1$. When the pair $(\mathbf{s}_1, \mathbf{s}_2)$ is labeled as E or N in Dialogue NLI, the example $(\mathbf{x}, \mathbf{y})$ is added to $\mathcal{D}^+$; otherwise it is added to $\mathcal{D}^-$.

	Selection Accuracy			Perplexity				
Data + Model	Entail	Tr-Entail	Neutral	Entail	Tr-Entail	Neutral	Contradict	ConvAI2
MLE Baseline	72%	41%	18%	8.54	17.5	36.7	12.5	11.4
UL (Dialogue NLI)	96%	85%	78%	9.1	26.6	39.4	248.9	11.9

Table 22: Test evaluation on the Dialogue NLI two utterance generation task, comparing standard MLE training with unlikelihood training (external unlikelihood with Dialogue NLI). Results are partitioned according to whether the premise and positive candidate are entailing, triple-entailing, or neutral. Selection Accuracy measures how often the model assigns lower perplexity to the positive candidate than to the negative candidate in the pair. The MLE model's perplexity is *lower* on contradicting utterances than on neutral or triple-entailing utterances, showing partial failure at the coherence task. Unlikelihood training yields large improvements on all coherence metrics, while minimally increasing overall perplexity.

Premise	Hypothesis	$\mathcal{L}_{\text{MLE}}$ PPL	$\mathcal{L}_{\text{UL}}$ PPL
Yes, I love watching baseball and basketball. I do not like running though.	(C) I love running.	25.5	226.9
	(E) I despise running.	29.9	9.4
Yes, I love watching baseball and basketball. I do like running though.	(E) I love running.	26.2	3.1
	(C) I despise running.	42.8	247.1
We did too but working in real estate for 12 years . sucked up a lot of time	(E) I have been working as a real estate agent for the past 12 years.	3.9	3.8
	(C) We did too but working in real estate for fifteen years sucked up a lot of time.	3.1	17.6

Table 23: Example perplexities of a baseline maximum likelihood model ($\mathcal{L}_{\text{MLE}}$) and an unlikelihood trained model ($\mathcal{L}_{\text{UL}}$) when generating the hypotheses, given the premise. The MLE-trained model assigns high probability (low perplexity) to contradictory generations, while unlikelihood does not.

We employ a large pre-trained seq2seq transformer as our base model.[18] We fine-tune using maximum-likelihood on next-utterance examples from the ConvAI2 persona-based dialogue dataset (Zhang et al. 2018) as well as examples from $\mathcal{D}^+$, and using unlikelihood on examples from $\mathcal{D}^-$ (Equation 38).

	Selection Accuracy (vs. Contradict)		Perplexity			
Data + Model	Triple-Entail	Neutral	Triple-Entail	Neutral	Contradict	ConvAI2
MLE Baseline	66.5%	36.8%	23.3	45.1	35.9	11.4
UL (Dialogue NLI)	89.0%	69.8%	21.5	40.3	63.5	11.8

Table 24: Test evaluation on the Full Dialogue NLI generation task. External unlikelihood training with Dialogue NLI improves coherence metrics compared to likelihood (MLE) training. The unlikelihood-trained model assigns higher probability (lower perplexity) to triple-entailing or neutral candidates than to contradicting candidates, with higher selection accuracy for coherent labels.

Results. Our baseline model ($\mathcal{L}_{\text{MLE}}$) obtains a perplexity of 11.4, in line with state-of-the-art systems on this task (Lewis et al. 2019). Unfortunately, despite being good on such standard metrics, our baseline models fail at the coherence task. As seen in Table 22 for the two utterance task, the perplexity of contradicting utterances (12.5) is on average *lower* than for neutral (36.7) or triple-entailing utterances

[18] See Appendix B.1.3 for pretraining details.

(17.5), although it is higher than entailing utterances. We believe this is due to contradicting utterances having high word overlap with the premise utterance, coupled with an inability to judge incoherence. Viewed as a selection task between utterances, picking the utterance with the lowest perplexity, this means the selection rates of non-contradicting utterances are very low, e.g. picking neutral utterances over contradicting utterances only 18% of the time. Even fully entailing utterances are only picked 73% of the time. Similar results are found on the full dialogue task as well; see Table 24.

Unlikelihood training brings large improvements in coherence metrics, whilst minimally impacting overall dialogue perplexity. After applying unlikelihood, perplexity for contradicting utterances has a clear signature, with very large average values compared to entailing or neutral utterances, e.g. 248.9 vs. 9.1 for contradict vs. entail on the two utterance task. This converts to corresponding large increases in selection accuracy across all types on both tasks, e.g., an increase from 18% to 78% on neutral statements on the two utterance task, and from 37.4% to 69.8% on the full dialogue task.

Some example model predictions are given in Figure 23, comparing the MLE baseline and unlikelihood model perplexities of generating the given hypotheses. The likelihood model cannot differentiate between contradicting and entailing statements easily, while there are large perplexity differences for the unlikelihood model in these cases.

8.3 Discussion

In this section, we studied *logical incoherence* in neural text generation. To do so, we constructed a dataset called Dialogue NLI which allowed us to measure incoherence on a persona-based dialogue modeling task. We demonstrated that logical incoherence occurs to a non-trivial degree in both retrieval-based and generative dialogue models. Even a large, state-of-the-art generative model frequently assigned higher probabilities to contradicting utterances than to coherent ones. Furthermore, we showed two ways that Dialogue NLI can be used to improve logical incoherence: (1) by re-ranking via a reduction to natural language inference; and (2) by penalizing contradictions with unlikelihood training.

9 Discussion and Future Directions

In this part of the thesis, we focused on text generation, studying downsides of autoregressive models trained with maximum-likelihood. We primarily characterized these downsides using three symptoms that are observed in generated text: non-termination, repetition, and logical incoherence, collectively termed *text degeneration*. Our investigations of inconsistency (§6), unlikelihood training (§7), and dialogue NLI (§8) provide new theory, algorithms, and data for analyzing and measuring text degeneration, as well as building improved text generators. As with most scientific endeavors, our investigation suggests future avenues of inquiry.

In terms of inconsistency, future work may seek an understanding of why, or under what conditions, maximum-likelihood results in a model whose induced distribution is inconsistent under an incomplete decoding algorithm. This might involve studying the effects of scale, identifying task properties that correlate with the degree of non-termination, or developing adversarial context distributions that result in inconsistency. A drawback of our proposed consistent sampling and self-terminating methods is that they are specifically designed for the consistency problem; thus another future direction is developing generic model, learning, and inference methods that guarantee consistency.

Unlikelihood training allows for controlled reduction of unwanted properties in generated sequences by specifying negative candidates and token-dependent weights. Unlikelihood training worked best when the candidates and weights were designed specifically for a property, such as repetition or token frequency. This can be seen as a benefit, allowing future exploration of more property-specific penalties that improve downstream task performance. It can also be seen as a limitation; an appeal to simplicity would strive for generic learning algorithms that do not require property-specific design choices. One step up in generality is to directly optimize a task cost – a function which assigns a quality score to (a collection of) sequences. In unlikelihood training we manually specify behaviors or properties that we do not want in the model's generations via negative candidates, whereas in task cost minimization the undesirable behaviors are implicitly defined by the task cost. Subsequent findings in Welleck and Cho (2020) suggest that text degeneration can be eliminated *as a byproduct of* minimizing a task cost, without explicitly specifying the degenerate properties that we want to eliminate.

More generally, unlikelihood training is part of a larger question of how to teach machine learning algorithms what *not* to do. Doing so may be crucial for preventing generation models from producing offensive, biased, or incorrect content (Gehman et al. 2020; Brown et al. 2020). Finally, furthering our understanding of *why* repetition occurs in MLE-trained autoregressive models is an important direction.

Our work on Dialogue NLI and its use with external unlikelihood training is a first step towards quantifying and addressing logical incoherence in neural generation models. A key limitation of Dialogue NLI is that it is specific to the persona-based dialogue generation task (and moreover, specific to the Persona-Chat dataset). Developing ways to measure and improve logical coherence in general settings is an interesting area of future work. On the other hand, the external unlikelihood approach is generic, in the sense that it requires only specifying positive and negative pairs. This opens up many possibilities for leveraging supervised datasets other than Dialogue NLI to improve text generation, e.g. by correcting causal or commonsense reasoning errors (Zellers et al. 2019; Qin et al. 2019).

Despite its drawbacks, maximum-likelihood estimation is a simple and scalable method for training autoregressive language models that is used in many state-of-the-art systems. This latter property, scalability, is likely to be of continued interest as the size of neural sequence models increases (Radford et al. 2018; Brown et al. 2020; Adiwardana et al. 2020; Roller et al. 2020; Lepikhin et al. 2020; Henighan et al. 2020). Although scaling the model has not yet succeeded in eliminating text degeneration, its ultimate effect on text degeneration might only be seen once even larger models are trained on even larger amounts of data. Alternatively, it is an open question whether we can characterize scaling effects using smaller, controlled, environments. Looking ahead, state-of-the-art neural text generators will either be trained with maximum likelihood, or they will not. Regardless of the outcome, a deeper scientific understanding of its properties and its potential alternatives will remain valuable into the future.

Part III

Non-Monotonic Generation

10 Non-Monotonic Generation

In this part of the thesis, we will depart from our preceding investigation into conventional text generation methods in two ways. First, we will not only consider generating text, but also multisets and tree-structured objects. Second, we will remove the restriction of a fixed generation order, thus requiring an alternative to the fixed-order maximum-likelihood objective that we used throughout Part II. Our investigation is centered around a new learning framework, called *non-monotonic generation*, that yields models capable of selecting input-dependent generation orders. This flexibility is natural for set- and tree-structured objects, which lack an inherent order. For text, the selected orders induce an interpretable latent structure and let us study whether the canonical left-to-right order is optimal for learning.

Non-monotonic generation is developed from the perspective of *imitation learning*, where we treat a generation model as a *policy* π_θ that navigates a state space and is tasked with imitating an *oracle* π_* that has knowledge of good actions to take in each state. We will provide background on this imitation learning perspective (§11), present the non-monotonic generation framework (§12), then adapt and apply it to generating multisets (§13), parse trees (§14), and text (§15).

11 Background

In this section we provide an overview of the *imitation learning* view of sequential structured prediction (Daumé III et al. 2009; Ross et al. 2011; Vlachos et al. 2017). After introducing notation, we will see how this perspective generalizes maximum-likelihood training. This will provide background for our subsequent investigation of non-monotonic generation.

In the imitation learning view of sequential structured prediction, a generation model is a *policy* π_θ that navigates a state space and is tasked with imitating an *oracle* π_* that has knowledge of good actions to take in each state. For example, when viewing standard autoregressive text generation in this way, the state consists of previously generated tokens along with an input, an action consists of choosing the next token to generate, and the oracle, which has access to the ground-truth sequence, places probability mass only on the true next-token.

The policy is trained by sampling states from a *roll-in* policy, and minimizing a cost at each sampled state. At first glance, this framework may appear to amount to a change in notation, but varying the roll-in, oracle, cost, as well as how we define states and actions, can lead to new formulations of sequential generation. To see this, we first present these components of the framework more precisely.

11.1 Imitation Learning for Structured Prediction

Let $\pi(a|\mathbf{s})$ denote a *policy*, which is a mapping from a *state* $\mathbf{s} \in \mathcal{S}$ to a distribution over *actions* $a \in \mathcal{A}$. A sequential generation involves a *trajectory* $\tau = (\mathbf{s}_0, a_1, \mathbf{s}_1, \ldots, a_T, \mathbf{s}_T)$, obtained by beginning in a start state $\mathbf{s}_0 \sim p(\mathbf{s}_0)$, taking actions $a_t \sim \pi(a|\mathbf{s}_t)$, and transitioning to new states $\mathbf{s}_{t+1} \sim p(\mathbf{s}|\mathbf{s}_t, a_t)$. The final generation is extracted from the trajectory, denoted $\hat{\mathbf{y}} = g(\tau)$. We will assume there is a ground-truth generation $\mathbf{y}$, which is formalized using an additional initial state, $(\mathbf{s}_0, \bar{\mathbf{s}}_0) \sim p(\mathbf{s}_0)$. Continuing our text generation example above, the initial state distribution is represented by a dataset, meaning $\mathbf{s}_0, \bar{\mathbf{s}}_0$ is a sampled $(\mathbf{x}, \mathbf{y})$ example, each state transition $\mathbf{s}_{t+1} \sim p(\mathbf{s}|\mathbf{s}_t, a_t)$ is deterministic, returning $\mathbf{s}_{t+1} = (\hat{y}_{<t+1}, \mathbf{x})$ where $a_t = \hat{y}_t$, and $g(\tau)$ returns the generated tokens $(\hat{y}_1, \ldots, \hat{y}_T)$ that are contained in $\mathbf{s}_T$.

In addition to the learned policy $\pi_\theta(a|\mathbf{s})$, the imitation learning framework considers an *oracle* policy $\pi_*(a|\bar{\mathbf{s}})$ and a *roll-in* policy π^{in}. The oracle policy has access to additional information in its state, $\bar{\mathbf{s}}$, from which it can derive supervision. In our text generation example, the oracle's state contains the ground truth, $\bar{\mathbf{s}}_t = (\mathbf{s}_t, \mathbf{y})$. The roll-in policy is used to obtain states to train on. In our example, selecting ground-truth next-tokens ($\hat{y}_t = y_t^*$) means the roll-in policy is the oracle policy (π_*^{in}), while sampling next-tokens from the learned policy ($\hat{y}_t \sim \pi_\theta(\cdot|\mathbf{s}_t)$) means the roll-in policy is the learned policy (π_θ^{in}).

Ideally, we want to minimize a *task cost* $C(\hat{\mathbf{y}}, \mathbf{y})$, such as exact match. However, the task cost only provides a single scalar of supervision for an entire trajectory, and lacks feedback about whether there are equally valid alternative trajectories or about which alternative actions are bad. Moreover, in some tasks, such as open-ended text generation, a good task cost is difficult to specify. Imitation learning instead minimizes a surrogate loss between the learned and oracle policy at each state, $\mathcal{L}_t\left(\pi_\theta(\cdot|\mathbf{s}_t), \pi_*(\cdot|\bar{\mathbf{s}}_t)\right)$, such as KL-divergence,

$$\mathcal{L}_t = -\sum_{a \in \mathcal{A}} \log \pi_*(a|\bar{\mathbf{s}}_t) \log\left(\frac{\pi_\theta(a|\mathbf{s}_t)}{\pi_*(a|\bar{\mathbf{s}}_t)}\right). \tag{39}$$

This loss is further generalized by specifying the *cost-to-go*, $Q(\mathbf{s}_t, a)$, of each possible action. An additional *roll-out* policy π^{out} can estimate the cost-to-go by taking action a then completing a trajectory, meaning $Q(\mathbf{s}_t, a) = C(g(\tau), \mathbf{y})$, where τ consists of reaching $\mathbf{s}_t$ with the roll-in policy, then taking action a, then sampling actions from the roll-out policy. One then defines $\mathcal{L}_t$ using KL-divergence with a policy $\pi(a|\mathbf{s}_t) \propto -Q(\mathbf{s}_t, a)$. The loss (39) is the special case $Q(\mathbf{s}_t, a) = \pi_*(a|\bar{\mathbf{s}}_t)$. The roll-out provides dense supervision involving the task-cost, and incorporates the learned policy's predictions when π^{out} is π_θ.

In summary, the general imitation learning for structured prediction framework solves,

$$\arg\min_{\theta} \mathbb{E}_{\mathbf{s}_0,\bar{\mathbf{s}}_0\sim p(\mathbf{s}_0)}\mathbb{E}_{\tau\sim\pi^{\text{in}}} \sum_{t=1}^{T} \mathcal{L}_t(\pi_\theta, \pi^{\text{out}}, \mathbf{s}_t, \bar{\mathbf{s}}_t). \tag{40}$$

By varying the choice of $\pi^{\text{in}}, \pi^{\text{out}}$, and $\mathcal{L}$ one obtains different variants of imitation learning algorithms, such as Searn (Daumé III et al. 2009), DAgger (Ross et al. 2011), v-DAgger (Vlachos and Clark 2014), AggreVaTe (Ross and Bagnell 2014) or LOLS (Chang et al. 2015). Defining the state space, action space, and oracle π_* yields different applications (He et al. 2012a,b, 2014; Goldberg and Nivre 2012, 2013; Goodman et al. 2016; Leblond et al. 2018). We now discuss how maximum-likelihood estimation is a special case.

Maximum likelihood estimation. Let us see how training a fixed-order, autoregressive model with maximum likelihood estimation is a special case of (40). Consider training a translation model on an example $(\mathbf{x}, \mathbf{y}) \sim \mathcal{D}$. As in the example we discussed above, define states as the tokens predicted so far along with the input sentence, $\mathbf{s}_t = (\hat{y}_{<t}, \mathbf{x})$, actions as next-tokens, $a_t = \hat{y}_t$, and let $g(\tau)$ return the predicted tokens in the final state, $\hat{y}_1, \ldots, \hat{y}_T$. The initial state distribution is represented by the dataset, meaning $\mathbf{s}_0, \bar{\mathbf{s}}_0 \sim p(\mathbf{s}_0)$ corresponds to sampling $(\mathbf{x}, \mathbf{y}) \sim \mathcal{D}$ and setting $\mathbf{s}_0 = (\emptyset, \mathbf{x})$, $\bar{\mathbf{s}}_0 = (\mathbf{s}_0, \mathbf{y})$. Define an oracle that at each step assigns all of its probability mass to the ground truth next-token,

$$\pi_*^{\text{MLE}}(y|\bar{\mathbf{s}}_t) = \begin{cases} 1 & y = y_t \\ 0 & \text{otherwise} \end{cases}, \tag{41}$$

where $\bar{\mathbf{s}}_t = (\mathbf{s}_t, \mathbf{y})$. Maximum-likelihood only trains on ground-truth histories $y_{<t}$, meaning that the roll-in policy is the oracle, $\pi^{\text{in}} = \pi_*^{\text{MLE}}$. The loss is KL-divergence without rollouts (39),

$$\begin{aligned}
\mathcal{L}_t(\pi_\theta, \pi_*^{\text{MLE}}, \mathbf{s}_t, \bar{\mathbf{s}}_t) &= \mathrm{D}_{\text{KL}}\left(\pi_*^{\text{MLE}}(\cdot|\bar{\mathbf{s}}_t) \| \pi(\cdot|\mathbf{s}_t)\right) && (42)\\
&= -\sum_{a_t\in\mathcal{A}} \pi_*^{\text{MLE}}(a_t|\bar{\mathbf{s}}_t) \log\left(\frac{\pi(a_t|\mathbf{s}_t)}{\pi_*^{\text{MLE}}(a_t|\bar{\mathbf{s}}_t)}\right) && (43)\\
&\propto -\log \pi(a_t|\mathbf{s}_t) && (44)\\
&\equiv -\log p_\theta(y_t|y_{<t}, \mathbf{x}), && (45)
\end{aligned}$$

where the proportionality discards terms that do not depend on θ. In summary, learning a fixed-order autoregressive model with maximum likelihood (§4.2.1), is a special case of imitation learning (40),

$$\arg\min_{\theta} \mathbb{E}_{(\mathbf{x},\mathbf{y})\sim\mathcal{D}}\mathbb{E}_{\tau\sim\pi_*^{\text{MLE}}} \sum_{t=1}^{T} \left[\mathcal{L}_t(\pi_\theta, \pi_*^{\text{MLE}}, \mathbf{s}_t, \bar{\mathbf{s}}_t)\right], \tag{46}$$

with the loss and states defined as above. From this perspective, one can analyze the behavior of maximum likelihood and develop alternatives. For instance, Ross et al. (2011) show that due to using oracle roll-in, maximum-likelihood (alternatively known as 'behavioral cloning') suffers more from compounding errors than a variant that uses a roll-in which is a mixture of the learned policy and the oracle, called DAgger; see Ross et al. (2011); Chang et al. (2015) for details. In the subsequent sections we will develop alternatives to the MLE oracle, and empirically evaluate the effect of roll-in choice.

12 Non-Monotonic Generation

Now we will introduce the general framework that we will apply to non-monotonic generation of sets, trees, and text in the subsequent sections. This section presents the framework at an abstract level, with the hope that a reader can apply it to problems beyond those in this thesis; subsequent sections will show concrete applications to generating structured objects without requiring a specified generation order. The framework can be seen as a special case of imitation learning for structured prediction that includes the notion of a set of *valid actions* at each state.

Valid actions. Let $\pi_\theta(a|\mathbf{s})$ be a policy, and suppose $\tau = (\mathbf{s}_0, a_1, \mathbf{s}_1, \ldots, a_T, \mathbf{s}_T) \sim \pi^{\text{in}}$. Our non-monotonic generation framework requires defining a set of *valid actions* $\mathcal{A}_t \subseteq \mathcal{A}$ at each state $\mathbf{s}_t \in \tau$. An action is *valid* if taking it can lead to a correct structured output; that is, there exists a continuation $\tau_{t+1:T}$ such that the overall trajectory $\tau = (\tau_{1:t-1}, a, \tau_{t+1:T})$ receives zero task cost, $C(g(\tau), \mathbf{y}) = 0$. Defining the valid actions is application-specific, so we will see concrete examples in the subsequent sections.

Oracles. We define an *oracle policy* as any policy that only assigns probability to valid actions,

$$\pi_*(a|\bar{\mathbf{s}}_t) \propto \begin{cases} p_a & a \in \mathcal{A}_t \\ 0 & \text{otherwise,} \end{cases} \tag{47}$$

where $\bar{\mathbf{s}}_t$ contains $\mathcal{A}_t$. We will consider three different oracles, though many other variants are possible. The first *deterministically* selects a single valid action,

$$\pi_*^{\text{det}}(a|\bar{\mathbf{s}}_t) = \begin{cases} 1 & a = \sigma(\bar{\mathbf{s}}_t) \\ 0 & \text{otherwise,} \end{cases} \tag{48}$$

where the selected action $\sigma(\bar{\mathbf{s}}_t) \in \mathcal{A}_t$ is a deterministic function of the oracle's state. The deterministic oracle π_*^{det} ignores alternative valid actions, and hence alternative valid trajectories, and requires specifying the function $\sigma(\cdot)$. At the other extreme, the *uniform* oracle treats all valid actions as equally

likely,

$$\pi_*^{\text{uniform}}(a|\bar{\mathbf{s}}_t) \propto \begin{cases} \frac{1}{|\mathcal{A}_t|} & a \in \mathcal{A}_t \\ 0 & \text{otherwise.} \end{cases} \tag{49}$$

An issue with the uniform oracle is that it does not prefer any specific subset of the valid actions, making it difficult for a parameterized policy to imitate. This gap between the learned policy and the oracle has been noticed as a factor behind learning difficulty (He et al. 2012a). Motivated by He et al. (2012a), we define a *coaching* oracle that weights each valid action according to the learned policy,

$$\pi_*^{\text{coaching}}(a|\bar{\mathbf{s}}_t) \propto \pi_*^{\text{uniform}}(a|\bar{\mathbf{s}}_t)\pi_\theta(a|\mathbf{s}_t). \tag{50}$$

Intuitively, the coaching oracle favors valid actions that are preferred by the current learned policy, and reinforces these preferences. The coaching oracle is closer to the learned policy than the uniform oracle,[19]

$$\mathrm{D}_{\mathrm{KL}}\left(\pi_*^{\text{coaching}} \| \pi_\theta\right) \leq \mathrm{D}_{\mathrm{KL}}\left(\pi_*^{\text{uniform}} \| \pi_\theta\right). \tag{51}$$

Thus we expect the coaching oracle to be easier to imitate, a claim that we evaluate empirically.

