

Non-Monotonic Sequential Text Generation

Sean Welleck¹ Kianté Brantley² Hal Daumé III^{2,3} Kyunghyun Cho^{1,4,5}

Abstract

Standard sequential generation methods assume a pre-specified generation order, such as text generation methods which generate words from left to right. In this work, we propose a framework for training models of text generation that operate in non-monotonic orders; the model directly learns good orders, without any additional annotation. Our framework operates by generating a word at an arbitrary position, and then recursively generating words to its left and then words to its right, yielding a binary tree. Learning is framed as imitation learning, including a coaching method which moves from imitating an oracle to reinforcing the policy’s own preferences. Experimental results demonstrate that using the proposed method, it is possible to learn policies which generate text without pre-specifying a generation order, while achieving competitive performance with conventional left-to-right generation.

1. Introduction

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 & Tsujii, 2005), parsing (Goldberg & Elhadad, 2010), and coreference (Stoyanov & Eisner, 2012), which allow a model to effectively learn their own ordering. In investigating this question, we are solely interested in considering non-monotonic generation that does not rely on external supervision, such as parse trees (Eriguchi et al., 2017; Aharoni & Goldberg, 2017).

¹New York University ²University of Maryland, College Park ³Microsoft Research ⁴Facebook AI Research ⁵CIFAR Azrieli Global Scholar. Correspondence to: Sean Welleck <wellecks@nyu.edu>.

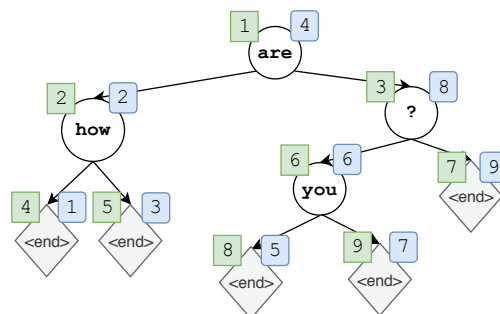


Figure 1. A sequence, “how are you?”, generated by the proposed approach trained on utterances from a dialogue dataset. The model first generated the word “are” and then recursively generated left and right subtrees (“how” and “you?”, respectively) of this word. At each production step, the model may 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 paper, we propose a framework for training sequential text generation models which learn a generation order without having to specifying an order in advance (§2). An example generation from our model is shown in Figure 1. 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 (§3). Because the tree structure is unknown, the oracle policy cannot know the exact correct actions to take; to remedy this we propose a method called *annealed coaching* which 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 (§5).¹

2. Non-Monotonic Sequence Generation

Formally, we consider the problem of sequentially generating a sequence of discrete tokens $Y = (w_1, \dots, w_N)$,

¹Code and trained models available at https://github.com/wellecks/nonmonotonic_text.

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. We propose a method 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 1.

We view the generation process as deterministically navigating a state space $\mathcal{S} = \tilde{V}^*$ where a state $s \in \mathcal{S}$ 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 s_0 is the empty sequence. For example, in Figure 1, $s_1 = \langle \text{are} \rangle$, $s_2 = \langle \text{are, how} \rangle$, ..., $s_4 = \langle \text{are, how, ?, } \langle \text{end} \rangle \rangle$. 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 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 $\langle a_{\tau(1)}, \dots, a_{\tau(T)} \rangle$ 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 1, τ maps from the numbers in the blue squares to those in the green squares.

A policy π is a (possibly) stochastic mapping from states to actions, and we denote the probability of an action $a \in \tilde{V}$ given a state s as $\pi(a|s)$. A 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 Y . Each of these trees has a different level-order traversal, thus the policy is capable of choosing from many different generation orders for Y , rather than a single predefined order. Note that left-to-right generation can be recovered if $\pi(\langle \text{end} \rangle|s_t) = 1$ if and only if t is odd (or non-zero and even for right-to-left generation).

3. Learning for Non-Monotonic Generation

Learning in our non-monotonic sequence generation model (§2) amounts to inferring a policy π from data. We first consider the *unconditional* generation problem (akin to language modeling) in which the data consists simply of sequences Y to be generated. Subsequently (§4.1) we consider the conditional case in which we wish to learn a mapping from inputs X to output sequences Y .

This learning problem is challenging because the sequences

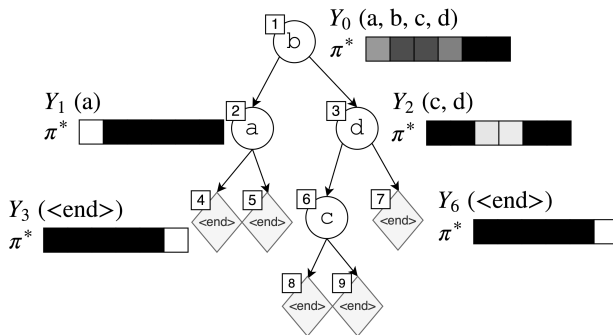


Figure 2. A sampled tree for the sentence “a b c d” with an action space $\tilde{V} = (a, b, c, d, e, \langle \text{end} \rangle)$, showing an oracle’s distribution π^* and consecutive subsequences (“valid actions”) Y_t for $t \in \{0, 1, 2, 3, 6\}$. Each oracle distribution is depicted as 6 boxes showing $\pi^*(a_{t+1}|s_t)$ (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 .