Roll-in. As the roll-in policy, we use either the learned policy, π_θ, the oracle π_*, or a mixture $z_t\pi_\theta + (1 - z_t)\pi_*$ where $z_t \sim \text{Bernoulli}(\beta_t)$ stochastically selects the policy to use at step t. In our applications, we will treat the roll-in policy as a hyper-parameter and investigate the choice of roll-in empirically.

Roll-out. Since our applications involve large neural models, performing explicit rollouts to estimate the cost-to-go of each action is prohibitively expensive. Thus in our framework we directly set the cost-to-go using the oracle probability, $Q(\mathbf{s}_t, a) = -\pi_*(a|\bar{\mathbf{s}}_t)$. This corresponds to doing an explicit oracle roll-out for each action, assuming a task cost that assigns maximal cost to non-optimal trajectories, and assigns cost to optimal trajectories proportional to the oracle's action probabilities. We leave incorporating explicit roll-outs into large neural model training for future work.

Loss. Finally, as our loss we use the KL-divergence loss $\mathcal{L}_t$ (Equation 39).

In summary, our non-monotonic framework optimizes the following imitation learning objective,

$$\arg\min_\theta \mathbb{E}_{\mathbf{s}_0, \bar{\mathbf{s}}_0 \sim p(\mathbf{s}_0)} \mathbb{E}_{\tau \sim \pi^{\text{in}}} \sum_{t=1}^{T} \mathrm{D}_{\mathrm{KL}}\left(\pi_*(\cdot|\bar{\mathbf{s}}_t) \| \pi_\theta(\cdot|\mathbf{s}_t)\right), \tag{52}$$

[19] Proof in Appendix A.2.

where we assume $\bar{\mathbf{s}}_t$ contains a valid action set $\mathcal{A}_t$, and the oracle is defined as above. We will now apply our framework to generating structured objects without requiring a pre-specified order, beginning with multisets.

13 Multisets: Multiset Prediction

In this section, we show how the non-monotonic generation framework that we presented in §12 is used for conditionally generating multisets[20], which we call *multiset prediction*. Multiset prediction appears in a variety of contexts. For instance, in the context of high-energy physics, one of the important problems in a particle physics data analysis is to count how many physics objects, such as electrons, muons, photons, taus, and jets, are in a collision event (Ehrenfeld et al. 2011). In computer vision, object counting and automatic alt-text can be framed as multiset prediction (Lempitsky and Zisserman 2010; Welleck et al. 2017). Our motivating application in this section will be multiple object classification, which we introduced in §3.1, and is illustrated in Figure 20.

In multiset prediction, there is no predefined order among the items in each target multiset, and each item can appear more than once. These properties make the problem of multiset prediction different from other well-studied problems. It is different from sequence prediction, because there is no known order among the items. It is not a ranking problem, since each item may appear more than once. It cannot be transformed into classification, because the number of possible multisets grows exponentially with respect to the maximum multiset size.

Fig. 20: Multiple object classification consists of mapping an image to a multiset of class labels. In this example, any ordering of the digits $\{0, 0, 0, 0, 0, 1, 1, 2, 2, 3, 5, 5, 5, 6, 6, 6, 7, 7, 9, 9\}$ is a valid labeling.

We propose a method for learning policies that sequentially generate multisets, without imposing a particular generation order during learning. Specifically, we use the non-monotonic generation framework (§12) with the valid actions equal to the remaining unpredicted items in the target multiset. We use the uniform oracle (Equation 49), which teaches the learned policy to generate any element of the target multiset that has not yet been generated, irrespective of order. These settings yield a simplified form of the non-monotonic generation objective (Equation 52), which we call the *multiset loss*.

We compare the multiset loss against an extensive set of baselines, including a sequential loss with an arbitrary ordering function, a sequential loss with an input-dependent ordering function, and an aggregated distribution matching loss and its one-step variant. We also test policy gradient, as was done in Welleck et al. (2017) for multiset prediction. Our evaluation is conducted on two sets of datasets with varying difficulties and properties. We find that the multiset loss outperforms all of the baselines.

[20] A set that allows multiple instances, e.g. $\{x, y, x\}$.

13.1 Method

A multiset prediction problem is a generalization of classification, where a target is not a single class but a multiset of classes. The goal is to find a mapping from an input x to a multiset $\mathcal{Y} = \{y_1, \ldots, y_{|\mathcal{Y}|}\}$, where $y_k \in \mathcal{C}$. Some of the core properties of multiset prediction are: (1) the input x is an arbitrary vector, (2) there is no predefined order among the items y_i in the target multiset $\mathcal{Y}$, (3) the size of $\mathcal{Y}$ may vary depending on the input x, and (4) each item in the class set $\mathcal{C}$ may appear more than once in $\mathcal{Y}$. Formally, $\mathcal{Y}$ is a multiset $\mathcal{Y} = (\mu, \mathcal{C})$, where $\mu : \mathcal{C} \to \mathbb{N}$ gives the number of occurrences of each class $c \in \mathcal{C}$ in the multiset.

We adapt the non-monotonic generation framework (§12) for multiset prediction. The action space consists of the class set $\mathcal{C}$, meaning an action a_t is a class label y_t, and a state is the previously predicted classes along with the input, $\mathbf{s}_t = (\hat{y}_{<t}, x)$. Thus the learned policy is of the form $\pi_\theta(y_t|\hat{y}_{<t}, x)$.

Given an example $(x, \mathcal{Y})$ sampled from a dataset, we generate a trajectory $\tau = (\hat{y}_1, \ldots, \hat{y}_T)$ from a roll-in policy π^{in}, stopping when either all the items in the target multiset have been predicted or a predefined maximum number of steps have passed. Starting with $\mathcal{A}_0 \leftarrow \mathcal{Y}$, we define the valid actions at time t as the multiset of labels that remain to be predicted after the first $t-1$ predictions by the roll-in policy,

$$\mathcal{A}_t \leftarrow \mathcal{A}_{t-1} \setminus \{\hat{y}_{t-1}\}. \tag{53}$$

We use the uniform oracle, which puts equal mass on all unpredicted items in the target multiset,

$$\pi_*(y_t|x, \mathcal{A}_t) = \begin{cases} \frac{1}{|\mathcal{A}_t|} & y_t \in \mathcal{A}_t \\ 0 & \text{otherwise,} \end{cases} \tag{54}$$

where as in (§12), the oracle's state contains the valid actions. Using the uniform oracle, the valid actions defined above, and the KL-divergence loss (Equation 39) lets us write a simplified non-monotonic objective (Equation 52), which we call the *multiset loss*,

Definition 17. *Multiset loss. The multiset loss for an example x and associated trajectory $\tau \sim \pi^{in}$ is,*

$$\mathcal{L}_{multi}(\tau, x, \theta) = -\sum_{t=1}^{T} \frac{1}{|\mathcal{A}_t|} \sum_{y \in \mathcal{A}_t} \log \pi_\theta(y|\hat{y}_{<t}, x), \tag{55}$$

which follows from expanding the KL-divergence loss (Equation 39) using the uniform oracle and valid actions defined above, and removing the entropy term since it is constant with respect to θ.

Permutation invariance. The multiset loss teaches the learned policy to generate any of the remaining correct elements. As a result, there is no prescribed order in which the learned policy must generate the target multiset. Formally, the loss is invariant to permuting a correct sequence of predictions,

$$\mathcal{L}_{\text{multi}}(\tau, x, \theta) = \mathcal{L}_{\text{multi}}(\sigma(\tau), x, \theta), \tag{56}$$

where $\sigma \in \mathbb{S}_T$ is an arbitrary permutation. This differs from the fixed-order maximum-likelihood objective (Equation 8), which treats a single permutation σ_* as the ground-truth generation order, meaning

$$\mathcal{L}_{\text{MLE}}(\tau, x, \theta) \neq \mathcal{L}_{\text{MLE}}(\sigma(\tau), x, \theta), \tag{57}$$

unless $\sigma = \sigma_*$, and in particular $\mathcal{L}_{\text{MLE}}$ can only minimized given the permutation $\sigma_*(\tau)$ (except in degenerate cases). Fixed-order maximum-likelihood does not account for the unordered nature of the multiset, in the sense that it penalizes all alternative generation orders.

Decreasing entropy. Intuitively, the uniform oracle policy over the valid actions defined above teaches the learned policy which items in the class set $\mathcal{C}$ can be generated at each time step t without decreasing the precision or recall of the resulting trajectory. By selecting an item according to the oracle, the valid actions decrease in size. As a result, the uniform oracle policy's entropy decreases over time,

$$\mathcal{H}(\pi_*^{(t)}) > \mathcal{H}(\pi_*^{(t+1)}), \tag{58}$$

where $\mathcal{H}(\pi_*^{(t)})$ denotes the Shannon entropy of the oracle policy at time t, $\pi_*(y|\hat{y}_{<t}, x, \mathcal{A}_t)$.

13.2 Related Problems in Supervised Learning

Variants of multiset prediction have been studied previously. We now discuss a taxonomy of approaches in order to differentiate the multiset loss function from previous work and define strong baselines.

13.2.1 Set prediction. A related variant predicts a set rather than a multiset. One approach to set prediction is **ranking**, which can be considered as learning a mapping from a pair of input x and one of the items $c \in \mathcal{C}$ to its score $s(x, c)$. All the items in the class set are then sorted according to the score, and this sorted order determines the rank of each item. Taking the top-k items from this sorted list results in a predicted set (e.g. Gong et al. (2013)). Similarly to multiset prediction, the input x is arbitrary, and the target is a set without any prespecific order. However, ranking differs from multiset prediction in that it is unable to handle multiple occurrences of a single item in the target set.

Another approach to set prediction is **multi-label classification**, which consists of learning a mapping from an input x to a subset of classes identified as $\mathbf{y} \in \{0,1\}^{|\mathcal{C}|}$. This problem can be reduced to $|\mathcal{C}|$ binary classification problems by learning a binary classifier for each possible class. Representative approaches include binary relevance, which assumes classes are conditionally independent, and probabilistic classifier chains which decompose the joint probability as $p(\mathbf{y}|x) = \prod_{c=1}^{|\mathcal{C}|} p(y_c|y_{<c}, x)$ (Dembczyński et al. 2010; Read et al. 2011; Hamid Rezatofighi et al. 2017). Since each $p(y_c|y_{<c}, x)$ models *binary* class membership, their predictions collectively form a set $\hat{\mathbf{y}} \in \{0,1\}^{|\mathcal{C}|}$ rather than a multiset $\hat{\mathbf{y}} \in \mathbb{N}^{|\mathcal{C}|}$.

13.2.2 Parallel prediction. A brute-force approach called **power multiset classification**, based on the combination method in multi-label classification (Tsoumakas and Katakis 2007; Read et al. 2011), is to transform the class set C into a set $M(C)$ of all possible multisets, then train a classifier π that maps an input x to an element in $M(C)$. However, the number of all possible multisets grows exponentially in the maximum size of a target multiset,[21] rendering this approach infeasible in practice.

Instead of considering the target multiset as an actual multiset, one can convert it into a distribution over the class set, using each item's multiplicity. That is, we consider a target multiset $\mathcal{Y}$ as a set of samples from a single, underlying distribution q^* over the class set C, empirically estimated as $q^*(c|x) = \frac{1}{|\mathcal{Y}|}\sum_{y\in\mathcal{Y}} \mathbb{I}_{y=c}$, where $\mathbb{I}_{\cdot}$ is an indicator function. A model then outputs a point $q_\theta(\cdot|x)$ in a $|C|$-dimensional simplex and is trained by minimizing a divergence between $q_\theta(\cdot|x)$ and $q^*(c|x)$, which we call **one-step distribution matching.** The model also predicts the size $\hat{l}_\theta(x)$ of the target multiset, so that each unique $c \in C$ has a predicted cardinality $\hat{\mu}(c) = \text{round}(q_\theta^c(x) \cdot \hat{l}(x))$, giving the objective:

$$\mathcal{L}_{\text{1-step}}(x, \mathcal{Y}, \theta) = \sum_{c\in C} q_*(c|x) \log q_\theta(c|x) + \lambda(\hat{l}_\theta(x) - |\mathcal{Y}|)^2, \tag{59}$$

where $\lambda > 0$ is a coefficient for balancing the contributions from the two terms. An un-normalized variant could directly regress the cardinality of each class. A major weakness of these methods is the lack of modeling dependencies among the items in the predicted multiset, a known issue in multi-label classification (Dembczyński et al. 2010; Nam et al. 2017). We test $\mathcal{L}_{\text{1-step}}$ in the experiments and observe substantially worse prediction accuracy than other baselines.

13.2.3 Sequential methods. A **sequence prediction** problem is characterized as finding a mapping from an input x to a sequence of classes $\mathcal{Y}_{\text{seq}} = (y_1, \ldots, y_T)$. It is different from multiset prediction since a sequence has a predetermined order of items, while a multiset is an unordered collection. Multiset prediction can however be treated as sequence prediction by defining an ordering for each multiset. We

[21] The number of all possible multisets of size $\leq K$ is $\sum_{k=1}^{K} \frac{(|C|+k-1)!}{k!(|C|-1)!}$.

can formalize this as a special case of our non-monotonic generation framework by using the deterministic oracle (Equation 48),

$$\pi_*^{\text{det}}(a|\bar{\mathbf{s}}_t) = \begin{cases} 1 & a = \sigma(\bar{\mathbf{s}}_t) \\ 0 & \text{otherwise.} \end{cases} \tag{60}$$

By specifying an ordering function σ and using π_*^{det} as the roll-in, each target multiset $\mathcal{Y}$ is deterministically transformed into an ordered sequence $\mathcal{Y}_{\text{seq}} = (y_1, \ldots, y_{|\mathcal{Y}|})$. Using the KL-divergence loss (39) and removing terms that are constant with respect to θ yields the objective,

$$\mathcal{L}_{\text{seq}}(x, \mathcal{Y}_{\text{seq}}, \theta) = -\sum_{t=1}^{T} \log \pi_\theta(y_t|y_{<t}, x),$$

which is equivalent to maximizing the likelihood of $\mathcal{Y}_{\text{seq}}$. Sequential prediction can also be seen as using the multiset loss with a deterministic oracle specifying a valid action at each step, $\mathcal{A}_t = \{\sigma(\bar{\mathbf{s}}_t)\}$.

Recently, multi-label classification (i.e. set prediction) was posed as sequence prediction with RNNs (Wang et al. 2016; Nam et al. 2017), improving upon methods that do not model conditional label dependencies. However, these approaches and the $\mathcal{L}_{seq}$ approach outlined above require a pre-specified, often ad-hoc, ordering function, and performance can significantly vary based on the choice. Vinyals et al. (2015) observed this variation in sequence-based set prediction (also observed in Nam et al. (2017); Wang et al. (2016)), which we confirm for multisets in the experiments. This shows the importance of our proposed method, which does not require a pre-specified ordering. We use $\mathcal{L}_{\text{seq}}$ as a baseline, finding that it underperforms the multiset loss.

Another baseline is a sequential version of distribution matching, called **aggregated distribution matching.** As in one-step distribution matching, a multiset is treated as a distribution q_* over classes. The sequential variant predicts a sequence of classes $(y_1, \ldots, y_T)$ by sampling from a predicted distribution $q_\theta^{(t)}(y_t|y_{<t}, x)$ at each step t. The per-step distributions $q_\theta^{(t)}$ are averaged into an aggregate distribution q_θ, and a divergence between q_* and q_θ is minimized. We test L_1 distance and KL-divergence,

$$\mathcal{L}_{\text{dm}}^{1}(x, \mathcal{Y}, \theta) = \|q_* - q_\theta\|_1, \tag{61}$$

$$\mathcal{L}_{\text{dm}}^{\text{KL}}(x, \mathcal{Y}, \theta) = -\sum_{c \in C} q_*(c|x) \log q_\theta(c|x). \tag{62}$$

A major issue with this approach is that it may assign non-zero probability to an incorrect sequence of predictions due to the aggregated distribution's invariance to the order of predictions. This is reflected in an increase in the entropy of $q_\theta^{(t)}$ over time, discussed in Experiment 3.

Reinforcement learning. In Welleck et al. (2017), an approach based on reinforcement learning (RL) was proposed for multiset prediction. Instead of assuming the existence of an oracle policy, this approach solely relies on a reward function r designed specifically for multiset prediction,

$$r(\hat{y}_t, \mathcal{A}_t) = \begin{cases} 1, & \text{if } \hat{y}_t \in \mathcal{A}_t \\ -1, & \text{otherwise.} \end{cases}$$

The goal is then to maximize the expected sum of rewards over trajectories sampled from π_θ,

$$\mathcal{L}_{\text{RL}}(x, \theta) = -\mathbb{E}_{\hat{\mathbf{y}} \sim \pi_\theta} \left[\sum_{t=1}^{T} r(\hat{y}_{<t}, \mathcal{A}_t) - \lambda \mathcal{H}(\pi_\theta(\hat{y}_{<t}, x)) \right], \tag{63}$$

where the valid actions are determined according to (Equation 53) for each sampled $\hat{\mathbf{y}}$ and the second term inside the expectation is the negative entropy multiplied with a regularization coefficient λ. The second term encourages exploration during training. As in (Welleck et al. 2017), we use REINFORCE (Williams 1992) to stochastically minimize the loss function above with respect to π_θ. This loss function is optimal in that the total reward is maximized when both the precision and recall are maximal ($= 1$).

13.2.4 Domain-Specific Methods In computer vision, object counting and object detection are instances of multiset prediction. Typical object counting approaches in computer vision, e.g. Lempitsky and Zisserman (2010); Zhang et al. (2016); Oñoro-Rubio and López-Sastre (2016), model the counting problem as density estimation over image space, and assume that each object is annotated with a dot specifying its location. Object detection methods (e.g. Stewart et al. (2016); Ren and Zemel (2017); He et al. (2017)) also require object location annotations. Since these approaches exploit the fact the input is an image and rely on additional annotated information, they are not directly comparable to our method which only assumes annotated class labels and is agnostic to the input modality.

13.3 Empirical Evaluation

We empirically evaluate the multiset loss on multiset image classification tasks, where the policy must generate the multiset of class labels present in each input image. In addition to comparing the multiset loss against an extensive collection of baselines, we study how the roll-in policy affects learning, how the sequential baseline performs based on the choice or ordering function, and how the learned policy's entropy evolves over time.

Datasets – MNIST Multiset. MNIST Multiset is a class of synthetic datasets. Each dataset consists of multiple 100x100 images, each of which contains a varying number of digits from the original MNIST

(LeCun et al. 1998). We vary the size of each digit and also add clutter. In the experiments, we consider variants of MNIST Multiset with $|\mathcal{Y}| = 4, |\mathcal{Y}| \in \{1, 2, 3, 4\}$, or $|\mathcal{Y} = 10|$, where $|\mathcal{Y}|$ is the number of digits in each image. Each variant has a training set with 70,000 examples and a test set with 10,000 examples. We randomly sample 7,000 examples from the training set to use as a validation set, and train with the remaining 63,000 examples.

Datasets – MS COCO. As a real-world dataset, we use Microsoft COCO (Lin et al. 2014) which includes natural images with multiple objects. Compared to MNIST Multi, each image in MS COCO has objects of more varying sizes and shapes, and there is a large variation in the number of object instances per image which spans from 1 to 91. The problem is made even more challenging with many overlapping and occluded objects. To better control the difficulty, we create two variants: **COCO Easy** with $|\mathcal{Y}| = 2$, 10k examples, and 24 classes, and **COCO Medium** with $|\mathcal{Y}| \in \{1, \ldots, 4\}$, 44k training examples, and 23 classes.

In both variants, we only include images whose $|\mathcal{Y}|$ objects are large and of common classes. An object is defined to be large if the object's area is above the 40-th percentile across the training set of MS COCO. After reducing the dataset to have $|\mathcal{Y}|$ large objects per image, we remove images containing only objects of rare classes. A class is considered rare if its frequency is less than $\frac{1}{|C|}$, where C is the class set. These two stages ensure that only images with a proper number of large objects are kept. We do not use fine-grained annotation (pixel-level segmentation and bounding boxes) except for creating input-dependent order functions for the sequential baseline.

For each variant, we hold out a randomly sampled 15% of the training examples as a validation set. We form separate test sets by applying the same filters to the COCO validation set. The test set sizes are 5,107 for COCO Easy and 21,944 for COCO Medium.

Models. For MNIST Multiset experiments, we use three convolutional layers of channel sizes 10, 10 and 32, followed by a convolutional long short-term memory (LSTM) layer (Xingjian et al. 2015). At each step, the feature map from the convolutional LSTM layer is average-pooled spatially and fed to a softmax classifier. For one-step distribution matching, the LSTM layer is skipped.

For MS COCO experiments, we use a ResNet-34 (He et al. 2016) pretrained on ImageNet (Deng et al. 2009) as a feature extractor. The final feature map from this ResNet-34 is fed to a convolutional LSTM layer, as described for MNIST Multi above. We do not finetune the ResNet-34 feature extractor.

In all experiments, for predicting variable-sized multisets we use the termination policy approach since it is easily applicable to all of the baselines, thus ensuring a fair comparison. An alternative is to predict

	MNIST Multiset (4)		COCO Easy	
	EM	F1	EM	F1
$\sigma_{\textbf{random}}$	0.088	0.507	0.459	0.555
$\sigma_{\textbf{random-1}}$	0.920	0.977	0.721	0.779
$\sigma_{\textbf{area}}$	0.529	0.830	0.700	0.763
$\sigma_{\textbf{spatial}}$	0.917	0.976	0.675	0.738

Table 25: Performance of the sequential baseline (§13.2.3) varies based on the ordering σ used by the deterministic oracle π_*^{det} (Equation 60).

	COCO Medium	
	EM	F1
Greedy	0.475 ± 0.006	0.645 ± 0.016
Stochastic	0.475 ± 0.004	0.649 ± 0.009
Oracle	0.469 ± 0.002	0.616 ± 0.009

Table 26: Influence of the roll-in policy π^{in} used in the multiset loss (Equation 55). Using π_θ as the roll-in policy with either greedy selection or sampling outperforms oracle π_* roll-in.

	Multiset (4)		Multiset (1-4)		Multiset (10)	
	EM	F1	EM	F1	EM	F1
$\mathcal{L}_{\text{multi}}$	**0.950**	**0.987**	**0.953**	**0.981**	**0.920**	**0.992**
$\mathcal{L}_{\text{RL}}$	0.912	0.977	0.945	0.980	0.665	0.970
$\mathcal{L}_{\text{dm}}^{1}$	0.921	0.978	0.918	0.969	0.239	0.714
$\mathcal{L}_{\text{dm}}^{\text{KL}}$	0.908	0.974	0.908	0.962	0.256	0.874
$\mathcal{L}_{\text{seq}}$	0.906	0.973	0.891	0.952	0.592	0.946
$\mathcal{L}_{\text{1-step}}$	0.210	0.676	0.055	0.598	0.032	0.854

Table 27: **MNIST Multiset** results.

	Easy		Medium	
	EM	F1	EM	F1
$\mathcal{L}_{\text{multi}}$	0.702	**0.788**	**0.481**	**0.639**
$\mathcal{L}_{\text{RL}}$	0.672	0.746	0.425	0.564
$\mathcal{L}_{\text{dm}}^{1}$	0.533	0.614	0.221	0.085
$\mathcal{L}_{\text{dm}}^{\text{KL}}$	**0.714**	0.763	0.444	0.591
$\mathcal{L}_{\text{seq}}$	0.709	0.774	0.457	0.592
$\mathcal{L}_{\text{1-step}}$	0.552	0.664	0.000	0.446

Table 28: **MS COCO** results.

a special ⟨eos⟩ token to determine termination, but it is unclear how to extend this approach to the distribution matching baselines.

Training and evaluation. For each loss, a model was trained for 200 epochs (350 for MNIST Multi 10). After each epoch, exact match was computed on the validation set. The model with the highest validation exact match was used for evaluation on the test set. When evaluating a trained policy, we use greedy decoding. Each predicted multiset is compared against the ground-truth target multiset, and we report the exact match accuracy (EM) and F-1 score (F1).

13.3.1 Experiment 1: Influence of order function on sequential baseline. First, we investigate the sequence loss function $\mathcal{L}_{\text{seq}}$ (§13.2.3), while varying the order function σ, which determines how each target multiset is transformed into a sequence (Equation 60). We test four alternatives: random ordering functions generated for each training batch (σ_{random}), a random order function generated before training and held fixed ($\sigma_{\text{random-1}}$), and two input-dependent order functions σ_{spatial} and σ_{area}. σ_{spatial} orders labels in left-to-right, top-to-bottom order, and σ_{area} orders labels by decreasing object area. We compare $\mathcal{L}_{\text{seq}}$ with these order functions on MNIST Multi (4) and COCO Easy.

We present the results in Table 25. It is clear from the results that the performance of the sequence prediction loss function is dependent on the choice of ordering function. On MNIST Multiset the area-based rank function was far worse than the other choices. However, this was not true on COCO Easy, where the spatial rank function was worst among the three. In both cases, we have observed that the fixed

random ordering function ($\sigma_{\text{random-1}}$) performed best, so for the remaining experiments we use $\sigma_{\text{random-1}}$ for the sequence prediction loss. This set of experiments firmly suggests the need of an order-invariant multiset loss function, such as our multiset loss function.

13.3.2 Experiment 2: Roll-in policy for the multiset loss. In this experiments, we compare three roll-in policies for the multiset loss function. They are **greedy** decoding, $\hat{y}_t = \arg\max_{y_t} \pi_\theta(y_t|\hat{y}_{<t}, x)$, **stochastic** sampling, $\hat{y}_t \sim \pi_\theta(y_t|\hat{y}_{<t}, x)$, and **oracle** sampling, $\hat{y}_t \sim \pi_*(y_t|x, \mathcal{A}_t)$. We test them on the most challenging dataset, COCO Medium, and report the mean and standard deviation for the evaluation metrics across 5 runs.

As shown in Table 26, greedy decoding and stochastic sampling, both of which consider states that are likely to be visited by the learned policy, outperform the oracle sampling, which only considers states on optimal trajectories. This is particularly apparent in the F1 score, which can be increased even after visiting a state that is not on an optimal trajectory. The results are consistent with the theory from Ross et al. (2011); Chang et al. (2015). The performance difference between greedy decoding and stochastic sampling was not significant, so from here on we choose the simpler method, greedy decoding, when training a model with the multiset loss.

13.3.3 Experiment 3: Loss function comparison. We now compare the proposed multiset loss function against the five baseline loss functions (see §13.2): reinforcement learning $\mathcal{L}_{\text{RL}}$, aggregate distribution matching ($\mathcal{L}^1_{\text{dm}}$ and $\mathcal{L}^{\text{KL}}_{\text{dm}}$), its one-step variant $\mathcal{L}_{\text{1-step}}$, and sequence prediction $\mathcal{L}_{\text{seq}}$.

MNIST Multiset. We present the results on the MNIST Multiset variants in Table 27). On all three variants and according to both metrics, the multiset loss function outperforms all the others. The reinforcement learning based approach closely follows behind. Its performance, however, drops as the number of items in a target multiset increases. This is understandable, as the variance of policy gradient grows as the trajectory length grows. A similar behavior was observed with sequence prediction as well as aggregate distribution matching. We were not able to train any decent models with the one-step variant of aggregate distribution matching. This was true especially in terms of exact match (EM), which we attribute to the one-step variant not being capable of modeling dependencies among the predictions.

MS COCO. Similar to the results on the variants of MNIST Multiset, the multiset loss function matches or outperforms all the others on the two variants of MS COCO, as presented in Table 28. On COCO Easy, with only two objects to predict per example, both aggregated distribution matching (with KL divergence) and the sequence loss functions are as competitive as the proposed multiset loss. The other loss functions significantly underperform these three loss functions, as they did on MNIST Multiset.

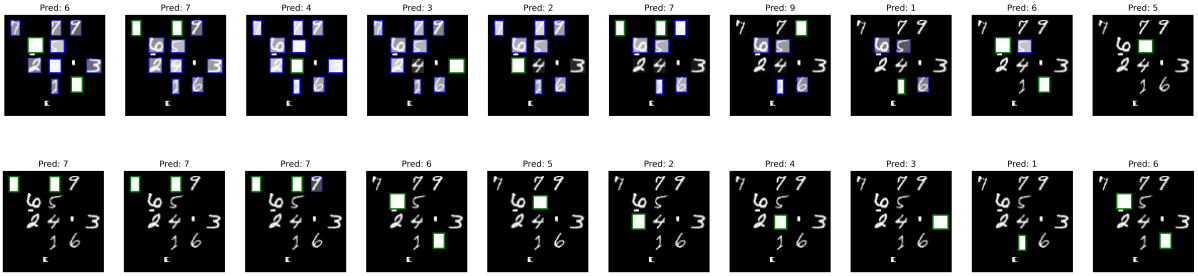


Fig. 21: Each row shows a trajectory from a learned policy π_θ, with each image representing a timestep. Each digit with class y_t is highlighted according to its probability, $\pi_\theta(y_t|\hat{y}_{<t}, x)$, normalized by the highest class's probability. Brighter squares represent higher probabilities. A green box denotes the class predicted by the policy, $\hat{y}_t = \arg\max_{y_t} \pi_\theta$, and a blue border denotes a class with normalized probability exceeding a threshold $\tau = 0.1$. **Top**: a policy trained with $\mathcal{L}_{\text{multiset}}$. **Bottom**: a policy trained with $\mathcal{L}_{\text{seq}}$ using the spatial ordering function.

The performance gap between the proposed loss and the others, however, grows substantially on the more challenging COCO Medium, which has more objects per example. The multiset loss outperforms aggregated distribution matching with KL divergence by 3.7 percentage points on exact match and 4.8 on F1. This is analogous to the experiments on MNIST Multiset, where the performance gap increased when moving from four to ten digits.

Analysis: Entropy evolution. Recall that the entropy of the uniform oracle's predictive distribution decreases over time, i.e., $\mathcal{H}(\pi_*^{(t)}) > \mathcal{H}(\pi_*^{(t+1)})$ (Equation 58). This naturally follows from the fact that there is no pre-specified rank function, because the oracle policy cannot prefer any item from the others in a free label multiset. Hence, we examine here how the policy learned based on each loss function compares to the oracle policy in terms of per-step entropy. We consider the policies trained on MNIST Multiset (10), where the differences among them were most clear. As shown in Fig. 22, the policy trained on MNIST Multiset (10) using the proposed multiset loss closely follows the oracle policy. The entropy decreases as the predictions are made. The decreases can be interpreted as concentrating probability mass on progressively smaller valid action sets. The variance is quite small, indicating that this strategy is uniformly applied for any input.

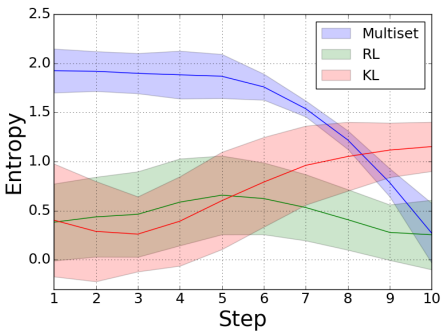


Fig. 22: Comparison of per-step entropies of predictive distributions (validation set).

Figure 21 shows an example which illustrates this learned strategy. Interestingly, we noticed that the policy learned to consistently place slightly higher probability on classes that have multiple unpredicted

instances (e.g. 6, 7 on the first two steps), despite the fact that the uniform oracle places equal probability on classes in the valid set independent of the number of instances.

The policy trained with reinforcement learning retains a relatively low entropy across steps, with a decreasing trend in the second half. We carefully suspect the low entropy in the earlier steps is due to the greedy nature of policy gradient. The policy receives a high reward more easily by choosing one of many possible choices in an earlier step than in a later step. This effectively discourages the policy from exploring all trajectories during training.

On the other hand, the policy found by aggregated distribution matching ($\mathcal{L}_{\text{dm}}^{\text{KL}}$) has the opposite behaviour. The entropy in general grows as more predictions are made. To see why this is sub-optimal, consider the final step. Assuming the first nine predictions were correct, there is only one correct class left for the final prediction. The high entropy, however, indicates that the model is placing a significant amount of probability on incorrect sequences. Such a policy may result because $\mathcal{L}_{\text{dm}}^{\text{KL}}$ cannot properly distinguish between policies with increasing and decreasing entropies.

13.4 Discussion

In this section we showed how the non-monotonic generation framework is used to learn policies that conditionally generate multisets, which we call multiset prediction. Using the uniform oracle along with valid actions that respect the unordered nature of multisets, we derived an objective specifically designed for multiset prediction, called the *multiset loss*. The multiset loss is invariant to the order in which the multiset is generated, unlike typical sequential losses. The experiments on two families of datasets, MNIST Multiset and MS COCO, demonstrated the effectiveness of the multiset loss over reinforcement learning, sequence, and aggregated distribution matching loss functions. This success brings new opportunities for applying the multiset loss – and more generally, the non-monotonic generation framework – to other domains. We do so in the next section, where we use a similar approach for dependency parsing.

14 Trees: Sequential Graph Dependency Parser

In this section, we will show how the non-monotonic generation framework (§12) is used for conditionally generating *dependency trees*. Our motivation here is to develop a sequential parsing method that does not impose a generation order like typical sequential dependency parsing methods (Ma et al. 2018; Fernández-González and Gómez-Rodríguez 2019). To do so, we view parsing as incrementally building a graph by adding edges to an edge set. Since edges do not have a pre-specified order, we adopt the non-monotonic generation framework to obtain parsers with learned, input-dependent generation orders by using a variant of the coaching oracle and a roll-in policy that only explores the space of valid dependency trees. Experimentally, we find that that these sequential, non-monotonic parsers give performance gains over strong one-step methods.

14.1 Method

Given a sentence $\mathbf{x} = (x_1, \ldots, x_N)$, a dependency parser constructs a graph $G = (V, E)$ with $V = (x_0, x_1, \ldots, x_N)$ and $E = \{(i,j)_1, \ldots (i,j)_N\}$, where x_0 is a special root node, and E forms a dependency tree.[22] First, we will describe a family of sequential graph-based dependency parsers that we will use in this work. Then we will discuss learning a parser with the non-monotonic generation framework.

14.1.1 Sequential graph dependency parser. We describe a family of graph-based dependency parsers which generate a *sequence* of graphs, where V is fixed and $E = \bigcup_{t=1}^{T} E_t$:

$$H^{\text{enc}} = f_{\text{enc}}(x_0, \ldots, x_N) \tag{64}$$

$$H_t^{\text{head}}, H_t^{\text{dep}} = f_{\text{V}}(H^{\text{enc}}, E_{<t}, h_{t-1}) \tag{65}$$

$$S_t = f_{\text{E}}(H_t^{\text{head}}, H_t^{\text{dep}}, S_{t-1}) \tag{66}$$

$$E_t = f_{\text{dec}}(S_t, E_{<t}). \tag{67}$$

Steps (2-4) run for $T \leq N$ time-steps. At each time-step, first f_V generates head and dependent representations for each vertex, $H_t^{\cdot} \in \mathbb{R}^{V \times d_H}$, based on vertex representations $H^{\text{enc}} \in \mathbb{R}^{V \times d}$, previously predicted edges $E_{<t}$, and a recurrent state $h_{t-1} \in \mathbb{R}^d$. Then f_E computes a score for every possible edge, $S_t \in \mathbb{R}^{V \times V}$, and the scores are used by f_{dec} to predict a set of edges E_t.

This general sequential family includes the biaffine parser of Dozat and Manning (2017) as a one-step special case, as well as a recurrent variant which we discuss below.

[22] See properties (1-5) in Appendix A.2.2.

Biaffine one-step. The Biaffine parser of Dozat and Manning (2017) is a one-step variant, implementing steps (1-4) using a bidirectional LSTM, head and dependent neural networks, a biaffine scorer, and a maximum spanning tree decoder, respectively:

$$\begin{aligned} H^{\text{enc}} &= \text{BiLSTM}(x_1, \ldots, x_N) \\ H^{\text{head}}, H^{\text{dep}} &= \text{MLP}^{\text{h}}(H^{\text{enc}}), \text{MLP}^{\text{d}}(H^{\text{enc}}) \\ S &= \text{BiAffine}(H^{\text{head}}, H^{\text{dep}}) \\ E &= \text{MST}(S), \end{aligned}$$

where each row of scores $S^{(i)}$ is interpreted as a distribution over i's potential head nodes:

$$p((j \to i)|x) \propto \text{softmax}_j(S^{(i)}),$$

and $\text{MST}(\cdot)$ is an off-the-shelf maximum-spanning-tree algorithm. This model assumes conditional independence of the edges.

Recurrent weight. We propose a variant which iteratively adjusts a distribution over edges at each step, based on the predictions so far. A recurrent function generates a weight matrix W which is used to form vertex embeddings and in turn adjust edge scores.

Specifically, we obtain an initial score matrix S_0 using the biaffine one-step parser, and initialize a recurrent hidden state h_0 using a linear transformation of f_{enc}'s final hidden state. We define f_V as:

$$\begin{aligned} W, h_t &= \text{LSTM}(f_{\text{emb}}(E_{t-1}), h_{t-1}) \\ H_t^{\text{head}} &= \text{emb}_h(0, \ldots, N)W \\ H_t^{\text{dep}} &= \text{emb}_d(0, \ldots, N)W, \end{aligned}$$

and $f_{\text{E}}(H_t^{\text{head}}, H_t^{\text{dep}}, S_{t-1})$ is defined as:

$$\begin{aligned} S_t^{\Delta} &= \text{BiAffine}(H_t^{\text{head}}, H_t^{\text{dep}}) \\ S_t &= S_{t-1} + S_t^{\Delta}, \end{aligned}$$

where t ranges from 1 to N, $W \in \mathbb{R}^{d_{\text{emb}} \times d_H}$, and each $\text{emb}_{(\cdot)} : \mathbb{N} \to \mathbb{R}^{d_{\text{emb}}}$ is a learned embedding layer, yielding $\text{emb}_{(\cdot)}(0, \ldots, N)$ in $\mathbb{R}^{V \times d_{\text{emb}}}$. We use a bidirectional LSTM as f_{enc}.

The scores at each step yield a distribution over all $V \times V$ edges, which we denote by π:

$$\pi((i \to j)|E_{<t}, x) \propto \text{softmax}(\text{flatten}(S_t)). \tag{68}$$

Unlike the one-step model, this recurrent model can predict edges based on past predictions.

Valid decoding. At inference time, we must ensure the incrementally decoded edges $E = \bigcup_{t=1}^{T} E_t$ form a valid dependency tree. To do so, we choose f_{dec} to be a decoder which greedily selects valid edges,

$$E_t = f_{\text{valid}}(S_t, E_{<t}),$$

which we refer to as the *valid decoder*, detailed in Appendix A.2.2. We only predict one edge per step ($|E_t| = 1$), leaving the setting of multiple predictions per step as future work.

Embedding edges. We embed a predicted edge $E_t = \{(\hat{i,j})\}$ as:

$$f_{\text{emb}}(E_t) = e_{\text{edge}}; e_{\text{head}}; e_{\text{dependent}},$$
$$e_{\text{edge}} = W_e H^{\text{enc}}_{(i)} - W_e H^{\text{enc}}_{(j)}, \quad e_{\text{head}} = \text{emb}_h(i), \quad e_{\text{dependent}} = \text{emb}_d(j),$$

where $H^{\text{enc}}_{(\cdot)} \in \mathbb{R}^d$ are row vectors, $W_e \in \mathbb{R}^{d_e \times d}$ is a learned weight matrix, $\text{emb}_{(\cdot)}$ are learned embedding layers, and ; is concatenation.

14.1.2 Non-monotonic dependency parsing. In this investigation, we restrict to the case of predicting a single edge $(\hat{i,j})$ per step, so that the recurrent weight model generates a sequence of edges with the goal of matching a target edge set, i.e. $\bigcup_{t=1}^{N} (\hat{i,j})_t = E$. Since the target edges E are a set, the model's generation order is not determined *a priori*. As we saw in the preceding section (§13), the non-monotonic generation framework is effective for predicting set-structured objects, and thus we adopt the non-monotonic generation framework for training our sequential graph parser.

For our sequential graph parser, an *action* is an edge $(i, j) \in \mathcal{E}$, and a *state* $\mathbf{s}_t$ is an input sentence $\mathbf{x}$ along with the edges predicted so far, $\hat{E}_{<t}$. The *policy* is a conditional distribution over $\mathcal{E}$,

$$\pi_\theta((i, j)|\hat{E}_{<t}, \mathbf{x}),$$

such as the distribution in Equation 68.

As we discussed in §12, learning consists of minimizing the KL-divergence between an *oracle* policy and the learned policy, computed on states from a *roll-in* policy π^{in} (Equation 52). In the case of dependency parsing this means,

$$\arg\min_{\theta} \mathbb{E}_{\mathbf{x}, E_* \sim \mathcal{D}} \mathbb{E}_{\mathbf{s}_1, \ldots, \mathbf{s}_{|\mathbf{x}|} \sim \pi^{\text{in}}} \sum_{t=1}^{|\mathbf{x}|} \mathrm{D}_{\mathrm{KL}} \left(\pi_*(\cdot \mid \bar{\mathbf{s}}_t) \| \pi_\theta(\cdot \mid \mathbf{s}_t) \right), \tag{69}$$

where $\mathbf{x}$ is a sentence sampled from a dataset with ground-truth tree E, and the states $\mathbf{s}_t = (\mathbf{x}, \hat{E}_{<t})$ are obtained by sampling a sequence of edges from the roll-in policy. We now describe our choices for valid actions, oracle, and roll-in.

Valid actions and oracles. Analogous to the valid actions that we used for multiset prediction (§13), we define a *free edge set* containing the un-predicted target edges at time t:

$$E^t_{\text{free}} = E \setminus \bigcup_{t'=1}^{t-1} (\hat{i,j})_{t'}, \tag{70}$$

where $E^0_{\text{free}} = E$. The free edges act as the valid actions, meaning an oracle policy places non-zero probability mass only on free edges. We will consider the three oracles introduced in §12. The *uniform* oracle assigns a uniform probability to each free edge:

$$\pi_*^{\text{uniform}}((i,j)|E^t_{\text{free}}) = \begin{cases} \frac{1}{|E^t_{\text{free}}|} & (i,j) \in E^t_{\text{free}} \\ 0 & \text{otherwise,} \end{cases}$$

which treats each permutation of the target edge set as equally likely. The *coaching* oracle weights free edges by π_θ:

$$\pi_*^{\text{coaching}}((i,j)|E^t_{\text{free}}) \propto \pi^*_{\text{unif}}(\cdot|E^t_{\text{free}}) \pi_\theta(\cdot|E_{<t}, X).$$

This oracle prefers certain edge permutations over others, reinforcing π_θ's preferences. We use two techniques when using the coaching oracle in practice. First, the coaching and uniform oracles are mixed to ensure each free edge receives probability mass:

$$\beta \pi_*^{\text{uniform}} + (1-\beta) \pi_*^{\text{coaching}}, \tag{71}$$

where $\beta \in [0,1]$. Second, we can begin training with the uniform oracle and gradually introducing the coaching oracle by annealing the β term as training progresses, which we call *annealed coaching*. This may prevent the coaching oracle from reinforcing sub-optimal permutations early in training.

Finally, we consider the *deterministic* oracle (Equation 48), which linearizes each edge set E into a sequence E_{seq}. The oracle selects the t'th element of E_{seq} at time t with probability 1. We linearize each edge set in increasing edge-index order: (i_1, j_1) precedes (i_2, j_2) if $(i_1, j_1) < (i_2, j_2)$. This oracle serves as a baseline that is analogous to the fixed generation orders used in conventional parsers.

Roll-in. The roll-in policy determines the state distribution that π_θ is trained on, which can address the mismatch between training and testing state distributions or narrow the set of training trajectories. Although existing theory suggests that learned policy roll-in ($(i, j) \sim \pi_\theta$) is optimal (e.g see Chang et al. (2015)), in practice we observed that for dependency parsing, using learned policy roll-in failed to train. We suspect this is since a poorly-trained policy generates many invalid dependency trees, making learning difficult.

As a result, we propose two variants of learned policy roll-in: **coaching roll-in**, $(i, j) \sim \pi_\theta \odot \pi_*^{\text{uniform}}$, and **valid-policy roll-in**, $(i, j) \sim \text{valid}(\pi_\theta)$, where $\text{valid}(\pi_\theta)$ restricts the policy's distribution to the set of edges that keeps the predicted tree as a valid dependency tree (i.e properties (1-5) hold). These roll-ins choose edge permutations that are preferred by the policy. Coaching roll-in incorporates preferences over *correct* edges, while valid-policy roll-in allows for certain incorrect predictions.

14.2 Empirical Evaluation

We first evaluate the effect of the multi-step model versus a single-step model, then evaluate the effect the varying the roll-in and oracle policy. For these initial evaluations we use English, German, Chinese, and Ancient Greek dependency parsing datasets since they vary with respect to projectivity, size, and performance in past work (Qi et al. 2018). Based on these development set results, we then test our strongest model on a large suite of languages.

Experimental setup. We use datasets from the CoNLL 2018 Shared Task (Zeman et al. 2018). We build our implementation from the open-source version of Qi et al. (2018),[23] and use their experimental setup (pre-processing, data-loading, pre-trained vectors, evaluation) which follows the shared task setup. Our model uses the same encoder from Qi et al. (2018). For the Qi et al. (2018) baseline, we use their pretrained models and evaluation script.[24] For the Dozat and Manning (2017) baseline, we use the Qi et al. (2018) im-

	En	**De**	**Grc**	**Zh**
D&M (2017)	91.14	90.38	78.99	86.50
Qi et al. (2018)	92.11	89.46	81.35	86.73
One-step	91.74	91.07	79.60	86.61
Recurrent (U)	91.92	91.02	79.15	86.69
Recurrent (C)	91.99	91.19	79.93	86.77

Table 29: Development set UAS for single vs. multi-step methods. (U) is uniform oracle and roll-in, (C) is coaching with valid roll-in. D&M is an abbreviation for (Dozat and Manning 2017).

[23] `https://github.com/stanfordnlp/stanfordnlp`.
[24] `https://stanfordnlp.github.io/stanfordnlp/installation_download.html`.