Y alone only tell us what the final output sequences of words should be, but not what tree(s) should be used to get there. In left-to-right generation, the observed sequence Y fully determines the sequence of actions to take. In our case, however, the tree structure is effectively a latent variable, which will be determined by the policy itself. This prevents us from using conventional supervised learning for training the parameterized policy. On the other hand, at training time, we do know *which words* should eventually appear, and their order; this substantially constrains the search space that needs to be explored, suggesting learning-to-search (Daumé et al., 2009) and imitation learning (Ross et al., 2011; Ross & Bagnell, 2014) as a learning strategy.²

The key idea in our imitation learning framework is that at the first step, an oracle policy’s action is to produce *any* word w that appears anywhere in Y . Once picked, in a quicksort-like manner, all words to the left of w in Y are generated recursively on the left (following the same procedure), and all words to the right of w in Y are generated recursively on the right. (See Figure 2 for an example.) Because the oracle is *non-deterministic* (many “correct” actions are available at any given time), we inform this oracle policy with the current learned policy, encouraging it to favor actions that are preferred by the current policy, inspired by work in direct loss minimization (Hazan et al., 2010) and related techniques (Chiang, 2012; He et al., 2012).

²One could consider applying reinforcement learning to this problem. This would ignore the fact that at training time we *know* which words will appear, reducing the size of the feasible search space from $O(|V|^T)$ to $O(|X|^T)$, a huge savings. Furthermore, even with a fixed generation order, RL has proven to be difficult without partially relying on supervised learning (Ranzato et al., 2015; Bahdanau et al., 2015; 2016).

3.1. Background: Learning to Search

In learning-to-search-style algorithms, we aim to learn a policy π that mimics an oracle (or “reference”) policy π^* . To do so, we define a *roll-in* policy π^{in} and *roll-out* policy π^{out} . We then repeatedly draw states s according to the state distribution induced by π^{in} , and compute cost-to-go under π^{out} , for all possible actions a at that state. The learned policy π is then trained to choose actions to minimize this cost-to-go estimate.

Formally, denote the uniform distribution over $\{1, \dots, T\}$ as $U[T]$ and denote by d_π^t the distribution of states induced by running π for t -many steps. Denote by $\mathcal{C}(\pi; \pi^{\text{out}}, s)$ a scalar cost measuring the loss incurred by π against the cost-to-go estimates under π^{out} (for instance, \mathcal{C} may measure the squared error between the vector $\pi(\cdot|s)$ and the cost-to-go estimates). Then, the quantity being optimized is:

$$\mathbb{E}_{Y \sim D} \mathbb{E}_{t \sim U[2|Y|+1]} \mathbb{E}_{s_t \sim d_{\pi^{\text{in}}}^t} [\mathcal{C}(\pi; \pi^{\text{out}}, s_t)] \quad (1)$$

Here, π^{in} and π^{out} can use information not available at test-time (e.g., the ground-truth Y). Learning consists of finding a policy which only has access to states s_t but performs as well or better than π^* . By varying the choice of π^{in} , π^{out} , and \mathcal{C} , one obtains different variants of learning-to-search algorithms, such as DAgger (Ross et al., 2011), AggreVaTe (Ross & Bagnell, 2014) or LOLS (Chang et al., 2015).

In the remainder of this section, we describe the cost function we use, a set of oracle policies and a set of roll-in policies, both of which are specifically designed for the proposed problem of non-monotonic sequential generation of a sequence. These sets of policies are empirically evaluated later in the experiments (§5).

3.2. Cost Measurement

There are many ways to measure the prediction cost $\mathcal{C}(\pi; \pi^{\text{out}}, s)$; arguably the most common is squared error between cost-predictions by π and observed costs obtained by π^{out} at the state s . However, recent work has found that, especially when dealing with recurrent neural network policies (which we will use; see §4), using a cost function more analogous to a cross-entropy loss can be preferred (Leblond et al., 2018; Cheng et al., 2018; Welleck et al., 2018). In particular, we use a KL-divergence type loss, measuring the difference between the action distribution produced by π and the action distribution preferred by π^{out} .

$$\begin{aligned} \mathcal{C}(\pi; \pi^{\text{out}}, s) &= D_{\text{KL}}(\pi^{\text{out}}(\cdot|s) \parallel \pi(\cdot|s)) \\ &= \sum_{a \in \mathcal{V}} \pi^{\text{out}}(a|s) \log \pi(a|s) + \text{const.} \end{aligned} \quad (2)$$

Our approach estimates the loss in Eq. (1) by first sampling one training sequence, running the roll-in policy for t steps,

and computing the KL divergence (2) at that state using π^* as π^{out} . Learning corresponds to minimizing this KL divergence iteratively with respect to the parameters of π .