Oracle	Roll-in	UAS	Loss
Determin.	π_*^{det}	81.03	0.04
Uniform	π^*_{uniform}	91.02	0.35
Coaching	π^*_{uniform}	91.04	0.17
Uniform	π^*_{coaching}	90.93	0.45
Coaching	π^*_{coaching}	91.17	0.33
Annealed	π^*_{coaching}	90.89	0.34
Uniform	$\pi_\theta^{\text{valid}}$	90.99	0.51
Coaching	$\pi_\theta^{\text{valid}}$	**91.19**	0.31
Annealed	$\pi_\theta^{\text{valid}}$	90.91	0.30

Table 30: Varying the oracle and roll-in policies (German dataset).

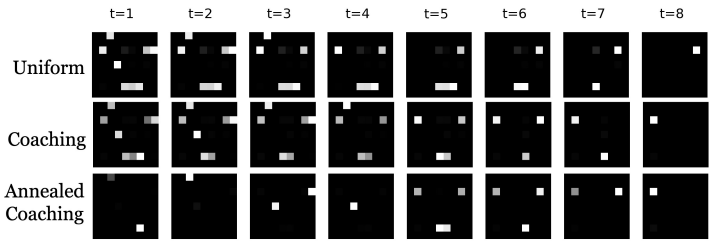


Fig. 23: Per-step edge distributions from recurrent weight models trained with the given oracle.

plementation with auxiliary outputs and losses disabled, and train with the default hyper-parameters and training script. For our models, we changed the learning rate schedule (and model-specific hyper-parameters), after observing diverging loss in preliminary experiments with the default learning rate. Our models did not require the additional AMSGrad technique used in Qi et al. (2018). We evaluate validation UAS every 2k steps (vs. 100 for the baseline). Models are trained for up to 100k steps, and the model with the highest validation unlabeled attachment score (UAS) is saved.

14.2.1 Experiment 1: Multi-step learning. In this experiment we evaluate the sequential aspect of the proposed recurrent model by comparing it with one-step baselines. We compare against a baseline ('one-step') that simply uses the first step's score matrix S_0 from the recurrent weight model and minimizes the loss for one time-step using a uniform oracle. At test time the valid decoder uses S_0 for all timesteps. We also compare against the biaffine one-step model of Dozat and Manning (2017) which uses Chu-Liu-Edmonds maximum spanning tree decoding instead of valid decoding. Since we only evaluate UAS, we disable its edge label output and loss. Finally, we compare against Qi et al. (2018) which is based on Dozat and Manning (2017) plus auxiliary losses for length and linearization prediction.

Results are shown in Table 29, including results for a recurrent model trained with coaching ('Recurrent (C)') using a mixture (eq. 71) with $\beta = 0.5$. The one-step baseline is strong, even outperforming the uniform recurrent variant on some languages. The recurrent weight model with coaching, however, outperforms the one-step and Dozat and Manning (2017) baselines on all four languages. Adding in auxiliary losses to the Dozat and Manning (2017) model yields improved UAS as seen in the Qi et al. (2018) performance, suggesting that our recurrent model might be improved further with auxiliary losses.

Temporal distribution adjustment. Figure 23 shows per-step edge distributions on an eight-edge example. The recurrent weight variants learned to adjust their distributions over time based on past

predictions. The model trained with the uniform oracle has a decreasing number of high probability edges per step since it aims to place equal mass on each free edge $(i, j) \in \hat{E}^t_{\text{free}}$. The model trained with coaching learned to prefer certain free edges over others, but with $\beta = 0.5$ the uniform term in the loss still encourages placing mass on multiple edges per step. By annealing β, however, the coaching model exhibits vastly different behavior than the uniform-trained policy. The low entropy distributions at early steps followed by higher entropy distributions later on (e.g. $t \in \{5, 6\}$) indicates easy-first behavior.

14.2.2 Experiment 2: Oracle and roll-in choice. In this experiment, we study the effects of varying the oracle and roll-in distributions. Table 30 shows results on German, analyzed below. Models trained with coaching (C) use a mixture with $\beta = 0.5$, after observing lower UAS in preliminary experiments with lower β. The π^*_{coaching} and $\pi^{\text{valid}}_\theta$ roll-ins use a mixture with $\beta = 0.5$ and greedy decoding, which generally outperformed stochastic sampling.

Set-based learning. The model trained with the deterministic oracle (UAS 81.03), which teaches the model to adhere to a pre-specified generation order, underperforms the set-based models (UAS $\geq$ 90.89), which do not have a pre-specified generation order and can learn strategies such as easy-first.

Coaching. Models trained with coaching (C, UAS $\geq$ 91.04) had higher UAS and lower loss than models trained with the uniform oracle (U, UAS $\leq$ 91.02), for all roll-in methods. This suggests that for the proposed model, weighting free edges in the loss based on the model's distribution is more effective than a uniform weighting. Annealing the β parameter generally did not further improve UAS (CA vs. C), possibly due to the annealing schedule or overfitting; despite lower losses with annealing, eventually validation UAS *decreased* as training progressed.

Roll-in policy. With the coaching oracle (C), the choice of roll-in impacted UAS, with coaching roll-in (π^*_{coaching}, 91.17) and valid roll-in ($\pi^{\text{valid}}_\theta$, 91.19) achieving higher UAS than uniform oracle roll-in (π^*_{uniform}, 91.04). This suggests that when using coaching, narrowing the set of training trajectories to those preferred by the policy may be more effective than sampling uniformly from the set of all correct trajectories. Based on these results, we use the coaching oracle and valid roll-in for training our final model in the next experiment.

14.2.3 Experiment 3: CoNLL 2018 comparison. In this experiment, we evaluate our best model on a diverse set of multi-lingual datasets. We use the CoNLL 2018 shared task datasets that have at least 200k examples, along with the four datasets used in the previous experiments. We train a recurrent weight model for each dataset using the coaching oracle and valid roll-in. We compare against Qi et al.

(2018) which placed highly in the CoNLL 2018 competition, reporting test UAS evaluated using their pre-trained models.

Table 31 shows the results on the 19 datasets from 17 different languages. The recurrent model trained with the coaching oracle and a valid roll-in achieves a higher UAS than the one-step Qi et al. (2018) model on 12 of the 19 datasets, plus two ties.

14.3 Related Work

Transition-based dependency parsing has a rich history, with methods generally varying by the choice of transition system and feature representation. Traditional stack-based arc-standard and arc-eager (Yamada and Matsumoto 2003; Nivre 2003) transition systems only parse projectively, requiring additional operations for pseudo-non-projectivity (Gómez-Rodríguez et al. 2014) or projectivity (Nivre 2009), while list-based non-projective systems have been developed (Nivre 2008). Recent variations assume a generation order such as top-down (Ma et al. 2018) or left-to-right (Fernández-González and Gómez-Rodríguez 2019). Other recent models focus on unsupervised settings (Kim et al. 2019). Our focus is a non-projective transition system and learning method which does not assume a particular generation order.

	Ours	**Qi et al.**
AR	88.22	**88.35**
CA	**94.13**	**94.13**
CS (CAC)	**93.53**	93.22
CS (PDT)	**93.80**	93.21
DE	**88.39**	87.21
EN (EWT)	**91.28**	91.21
ES	**93.70**	93.38
ET	**89.56**	89.40
FR (GSD)	**91.07**	90.90
GRC (Perseus)	80.90	**82.77**
HI	**96.78**	**96.78**
IT (ISDT)	94.06	**94.24**
KO (KAIST)	**91.02**	90.55
LA (ITTB)	**93.66**	93.00
NO (Bokmaal)	**94.63**	94.27
NO (Nynorsk)	**94.44**	94.02
PT	91.22	**91.67**
RU (SynTagRus)	**94.57**	94.42
ZH	87.31	**88.49**

Table 31: Test set results (UAS) on datasets from the CoNLL 2018 shared task with greater than 200k examples, plus the Ancient Greek (GRC) and Chinese (ZH) datasets. Bold denotes the highest UAS on each dataset.

A separate thread of research in sequential modeling has demonstrated that generation order can affect performance (Vinyals et al. 2015), in tasks with set-structured outputs as we discussed in section 13, graphs (Li et al. 2018), and in NLP tasks such as language modeling (Ford et al. 2018). We investigate this for dependency parsing, framing the problem as sequential set generation.

Finally, our work is inspired by techniques for improving upon maximum likelihood training through error exploration and dynamic oracles (Goldberg and Nivre 2012, 2013), and related techniques in imitation learning for structured prediction (Daumé III et al. 2009; Ross et al. 2011; He et al. 2012a; Goodman et al. 2016). We detailed this relationship when we introduced our non-monotonic generation framework from the perspective of imitation learning (§12).

14.4 Discussion

In this section we showed how the non-monotonic generation framework is used to learn policies that conditionally generate trees for the application of dependency parsing. We described a family of dependency parsers which construct a dependency tree by generating a sequence of edge sets, and use non-monotonic generation for learning without a prescribed generation order. Experimentally, we found that coaching, which weights actions in the loss according to the model, improves parsing accuracy compared to a uniform weighting and allows the parser to learn preferred, input-dependent generation orders. The model's sequential aspect, along with coaching and training on a state distribution which resembles the model's own behavior, yielded improvements in dependency parsing over strong one-step baselines.

In this section and the preceding section on multiset prediction (§13), we generated objects that do not have a natural order. In the next section, we will show that even when the underlying data has a sequential order, we can still use non-monotonic generation to vary the model's generation order.

15 Sequences: Binary Tree Generation Policy

In this section, we will show how the non-monotonic generation framework (§12) is used for generating *text*. Unlike our previous applications of non-monotonic generation (§13, §14), text is a *sequence*. Most sequence generation models, from n-grams (Bahl et al. 1983) to neural language models (Bengio et al. 2003) generate sequences in a purely left-to-right, monotonic order. This raises the question of whether alternative, non-monotonic orders are worth considering (Ford et al. 2018), especially given the success of "easy first" techniques in natural language tagging (Tsuruoka and Tsujii 2005), parsing (Goldberg and Elhadad 2010), and coreference (Stoyanov and Eisner 2012), which allow a model to effectively learn their own ordering.

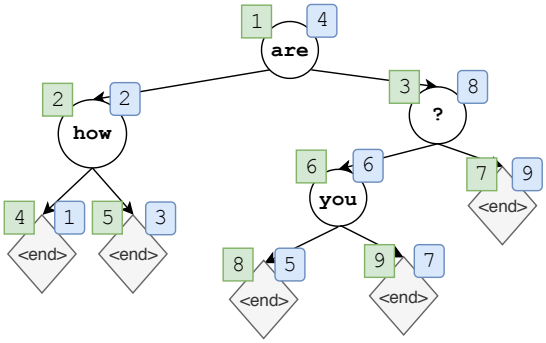


Fig. 24: A sequence, "how are you ?", generated by the proposed approach trained on utterances from a dialogue dataset. The model first generated the word "are" and then recursively generated left and right subtrees ("how" and "you ?", respectively) of this word. At each production step, the model may either generate a token, or an ⟨end⟩ token, which indicates that this subtree is complete. The full generation is performed in a level-order traversal, and the output is read off from an in-order traversal. The numbers in green squares denote generation order (level-order); those in rounded blue squares denote location in the final sequence (in-order).

In this section, we use our non-monotonic generation framework (§12) for training sequential text generation models which learn a generation order without having to specifying an order in advance. An example generation from our model is shown in Figure 24. As before, we frame the learning problem as an imitation learning problem, in which we aim to learn a generation policy that mimics the actions of an oracle generation policy. Because the best generation order is unknown, the oracle policy cannot know the exact correct actions to take; to remedy this we show that annealing towards the coaching oracle (Equation 50) can yield a policy with learned generation orders, by gradually moving from imitating a maximum entropy oracle to reinforcing the policy's own preferences. Experimental results demonstrate that using the proposed framework, it is possible to learn policies which generate text without pre-specifying a generation order, achieving easy first-style behavior. The policies achieve performance metrics that are competitive with or superior to conventional left-to-right generation in language modeling, word reordering, and machine translation.[25]

[25] Code and trained models available at `https://github.com/wellecks/nonmonotonic_text`.

15.1 Method

We consider the problem of sequentially generating a sequence of discrete tokens $\mathbf{y} = (w_1, \ldots, w_N)$, such as a natural language sentence, where $w_i \in V$, a finite vocabulary. Let $\tilde{V} = V \cup \{\langle \text{end} \rangle\}$. Unlike conventional approaches with a fixed generation order, often left-to-right (or right-to-left), our goal is to build a sequence generator that generates these tokens in an order automatically determined by the sequence generator, without any extra annotation nor supervision of what might be a good order.

Binary tree generation policy. We propose a *binary tree generation policy* which does so by generating a word at an arbitrary position, then recursively generating words to its left and words to its right, yielding a binary tree like that shown in Figure 24. In order to learn such a policy with the non-monotonic generation framework (§12) we must first define the states and actions.

Each *state* $\mathbf{s} \in \mathcal{S}$ corresponds to a sequence of tokens from $\tilde{V}$. We interpret this sequence of tokens as a top-down traversal of a binary tree, where $\langle \text{end} \rangle$ terminates a subtree. The initial state $\mathbf{s}_0$ is the empty sequence. For example, in Figure 24, $\mathbf{s}_1 = \langle \text{are} \rangle$, $\mathbf{s}_2 = \langle \text{are}, \text{how} \rangle$, $\cdots$, $\mathbf{s}_4 = \langle \text{are}, \text{how}, ?, \langle \text{end} \rangle \rangle$. An *action* a is an element of $\tilde{V}$ which is deterministically appended to the state. Terminal states are those for which all subtrees have been $\langle \text{end} \rangle$'ed. If a terminal state $\mathbf{s}_T$ is reached, we have that $T = 2N + 1$, where N is the number of words (non-$\langle \text{end} \rangle$ tokens) in the tree. We use $\tau(t)$ to denote the level-order traversal index of the t-th node in an in-order traversal of a tree, so that $\langle a_{\tau(1)}, \ldots, a_{\tau(T)} \rangle$ corresponds to the sequence of discrete tokens generated. The final sequence returned is this, postprocessed by removing all $\langle \text{end} \rangle$'s. In Figure 24, τ maps from the numbers in the blue squares to those in the green squares.

The policy π is then a (possibly) stochastic mapping from states to actions, and we denote the probability of an action $a \in \tilde{V}$ given a state $\mathbf{s}$ as $\pi(a \mid \mathbf{s})$. The binary tree generation policy π's behavior decides which and whether words appear before and after the token of the parent node. Typically there are many unique binary trees with an in-order traversal equal to a sequence $\mathbf{y}$. Each of these trees has a different level-order traversal, thus the policy is capable of choosing from many different generation orders for $\mathbf{y}$, rather than a single predefined order. Note that left-to-right generation can be recovered if $\pi(\langle \text{end} \rangle \mid \mathbf{s}_t) = 1$ if and only if t is odd (or non-zero and even for right-to-left generation).

Objective. As in multiset prediction (§13) and graph-based dependency parsing (§14), we will use the non-monotonic generation objective, which involves minimizing the distance between the learned policy π_θ and an *oracle* policy π_* on states sampled from a *roll-in* policy π^{in} (Equation 52),

$$\arg\min_\theta \mathbb{E}_{\mathbf{x},\mathbf{y}\sim\mathcal{D}} \mathbb{E}_{\mathbf{s}_1,\ldots,\mathbf{s}_T\sim\pi^{\text{in}}} \sum_{t=1}^{T} \mathrm{D}_{\mathrm{KL}}\left(\pi_*(\cdot \mid \bar{\mathbf{s}}_t) \| \pi_\theta(\cdot \mid \mathbf{s}_t)\right), \tag{72}$$

where $\mathbf{x}$ is a sentence sampled from a dataset, and the states are obtained by sampling a sequence of edges from the roll-in policy. We now describe our choices for roll-in, valid actions, and oracle.

Roll-in. As we have seen in preceding sections, the *roll-in* policy determines the state distribution over which the learned policy π_θ is to be trained. In most formal analyses, the roll-in policy is a stochastic mixture of the learned policy π and the oracle policy π^*, ensuring that π is eventually trained on its own state distribution (Daumé III et al. 2009; Ross et al. 2011; Ross and Bagnell 2014; Chang et al. 2015). Despite this, *experimentally*, it has often been found that simply using the oracle's state distribution is optimal (Ranzato et al. 2016; Leblond et al. 2018). This is likely because the noise incurred early on in learning by using π's state distribution is not overcome by the benefit of matching state distributions, especially when the the policy class is sufficiently high capacity so as to be nearly realizable on the training data (Leblond et al. 2018). In preliminary experiments, we observed the same is true in our setting: simply rolling in according to the oracle policy yielded the best results experimentally. Therefore, despite the fact that this can lead to inconsistency in the learned model (Chang et al. 2015), all experiments use oracle roll-ins.

Valid actions. To use the non-monotonic generation framework, we must specify the valid actions which keep the trajectory on a path towards a correct prediction (i.e. a tree whose in-order traversal equals the ground truth sequence $\mathbf{y}$). The **key idea** is to define the valid actions at the first step as *all* words in $\mathbf{y}$, then after picking a valid action w, in a quicksort-esque manner, all words to the left of w in $\mathbf{y}$ are defined as the valid actions on the left, all words to the right are defined as the valid actions on the right, and the procedure repeats recursively.

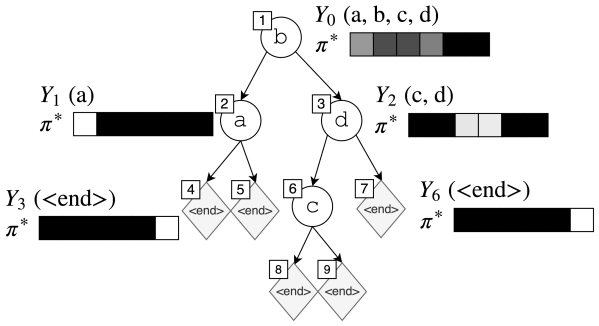


Fig. 25: A sampled tree for the sentence "a b c d" with an action space $\tilde{V}$ = (a,b,c,d,e,⟨end⟩), showing an oracle's distribution π^* and valid actions (consecutive subsequences) Y_t for $t \in \{0, 1, 2, 3, 6\}$. Each oracle distribution is depicted as 6 boxes showing $\pi^*(a_{t+1}|\mathbf{s}_t)$ (lighter = higher probability). After `b` is sampled at the root, two empty left and right child nodes are created, associated with valid actions (a) and (c, d), respectively. Here, π^* only assigns positive probability to tokens in Y_t.

Specifically, let $\mathbf{y}$ be the ground-truth output and $\mathbf{s}_t$ be the current state. We interpret the state $\mathbf{s}_t$ as a partial binary tree and a "current node" in that binary tree where the next prediction will go. A roll-in trajectory proceeds as a top-down level-order construction of a tree, with valid actions, denoted Y_t, defined at each state $\mathbf{s}_t$. At $\mathbf{s}_0$ (the root node), the valid actions contains all words, $Y_0 = \mathbf{y}$. When an action a is selected at s_t, this "splits" the sub-sequence $Y_t = (w'_1, \ldots, w'_{N'})$ into left and right sub-sequences, $\overleftarrow{Y}_t = (w'_1, \ldots, w'_{i-1})$ and $\overrightarrow{Y}_t = (w'_{i+1}, \ldots, w'_{N'})$, where i is the index of a in Y_t. (This split

may not be unique due to duplicated words in Y_t, in which case we choose a valid split arbitrarily.) These are "passed" to the left and right child nodes, respectively. This continues as the tree is descended, so that at each $\mathbf{s}_t$ the valid actions Y_t contain a consecutive subsequence $Y_t = (w'_j, \ldots, w'_k)$. For instance, in Figure 25, after sample b for the root, the valid actions (a, b, c, d) are split into (a) for the left child and (c, d) for the right child. When no tokens remain after a split, the valid actions are set to $(\langle\text{end}\rangle)$; see Y_3 and Y_6 in Figure 25.

Oracle policies. An oracle policy is a policy that places probability mass only over valid actions,

$$\pi_*(a \mid \bar{\mathbf{s}}_t) = \begin{cases} p_a & \text{if } a \in Y_t \\ 0 & \text{otherwise,} \end{cases} \tag{73}$$

where the oracle's state contains the valid actions, $\bar{\mathbf{s}}_t = (Y_t, \mathbf{s}_t)$, and the p_a's satisfy $p_a \geq 0$ and $\sum_{a \in Y} p_a = 1$. See Figure 25 for an example.

As in our investigation of parsing (§14), we consider the three oracles introduced in §12. The *uniform oracle* treats all possible generation orders that lead to the target sequence $\mathbf{y}$ as equally likely,

$$\pi_*^{\text{uniform}}(a \mid \bar{\mathbf{s}}_t) = \begin{cases} \frac{1}{|Y_t|} & a \in Y_t \\ 0 & \text{otherwise.} \end{cases}$$

As we discussed in (§12), the uniform oracle may be difficult for a learned policy to imitate since the oracle does not prefer any particular orders; intuitively, it may be too difficult for a policy to learn *all* valid generation orders. As a result, we consider the *coaching oracle* (Equation 50):

$$\pi_*^{\text{coaching}}(a \mid \bar{\mathbf{s}}_t) \propto \pi_*^{\text{uniform}}(a \mid \bar{\mathbf{s}}_t) \pi_\theta(a \mid \mathbf{s}_t).$$

The coaching oracle prefers actions according to the current learned policy, reinforcing the selection by the current policy if it is valid. The multiplicative nature of the coaching oracle gives rise to an issue, especially in the early stage of learning, as it does not encourage learning to explore a diverse set of generation orders. We thus design a mixture of the uniform and coaching policies, which we refer to as the *annealed coaching oracle*:

$$\pi_*^{\text{annealed}}(a|\bar{\mathbf{s}}_t) = \beta \pi_*^{\text{uniform}}(a \mid \bar{\mathbf{s}}_t) + (1 - \beta) \pi_*^{\text{coaching}}(a \mid \bar{\mathbf{s}}_t). \tag{74}$$

We anneal β from 1 to 0 over learning, on a linear schedule.

We also experiment with a *deterministic oracle* that corresponds to generating the target sequence from left to right: $\pi_*^{\text{left-right}}$ always selects the first un-produced word as the correct action, with probability 1. When the roll-in and oracle policies are both set to the left-to-right oracle $\pi_*^{\text{left-right}}$, the proposed approach recovers fixed-order maximum likelihood learning (Equation 8).

15.1.1 Neural network parameterization. We use a neural network to implement the binary tree generating policy, which takes as input a partial binary tree, or equivalently a sequence of nodes in this partial tree by level-order traversal, and outputs a distribution over the action set $\tilde{V}$.

LSTM policy. The first policy we consider is implemented as a recurrent network with long short-term memory (LSTM) units (Hochreiter and Schmidhuber 1997) by considering the partial binary tree as a flat sequence of nodes in a level-order traversal $(a_1, \ldots, a_t)$. The recurrent network encodes the sequence into a vector h_t and computes a categorical distribution over the action set:

$$\pi(a \mid \mathbf{s}_t) \propto \exp(u_a^\top h_t + b_a) \tag{75}$$

where u_a and b_a are weights and bias associated with a. This LSTM structure relies entirely on the linearization of a partial binary tree, and minimally takes advantage of the actual tree structure *or* the surface order. We did experiment with additionally conditioning π's action distribution on the *parent* of the current node in the tree, but preliminary experiments did not show gains.

Transformer policy. We additionally implement a policy using a transformer Vaswani et al. (2017). The level-order sequence $a_1, \ldots, a_t$ is again summarized by a vector h_t, here computed using a multi-head attention mechanism; see (§4.1.3) for background. As in the LSTM policy, the vector h_t is used to compute a categorical distribution over the action set (Equation 75).

Auxiliary ⟨end⟩ prediction. We also consider separating the action prediction into token ($a_i \in \mathcal{V}$) prediction and ⟨end⟩ prediction. The policy under this view consists of a categorical distribution over tokens (Equation 75) as well as an ⟨end⟩ predictor that parameterizes a Bernoulli distribution, $\pi_{\text{end}}(\langle\text{end}\rangle \mid \mathbf{s}_t) \propto \sigma(u_e^\top h_t + b_e)$, where $\pi_{\text{end}}(\langle\text{end}\rangle = 1 \mid \mathbf{s}_t)$ means a_t is ⟨end⟩, and a_t is determined by π according to (Equation 75) otherwise. At test time, we threshold the predicted ⟨end⟩ probability at a threshold τ. In our experiments, we only use this approach with the transformer policy.

15.2 Empirical Evaluation

We evaluate our non-monotonic text generation framework across four tasks. The first two are *unconditional* generation tasks: language modeling and non-monotonic text completion. Our analysis in these

tasks is primarily qualitative: we seek to understand what the non-monotonic policy is learing and how it compares to a left-to-right model. The second two tasks are *conditional* generation tasks: word reordering and machine translation.

15.2.1 Language modeling. We begin by considering generating samples from our model, trained as a language model. Our goal in this section is to qualitatively understand what our model has learned. It would be natural also to evaluate our model according to a score like perplexity. Unfortunately, unlike conventional autoregressive language models, it is intractable to compute the probability of a given sequence in the non-monotonic generation setting, as it requires us to marginalize out all possible binary trees that lead to the sequence.

Oracle	Novel	Unique	Tokens	Span	Bleu
left-right	17.8	97.0	11.9	1.00	47.0
uniform	98.3	99.9	13.0	1.43	40.0
annealed	93.1	98.2	10.6	1.31	56.2
validation	97.0	100	12.1	-	-

Table 32: Statistics computed over 10,000 sampled sentences (in-order traversals of sampled trees with ⟨*end*⟩ tokens removed) for policies trained on Persona-Chat. A sample is novel when it is not in the training set. Percent unique is the cardinality of the set of sampled sentences divided by the number of sampled sentences.

Setup. We use a dataset derived from the Persona-Chat (Zhang et al. 2018) dialogue dataset, which consists of multi-turn dialogues between two agents. Our dataset here consists of all unique persona sentences and utterances in Persona-Chat. We derive the examples from the same train, validation, and test splits as Persona-Chat, resulting in 133,176 train, 16,181 validation, and 15,608 test examples. Sentences are tokenized by splitting on spaces and punctuation. The training set has a vocabulary size of 20,090 and an average of 12.0 tokens per example. We use a uni-directional LSTM that has 2 layers of 1024 LSTM units. See Appendix B.2.1 for more details.

Basic statistics. We draw 10,000 samples from each trained policy (by varying the oracle) and analyze the results using the following metrics: percentage of novel sentences, percentage of unique, average number of tokens, average span size[26] and Bleu (Figure 32). We use Bleu to quantify the sample quality by computing the Bleu score of the samples using the validation set as reference, following (Yu et al. 2016; Zhu et al. 2018). We see that the non-monotonically trained policies generate many more novel sentences, and build trees that are bushy (span ≈ 1.3), but not complete binary trees. The policy trained with the annealed oracle is most similar to the validation data according to Bleu.

Content analysis. We investigate the content of the models in Table 33, which shows samples from policies trained with different oracles. Each of the displayed samples are not a part of the training set.

[26] The average span is the average number of children for non-leaf nodes excluding the special token ⟨end⟩, ranging from 1.0 (chain, as induced by the left-right oracle) to 2.0 (full binary tree).

We additionally examined word frequencies and part-of-speech tag frequencies, finding that the samples from each policy typically follow the validation set's word and tag frequencies.

Generation order. We analyze the generation order of our various models by inspecting the part-of-speech (POS) tags each model tends to put at different tree depths (i.e. number of edges from node to root). Figure 26 shows POS counts by tree depth, normalized by the sum of counts at each depth (we only show the four most frequent POS categories). We also show POS counts for the validation set's dependency trees, obtained with an off-the-shelf parser. Not surprisingly, policies trained with the uniform oracle tend to generate words with a variety of POS tags at each level. Policies trained with the annealed oracle on the other hand, learned to frequently generate punctuation at the root node, often either the sentence-final period or a comma, in an "easy first" style, since most sentences contain a period. Furthermore, we see that the policy trained with the annealed oracle tends to generate a pronoun before a noun or a verb (tree depth 1), which is a pattern that policies trained with the left-right oracle also learn. Nouns typically appear in the middle of the policy trained with the annealed oracle's trees. Aside from verbs, the annealed policy's trees, which place punctuation and pronouns near the root and nouns deeper, follow a similar structure as the dependency trees.

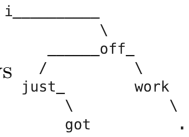
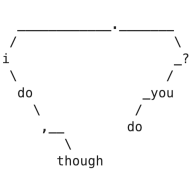
π^*	**Samples**	
left-right	∘ hey there , i should be ! ∘ not much fun . what are you doing ? ∘ not . not sure if you . ∘ i love to always get my nails done . ∘ sure , i can see your eye underwater while riding a footwork .	
uniform	∘ i just got off work . ∘ yes but believe any karma , it is . ∘ i bet you are . i read most of good tvs on that horror out . cool . ∘ sometimes , for only time i practice professional baseball . ∘ i am rich , but i am a policeman .	
annealed	∘ i do , though . do you ? ∘ i like iguanas . i have a snake . i wish i could win . you ? ∘ i am a homebody . ∘ i care sometimes . i also snowboard . ∘ i am doing okay . just relaxing , and you ?	

Table 33: Samples from unconditional generation policies trained on Persona-Chat for each training oracle. The first sample's underlying tree is shown.

15.2.2 Non-monotonic text completion. In *text completion*, where given a context $\mathbf{x}$ the model 'fills in' a completion $\mathbf{y}$ (§3.3.1). A major weakness of the conventional left-to-right autoregressive model is that it cannot be easily used to fill in missing parts of a sentence except at the end. This is especially true when the number of tokens per missing segment is not provided.

Our proposed approach, on the other hand, can naturally fill in missing segments in a sentence, which we call *non-monotonic text completion*. Using models trained as language models (§15.2.1), we can

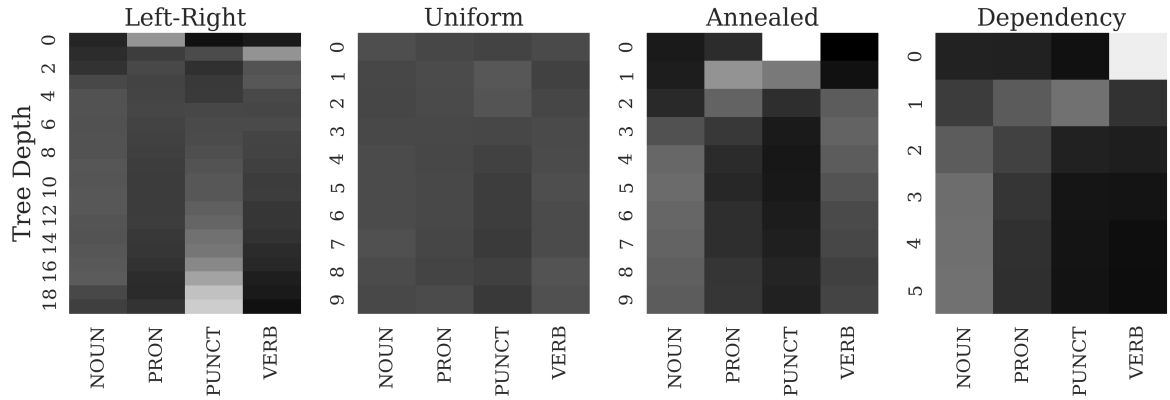


Fig. 26: POS tag counts by tree-depth, computed by tagging 10,000 sampled sentences. Counts are normalized across each row (depth), then the marginal tag probabilities are subtracted. Light values mean the probability of the tag occurring at that depth exceeds the prior probability of the tag occurring.

achieve this by initializing a binary tree with observed tokens in a way that they respect their relative positions. Generally, an initial tree with nodes $(w_i, \ldots, w_k)$ ensures that each w_j appears in the completed sentence, and that w_i appears at *some* position to the left of w_j in the completed sentence when w_i is a left-descendant of w_j (analogously for right-descendants).

For instance, the first example shown in Table 34 can be seen as the template "_____ favorite _____ food _____ ! _____" with variable-length missing segments. Concretely, we forward an action sequence $\mathbf{s}_t = (a_1, \ldots, a_t)$ corresponding to a tree containing $(w_i, \ldots, w_k)$ through the policy π_θ, then sample action sequences $(a_{t+1}, \ldots, a_T) \sim \pi_\theta$ beginning at $\mathbf{s}_t$. The resulting sequence $(a_1, \ldots, a_T)$ is a tree whose level order traversal is the completed sequence.

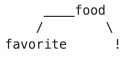
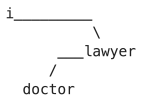
Initial Tree	Samples
	∘ lasagna is my **favorite food !** ∘ my **favorite food** is mac and cheese **!** ∘ what is your **favorite food** ? pizza , i love it **!** ∘ whats your **favorite food** ? mine is pizza **!** ∘ seafood is my **favorite** . and mexican **food !** what is yours ?
	∘ **hello** ! i like classical **music** . do you **?** ∘ **hello** , do you enjoy playing **music ?** ∘ **hello** just relaxing at home listening to fine **music** . you **?** ∘ **hello** , do you like to listen to **music ?** ∘ **hello** . what kind of **music** do you like **?**
	∘ **i** am a **doctor** or a **lawyer** . ∘ **i** would like to feed my **doctor** , i aspire to be a **lawyer** . ∘ **i** am a **doctor lawyer** . 4 years old . ∘ **i** was a **doctor** but went to a **lawyer** . ∘ **i** am a **doctor** since i want to be a **lawyer** .

Table 34: Non-monotonic text completion samples from a policy trained on Persona-Chat with the uniform oracle. The left column shows the initial seed tree. Seed words are in bold.

To quantify completion quality, we first create a collection of initial trees by randomly sampling three words (w_i, w_j, w_k) from each sentence $\mathbf{y} = (w_1, \ldots, w_N)$ from the Persona-Chat validation set. We then sample one completion for each initial tree and measure the BLEU of each sample using the validation set as

reference. According to BLEU, the policy trained with the annealed oracle sampled completions that were more similar to the validation data (BLEU 44.7) than completions from the policies trained with the uniform (BLEU 38.9) or left-to-right (BLEU 14.3) oracles. In Table 34, we present some sample completions using the policy trained with the uniform oracle. The completions illustrate a property of non-monotonic generation that is not available in left-to-right generation.

15.2.3 Word reordering. We first evaluate the non-monotonic models for conditional generation on the word reordering task, also known as bag translation (Brown et al. 1990) or linearization (Schmaltz et al. 2016). In this task, a sentence $\mathbf{y} = (w_1, \ldots, w_N)$ is given as an unordered collection $\mathbf{x} = \{w_1, \ldots, w_N\}$, and the task is to reconstruct $\mathbf{y}$ from $\mathbf{x}$. We assemble a dataset of $(\mathbf{x}, \mathbf{y})$ pairs using sentences $\mathbf{y}$ from the Persona-Chat sentence dataset (§15.2.1). We do not force the the non-monotonic policies to produce a particular permutation of the input, instead letting them learn this automatically.

Model. For encoding each unordered input $\mathbf{x} = \{w_1, \ldots, w_N\}$, we use a simple bag-of-words encoder: $f^{\text{enc}}(\{w_1, \ldots, w_N\}) = \frac{1}{N} \sum_{i=1}^{N} \text{emb}(w_i)$. We implement $\text{emb}(w_i)$ using an embedding layer followed by a linear transformation. The embedding layer is initialized with GloVe (Pennington et al. 2014) vectors and updated during training. As the policy (decoder) we use a flat LSTM with 2 layers of 1024 LSTM units. The decoder hidden state is initialized with a linear transformation of $f^{\text{enc}}(\{w_1, \ldots, w_N\})$.

Results. Figure 35 shows BLEU, F1 score, and exact match for policies trained with each oracle. The non-monotonic policies (uniform, annealed) outperform the left-right policy in F1 score (0.96 and 0.95 vs. 0.903). The policy trained using the annealed oracle also matches the left-right policy's performance in terms of BLEU score (46.0 vs. 46.3) and exact match (0.212 vs. 0.208). The model trained with the uniform policy does not fare as well on BLEU or exact match.

Easy-first analysis. Figure 27 shows the entropy of each model as a function of depth in the tree (normalized to fall in $[0, 1]$). The left-right-trained policy has high entropy on the first word and then drops dramatically as additional conditioning from prior context kicks in. The uniform-trained policy exhibits similar behavior. The annealed-trained policy, however, makes its highest confidence ("easiest") predictions at the beginning (consistent with Figure 26) and defers harder decisions until later.

15.2.4 Machine translation. Next we evaluate the non-monotonic models on machine translation, specifically the IWSLT'16 German $\rightarrow$ English (196k pairs) translation task. The data sets consist of TED talks. We use TED tst2013 as a validation dataset and tst-2014 as test.

We use a transformer policy, following the architecture of Vaswani et al. (2017). We use auxiliary $\langle\text{end}\rangle$ prediction by introducing an additional output head, after observing a low brevity penalty in preliminary

	Validation			Test		
Oracle	**Bleu**	**F1**	**EM**	**Bleu**	**F1**	**EM**
left-right	46.6	0.910	0.230	46.3	0.903	0.208
uniform	44.7	0.968	0.209	44.3	0.960	0.197
annealed	46.8	0.960	0.230	46.0	0.950	0.212

Table 35: Word Reordering results on PersonaChat, reported according to BLEU score, F1 score, and exact match.

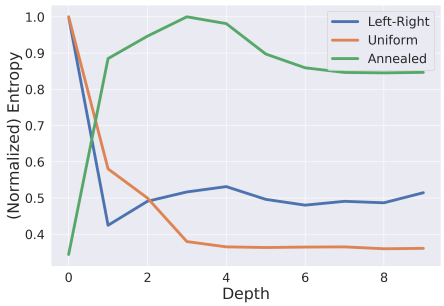


Fig. 27: Normalized entropy of $\pi(\cdot|s)$ as a function of tree depth for policies trained with each of the oracles. The annealing-trained policy, unlike the others, makes low entropy decisions early.

	Validation				Test			
Oracle	**Bleu** (BP)	**Meteor**	**YiSi**	**Ribes**	**Bleu** (BP)	**Meteor**	**YiSi**	**Ribes**
left-right	32.30 (0.95)	31.96	69.41	84.80	28.00 (1.00)	30.10	65.22	82.29
uniform	24.50 (0.84)	27.98	66.40	82.66	21.40 (0.86)	26.40	62.41	80.00
annealed	26.80 (0.88)	29.67	67.88	83.61	23.30 (0.91)	27.96	63.38	80.91
+tree-encoding	28.00 (0.86)	30.15	68.43	84.36	24.30 (0.91)	28.59	63.87	81.64
+⟨end⟩-tuning	29.10 (0.99)	31.00	68.81	83.51	24.60 (1.00)	29.30	64.18	80.53

Table 36: Machine translation results for different training oracles across four different evaluation metrics.

experiments. For the ⟨end⟩ prediction threshold τ we use 0.5, and also report a variant (+⟨end⟩ tuning) in which τ is tuned based on validation BLEU ($\tau = 0.67$). Finally, we report a variant which embeds each token by additionally encoding its path from the root (+tree-encoding) based on (Shiv and Quirk 2019). See Appendix B.2.1 for additional details.

Results. Results on validation and test data are in Table 36 according to four (very) different evaluation measures: BLEU, Meteor (Banerjee and Lavie 2005), YiSi (Lo 2018), and Ribes (Isozaki et al. 2010a). First focusing on the non-monotonic models, the annealed policy outperforms the uniform policy on all metrics, with tree-encoding yielding further gains. Adding ⟨end⟩ tuning to the tree-encoding model decreases the Ribes score but improves the other metrics, notably increasing the BLEU brevity penalty.

Compared to the best non-monotonic model, the left-to-right model has superior performance according to BLEU. As previously observed (Callison-Burch et al. 2006; Wilks 2008), BLEU tends to strongly prefer left-to-right language models because it focuses on getting a large number of 4-grams correct. The other three measures of translation quality are significantly less sensitive to exact word order and focus more on whether the "semantics" is preserved (for varying definitions of "semantics"). For those, we see that the best annealed model is more competitive, typically within one percentage point of left-to-right.

15.3 Discussion

We described a method that uses the non-monotonic generation framework (§12) to learn policies capable of generating text in non-monotonic orders that fall out naturally as the result of learning. We explored several different oracle models for imitation, and found that an annealed "coaching" oracle performed best, and learned a "best-first" strategy for language modeling, where it appears to significantly outperform alternatives. On a word re-ordering task, we found that this approach essentially ties left-to-right decoding, a rather promising finding given the decades of work on left-to-right models. In a machine translation setting, we found that the model learns to translate in a way that tends to preserve meaning but not n-grams.

16 Discussion and Future Directions

In this part of the thesis, we focused on sequential generation without a pre-specified generation order. We introduced a learning framework based on imitation learning that yields generation policies which learn to determine their own input-dependent generation orders. The key idea behind the framework, called *non-monotonic generation* (§12), is to define a *set* of valid actions at each generation step, then imitate an oracle policy which places probability only on these valid actions. We saw how the framework can be used for generating multisets (§13), dependency trees (§14), and text (§15). Our investigations suggest several avenues for future work.

One avenue deals with settings that our non-monotonic framework does not currently cover. In multiset prediction, we considered only discrete-valued elements. While this setting is still of interest (e.g. Locatello et al. (2020)), a more generic setting involves predicting (multi-)sets with vector-valued elements (Kosiorek et al. 2020), which would require extensions to our framework.

For tree generation, we treated the tree as an edge set, relied on the decoding algorithm to enforce tree constraints, and used an architecture that is agnostic to the tree structure. Future work could incorporate node labels and evaluate the effect of architectures that leverage tree structure. One promising application is modeling symbolic mathematical statements, which may be represented by expression trees (Piotrowski et al. 2019; Lample and Charton 2020); little investigation has been done on whether the generation order or tree representation impacts generalization and sample efficiency.

In general, graphs do not come with a specified generation order, and face the additional complication of needing to deal with isomorphic structures under node relabeling, making graph generation a challenging application area for non-monotonic generation. Several sequential methods have been developed for graphs, which rely on hand-specified generation orders (Li et al. 2018; You et al. 2018; Liao et al. 2019). Though these could be sufficient for practical purposes, it is of scientific interest to develop a method that does not require a specified generation order.

For text – and generally, sequences – a natural direction is developing models that can consider *any* generation order (rather than those considered by our binary tree formulation), and seeking to exceed performance of left-to-right models. Since our work presented in (§15), several non-monotonic methods have been developed (e.g. Stern et al. (2019); Gu et al. (2019); Emelianenko et al. (2019); Mansimov et al. (2019)). Typically machine translation is used as the gold-standard for evaluating these methods, where the non-monotonic methods either underperform or show marginal gains. It is reasonable to expect left-to-right generation to be strong in machine translation, but it is also reasonable to suspect that there may be other tasks that might benefit from alternative generation orders more than machine translation.

Some potential tasks might include language tasks such as table-to-text (Chen et al. 2020), but also non-language tasks such as music (Huang et al. 2019; Roberts et al. 2018; Payne 2019) or image (Oord et al. 2016; Jain et al. 2020) generation.

Rather than aiming to outperform left-to-right models, one can further study or leverage the unique properties of non-monotonic models. Non-monotonic text completion (§3.3.1) offers a setting that is natural for non-monotonic models, while training with multiple generation orders has shown promise as a pre-training objective (Yang et al. 2019). Finally, an intriguing finding of our investigation was that the 'bottleneck' of choosing a generation order can induce a latent structure in the data (§15.2.1). Exploring this further by modeling the generation order as a latent variable, or leveraging the latent structure for planning are interesting directions for future investigation.

Part IV

Appendices

A Derivations & Proofs

A.1 Neural Text Generation (Part II)

A.1.1 Theory: Inconsistency

Lemma 1 *If a sequence distribution $p(\mathbf{y})$ is consistent, $p(|\mathbf{y}| = \infty \mid \mathbf{x}) = 0$ for any probable context $\mathbf{x}$.*

Proof. Suppose there exists a probable context $\tilde{\mathbf{x}}$ such that $p(|\mathbf{y}| = \infty \mid \tilde{\mathbf{x}}) > 0$. Then

$$\begin{aligned} p_\theta(|\mathbf{y}| = \infty) &= \mathbb{E}\left[p_\theta(|\mathbf{y}| = \infty \mid \mathbf{x})\right] \\ &\geq p(\tilde{\mathbf{x}}) p_\theta(|\mathbf{y}| = \infty \mid \tilde{\mathbf{x}}) > 0, \end{aligned}$$

which contradicts the consistency of $p(\mathbf{y})$.

Lemma 2 *A recurrent language model p_θ is consistent if $\|h_t\|_p$ is uniformly bounded for some $p \geq 1$.*

Proof. Let $B > 0$ be an upper bound such that $\|h_t\|_p < B$ for all t. Let q be the conjugate of p satisfying $1/p + 1/q = 1$. Then we have from Hölder's inequality, for all $v \in V$ and t,

$$u_v^\top h_t \leq \|u_v^\top h_t\|_1 \leq \|h_t\|_p \|u_v\|_q < B u^+,$$

where $u^+ = \max_{v \in V} \|u_v\|_q$. Note that

$$\begin{aligned} \log \sum_{v \in V} e^{u_v^\top h_t + c_v} &\leq \log \left(\max_{v \in V} e^{u_v^\top h_t + c_v} \times |V| \right) \\ &\leq \max_{v \in V} \{u_v^\top h_t + c_v\} + \log |V| \\ &< B u^+ + c^+ + \log |V|, \end{aligned}$$

where $c^+ = \max_{v \in V} c_v$. For a given $y_{<t}$ and context $\mathbf{x}$,

$$\begin{aligned} &\log p_\theta(\langle \text{eos} \rangle \mid y_{<t}, \mathbf{x}) \\ =&(u_{\langle \text{eos} \rangle}^\top h_t + c_{\langle \text{eos} \rangle}) - \log \sum_{v \in V} e^{u_v^\top h_t + c_v} \\ >&(-B u^+ + c_{\langle \text{eos} \rangle}) - (B u^+ + c^+ + \log |V|) > -\infty, \end{aligned}$$

and it follows that $p_\theta(\langle \text{eos} \rangle \mid y_{<t}, \mathbf{x}) > \xi > 0$ for some strictly positive constant ξ. Then

$$\begin{aligned} p_\theta(|\mathbf{y}| = \infty) &= \lim_{t \to \infty} p_\theta(|\mathbf{y}| > t) \\ &= \lim_{t \to \infty} \mathbb{E}\left[p_\theta(|\mathbf{y}| > t \mid \mathbf{x})\right] \\ &= \mathbb{E}\left[\lim_{t \to \infty} p_\theta(|\mathbf{y}| > t \mid \mathbf{x})\right] \\ &\leq \mathbb{E}\left[\lim_{t \to \infty} (1 - \xi)^t\right] = 0, \end{aligned}$$

and hence p_θ is consistent.