3.3. Roll-In Policies

The *roll-in* policy determines the state distribution over which the learned policy π 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 et al., 2009; Ross et al., 2011; Ross & 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., 2015; 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 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 (§3.4) 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 are with oracle roll-ins.

3.4. Oracle Policies

In this section we formalize the oracle policies that we consider. To simplify the discussion (we assume that the roll-in distribution is the oracle), we only need to define an oracle policy that takes actions on states it, itself, visits. All the oracles we consider have access to the ground truth output Y , and the current state s . We interpret the state s as a partial binary tree and a “current node” in that binary tree where the next prediction will go. It is easiest to consider the behavior of the oracle as a top-down, level-order traversal of the tree, where in each state it maintains a sequence of “possible tokens” at that state. An oracle policy $\pi^*(\cdot|s_t)$ is defined with respect to Y_t , a consecutive subsequence of Y . At $s_0 = \langle \rangle$, π^* uses the full $Y_0 = Y$. This is subdivided as the tree is descended. At each state s_t , Y_t contains “valid actions”; labeling the current node with *any* token from Y_t keeps the generation leading to Y . For instance, in Figure 2, after sampling 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.

Given the consecutive subsequence $Y_t = (w'_1, \dots, w'_{N'})$, an oracle policy is defined as:

$$\pi^*(a|s_t) = \begin{cases} 1 & \text{if } a = \langle \text{end} \rangle \text{ and } Y_t = \langle \rangle \\ p_a & \text{if } a \in Y_t \\ 0 & \text{otherwise} \end{cases} \quad (3)$$

where the p_a s are arbitrary such that $\sum_{a \in Y} p_a = 1$. An oracle policy places positive probability only on valid actions,

and forces an $\langle \text{end} \rangle$ output if there are no more words to produce. This is guaranteed to *always* generate Y , regardless of how the random coin flips come up.

When an action a is chosen, at s_t , this “splits” the sub-sequence $Y_t = (w'_1, \dots, w'_{N'})$ into left and right sub-sequences, $\overleftarrow{Y}_t = (w'_1, \dots, w'_{i-1})$ and $\overrightarrow{Y}_t = (w'_{i+1}, \dots, w'_N)$, 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.

There are many possible oracle policies, and each of them is characterized by how p_a in Eq. (3) is defined. Specifically, we propose three variants.

Uniform Oracle. Motivated by Welleck et al. (2018) who applied learning-to-search to the problem of multiset prediction, we design a uniform oracle π_{uniform}^* . This oracle treats all possible generation orders that lead to the target sequence Y as equally likely, without preferring any specific set of orders. Formally, π_{uniform}^* gives uniform probabilities $p_a = 1/n$ for all words in Y_t where n is the number of unique words in Y_t . (Daumé (2009) used a similar oracle for unsupervised structured prediction, which has a similar non-deterministic oracle complication.)

Coaching Oracle. An issue with the uniform oracle is that it does not prefer any specific set of generation orders, making it difficult for a parameterized policy to imitate. This gap has been noticed as a factor behind the difficulty in learning-to-search by He et al. (2012), who propose the idea of coaching. In coaching, the oracle takes into account the preference of a parameterized policy in order to facilitate its learning. Motivated by this, we design a coaching oracle as the product of the uniform oracle and current policy π :

$$\pi_{\text{coaching}}^*(a|s) \propto \pi_{\text{uniform}}^*(a|s) \pi(a|s) \quad (4)$$

This coaching oracle ensures that no invalid action is assigned any probability, while preferring actions that are preferred by the current parameterized policy, reinforcing the selection by the current policy if it is valid.

Annealed Coaching Oracle. 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 an annealed coaching oracle:

$$\pi_{\text{annealed}}^*(a|s) = \beta \pi_{\text{uniform}}^*(a|s) + (1 - \beta) \pi_{\text{coaching}}^*(a|s) \quad (5)$$

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

Deterministic Left-to-Right Oracle. In addition to the proposed oracle policies above, 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 both roll-in and oracle policies are set to the left-to-right oracle $\pi_{\text{left-right}}^*$, the proposed approach recovers to maximum likelihood learning of an autoregressive sequence model, which is *de facto* standard in neural sequence modeling. In other words, supervised learning of an autoregressive sequence model is a special case of the proposed approach.