Theorem 2 *Suppose a recurrent language model p_θ has uniformly bounded $\|h_t\|_p$ for some $p \geq 1$. If a decoding algorithm $\mathcal{F}$ satisfies $q_\mathcal{F}(\langle eos \rangle \mid y_{<t}, \mathbf{x}) \geq p_\theta(\langle eos \rangle \mid y_{<t}, \mathbf{x})$ for every prefix $y_{<t}$ and context $\mathbf{x}$, then the decoding algorithm $\mathcal{F}$ is consistent with respect to the model p_θ.*

Proof. By Lemma 2 the model p_θ is consistent and $p_\theta(\langle \text{eos} \rangle \mid y_{<t}, \mathbf{x}) > \delta$ for some positive value δ. Thus, $q_\mathcal{F}(\langle \text{eos} \rangle \mid y_{<t}, \mathbf{x}) \geq p_\theta(\langle \text{eos} \rangle \mid y_{<t}, \mathbf{x}) > \delta$. For $t \geq 1$,

$$\begin{aligned} q_\mathcal{F}(|\mathbf{y}| > t \mid \mathbf{x}) &= q_\mathcal{F}(y_1 \neq \langle \text{eos} \rangle, \cdots, y_t \neq \langle \text{eos} \rangle \mid \mathbf{x}) \\ &\leq (1 - \delta)^t. \end{aligned}$$

Taking the limit $t \to \infty$ and expectation over $\mathbf{x}$, we have

$$\begin{aligned} q_\mathcal{F}(|\mathbf{y}| = \infty) &= \mathbb{E}_\mathbf{x}\left[\lim_{t \to \infty} q_\mathcal{F}(|\mathbf{y}| > t \mid \mathbf{x})\right] \\ &\leq \lim_{t \to \infty} (1 - \delta)^t = 0, \end{aligned}$$

from which the decoding algorithm is consistent.

Theorem 4.2 *Greedy decoding is consistent with respect to any self-terminating recurrent LM.*

Proof. Let $p_t^{\langle \text{eos} \rangle}$ denote $p_\theta(\langle \text{eos} \rangle \mid y_{<t}, \mathbf{x})$ and $a_t^{\langle \text{eos} \rangle}$ denote $u_{\langle \text{eos} \rangle}^\top h_t + c_{\langle \text{eos} \rangle}$. By Definition 16 we have

$$\begin{aligned} p_t^{\langle \text{eos} \rangle} &= 1 - \sigma(a_t^{\langle \text{eos} \rangle})(1 - p_{t-1}^{\langle \text{eos} \rangle}) \\ &= 1 - \prod_{t'=0}^{t} \sigma(a_{t'}^{\langle \text{eos} \rangle}) \geq 1 - (1 - \epsilon)^{t+1}. \end{aligned}$$

Take $B = -\log 2 / \log(1 - \epsilon)$. We then have $p_t^{\langle \text{eos} \rangle} > 1/2$ for all $t > B$, which implies that $\langle \text{eos} \rangle$ is always the most probable token after time step B. Hence, the sequence length is less than B with probability 1.

Theorem 4.3 *Beam search with width k is consistent with respect to any self-terminating recurrent language model.*

Proof. Let $S(\rho)$ be the size-k set of sequences kept by $\mathcal{F}_{\text{beam}-k}$ that start with a prefix ρ.

Take $B = -\log 2/\log(1-\epsilon)$ as in the proof of Theorem 4.2. Suppose that there exists at least one prefix $\hat{\rho} \in P_B^{\text{top}}$ which does not end with $\langle\text{eos}\rangle$.

We first want to show that $\hat{\rho}$ induces at most k more steps in beam search with width k, that is, $\mathbf{y} \in S(\hat{\rho})$ implies $|\mathbf{y}| \leq B + k$.

We know from the proof of Theorem 4.2 that an STRLM p_θ satisfies: for any context $\mathbf{x}$ and $v \in V \setminus \{\langle\text{eos}\rangle\}$,

$$p_\theta(\langle\text{eos}\rangle \mid \hat{\rho}, \mathbf{x}) > p_\theta(v \mid \hat{\rho}, \mathbf{x}).$$

For any subsequence $y = (y_1, \ldots, y_l)$ with $y_1 \neq \langle\text{eos}\rangle$,

$$\begin{aligned} p_\theta(\hat{\rho} \circ y \mid \hat{\rho}, \mathbf{x}) &= \prod_{i=1}^{l} p_\theta(y_i \mid \hat{\rho} \circ y_{<i}, C) \\ &\leq p_\theta(y_1 \mid \hat{\rho}, \mathbf{x}) \\ &< p_\theta(\langle\text{eos}\rangle \mid \hat{\rho}, \mathbf{x}). \end{aligned}$$

Thus, $\hat{\rho} \circ \langle\text{eos}\rangle$ is the most probable sequence among sequences starting with the prefix $\hat{\rho}$, and it follows that $\hat{\rho} \circ \langle\text{eos}\rangle \in S(\hat{\rho})$.

Thus, in $S(\hat{\rho})$, there are $(k-1)$ sequences starting with $\hat{\rho} \circ v$ for $v \in V \setminus \{\langle\text{eos}\rangle\}$. By the same argument, at each step at least one sequence ending with $\langle\text{eos}\rangle$ is added to $S(\hat{\rho})$, and therefore at time step $(B+k)$, k sequences ending with $\langle\text{eos}\rangle$ are in $S(\hat{\rho})$.

Note that the result set S by $\mathcal{F}_{\text{beam}-k}$ (Definition 2.11) satisfies

$$S \subseteq \bigcup_{\rho \in P_B^{\text{top}}} S(\rho).$$

Since each $\rho \in P_B^{\text{top}}$ induces sequences of length at most $B + k$, we have

$$p_\theta(|\mathbf{y}| > B + k \mid \mathbf{x}) = 0.$$

Taking the expectation over $\mathbf{x}$ yields consistency of the model p_θ.

A.1.2 Learning: Unlikelihood

A.1.2.1 Unlikelihood gradient. Let y_t^* be the true next-token (index $i^* \in \mathcal{V}$) at step t, and let y_{neg} be a negative candidate (index i_{neg}). Let $p = p(y_t|y_{<t}, \mathbf{x}) \in \mathbb{R}^{\mathcal{V}}$ be the output of softmax(a) where $a \in \mathbb{R}^{\mathcal{V}}$.

Denote the probability of an element $i \in \{1, \ldots, V\}$ as $p_i = p(y_t^i|y_{<t})$, and let p_*, p_{neg}, and $\tilde{p}_i$ be probabilities of the true next-token, negative-candidate token, and any other token with $i \notin \{i^*, \bar{i}\}$.

The (negative) token-level loss with a single candidate is,

$$\mathcal{L}_t = \log p(y_t^*|y_{<t}) + \alpha \cdot \log(1 - p(y_{\text{neg}}|y_{<t})),$$

and its gradient with respect to a logit $a_i \in \mathbb{R}$ is:

$$\frac{\partial \mathcal{L}}{\partial p_i} \frac{\partial p_i}{\partial a_i} = (\mathbb{I}[i = i^*] - p_i) - \alpha \frac{p_{\text{neg}}}{1 - p_{\text{neg}}} (\mathbb{I}[i = i_{\text{neg}}] - p_i).$$

We consider the gradient when i is the true next-token, a negative-candidate, and any other token.

True Next-Token ($i = i^*$)

$$\begin{aligned}\frac{\partial \mathcal{L}}{\partial p_*} \frac{\partial p_*}{\partial a_{i^*}} &= (1 - p_*) - \alpha \frac{p_{\text{neg}}}{1 - p_{\text{neg}}}(0 - p_*) \\ &= 1 - p_*(1 - \alpha \frac{p_{\text{neg}}}{1 - p_{\text{neg}}}).\end{aligned}$$

Negative Candidate ($i = i_{\text{neg}}$)

$$\begin{aligned}\frac{\partial \mathcal{L}}{\partial p_{\text{neg}}} \frac{\partial p_{\text{neg}}}{\partial a_{\text{neg}}} &= (0 - p_{\text{neg}}) - \alpha \frac{p_{\text{neg}}}{1 - p_{\text{neg}}}(1 - p_{\text{neg}}) \\ &= -p_{\text{neg}}(1 + \alpha).\end{aligned}$$

Other Token ($i \notin \{i^*, i_{\text{neg}}\}$)

$$\begin{aligned}\frac{\partial \mathcal{L}}{\partial \tilde{p}_i} \frac{\partial \tilde{p}_i}{\partial a_i} &= (0 - \tilde{p}_i) - \alpha \frac{p_{\text{neg}}}{1 - p_{\text{neg}}}(0 - \tilde{p}_i) \\ &= -\tilde{p}_i(1 - \alpha \frac{p_{\text{neg}}}{1 - p_{\text{neg}}}).\end{aligned}$$

Combining the three cases above, we get:

$$\nabla \mathcal{L}_a = y^* - m \odot p,$$

where $y^* \in \{0,1\}^V$ is 1 at index i^* and 0 otherwise, and $m \in \mathbb{R}^V$ is:

$$m_i = \begin{cases} (1 - \alpha \frac{p_{\text{neg}}}{1-p_{\text{neg}}}) & i \neq i_{\text{neg}} \\ (1+\alpha) & i = i_{\text{neg}}. \ \square \end{cases} \tag{76}$$

A.1.2.2 Unlikelihood gradient – multiple candidates. In general, the token-level unlikelihood objective considers multiple negative candidates (i.e. $|\mathcal{C}_t| \geq 1$):

$$\mathcal{L}^t_{\text{ULE-token}}(p_\theta(\cdot|y_{<t}, \mathbf{x}), \mathcal{C}^t) = -\alpha \cdot \sum_{y_c \in \mathcal{C}^t} \log(1 - p_\theta(y_c|y_{<t}, \mathbf{x})) - \log p_\theta(y_t|y_{<t}, \mathbf{x}).$$

We regroup the token-level objective to be a weighted sum of per-candidate losses:

$$-\mathcal{L}^t_{\text{UL-token}} = \frac{1}{|\mathcal{C}^t|} \sum_{y_c \in \mathcal{C}^t} \big(\log p_\theta(y_t|y_{<t}, \mathbf{x}) + \alpha_c \cdot \log(1 - p_\theta(y_c|y_{<t}, \mathbf{x})) \big),$$

where $\alpha_c = \alpha \cdot |\mathcal{C}^t|$. Now the gradient takes the same form as Eqn. 76, but with α_c in place of α.

A.2 Non-Monotonic Generation (Part III)

A.2.1 Coaching oracle KL-divergence.

$$\mathrm{D}_{\mathrm{KL}}\left(\pi_*^{\text{coaching}} \| \pi_\theta\right) \leq \mathrm{D}_{\mathrm{KL}}\left(\pi_*^{\text{uniform}} \| \pi_\theta\right).$$

Proof: Let $\mathcal{A}_t$ denote the valid actions at time t, and π_u denote the uniform oracle. For the uniform oracle, we have

$$\begin{aligned} \mathrm{D}_{\mathrm{KL}}\left(\pi_*^{\text{uniform}} \| \pi_\theta\right) &= \sum_{a \in \mathcal{A}_t} \pi_u \log\left(\frac{\pi_u}{\pi_\theta}\right) \\ &= \frac{1}{|\mathcal{A}_t|} \sum_{a \in \mathcal{A}_t} \left[\log \frac{1}{|\mathcal{A}_t|} - \log \pi_\theta(a)\right] \\ &= -\log|\mathcal{A}_t| - \frac{1}{|\mathcal{A}_t|} \sum_{a \in \mathcal{A}_t} \log \pi_\theta(a). \end{aligned}$$

Now we denote the coaching as π_c and write it as,

$$\begin{aligned}
\pi_c(a) &= \frac{\pi_u(a)\pi_\theta(a)}{\sum_{a' \in \mathcal{A}_t} \pi_u(a')\pi_\theta(a')} \\
&= \frac{\frac{1}{|\mathcal{A}_t|}\pi_\theta(a)}{\frac{1}{|\mathcal{A}_t|}\sum_{a' \in \mathcal{A}_t} \pi_\theta(a')} \\
&= \frac{\pi_\theta(a)}{\sum_{a' \in \mathcal{A}_t} \pi_\theta(a')} \\
&= \frac{\pi_\theta(a)}{Z(\theta)}.
\end{aligned}$$

Then we have,

$$\begin{aligned}
\mathrm{D}_{\mathrm{KL}}\left(\pi_*^{\text{coaching}} \| \pi_\theta\right) &= \sum_{a \in \mathcal{A}_t} \pi_c \log\left(\frac{\pi_c}{\pi_\theta}\right) \\
&= \sum_{a \in \mathcal{A}_t} \frac{\pi_\theta(a)}{Z(\theta)} \log\left(\frac{\pi_\theta(a)/Z(\theta)}{\pi_\theta(a)}\right) \\
&= -\sum_{a \in \mathcal{A}_t} \frac{\pi_\theta(a)}{Z(\theta)} \log\left(Z(\theta)\right) \\
&= -\frac{\log Z(\theta)}{Z(\theta)} \underbrace{\sum_{a \in \mathcal{A}_t} \pi_\theta(a)}_{Z(\theta)} \\
&= -\log Z(\theta) \\
&= -\log\left(\sum_{a \in \mathcal{A}_t} \pi_\theta(a)\right) \\
&= -\log|\mathcal{A}_t| - \log\left(\frac{1}{|\mathcal{A}_t|}\sum_{a \in \mathcal{A}_t} \pi_\theta(a)\right) \\
&\leq -\log|\mathcal{A}_t| - \frac{1}{|\mathcal{A}_t|}\sum_{a \in \mathcal{A}_t} \log \pi_\theta(a) \\
&= \mathrm{D}_{\mathrm{KL}}\left(\pi_*^{\text{uniform}} \| \pi_\theta\right)
\end{aligned}$$

where the inequality invokes Jensen's inequality on the convex function $f(x) = -\log(x)$. □

A.2.2 Dependency parsing – sequential valid decoder. We wish to sequentially sample $E_1, E_2, \ldots, E_T$ from score matrices $S_1, S_2, \ldots, S_T$, respectively, such that $E = \bigcup_t E_t$ is a dependency tree. A dependency tree must satisfy:

1. The root node has no incoming edges.

2. Each non-root node has exactly one incoming edge.

3. There are no duplicate edges.

4. There are no self-loops.

5. There are no cycles.

We first consider predicting one edge per step $|E_t| = 1$, then address the case $|E_t| \geq 1$.

A.2.2.1 One Edge Per Step Let $x = x_0, x_1, \ldots, x_N$ where x_0 is a root node. We define a function $f_{\text{valid}}(S_t, E_{<t}) \rightarrow (i, j)$ which chooses the highest scoring edge (i, j) such that $E_{<t} \cup \{(i, j)\}$ is a dependency tree, given edges $E_{<t}$ and scores S_t. We represent $E_{<t}$ as an adjacency matrix $A_{<t}$, and implement $f_{\text{valid}}(S_t, A_{<t})$ by masking S_t to yield scores $\tilde{S}$ that satisfy (1-5) as follows:

1. $\tilde{S}_{\cdot,0} = -\infty$

2. $A_{i,j} = 1$ implies $\tilde{S}_{\cdot,j} = -\infty$

3. $A_{i,j} = 1$ implies $\tilde{S}_{i,j} = -\infty$

4. $\tilde{S}_{i,i} = -\infty$ for all i

5. $R_{i,j} = 1$ implies $\tilde{S}_{j,i} = -\infty$, where $R \in \{0, 1\}^{N \times N}$ is the reachability matrix (transitive closure) of A. That is, $R_{i,j} = 1$ when there is a directed path from i to j. [27]

The selected edge is then $\arg\max_{(i,j)} \tilde{S}_{i,j}$.

A full tree is decoded by calling f_{valid} for T steps, using the current step scores S_t and an adjacency matrix $A_{<t} = \bigcup_{t'=1}^{t-1} \{(i, j)_{t'}\}$.

A.2.2.2 Multiple Edges Per Step To decode multiple edges per step, i.e. $|E_t| \geq 1$, we propose to repeatedly call f_{valid}, adding the returned edge to the adjacency matrix after each call, and stopping once the returned edge's score is below a pre-defined threshold τ.

B Experimental Details

B.1 Neural Text Generation (Part II)

B.1.1 Theory: Inconsistency Experiments 1 and 2. Each model is trained on a single Nvidia P40 GPU for up to 100 epochs, stopping when validation perplexity does not decrease for 10 consecutive epochs. A grid search was performed on hidden size $\in \{\boldsymbol{256}, 512, 1024\}$, dropout $\in \{0.1, \mathbf{0.3}, \textit{0.5}\}$, and embedding weight tying $\in$ {True, ***False***}. The values selected for the LSTM-RNN are shown in bold, and for the tanh-RNN are shown in italics.

[27] The reachability matrix R can be computed with batched matrix multiplication as $\sum_{k=1}^{t} A^k$ where t is the maximum path length; other methods could potentially improve speed.

B.1.2 Learning: Unlikelihood

B.1.2.1 Dialogue pretraining. Following previous work (Humeau et al. 2019), we pre-train our model on dialogue data, using a previously existing Reddit dataset extracted and obtained by a third party and made available on pushshift.io, training to generate a comment conditioned on the full thread leading up to the comment, spanning $\sim 2200M$ training examples. Our Transformer model consists of an 8 layer encoder, 8 layer decoder with 512-dimensional embeddings and 16 attention heads, and is based on the ParlAI implementation of Miller et al. (2017). The model was trained with a batch size of 3072 sequences for approximately 3M updates using a learning rate of 5e-4, and an inverse square root scheduler. This pre-training took approximately two weeks using 64 NVIDIA V100s.

B.1.3 Data: Dialogue Natural Language Inference

B.1.3.1 Human evaluation. We use ParlAI Miller et al. (2017) which integrates with Amazon Mechanical Turk for human evaluation. A human annotator is paired with a model, and each is randomly assigned a persona from 1,155 persona sets. The human and model are then asked to make a conversation of at least either five or six turns (randomly decided). After the conversation, the annotator assigns three scores to the conversation, described below. Each annotator is allowed to participate in at most ten conversations per model, and we collect 100 conversations per model. We evaluate the key-value memory network without re-ranking (**KV-Mem**), and with re-ranking (**KV-Mem + NLI**).

Following a conversation, an annotator is shown the conversation and the model's persona, and assigns three scores: an overall score of how well the model represented its persona ($\{1,2,3,4,5\}$), a marking of each model utterance that was consistent with the model's persona ($\{0,1\}$), and a marking of each model utterance that contradicted a previous utterance or the model's persona ($\{0,1\}$).

We use Bayesian calibration to adjust for annotator bias, following Kulikov et al. (2019). We assume a model with observed scores S_{ij} and latent variables M_i for the unobserved score of model i and B_j for the bias of annotator j. We then estimate the posterior mean and variance for the unobserved scores given the observed scores. We use Pyro Bingham et al. (2018) and the no-u-turn sampler Hoffman and Gelman (2014) for posterior inference. Refer to Kulikov et al. (2019) for further details on the Bayesian calibration.

B.1.3.2 Schema. Relation t ypes: place_origin, live_in_citystatecountry, live_in_general, nationality, employed_by_company, employed_by_general, has_profession, previous_profession, job_status, teach, school_status, has_degree, attend_school, like_general, like_food, like_drink, like_animal, like_movie, like_music, like_read, like_sports, like_watching, like_activity, like_goto, dislike, has_hobby, has_ability, member_of, want_do,

want_job, want, favorite_food, favorite_color, favorite_book, favorite_movie, favorite_music, favorite_music_artist, favorite_activity, favorite_drink, favorite_show, favorite_place, favorite_hobby, favorite_season, favorite_animal, favorite_sport, favorite, own, have, have_pet, have_sibling, have_children, have_family, have_vehicle, physical_attribute, misc_attribute, has_age, marital_status, gender, other.

Additional triples with a not_have relation were extracted using a dependency tree pattern.

Entity categories: ability, activity, animal, color, citystate, country, company, cuisine, degree_type, drink, family, food, gender, general_location, job_status, language, marital, media_genres, media_other, movie_title, music_artist, music_genre, music_instrument, noun, number, organization, person, person_attribute, person_label, personality_trait, profession, read_author, read_genre, read_title, read_other, school_name, school_status, school_type, season, sport_type, subject, time, vehicle, location, other.

B.1.3.3 Relation swaps. Relation swaps for contradictions include `(have_*, not_have)`,
`(own, not_have)`,
`(has_hobby, not_have)`,
`(like_*, dislike)`,
`(favorite_*, dislike)`.

Neutral relation swaps include `(have_x, have_y)`, e.g. `have_pet`, `have_sibling`. Additional `(have_* A, not_have B)` swaps were defined for entities A which are a super-type of B, namely (A,B) pairs ({pet, animal}, {dog, cat}), ({sibling}, {brother, sister}), ({child, kid}, {son, daughter}), ({vehicle}, {car, truck}); this includes sentence pairs such as "i have a sibling", "i do not have a sister". Similarly, `(not_have B, have_* A)` swaps were defined using the (A, B) pairs above.

B.1.3.4 Entity swaps. For contradictions, swapping entities for the following relation types was assumed to yield a contradiction:

attend_school, employed_by_company, employed_by_general, favorite_animal, favorite_book, favorite_color, favorite_drink, favorite_food, favorite_hobby, favorite_movie, favorite_music, favorite_music_artist, favorite_place, favorite_season, favorite_show, favorite_sport, gender, has_profession, job_status, live_in_citystatecountry, marital_status, nationality, place_origin, previous_profession, school_status, want_job.

Additionally, for `physical_attribute`, `misc_attribute`, or `other` relations, an entity swap was done using all WordNet antonym pairs in the personality_trait and person_attribute entity categories, as well as the swaps ({blonde}, {brunette}), ({large}, {tiny}), ({carnivore, omnivore}, {vegan, vegetarian}), ({depressed}, {happy, cheerful}), ({clean}, {dirty}) where each entity in the left set is swapped with each entity in the right set.

B.2 Non-Monotonic Generation (Part III)

B.2.1 Sequences: Binary Tree Generation Policy

B.2.1.1 Language modeling and word reordering. The decoder is a 2-layer LSTM with 1024 hidden units, dropout of 0.0, based on a preliminary grid search of $n_{\text{layers}} \in \{1, 2\}, n_{\text{hidden}} \in \{512, 1024, 2048\}$, dropout $\in \{0.0, 0.2, 0.5\}$. Word embeddings are initialized with GloVe vectors and updated during training. All presented word reordering results use greedy decoding.

For π_*^{annealed}, β is linearly annealed from 1.0 to 0.0 at a rate of 0.05 each epoch, after a burn-in period of 20 epochs in which β is not decreased. We use greedy decoding when π_*^{coaching} is selected at a roll-in step; we did not observe significant performance variations with stochastically sampling from π_*^{coaching}. These settings are based on a grid search of $\beta_{\text{rate}} \in \{0.01, 0.05\}, \beta_{\text{burn-in}} \in \{0, 20\}$, coaching-rollin $\in$ {greedy, stochastic} using the model selected based on the preliminary grid search above.

Each model was trained on a single GPU using a maximum of 500 epochs, batch size of 32, Adam optimizer, gradient clipping with maximum ℓ_2-norm of 1.0, and a learning rate starting at 0.001 and multiplied by a factor of 0.5 every 20 epochs. For evaluation we select the model state which had the highest validation BLEU score, which is evaluated after each training epoch.

For language modeling, we use the same settings as word reordering, except we always use stochastic sampling from π^*_{coaching} during roll-in. For evaluation we select the model state at the end of training.

B.2.1.2 Machine translation. We use the default Moses tokenizer script (Koehn et al. 2007) and segment each word into a subword using BPE (Sennrich et al. 2015) creating 40k tokens for both source and target. Similar to (Bahdanau et al. 2015), during training we filter sentence pairs that exceed 50 words.

Transformer policy. The transformer policy uses 4 layers, 4 attention heads, hidden dimension 256, feed-forward dimension 1024, and is trained with batch-size 32 and a learning rate $1e^{-5}$. For this model and experiment, we define an epoch as 1,000 model updates. The learning rate is divided by a factor of 1.1 every 100 epochs. For π_*^{annealed}, β is linearly annealed from 1.0 to 0.0 at a rate of 0.01 each epoch, after a burn-in period of 100 epochs. We compute metrics after each validation epoch, and following training we select the model with the highest validation BLEU.

Loss with auxiliary ⟨end⟩ predictor A binary cross-entropy loss is used for the ⟨end⟩ predictor for all time-steps, so that the total loss is $\mathcal{L}_{\text{bce}}(\pi_*, \pi_{\text{end}}) + \mathcal{L}_{\text{KL}}(\pi_*, \pi)$. For time-steps in which ⟨end⟩ is sampled, $\mathcal{L}_{\text{KL}}$ is masked, since the policy's token distribution is not used when a_t is ⟨end⟩. $\mathcal{L}_{\text{KL}}$ is averaged over time by summing the loss from unmasked time-steps, then dividing by the number of unmasked time-steps.

Tree position encodings. We use an additional *tree position encoding*, based on (Shiv and Quirk 2019), which may make it easier for the policy to identify and exploit structural relationships in the partially decoded tree. Each node is encoded using its path from the root, namely a sequence of left or right steps from parent to child. Each step is represented as a 2-dimensional binary vector ($[0, 0]$ for the root, $[1, 0]$ for left and $[0, 1]$ for right), so that the path is a vector $e(a_i) \in \{0, 1\}^{2*\text{max-depth}}$ after zero-padding. Finally, $e(a_i)$ is multiplied element-wise by a geometric series of a learned parameter p, that is, $e(a_i) \cdot \left[1, p, p, p^2, p^3, ...\right]$.

C Additional Results

C.1 Neural Text Generation (Part II)

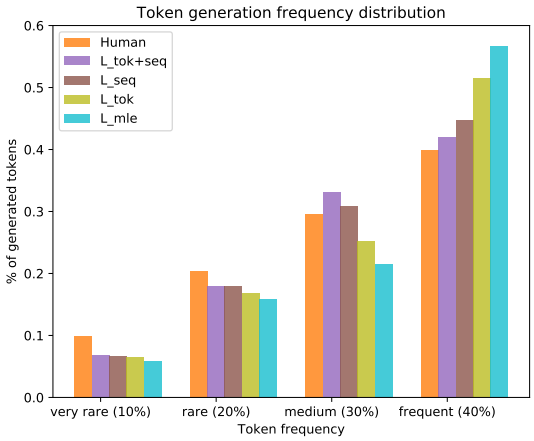


Fig. 28: Different combinations of unlikelihood

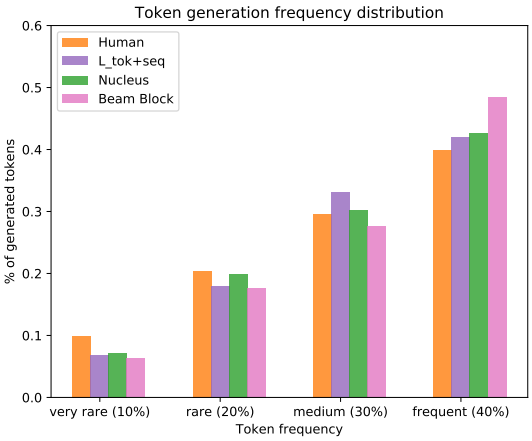


Fig. 29: Unlikelihood vs. stochastic decoding

Bibliography

Daniel Adiwardana, Minh-Thang Luong, David R So, Jamie Hall, Noah Fiedel, Romal Thoppilan, Zi Yang, Apoorv Kulshreshtha, Gaurav Nemade, Yifeng Lu, and Quoc V. Le. Towards a Human-like Open-Domain Chatbot. 2020. URL `https://www.msxiaobing.com/http://arxiv.org/abs/2001.09977`.

Daniel Andor, Chris Alberti, David Weiss, Aliaksei Severyn, Alessandro Presta, Kuzman Ganchev, Slav Petrov, and Michael Collins. Globally normalized transition-based neural networks. In *54th Annual Meeting of the Association for Computational Linguistics, ACL 2016 - Long Papers*, 2016. ISBN 9781510827585. doi: 10.18653/v1/p16-1231.

Yannis Assael, Thea Sommerschield, and Jonathan Prag. Restoring ancient text using deep learning: A case study on Greek epigraphy. In *EMNLP-IJCNLP 2019 - 2019 Conference on Empirical Methods in Natural Language Processing and 9th International Joint Conference on Natural Language Processing, Proceedings of the Conference*, 2020. ISBN 9781950737901. doi: 10.18653/v1/d19-1668.

Alexei Baevski and Michael Auli. Adaptive input representations for neural language modeling. In *7th International Conference on Learning Representations, ICLR 2019*, 2019.

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015. URL `http://arxiv.org/abs/1409.0473`.

Lalit R Bahl, Frederick Jelinek, and Robert L Mercer. A maximum likelihood approach to continuous speech recognition. *IEEE transactions on pattern analysis and machine intelligence*, 5(2):179–190, 1983.

Satanjeev Banerjee and Alon Lavie. METEOR: An automatic metric for MT evaluation with improved correlation with human judgments. In *Proceedings of the acl workshop on intrinsic and extrinsic evaluation measures for machine translation and/or summarization*, 2005.

Samy Bengio, Oriol Vinyals, Navdeep Jaitly, and Noam Shazeer. Scheduled sampling for sequence prediction with recurrent neural networks. In *Advances in Neural Information Processing Systems*, 2015.

Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Jauvin. A neural probabilistic language model. *Journal of machine learning research*, 3(Feb):1137–1155, 2003.

Eli Bingham, Jonathan P Chen, Martin Jankowiak, Fritz Obermeyer, Neeraj Pradhan, Theofanis Karaletsos, Rohit Singh, Paul Szerlip, Paul Horsfall, and Noah D Goodman. Pyro: Deep Universal Probabilistic Programming. *arXiv preprint arXiv:1810.09538*, 2018. URL `https://arxiv.org/pdf/1810.09538.pdf`.

Samuel R Bowman, Gabor Angeli, Christopher Potts, Christopher D Manning, and Stanford Linguistics. A large annotated corpus for learning natural language inference. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 632–642. Association for Computational Linguistics, 2015. URL `https://doi.org/10.18653/v1/D15-1075`.

Peter F. Brown, John Cocke, Stephen A. Della Pietra, Vincent J. Della Pietra, Fredrick Jelinek, John D. Lafferty, Robert L. Mercer, and Paul S. Roossin. A statistical approach to machine translation. *Comput. Linguist.*, 16(2):79–85, June 1990. ISSN 0891-2017. URL `http://dl.acm.org/citation.cfm?id=92858.92860`.

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. Language models are few-shot learners, 2020.

Chris Callison-Burch, Miles Osborne, and Philipp Koehn. Re-evaluation the role of bleu in machine translation research. In *11th Conference of the European Chapter of the Association for Computational Linguistics*, 2006.

Kai Wei Chang, Akshay Krishnamurthy, Alekh Agarwal, Hal Daumé, and John Langford. Learning to search better than your teacher. In *32nd International Conference on Machine Learning, ICML 2015*, 2015. ISBN 9781510810587.

Qian Chen, Xiaodan Zhu, Zhenhua Ling, Si Wei, Hui Jiang, and Diana Inkpen. Enhanced LSTM for Natural Language Inference. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1657–1668. Association for Computational Linguistics, 2017a. URL `https://doi.org/10.18653/v1/P17-1152`.

Stanley F. Chen and Joshua Goodman. An empirical study of smoothing techniques for language modeling. 1996. doi: 10.3115/981863.981904.

Wenhu Chen, Jianshu Chen, Yu Su, Zhiyu Chen, and William Yang Wang. Logical natural language generation from open-domain tables. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 7929–7942, Online, July 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.acl-main.708. URL https://www.aclweb.org/anthology/2020.acl-main.708.

Yining Chen, Sorcha Gilroy, Andreas Maletti, Jonathan May, and Kevin Knight. Recurrent neural networks as weighted language recognizers. *arXiv preprint arXiv:1711.05408*, 2017b.

Kyunghyun Cho, Bart van Merriënboer, Dzmitry Bahdanau, and Yoshua Bengio. On the properties of neural machine translation: Encoder–decoder approaches. In *Proceedings of SSST-8, Eighth Workshop on Syntax, Semantics and Structure in Statistical Translation*, pages 103–111, Doha, Qatar, October 2014. Association for Computational Linguistics. doi: 10.3115/v1/W14-4012. URL https://www.aclweb.org/anthology/W14-4012.

Yejin Choi. The missing representation in neural (language) models. *3rd Workshop on Representation Learning for NLP (RepL4NLP)*, 2018.

Alexis Conneau, Douwe Kiela, Holger Schwenk, Loic Barrault, and Antoine Bordes. Supervised Learning of Universal Sentence Representations from Natural Language Inference Data. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 670–680, Copenhagen, Denmark, 2017. Association for Computational Linguistics. URL https://www.aclweb.org/anthology/D17-1070.

Ido Dagan, Oren Glickman, and Bernardo Magnini. The PASCAL Recognising Textual Entailment Challenge. pages 177–190. Springer, Berlin, Heidelberg, 2006. doi: 10.1007/11736790{_}9. URL http://link.springer.com/10.1007/11736790_9.

Hal Daumé III, John Langford, and Daniel Marcu. Search-based Structured Prediction. Technical report, 2009. URL https://arxiv.org/pdf/0907.0786.pdf.

Krzysztof Dembczyński, Weiwei Cheng, and Eyke Hüllermeier. Bayes optimal multilabel classification via probabilistic classifier chains. In *Proceedings of the 27th International Conference on International Conference on Machine Learning*, ICML'10, pages 279–286, USA, 2010. Omnipress. ISBN 978-1-60558-907-7. URL http://dl.acm.org/citation.cfm?id=3104322.3104359.

Dorottya Demszky, Kelvin Guu, and Percy Liang. Transforming Question Answering Datasets Into Natural Language Inference Datasets. *arXiv preprint arXiv:1809.02922*, 2018.

Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 248–255. IEEE, 2009.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. 2018. URL `http://arxiv.org/abs/1810.04805`.

Emily Dinan, Varvara Logacheva, Valentin Malykh, Alexander Miller, Kurt Shuster, Jack Urbanek, Douwe Kiela, Arthur Szlam, Iulian Serban, Ryan Lowe, et al. The second conversational intelligence challenge (convai2). *arXiv preprint arXiv:1902.00098*, 2019a. URL `https://arxiv.org/pdf/1902.00098.pdf`.

Emily Dinan, Stephen Roller, Kurt Shuster, Angela Fan, Michael Auli, and Jason Weston. Wizard of wikipedia: Knowledge-powered conversational agents. In *Proceedings of the International Conference on Learning Representations*, 2019b. URL `https://openreview.net/forum?id=r1l73iRqKm`.

Chris Donahue, Mina Lee, and Percy Liang. Enabling language models to fill in the blanks. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 2492–2501, Online, July 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.acl-main.225. URL `https://www.aclweb.org/anthology/2020.acl-main.225`.

Timothy Dozat and Christopher D Manning. Deep Biaffine Attention for Neural Dependency Parsing. In *International Conference on Learning Representations (ICLR)*, 2017. URL `https://arxiv.org/pdf/1611.01734.pdf`.

W Ehrenfeld, R Buckingham, J Cranshaw, T Cuhadar Donszelmann, T Doherty, E Gallas, J Hrivnac, D Malon, M Nowak, M Slater, F Viegas, E Vinek, Q Zhang, and the ATLAS Collaboration. Using tags to speed up the atlas analysis process. *Journal of Physics: Conference Series*, 331(3):032007, 2011. URL `http://stacks.iop.org/1742-6596/331/i=3/a=032007`.

Bryan Eikema and Wilker Aziz. Is map decoding all you need? the inadequacy of the mode in neural machine translation, 2020.

Jeffrey L Elman. Finding structure in time. *Cognitive science*, 14(2):179–211, 1990.

Dmitrii Emelianenko, Elena Voita, and Pavel Serdyukov. Sequence modeling with unconstrained generation order. In H. Wallach, H. Larochelle, A. Beygelzimer, F. dAlché

Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 7700–7711. Curran Associates, Inc., 2019. URL http://papers.nips.cc/paper/8986-sequence-modeling-with-unconstrained-generation-order.pdf.

Angela Fan, Mike Lewis, and Yann Dauphin. Hierarchical neural story generation. In *ACL 2018 - 56th Annual Meeting of the Association for Computational Linguistics, Proceedings of the Conference (Long Papers)*, 2018. ISBN 9781948087322. doi: 10.18653/v1/p18-1082.

Angela Fan, Yacine Jernite, Ethan Perez, David Grangier, Jason Weston, and Michael Auli. ELI5: Long form question answering. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 3558–3567, Florence, Italy, July 2019. Association for Computational Linguistics. doi: 10.18653/v1/P19-1346. URL https://www.aclweb.org/anthology/P19-1346.

Jillian H. Fecteau and Douglas P. Munoz. Salience, relevance, and firing: a priority map for target selection. *Trends in Cognitive Sciences*, 10(8):382 – 390, 2006. ISSN 1364-6613. doi: https://doi.org/10.1016/j.tics.2006.06.011.

Daniel Fernández-González and Carlos Gómez-Rodríguez. Left-to-right dependency parsing with pointer networks. 2019.

Nicolas Ford, Daniel Duckworth, Mohammad Norouzi, and George E Dahl. The importance of generation order in language modeling. *arXiv preprint arXiv:1808.07910*, 2018.

Samuel Gehman, Suchin Gururangan, Maarten Sap, Yejin Choi, and Noah A. Smith. Realtoxicityprompts: Evaluating neural toxic degeneration in language models, 2020.

Yoav Goldberg and Michael Elhadad. An efficient algorithm for easy-first non-directional dependency parsing. In *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pages 742–750. Association for Computational Linguistics, 2010.

Yoav Goldberg and Joakim Nivre. A dynamic oracle for arc-eager dependency parsing. In *24th International Conference on Computational Linguistics - Proceedings of COLING 2012: Technical Papers*, 2012.

Yoav Goldberg and Joakim Nivre. Training Deterministic Parsers with Non-Deterministic Oracles. *Transactions of the Association for Computational Linguistics*, 2013. ISSN 2307-387X. doi: 10.1162/tacl_a_00237.

Carlos Gómez-Rodríguez, Francesco Sartorio, and Giorgio Satta. A polynomial-time dynamic oracle for non-projective dependency parsing. In *Proceedings of the 2014 Conference on Empirical Methods in*

Natural Language Processing (EMNLP), pages 917–927. Association for Computational Linguistics, 2014. doi: 10.3115/v1/D14-1099. URL http://aclweb.org/anthology/D14-1099.

Yunchao Gong, Yangqing Jia, Thomas Leung, Alexander Toshev, and Sergey Ioffe. Deep convolutional ranking for multilabel image annotation. *arXiv preprint arXiv:1312.4894*, 2013.

James Goodman, Andreas Vlachos, and Jason Naradowsky. Noise reduction and targeted exploration in imitation learning for Abstract Meaning Representation parsing. In *54th Annual Meeting of the Association for Computational Linguistics, ACL 2016 - Long Papers*, 2016. ISBN 9781510827585. doi: 10.18653/v1/p16-1001.

Edouard Grave, Armand Joulin, and Nicolas Usunier. Improving neural language models with a continuous cache. In *5th International Conference on Learning Representations, ICLR 2017 - Conference Track Proceedings*, 2017.

Alex Graves. Generating Sequences With Recurrent Neural Networks. 2013. URL http://arxiv.org/abs/1308.0850.

Jiatao Gu, Changhan Wang, and Jake Zhao. Levenshtein Transformer. *Neural Information Processing Systems (NeurIPS)*, 2019. URL https://github.com/pytorch/fairseq/tree/http://arxiv.org/abs/1905.11006.

Suchin Gururangan, Swabha Swayamdipta, Omer Levy, Roy Schwartz, Samuel R Bowman, and Noah A Smith. Annotation Artifacts in Natural Language Inference Data. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Short Papers)*, pages 107–112, New Orleans, Louisiana, 2018. Association for Computational Linguistics. URL http://aclweb.org/anthology/N18-2017.

S. Hamid Rezatofighi, Vijay Kumar B G, Anton Milan, Ehsan Abbasnejad, Anthony Dick, and Ian Reid. Deepsetnet: Predicting sets with deep neural networks. In *The IEEE International Conference on Computer Vision (ICCV)*, Oct 2017.

He He, Hal Daumé, and Jason Eisner. Imitation learning by coaching. In *Advances in Neural Information Processing Systems*, 2012a. ISBN 9781627480031.

He He, Hal Daumé, and Jason Eisner. Cost-sensitive Dynamic Feature Selection. In *International Conference on Machine Learning (ICML) workshop on Inferning: Interactions between Inference and Learning*, 2012b.

He He, Hal Daumé, and Jason Eisner. Learning to search in branch-and-bound algorithms. In *Advances in Neural Information Processing Systems*, 2014.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask R-CNN. *arXiv preprint arXiv:1703.06870*, 2017.

Tom Henighan, Jared Kaplan, Mor Katz, Mark Chen, Christopher Hesse, Jacob Jackson, Heewoo Jun, Tom B. Brown, Prafulla Dhariwal, Scott Gray, Chris Hallacy, Benjamin Mann, Alec Radford, Aditya Ramesh, Nick Ryder, Daniel M. Ziegler, John Schulman, Dario Amodei, and Sam McCandlish. Scaling laws for autoregressive generative modeling, 2020.

Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the Knowledge in a Neural Network. 2015. URL `http://arxiv.org/abs/1503.02531`.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997.

Matthew D. Hoffman and Andrew Gelman. The no-u-turn sampler: Adaptively setting path lengths in hamiltonian monte carlo. *J. Mach. Learn. Res.*, 15(1):1593–1623, 2014. ISSN 1532-4435. URL `http://dl.acm.org/citation.cfm?id=2627435.2638586`.

Ari Holtzman, Jan Buys, Li Du, Maxwell Forbes, and Yejin Choi. The curious case of neural text degeneration. In *International Conference on Learning Representations*, 2020. URL `https://openreview.net/forum?id=rygGQyrFvH`.

Cheng-Zhi Anna Huang, Ashish Vaswani, Jakob Uszkoreit, Ian Simon, Curtis Hawthorne, Noam Shazeer, Andrew M. Dai, Matthew D. Hoffman, Monica Dinculescu, and Douglas Eck. Music transformer. In *International Conference on Learning Representations*, 2019. URL `https://openreview.net/forum?id=rJe4ShAcF7`.

Samuel Humeau, Kurt Shuster, Marie-Anne Lachaux, and Jason Weston. Poly-encoders: Transformer architectures and pre-training strategies for fast and accurate multi-sentence scoring. *arXiv preprint arXiv:1905.01969*, 2019. URL `https://arxiv.org/abs/1905.01969`.

Hideki Isozaki, Tsutomu Hirao, Kevin Duh, Katsuhito Sudoh, and Hajime Tsukada. Automatic evaluation of translation quality for distant language pairs. In *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing*, pages 944–952. Association for Computational Linguistics, 2010a.

Hideki Isozaki, Tsutomu Hirao, Kevin Duh, Katsuhito Sudoh, and Hajime Tsukada. Automatic evaluation of translation quality for distant language pairs. In *EMNLP 2010 - Conference on Empirical Methods in Natural Language Processing, Proceedings of the Conference*, 2010b. ISBN 1932432868.

L. Itti, C. Koch, and E. Niebur. A model of saliency-based visual attention for rapid scene analysis. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(11):1254–1259, 1998. ISSN 01628828. doi: 10.1109/34.730558. URL `http://ieeexplore.ieee.org/document/730558/`.

Ajay Jain, Pieter Abbeel, and Deepak Pathak. Locally masked convolution for autoregressive models. In *Conference on Uncertainty in Artificial Intelligence (UAI)*, 2020.

Yoon Kim, Alexander M Rush, Lei Yu, Adhiguna Kuncoro, Chris Dyer, and Gábor Melis. Unsupervised Recurrent Neural Network Grammars. 2019. URL `https://github.com/harvardnlp/urnnghttp://arxiv.org/abs/1904.03746`.

C Koch and S Ullman. Shifts in selective visual attention: towards the underlying neural circuitry. *Human neurobiology*, 4(4):219–27, 1985. ISSN 0721-9075. URL `http://www.ncbi.nlm.nih.gov/pubmed/3836989`.

Philipp Koehn and Rebecca Knowles. Six challenges for neural machine translation. In *Proceedings of the First Workshop on Neural Machine Translation*, pages 28–39, Vancouver, August 2017. Association for Computational Linguistics. doi: 10.18653/v1/W17-3204. URL `https://www.aclweb.org/anthology/W17-3204`.

Philipp Koehn, Hieu Hoang, Alexandra Birch, Chris Callison-Burch, Marcello Federico, Nicola Bertoldi, Brooke Cowan, Wade Shen, Christine Moran, Richard Zens, Chris Dyer, Ondřej Bojar, Alexandra Constantin, and Evan Herbst. Moses: Open source toolkit for statistical machine translation. In *Proceedings of the 45th Annual Meeting of the ACL on Interactive Poster and Demonstration Sessions*, pages 177–180. Association for Computational Linguistics, 2007.

Adam R Kosiorek, Hyunjik Kim, and Danilo J Rezende. Conditional set generation with transformers, 2020.

Ilia Kulikov, Alexander Miller, Kyunghyun Cho, and Jason Weston. Importance of search and evaluation strategies in neural dialogue modeling. In *Proceedings of the 12th International Conference on Natural Language Generation*, pages 76–87, Tokyo, Japan, October–November 2019. Association for Computational Linguistics. doi: 10.18653/v1/W19-8609. URL `https://www.aclweb.org/anthology/W19-8609`.

Shankar Kumar and William Byrne. Minimum Bayes-Risk Decoding for Statistical Machine Translation. In *HLT-NAACL 2004: Main Proceedings*, 2004.

John Lafferty, Andrew McCallum, and Fernando C N Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. *ICML '01 Proceedings of the Eighteenth International Conference on Machine Learning*, 2001a. ISSN 1750-2799. doi: 10.1038/nprot.2006.61.

John Lafferty, Andrew McCallum, and Fernando C N Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. *ICML '01 Proceedings of the Eighteenth International Conference on Machine Learning*, 2001b. ISSN 1750-2799. doi: 10.1038/nprot.2006.61.