4. Neural Net Policy Structure

We use a neural network to implement the proposed binary tree generating policy, as it has been shown to encode a variable-sized input and predict a structured output effectively (Cleeremans et al., 1989; Forcada & Neco, 1997; Sutskever et al., 2014; Cho et al., 2014b; Tai et al., 2015; Bronstein et al., 2017; Battaglia et al., 2018). This neural network 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 & Schmidhuber, 1997) by considering the partial binary tree as a flat sequence of nodes in a level-order traversal (a_1, \dots, a_t) . The recurrent network encodes the sequence into a vector h_t and computes a categorical distribution over the action set:

$$\pi(a|s_t) \propto \exp(u_a^\top h_t + b_a) \quad (6)$$

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. It may be possible to exploit the tree structure more thoroughly using a recurrent architecture that is designed to encode a tree (Zhang et al., 2015; Alvarez-Melis & Jaakkola, 2016; Dyer et al., 2015; Bowman et al., 2016), which we leave for future investigation. 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, \dots, a_t is again summarized by a vector h_t , here computed using a multi-head attention mechanism. As in the LSTM policy, the vector h_t is used to compute a categorical distribution over the action set (6).

Auxiliary $\langle \text{end} \rangle$ Prediction. We also consider separating the action prediction into token ($a_i \in \mathcal{V}$) prediction and

$\langle \text{end} \rangle$ prediction. The policy under this view consists of a categorical distribution over tokens (6) as well as an $\langle \text{end} \rangle$ predictor which parameterizes a Bernoulli distribution, $\pi_{\text{end}}(\langle \text{end} \rangle | s_t) \propto \sigma(u_e^\top h_t + b_e)$, where $\pi_{\text{end}}(\langle \text{end} \rangle) = 1 | s_t$ means a_t is $\langle \text{end} \rangle$, and a_t is determined by π according to (6) otherwise. At test time, we threshold the predicted $\langle \text{end} \rangle$ probability at a threshold τ . In our experiments, we only use this approach with the Transformer policy (§5.4).

4.1. Conditional Sentence Generation

An advantage of using a neural network to implement the proposed policy is that it can be easily conditioned on an extra context. It allows us to build a conditional non-monotonic sequence generator that can for instance be used for machine translation, image caption generation, speech recognition and generally multimedia description generation (Cho et al., 2015). To do so, we assume that a conditioning input (e.g. an image or sentence) X can be represented as a set of d_{enc} -dimensional context vectors, obtained with a learned *encoder* function $f^{\text{enc}}(X)$ whose parameters are learned jointly with the policy’s.

For word-reordering experiments §5.3, the encoder outputs a single vector which is used to initialize the LSTM policy’s state h_0 . In the machine translation experiments §5.4, the Transformer encoder outputs $|X|$ vectors, $H \in \mathbb{R}^{|X| \times d_{\text{enc}}}$, which are used as input to a decoder (i.e. policy) attention function; see (Vaswani et al., 2017) for further details.

5. Experimental Results

In this section we experiment with our non-monotonic sequence generation model across four tasks. The first two are *unconditional* generation tasks: language modeling (§5.1) and out-of-order sentence completion (§5.2). Our analysis in these tasks is primarily qualitative: we seek to understand what the non-monotone policy is learning and how it compares to a left-to-right model. The second two tasks are *conditional* generation tasks, which generate output sequences based on some given input sequence: word reordering (§5.3) and machine translation (§5.4).

5.1. 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	Avg. Tokens	Avg. Span	BLEU
left-right	17.8	97.0	11.9	1.0	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 1. Statistics computed over 10,000 sampled sentences (in-order traversals of sampled trees with $\langle \text{end} \rangle$ 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.

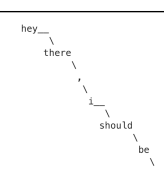
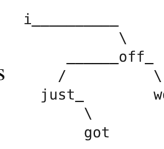
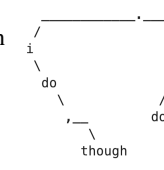
π^* Samples	
left-right	<ul style="list-style-type: none"> hey there , i should be ! not much fun . what are you doing ? not . not sure if you . i love to always get my nails done . sure , i can see your eye underwater while riding a footwork . 
uniform	<ul style="list-style-type: none"> i just got off 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	<ul style="list-style-type: none"> 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 2. Samples from unconditional generation policies trained on Persona-Chat for each training oracle. The first sample’s underlying tree is shown. See Appendix A.2 for more samples.

Dataset. 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.

Model. We use a uni-directional LSTM that has 2 layers of 1024 LSTM units. See Appendix A.2 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

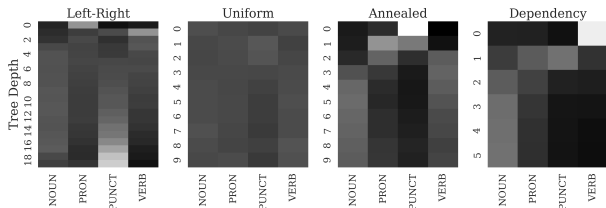


Figure 3. POS tag counts by tree-depth, computed by tagging 10,000 sampled sentences. Counts are normalized across each row (depth), then the marginal tag probabilities are subtracted. A light value means the probability of the tag occurring at that depth is higher than the prior probability of the tag occurring.

span size³ and BLEU (Table 1). 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) and Zhu et al. (2018). In Appendix Table 1 we report additional scores. 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 2, which shows samples from policies trained with different oracles. Each of the displayed samples are not a part of the training set. We provide additional samples organized by length in Appendix Tables 3 and 4, and samples showing the underlying trees that generated them in Appendix Figures 1-3. 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 3 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

³The average span is the average number of children for non-leaf nodes excluding the special token $\langle \text{end} \rangle$, ranging from 1.0 (chain, as induced by the left-right oracle) to 2.0 (full binary tree).

Initial Tree	Samples
	<ul style="list-style-type: none"> o lasagna is my favorite food ! o my favorite food is mac and cheese ! o what is your favorite food ? pizza , i love it ! o whats your favorite food ? mine is pizza ! o seafood is my favorite . and mexican food ! what is yours ?
	<ul style="list-style-type: none"> o hello ! i like classical music . do you ? o hello , do you enjoy playing music ? o hello just relaxing at home listening to fine music . you ? o hello , do you like to listen to music ? o hello . what kind of music do you like ?
	<ul style="list-style-type: none"> o i am a doctor or a lawyer . o i would like to feed my doctor , i aspire to be a lawyer . o i am a doctor lawyer . 4 years old . o i was a doctor but went to a lawyer . o i am a doctor since i want to be a lawyer .

Table 3. Sentence completion samples from a policy trained on Persona-Chat with the uniform oracle. The left column shows the initial seed tree. In the sampled sentences, seed words are bold.

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.

5.2. Sentence Completion

A major weakness of the conventional autoregressive model, especially with unbounded context, 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 given in advance. Achieving this requires significant changes to both model architecture, learning and inference (Berglund et al., 2015).

Our proposed approach, on the other hand, can naturally fill in missing segments in a sentence. Using models trained as language models from the previous section (§5.1), we can achieve this by initializing a binary tree with observed tokens in a way that they respect their relative positions. For instance, the first example shown in Table 3 can be seen as the template “___ favorite ___ food ___ ! ___” with variable-length missing segments. Generally, an initial tree with nodes (w_i, \dots, 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).

To quantify the completion quality, we first create a collection of initial trees by randomly sampling three words (w_i, w_j, w_k) from each sentence $Y = (w_1, \dots, w_T)$ from the Persona-Chat validation set of §5.1. We then sample one

Oracle	Validation			Test		
	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 4. Word Reordering results on Persona-Chat, reported according to BLEU score, F1 score, and percent exact match.

completion for each initial tree and measure the BLEU of each sample using the validation set as reference as in §5.1. 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 3, we present some sample completions using the policy trained with the uniform oracle. The completions illustrate a property of the proposed non-monotonic generation that is not available in left-to-right generation.

5.3. Word Reordering

We first evaluate the proposed 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 $Y = (w_1, \dots, w_N)$ is given as an unordered collection $X = \{w_1, \dots, w_N\}$, and the task is to reconstruct Y from X . We assemble a dataset of (X, Y) pairs using sentences Y from the Persona-Chat sentence dataset of §5.1. In our approach, we do not explicitly force the policies trained with our non-monotonic oracles to produce a permutation of the input and instead let them learn this automatically.

Model. For encoding each unordered input $x = \{w_1, \dots, w_N\}$, we use a simple bag-of-words encoder: $f^{\text{enc}}(\{w_1, \dots, w_N\}) = \frac{1}{T} \sum_{i=1}^N \text{emb}(w_i)$. 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 $f^{\text{enc}}(\{w_1, \dots, w_T\})$.

Results. Table 4 shows BLEU, F1 score, and exact match for policies trained with each oracle. The uniform and annealed policies 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. See Appendix Figure 6 for example predictions.

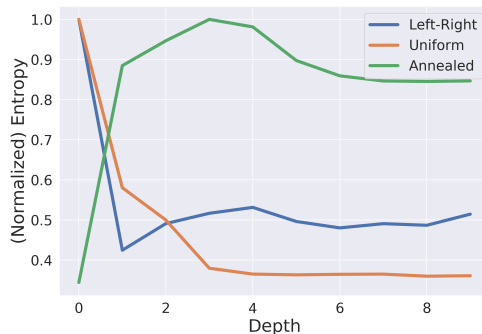


Figure 4. Normalized entropy of $\pi(\cdot|s)$ as a function of tree depth for policies trained with each of the oracles. The anneal-trained policy, unlike the others, makes low entropy decisions early.

Easy-First Analysis. Figure 4 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 3) and defers harder decisions until later.

5.4. Machine Translation

Dataset. We evaluate the proposed models on 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.

Model & Training. 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 experiments. For the $\langle \text{end} \rangle$ 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 & Quirk, 2019). See Appendix A.3 for additional details and results with a Bi-LSTM encoder-decoder architecture.