Victor A.F. Lamme and Pieter R. Roelfsema. The distinct modes of vision offered by feedforward and recurrent processing. *Trends in Neurosciences*, 23(11):571 – 579, 2000. ISSN 0166-2236. doi: https://doi.org/10.1016/S0166-2236(00)01657-X. URL `http://www.sciencedirect.com/science/article/pii/S016622360001657X`.

Guillaume Lample and François Charton. Deep learning for symbolic mathematics. In *International Conference on Learning Representations*, 2020. URL `https://openreview.net/forum?id=S1eZYeHFDS`.

Wuwei Lan and Wei Xu. Neural Network Models for Paraphrase Identification, Semantic Textual Similarity, Natural Language Inference, and Question Answering. In *Proceedings of the 27th International Conference on Computational Linguistics*, pages 3890–3902, Santa Fe, New Mexico, USA, 2018. Association for Computational Linguistics. URL `http://aclweb.org/anthology/C18-1328`.

Rémi Leblond, Jean Baptiste Alayrac, Anton Osokin, and Simon Lacoste-Julien. SeaRNN: Training rnNs with global-local losses. In *6th International Conference on Learning Representations, ICLR 2018 - Conference Track Proceedings*, 2018.

Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

Jason Lee, Dustin Tran, Orhan Firat, and Kyunghyun Cho. On the discrepancy between density estimation and sequence generation. *arXiv preprint arXiv:2002.07233*, 2020.

V. Lempitsky and A. Zisserman. Learning to count objects in images. In *Advances in Neural Information Processing Systems*, 2010.

Dmitry Lepikhin, HyoukJoong Lee, Yuanzhong Xu, Dehao Chen, Orhan Firat, Yanping Huang, Maxim Krikun, Noam Shazeer, and Zhifeng Chen. Gshard: Scaling giant models with conditional computation and automatic sharding, 2020.

Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Ves Stoyanov, and Luke Zettlemoyer. BART: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. *arXiv preprint arXiv:1910.13461*, 2019. URL https://arxiv.org/abs/1910.13461.

Jiwei Li, Michel Galley, Chris Brockett, Georgios P Spithourakis, Jianfeng Gao, and Bill Dolan. A Persona-Based Neural Conversation Model. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 994–1003, Berlin, Germany, 2016. Association for Computational Linguistics. URL http://www.aclweb.org/anthology/P16-1094.

Margaret Li, Jason Weston, and Stephen Roller. Acute-eval: Improved dialogue evaluation with optimized questions and multi-turn comparisons. *arXiv preprint arXiv:1909.03087*, 2019a. URL https://arxiv.org/pdf/1909.03087.pdf.

Margaret Li, Jason Weston, and Stephen Roller. Acute-eval: Improved dialogue evaluation with optimized questions and multi-turn comparisons, 2019b.

Margaret Li, Stephen Roller, Ilia Kulikov, Sean Welleck, Y-Lan Boureau, Kyunghyun Cho, and Jason Weston. Don't say that! making inconsistent dialogue unlikely with unlikelihood training. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 4715–4728, Online, July 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.acl-main.428. URL https://www.aclweb.org/anthology/2020.acl-main.428.

Yujia Li, Oriol Vinyals, Chris Dyer, Razvan Pascanu, and Peter Battaglia. Learning deep generative models of graphs. In *ICML 2018*, 2018.

Renjie Liao, Yujia Li, Yang Song, Shenlong Wang, Will Hamilton, David K Duvenaud, Raquel Urtasun, and Richard Zemel. Efficient graph generation with graph recurrent attention networks. In H. Wallach, H. Larochelle, A. Beygelzimer, F. dAlché Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 4255–4265. Curran Associates, Inc., 2019. URL http://papers.nips.cc/paper/8678-efficient-graph-generation-with-graph-recurrent-attention-networks.pdf.

Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft coco: Common objects in context. In *European conference on computer vision*, pages 740–755. Springer, 2014.

Chi-kiu Lo. YiSi: A semantic machine translation evaluation metric for evaluating languages with different levels of available resources. Unpublished, 2018. URL http://chikiu-jackie-lo.org/home/

index.php/yisi.

Francesco Locatello, Dirk Weissenborn, Thomas Unterthiner, Aravindh Mahendran, Georg Heigold, Jakob Uszkoreit, Alexey Dosovitskiy, and Thomas Kipf. Object-centric learning with slot attention, 2020.

Xuezhe Ma, Zecong Hu, Jingzhou Liu, Nanyun Peng, Graham Neubig, and Eduard Hovy. Stack-Pointer Networks for Dependency Parsing. Technical report, 2018. URL https://arxiv.org/pdf/1805.01087.pdf.

Bill Maccartney and Christopher D Manning. An extended model of natural logic. Technical report, 2009. URL http://www.aclweb.org/anthology/W09-3714.

Elman Mansimov, Alex Wang, and Kyunghyun Cho. A generalized framework of sequence generation with application to undirected sequence models. 2019.

M Marelli, S Menini, M Baroni, L Bentivogli, R Bernardi, and R Zamparelli. A SICK cure for the evaluation of compositional distributional semantic models. In *Proceedings of the Ninth International Conference on Language Resources and Evaluation (LREC-2014)*, Reykjavik, Iceland, 2014. European Language Resources Association (ELRA). URL http://www.lrec-conf.org/proceedings/lrec2014/pdf/363_Paper.pdf.

Pedro Henrique Martins, Zita Marinho, and André F. T. Martins. Sparse text generation, 2020.

Andrew McCallum, Dayne Freitag, and Fernando C N Pereira. Maximum Entropy Markov Models for Information Extraction and Segmentation. In *Proceedings of the Seventeenth International Conference on Machine Learning*, 2000. ISBN 1-55860-707-2.

Gábor Melis, Chris Dyer, and Phil Blunsom. On the state of the art of evaluation in neural language models. In *6th International Conference on Learning Representations, ICLR 2018 - Conference Track Proceedings*, 2018.

Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. Pointer sentinel mixture models. *arXiv preprint arXiv:1609.07843*, 2016.

Stephen Merity, Nitish Shirish Keskar, and Richard Socher. Regularizing and optimizing LSTM language models. In *6th International Conference on Learning Representations, ICLR 2018 - Conference Track Proceedings*, 2018.

Alexander H Miller, Will Feng, Adam Fisch, Jiasen Lu, Dhruv Batra, Antoine Bordes, Devi Parikh, and Jason Weston. ParlAI: A Dialog Research Software Platform. *arXiv preprint:1705.06476*, 2017. URL http://parl.ai.

Kenton Murray and David Chiang. Correcting length bias in neural machine translation. In *Proc. WMT*, pages 212–223, 2018.

Jinseok Nam, Eneldo Loza Mencía, Hyunwoo J Kim, and Johannes Fürnkranz. Maximizing subset accuracy with recurrent neural networks in multi-label classification. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 5419–5429. Curran Associates, Inc., 2017. URL http://papers.nips.cc/paper/7125-maximizing-subset-accuracy-with-recurrent-neural-networks-in-multi-label-classification.pdf.

Joakim Nivre. An efficient algorithm for projective dependency parsing. In *Proceedings of the Eighth International Workshop on Parsing Technologies (IWPT*, pages 149–160, Nancy, France, 2003. URL https://www.aclweb.org/anthology/W03-3017.

Joakim Nivre. Algorithms for deterministic incremental dependency parsing. *Comput. Linguist.*, 34(4): 513–553, December 2008. ISSN 0891-2017. doi: 10.1162/coli.07-056-R1-07-027. URL http://dx.doi.org/10.1162/coli.07-056-R1-07-027.

Joakim Nivre. Non-projective dependency parsing in expected linear time. In *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP*, pages 351–359. Association for Computational Linguistics, 2009. URL http://aclweb.org/anthology/P09-1040.

Franz Josef Och. Minimum error rate training in statistical machine translation. 2003. doi: 10.3115/1075096.1075117.

Daniel Oñoro-Rubio and Roberto J. López-Sastre. Towards perspective-free object counting with deep learning. In Bastian Leibe, Jiri Matas, Nicu Sebe, and Max Welling, editors, *Computer Vision – ECCV 2016*, pages 615–629, Cham, 2016. Springer International Publishing. ISBN 978-3-319-46478-7.

Aaron Van Oord, Nal Kalchbrenner, and Koray Kavukcuoglu. Pixel recurrent neural networks. volume 48 of *Proceedings of Machine Learning Research*, pages 1747–1756, New York, New York, USA, 20–22 Jun 2016. PMLR. URL http://proceedings.mlr.press/v48/oord16.html.

Myle Ott, Michael Auli, David Grangier, and Marc'aurelio Ranzato. Analyzing uncertainty in neural machine translation. In *35th International Conference on Machine Learning, ICML 2018*, 2018. ISBN 9781510867963.

Myle Ott, Sergey Edunov, Alexei Baevski, Angela Fan, Sam Gross, Nathan Ng, David Grangier, and Michael Auli. fairseq: A fast, extensible toolkit for sequence modeling. In *Proceedings of NAACL-HLT 2019: Demonstrations*, 2019.

Kishore Papineni, Salim Roukos, Todd Ward, and Wei-jing Zhu. BLEU : a Method for Automatic Evaluation of Machine Translation. *Computational Linguistics*, 2002.

Christine Payne. MuseNet, apr 2019. URL `https://openai.com/blog/musenet/`.

Jeffrey Pennington, Richard Socher, and Christopher D Manning. GloVe: Global Vectors for Word Representation. In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, 2014. URL `http://www.aclweb.org/anthology/D14-1162`.

R.J. Peters, A. Iyer, L. Itti, and C. Koch. Components of bottom-up gaze allocation in natural images. *Vision Research*, 2005. ISSN 0042-6989.

Bartosz Piotrowski, Josef Urban, Chad E Brown, and Cezary Kaliszyk. Can Neural Networks Learn Symbolic Rewriting? *ICML 2019 Workshop on Learning and Reasoning with Graph-Structured Data*, 2019. URL `http://arxiv.org/abs/1911.04873`.

Adam Poliak, Aparajita Haldar, Rachel Rudinger, J Edward Hu, Ellie Pavlick, Aaron Steven White, and Benjamin Van Durme. Collecting Diverse Natural Language Inference Problems for Sentence Representation Evaluation. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 67–81. Association for Computational Linguistics, 2018a. URL `http://aclweb.org/anthology/D18-1007`.

Adam Poliak, Jason Naradowsky, Aparajita Haldar, Rachel Rudinger, and Benjamin Van Durme. Hypothesis Only Baselines in Natural Language Inference. In *The Seventh Joint Conference on Lexical and Computational Semantics (*SEM)*, 2018b. URL `https://leonidk.com/`.

Peng Qi, Timothy Dozat, Yuhao Zhang, and Christopher D. Manning. Universal dependency parsing from scratch. In *Proceedings of the CoNLL 2018 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies*, pages 160–170, Brussels, Belgium, October 2018. Association for Computational Linguistics. URL `https://nlp.stanford.edu/pubs/qi2018universal.pdf`.

Lianhui Qin, Antoine Bosselut, Ari Holtzman, Chandra Bhagavatula, Elizabeth Clark, and Yejin Choi. Counterfactual story reasoning and generation. In *Proceedings of the 2019 Conference on Empiri-*

cal Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP), pages 5043–5053, Hong Kong, China, November 2019. Association for Computational Linguistics. doi: 10.18653/v1/D19-1509. URL https://www.aclweb.org/anthology/D19-1509.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language Models are Unsupervised Multitask Learners. 2018.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of Machine Learning Research*, 21(140):1–67, 2020. URL http://jmlr.org/papers/v21/20-074.html.

Marc'Aurelio Ranzato, Sumit Chopra, Michael Auli, and Wojciech Zaremba. Sequence level training with recurrent neural networks. In *4th International Conference on Learning Representations, ICLR 2016 - Conference Track Proceedings*, 2016.

Raymond M. Klein. Inhibition of return. *Trends in cognitive sciences*, 4(4):138–147, 4 2000. ISSN 1879-307X. URL http://www.ncbi.nlm.nih.gov/pubmed/10740278.

Jesse Read, Bernhard Pfahringer, Geoff Holmes, and Eibe Frank. Classifier chains for multi-label classification. *Machine Learning*, 85(3):333, Jun 2011. ISSN 1573-0565. doi: 10.1007/s10994-011-5256-5. URL https://doi.org/10.1007/s10994-011-5256-5.

Mengye Ren and Richard S. Zemel. End-to-end instance segmentation with recurrent attention. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017.

Adam Roberts, Jesse Engel, Colin Raffel, Curtis Hawthorne, and Douglas Eck. A hierarchical latent vector model for learning long-term structure in music. volume 80 of *Proceedings of Machine Learning Research*, pages 4364–4373, Stockholmsmässan, Stockholm Sweden, 10–15 Jul 2018. PMLR. URL http://proceedings.mlr.press/v80/roberts18a.html.

Stephen Roller, Emily Dinan, Naman Goyal, Da Ju, Mary Williamson, Yinhan Liu, Jing Xu, Myle Ott, Kurt Shuster, Eric M. Smith, Y-Lan Boureau, and Jason Weston. Recipes for building an open-domain chatbot, 2020.

Stéphane Ross and J. Andrew Bagnell. Reinforcement and imitation learning via interactive no-regret learning. *CoRR*, abs/1406.5979, 2014. URL http://arxiv.org/abs/1406.5979.

Stéphane Ross, Geoffrey J. Gordon, and J. Andrew Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. In *Journal of Machine Learning Research*, 2011.

Alexander M. Rush, Sumit Chopra, and Jason Weston. A neural attention model for sentence summarization. In *Conference Proceedings - EMNLP 2015: Conference on Empirical Methods in Natural Language Processing*, 2015. ISBN 9781941643327.

Allen Schmaltz, Alexander M. Rush, and Stuart Shieber. Word ordering without syntax. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 2319–2324. Association for Computational Linguistics, 2016. doi: 10.18653/v1/D16-1255. URL `http://aclweb.org/anthology/D16-1255`.

Rico Sennrich, Barry Haddow, and Alexandra Birch. Neural machine translation of rare words with subword units. *arXiv preprint arXiv:1508.07909*, 2015.

Shiqi Shen, Yong Cheng, Zhongjun He, Wei He, Hua Wu, Maosong Sun, and Yang Liu. Minimum risk training for neural machine translation. In *54th Annual Meeting of the Association for Computational Linguistics, ACL 2016 - Long Papers*, 2016. ISBN 9781510827585. doi: 10.18653/v1/p16-1159.

Tianxiao Shen, Victor Quach, Regina Barzilay, and Tommi Jaakkola. Blank language models. 2020.

Vighnesh Leonardo Shiv and Chris Quirk. Novel positional encodings to enable tree-structured transformers. 2019. URL `https://openreview.net/forum?id=SJerEhR5Km`.

Matthew Snover, Bonnie Dorr, Richard Schwartz, Linnea Micciulla, and John Makhoul. A study of translation edit rate with targeted human annotation. In *AMTA 2006 - Proceedings of the 7th Conference of the Association for Machine Translation of the Americas: Visions for the Future of Machine Translation*, 2006.

Pavel Sountsov and Sunita Sarawagi. Length bias in encoder decoder models and a case for global conditioning. In *EMNLP 2016 - Conference on Empirical Methods in Natural Language Processing, Proceedings*, 2016a. ISBN 9781945626258. doi: 10.18653/v1/d16-1158.

Pavel Sountsov and Sunita Sarawagi. Length bias in encoder decoder models and a case for global conditioning. In *EMNLP 2016 - Conference on Empirical Methods in Natural Language Processing, Proceedings*, 2016b. ISBN 9781945626258. doi: 10.18653/v1/d16-1158.

Felix Stahlberg and Bill Byrne. On NMT search errors and model errors: Cat got your tongue? In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 3354–3360, Hong Kong, China, November 2019a. Association for Computational Linguistics. doi: 10.18653/v1/D19-1331. URL `https://www.aclweb.org/anthology/D19-1331`.

Felix Stahlberg and Bill Byrne. On NMT search errors and model errors: Cat got your tongue? In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 3356–3362, Hong Kong, China, November 2019b. Association for Computational Linguistics. doi: 10.18653/v1/D19-1331. URL https://www.aclweb.org/anthology/D19-1331.

Mitchell Stern, William Chan, Jamie Kiros, and Jakob Uszkoreit. Insertion Transformer: Flexible sequence generation via insertion operations. In *36th International Conference on Machine Learning, ICML 2019*, 2019. ISBN 9781510886988.

Russell Stewart, Mykhaylo Andriluka, and Andrew Y. Ng. End-to-end people detection in crowded scenes. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016.

Veselin Stoyanov and Jason Eisner. Easy-first coreference resolution. *Proceedings of COLING 2012*, pages 2519–2534, 2012.

Ilya Sutskever, James Martens, and Geoffrey Hinton. Generating text with recurrent neural networks. In *Proceedings of the 28th International Conference on Machine Learning, ICML 2011*, 2011. ISBN 9781450306195.

Grigorios Tsoumakas and Ioannis Katakis. Multi-label classification: An overview. *Int J Data Warehousing and Mining*, 2007:1–13, 2007.

Yoshimasa Tsuruoka and Jun'ichi Tsujii. Bidirectional inference with the easiest-first strategy for tagging sequence data. In *Proceedings of the conference on human language technology and empirical methods in natural language processing*, pages 467–474. Association for Computational Linguistics, 2005.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, 2017.

Richard Veale, Ziad M. Hafed, and Masatoshi Yoshida. How is visual salience computed in the brain? Insights from behaviour, neurobiology and modelling. *Philosophical Transactions of the Royal Society of London B: Biological Sciences*, 372(1714), 2017. URL http://rstb.royalsocietypublishing.org/content/372/1714/20160113.

Oriol Vinyals, Google Quoc, and V Le. A Neural Conversational Model. In *ICML Deep Learning Workshop*, 2015.

Andreas Vlachos and Stephen Clark. A new corpus and imitation learning framework for context-dependent semantic parsing. *Transactions of the Association for Computational Linguistics*, 2:547–560, 2014. doi: 10.1162/tacl_a_00202. URL https://www.aclweb.org/anthology/Q14-1042.

Andreas Vlachos, Gerasimos Lampouras, and Sebastian Riedel. Imitation learning for structured prediction in natural language processing. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Tutorial Abstracts*, Valencia, Spain, April 2017. Association for Computational Linguistics. URL https://www.aclweb.org/anthology/E17-5003.

Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R Bowman. GLUE: A Multi-Task Benchmark and Analysis Platform for Natural Language Understanding. *arXiv preprint arXiv:1804.07461*, 2018. URL https://arxiv.org/pdf/1804.07461.pdf.

Jiang Wang, Yi Yang, Junhua Mao, Zhiheng Huang, Chang Huang, and Wei Xu. Cnn-rnn: A unified framework for multi-label image classification. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016.

Sean Welleck and Kyunghyun Cho. Sequential graph dependency parser. In *Proceedings of the International Conference on Recent Advances in Natural Language Processing (RANLP 2019)*, pages 1338–1345, Varna, Bulgaria, September 2019. INCOMA Ltd. doi: 10.26615/978-954-452-056-4_153. URL https://www.aclweb.org/anthology/R19-1153.

Sean Welleck and Kyunghyun Cho. Mle-guided parameter search for task loss minimization in neural sequence modeling, 2020.

Sean Welleck, Jialin Mao, Kyunghyun Cho, and Zheng Zhang. Saliency-based sequential image attention with multiset prediction. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 5173–5183. Curran Associates, Inc., 2017. URL http://papers.nips.cc/paper/7102-saliency-based-sequential-image-attention-with-multiset-prediction.pdf.

Sean Welleck, Ilia Kulikov, Stephen Roller, Emily Dinan, Kyunghyun Cho, and Jason Weston. Neural text generation with unlikelihood training. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=SJeYe0NtvH.

Yorick Wilks. *Machine translation: its scope and limits*. Springer Science & Business Media, 2008.

Adina Williams, Nikita Nangia, and Samuel R Bowman. A Broad-Coverage Challenge Corpus for Sentence Understanding through Inference. 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 1112–1122, New Orleans, Louisiana, 2018. Association for Computational Linguistics. URL http://aclweb.org/anthology/N18-1101.

Ronald J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 1992. ISSN 0885-6125. doi: 10.1007/bf00992696.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, R'emi Louf, Morgan Funtowicz, and Jamie Brew. Huggingface's transformers: State-of-the-art natural language processing. *ArXiv*, abs/1910.03771, 2019.

SHI Xingjian, Zhourong Chen, Hao Wang, Dit-Yan Yeung, Wai-Kin Wong, and Wang-chun Woo. Convolutional lstm network: A machine learning approach for precipitation nowcasting. In *Advances in neural information processing systems*, pages 802–810, 2015.

H. Yamada and Y. Matsumoto. Statistical Dependency Analysis with Support Vector machines. In *The 8th International Workshop of Parsing Technologies (IWPT2003)*, 2003.

Zhilin Yang, Zihang Dai, Yiming Yang, Jaime Carbonell, Russ R Salakhutdinov, and Quoc V Le. Xlnet: Generalized autoregressive pretraining for language understanding. In H. Wallach, H. Larochelle, A. Beygelzimer, F. dAlché Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 5753–5763. Curran Associates, Inc., 2019. URL http://papers.nips.cc/paper/8812-xlnet-generalized-autoregressive-pretraining-for-language-understanding.pdf.

Jiaxuan You, Rex Ying, Xiang Ren, William L. Hamilton, and Jure Leskovec. GraphRNN: Generating realistic graphs with deep auto-regressive models. In *35th International Conference on Machine Learning, ICML 2018*, 2018. ISBN 9781510867963.

Lantao Yu, Weinan Zhang, Jun Wang, and Yong Yu. Seqgan: Sequence generative adversarial nets with policy gradient. *CoRR*, abs/1609.05473, 2016. URL http://dblp.uni-trier.de/db/journals/corr/corr1609.html#YuZWY16.

Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. HellaSwag: Can a machine really finish your sentence? In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 4791–4800, Florence, Italy, July 2019. Association for Computational Linguistics. doi: 10.18653/v1/P19-1472. URL https://www.aclweb.org/anthology/P19-1472.

Daniel Zeman, Jan Hajič, Martin Popel, Martin Potthast, Milan Straka, Filip Ginter, Joakim Nivre, and Slav Petrov. CoNLL 2018 shared task: Multilingual parsing from raw text to universal dependencies. In *Proceedings of the CoNLL 2018 Shared Task: Multilingual Parsing from Raw Text to Universal*

Dependencies, pages 1–21, Brussels, Belgium, October 2018. Association for Computational Linguistics. URL http://www.aclweb.org/anthology/K18-2001.

Hugh Zhang, Daniel Duckworth, Daphne Ippolito, and Arvind Neelakantan. Trading off diversity and quality in natural language generation, 2020.

Saizheng Zhang, Emily Dinan, Jack Urbanek, Arthur Szlam, Douwe Kiela, and Jason Weston. Personalizing Dialogue Agents: I have a dog, do you have pets too? In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2204–2213, Melbourne, Australia, 2018. Association for Computational Linguistics. URL http://aclweb.org/anthology/P18-1205.

Y. Zhang, D. Zhou, S. Chen, S. Gao, and Y. Ma. Single-image crowd counting via multi-column convolutional neural network. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 589–597, June 2016. doi: 10.1109/CVPR.2016.70.

Yaoming Zhu, Sidi Lu, Lei Zheng, Jiaxian Guo, Weinan Zhang, Jun Wang, and Yong Yu. Texygen: A benchmarking platform for text generation models. *SIGIR*, 2018.