Results. Results on validation and test data are in Table 5 according to four (very) different evaluation measures: BLEU, Meteor (Lavie & Agarwal, 2007), YiSi (Lo, 2018), and Ribes (Isozaki et al., 2010). First focusing on the non-monotonic models, we see that the annealed policy outperforms the uniform policy on all metrics, with tree-encoding yielding further gains. Adding $\langle \text{end} \rangle$ tuning to the tree-encoding model decreases the Ribes score but improves the other metrics, notably increasing the BLEU brevity penalty.

Oracle	Validation				Test			
	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 5. Results of machine translation experiments for different training oracles across four different evaluation metrics.

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 models with 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.

6. Related Work

Arguably one of the most successful approaches for generating discrete sequences, or sentences, is neural autoregressive modeling (Sutskever et al., 2011; Tomas, 2012). It has become *de facto* standard in machine translation (Cho et al., 2014a; Sutskever et al., 2014) and is widely studied for dialogue response generation (Vinyals & Le, 2015) as well as speech recognition (Chorowski et al., 2015). On the other hand, recent works have shown that it is possible to generate a sequence of discrete tokens in parallel by capturing strong dependencies among the tokens in a non-autoregressive way (Gu et al., 2017; Lee et al., 2018; Oord et al., 2017). Stern et al. (2018) and Wang et al. (2018) proposed to mix in these two paradigms and build a semi-autoregressive sequence generator, while largely sticking to left-to-right generation. Our proposal radically departs from these conventional approaches by building an algorithm that automatically captures a distinct generation order.

In (neural) language modeling, there is a long tradition of modeling the probability of a sequence as a tree or directed graph. For example, Emami & Jelinek (2005) proposed to factorize the probability over a sentence following its syntactic structure and train a neural network to model conditional distributions, which was followed more recently by Zhang et al. (2015) and by Dyer et al. (2016). This approach was applied to neural machine translation by Eriguchi et al. (2017) and Aharoni & Goldberg (2017). In all cases, these approaches require the availability of the ground-truth parse of a sentence or access to an external parser during training or inference time. This is unlike the proposed approach

which does not require any such extra annotation or tool and learns to sequentially generate a sequence in an automatically determined non-monotonic order.

7. Conclusion, Limitations & Future Work

We described an approach to 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.

There are several potentially interesting avenues for future work. One is to solve the “learning to stop” problem directly, rather than through an after-the-fact tuning step. Another is to better understand how to construct an oracle that generalizes well after mistakes have been made, in order to train off of the gold path(s).

Moreover, the proposed formulation of sequence generation by tree generation is limited to binary trees. It is possible to extend the proposed approach to n -ary trees by designing a policy to output up to $n + 1$ decisions at each node, leading to up to n child nodes. This would bring a set of generation orders, that could be captured by the proposed approach, which includes all projective dependency parses. A new oracle must be designed for n -ary trees, and we leave this as a follow-up work.

Finally, although the proposed approach indeed learns to sequentially generate a sequence in a non-monotonic order, it cannot consider all possible orders. It is due to the constraint that there cannot be any crossing of two edges when the nodes are arranged on a line following the inorder traversal, which we refer to as projective generation. Extending the proposed approach to non-projective generation, which we leave as future work, would expand the number of generation orders considered during learning.

Acknowledgements

We thank support by eBay, TenCent and NVIDIA. This work was partly supported by Samsung Advanced Institute of Technology (Next Generation Deep Learning: from pattern recognition to AI), Samsung Electronics (Improving Deep Learning using Latent Structure), Sloan Foundation Research Fellowship, NSF Louis Stokes Alliances for Minority Participation Bridge to Doctorate (#1612736) and ACM SIGHPC/Intel Computational and Data Science Fellowship.

References

- Aharoni, R. and Goldberg, Y. Towards string-to-tree neural machine translation. *arXiv preprint arXiv:1704.04743*, 2017.
- Alvarez-Melis, D. and Jaakkola, T. S. Tree-structured decoding with doubly-recurrent neural networks. 2016.
- Bahdanau, D., Serdyuk, D., Brakel, P., Ke, N. R., Chorowski, J., Courville, A., and Bengio, Y. Task loss estimation for sequence prediction. *arXiv preprint arXiv:1511.06456*, 2015.
- Bahdanau, D., Brakel, P., Xu, K., Goyal, A., Lowe, R., Pineau, J., Courville, A., and Bengio, Y. An actor-critic algorithm for sequence prediction. *arXiv preprint arXiv:1607.07086*, 2016.
- Bahl, L. R., Jelinek, F., and Mercer, R. L. A maximum likelihood approach to continuous speech recognition. *IEEE transactions on pattern analysis and machine intelligence*, (2):179–190, 1983.
- Battaglia, P. W., Hamrick, J. B., Bapst, V., Sanchez-Gonzalez, A., Zambaldi, V., Malinowski, M., Tacchetti, A., Raposo, D., Santoro, A., Faulkner, R., et al. Relational inductive biases, deep learning, and graph networks. *arXiv preprint arXiv:1806.01261*, 2018.
- Bengio, Y., Ducharme, R., Vincent, P., and Jauvin, C. A neural probabilistic language model. *Journal of machine learning research*, 3(Feb):1137–1155, 2003.
- Berglund, M., Raiko, T., Honkala, M., Kärkkäinen, L., Vetek, A., and Karhunen, J. T. Bidirectional recurrent neural networks as generative models. In *Advances in Neural Information Processing Systems*, pp. 856–864, 2015.
- Bowman, S. R., Gauthier, J., Rastogi, A., Gupta, R., Manning, C. D., and Potts, C. A fast unified model for parsing and sentence understanding. *arXiv preprint arXiv:1603.06021*, 2016.
- Bronstein, M. M., Bruna, J., LeCun, Y., Szlam, A., and Vandergheynst, P. Geometric deep learning: going beyond euclidean data. *IEEE Signal Processing Magazine*, 34(4): 18–42, 2017.
- Brown, P. F., Cocke, J., Pietra, S. A. D., Pietra, V. J. D., Jelinek, F., Lafferty, J. D., Mercer, R. L., and Roossin, P. S. 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>.
- Callison-Burch, C., Osborne, M., and Koehn, P. Re-evaluation the role of bleu in machine translation research. In *11th Conference of the European Chapter of the Association for Computational Linguistics*, 2006.
- Chang, K.-W., Krishnamurthy, A., Agarwal, A., Daumé III, H., and Langford, J. Learning to search better than your teacher. *arXiv preprint arXiv:1502.02206*, 2015.
- Cheng, C.-A., Yan, X., Wagener, N., and Boots, B. Fast Policy Learning through Imitation and Reinforcement. *arXiv preprint 1805.10413*, 2018. URL <https://arxiv.org/pdf/1805.10413.pdf>.
- Chiang, D. Hope and fear for discriminative training of statistical translation models. *Journal of Machine Learning Research*, 13(Apr):1159–1187, 2012.
- Cho, K., Van Merriënboer, B., Bahdanau, D., and Bengio, Y. On the properties of neural machine translation: Encoder-decoder approaches. *arXiv preprint arXiv:1409.1259*, 2014a.
- Cho, K., Van Merriënboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., and Bengio, Y. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014b.
- Cho, K., Courville, A., and Bengio, Y. Describing multimedia content using attention-based encoder-decoder networks. *IEEE Transactions on Multimedia*, 17(11): 1875–1886, 2015.
- Chorowski, J. K., Bahdanau, D., Serdyuk, D., Cho, K., and Bengio, Y. Attention-based models for speech recognition. In *Advances in neural information processing systems*, pp. 577–585, 2015.
- Cleeremans, A., Servan-Schreiber, D., and McClelland, J. L. Finite state automata and simple recurrent networks. *Neural computation*, 1(3):372–381, 1989.
- Daumé, H., Langford, J., and Marcu, D. Search-based structured prediction. *Machine learning*, 75(3):297–325, 2009.

- Daumé, III, H. Unsupervised search-based structured prediction. In *International Conference on Machine Learning (ICML)*, Montreal, Canada, 2009.
- Dyer, C., Ballesteros, M., Ling, W., Matthews, A., and Smith, N. A. Transition-based dependency parsing with stack long short-term memory. *arXiv preprint arXiv:1505.08075*, 2015.
- Dyer, C., Kuncoro, A., Ballesteros, M., and Smith, N. A. Recurrent neural network grammars. *arXiv preprint arXiv:1602.07776*, 2016.
- Emami, A. and Jelinek, F. A neural syntactic language model. *Machine learning*, 60(1-3):195–227, 2005.
- Eriguchi, A., Tsuruoka, Y., and Cho, K. Learning to parse and translate improves neural machine translation. *arXiv preprint arXiv:1702.03525*, 2017.
- Forcada, M. L. and Neco, R. Recursive hetero-associative memories for translation. In *International Workshop Conference on Artificial Neural Networks*, 1997.
- Ford, N., Duckworth, D., Norouzi, M., and Dahl, G. E. The importance of generation order in language modeling. *arXiv preprint arXiv:1808.07910*, 2018.
- Goldberg, Y. and Elhadad, M. 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*, pp. 742–750. Association for Computational Linguistics, 2010.
- Gu, J., Bradbury, J., Xiong, C., Li, V. O., and Socher, R. Non-autoregressive neural machine translation. *arXiv preprint arXiv:1711.02281*, 2017.
- Hazan, T., Keshet, J., and McAllester, D. A. Direct loss minimization for structured prediction. In *Advances in Neural Information Processing Systems*, pp. 1594–1602, 2010.
- He, H., Eisner, J., and Daume, H. Imitation learning by coaching. In *Advances in Neural Information Processing Systems*, pp. 3149–3157, 2012.
- Hochreiter, S. and Schmidhuber, J. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.
- Isozaki, H., Hirao, T., Duh, K., Sudoh, K., and Tsukada, H. Automatic evaluation of translation quality for distant language pairs. In *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing*, pp. 944–952. Association for Computational Linguistics, 2010.
- Lavie, A. and Agarwal, A. Meteor: An automatic metric for mt evaluation with high levels of correlation with human judgments. In *Proceedings of the Second Workshop on Statistical Machine Translation*, pp. 228–231. Association for Computational Linguistics, 2007.
- Leblond, R., Alayrac, J.-B., Osokin, A., and Lacoste-Julien, S. SeaRNN: Training RNNs with global-local losses. In *ICLR*, 2018.
- Lee, J., Mansimov, E., and Cho, K. Deterministic non-autoregressive neural sequence modeling by iterative refinement. *arXiv preprint arXiv:1802.06901*, 2018.
- Lo, C. YiSi: A semantic machine translation evaluation metric for evaluating languages with different levels of available resources. 2018. URL <http://chikiu-jackie-lo.org/home/index.php/yisi>.
- Oord, A. v. d., Li, Y., Babuschkin, I., Simonyan, K., Vinyals, O., Kavukcuoglu, K., Driessche, G. v. d., Lockhart, E., Cobo, L. C., Stimberg, F., et al. Parallel wavenet: Fast high-fidelity speech synthesis. *arXiv preprint arXiv:1711.10433*, 2017.
- Pennington, J., Socher, R., and Manning, C. D. Glove: Global vectors for word representation. In *Empirical Methods in Natural Language Processing (EMNLP)*, pp. 1532–1543, 2014. URL <http://www.aclweb.org/anthology/D14-1162>.
- Ranzato, M., Chopra, S., Auli, M., and Zaremba, W. Sequence level training with recurrent neural networks. *arXiv preprint arXiv:1511.06732*, 2015.
- Ross, S. and Bagnell, J. A. Reinforcement and imitation learning via interactive no-regret learning. *arXiv preprint arXiv:1406.5979*, 2014.
- Ross, S., Gordon, G., and Bagnell, D. A reduction of imitation learning and structured prediction to no-regret online learning. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pp. 627–635, 2011.
- Schmaltz, A., Rush, A. M., and Shieber, S. Word ordering without syntax. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pp. 2319–2324. Association for Computational Linguistics, 2016. doi: 10.18653/v1/D16-1255. URL <http://aclweb.org/anthology/D16-1255>.
- Shiv, V. L. and Quirk, C. Novel positional encodings to enable tree-structured transformers. 2019. URL <https://openreview.net/forum?id=SJerEhR5Km>.
- Stern, M., Shazeer, N., and Uszkoreit, J. Blockwise parallel decoding for deep autoregressive models. In *Advances*

- in *Neural Information Processing Systems*, pp. 10107–10116, 2018.
- Stoyanov, V. and Eisner, J. Easy-first coreference resolution. *Proceedings of COLING 2012*, pp. 2519–2534, 2012.
- Sutskever, I., Martens, J., and Hinton, G. E. Generating text with recurrent neural networks. In *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, pp. 1017–1024, 2011.
- Sutskever, I., Vinyals, O., and Le, Q. V. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pp. 3104–3112, 2014.
- Tai, K. S., Socher, R., and Manning, C. D. Improved semantic representations from tree-structured long short-term memory networks. *arXiv preprint arXiv:1503.00075*, 2015.
- Tomas, M. Statistical language models based on neural networks. *Brno University of Technology*, 2012.
- Tsuruoka, Y. and Tsujii, J. 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*, pp. 467–474. Association for Computational Linguistics, 2005.
- Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L. u., and Polosukhin, I. Attention is all you need. In Guyon, I., Luxburg, U. V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems 30*, pp. 5998–6008. Curran Associates, Inc., 2017. URL <http://papers.nips.cc/paper/7181-attention-is-all-you-need.pdf>.
- Vinyals, O. and Le, Q. A neural conversational model. *arXiv preprint arXiv:1506.05869*, 2015.
- Wang, C., Zhang, J., and Chen, H. Semi-autoregressive neural machine translation. *arXiv preprint arXiv:1808.08583*, 2018.
- Welleck, S., Yao, Z., Gai, Y., Mao, J., Zhang, Z., and Cho, K. Loss functions for multiset prediction. In *Advances in Neural Information Processing Systems*, pp. 5788–5797, 2018.
- Wilks, Y. *Machine translation: its scope and limits*. Springer Science & Business Media, 2008.
- Yu, L., Zhang, W., Wang, J., and Yu, Y. 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>.
- Zhang, S., Dinan, E., Urbanek, J., Szlam, A., Kiela, D., and Weston, J. 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)*, pp. 2204–2213, Melbourne, Australia, 2018. Association for Computational Linguistics.
- Zhang, X., Lu, L., and Lapata, M. Top-down tree long short-term memory networks. *arXiv preprint arXiv:1511.00060*, 2015.
- Zhu, Y., Lu, S., Zheng, L., Guo, J., Zhang, W., Wang, J., and Yu, Y. Taxygen: A benchmarking platform for text generation models. *SIGIR*, 2018.