Insertion-based Decoding with Automatically Inferred Generation Order

Jiatao Gu[†], **Qi Liu**[◇*], **and Kyunghyun Cho**[‡†]

[†]Facebook AI Research
[◇]University of Oxford
[‡]New York University, CIFAR Azrieli Global Scholar
[†]{jgu, kyunghyuncho}@fb.com [‡]qi.liu@st-hughs.ox.ac.uk

Abstract

Conventional neural autoregressive decoding commonly assumes a fixed left-to-right generation order, which may be sub-optimal. In this work, we propose a novel decoding algorithm—InDIGO—which supports flexible sequence generation in arbitrary orders through insertion operations. We extend Transformer, a state-of-the-art sequence generation model, to efficiently implement the proposed approach, enabling it to be trained with either a pre-defined generation order or adaptive orders obtained from beam-search. Experiments on four real-world tasks, including word order recovery, machine translation, image caption, and code generation, demonstrate that our algorithm can generate sequences following arbitrary orders, while achieving competitive or even better performance compared with the conventional left-to-right generation. The generated sequences show that InDIGO adopts adaptive generation orders based on input information.

1 Introduction

Neural autoregressive models have become the *de facto* standard in a wide range of sequence generation tasks, such as machine translation (Bahdanau et al., 2015), summarization (Rush et al., 2015), and dialogue systems (Vinyals and Le, 2015). In these studies, a sequence is modeled autoregressively with the left-to-right generation order, which raises the question of whether generation in an arbitrary order is worth considering (Vinyals et al., 2016; Ford et al., 2018). Nevertheless, previous studies on generation orders mostly resort to a fixed set of generation orders, showing particular choices of ordering are helpful (Wu et al., 2018; Ford et al., 2018; Mehri and Sigal, 2018), without providing an efficient algorithm for finding adaptive generation orders, or restrict the problem scope to n-gram segment generation (Vinyals et al., 2016).

In this paper, we propose a novel decoding algorithm, *Insertion-based Decoding with Inferred Generation Order* (InDIGO), which models generation orders as latent variables and automatically infers the generation orders by simultaneously predicting a word and its position to be inserted at each decoding step. Given that absolute positions are unknown before generating the whole sequence, we use a relative-position-based representation to capture generation orders. We show that decoding consists of a series of insertion operations with a demonstration shown in Figure 1.

We extend Transformer (Vaswani et al., 2017) for supporting insertion operations, where the generation order is directly captured as relative positions through self-attention inspired by Shaw et al. (2018). For learning, we maximize the evidence lower-bound (ELBO) of the maximum likelihood objective, and study two approximate posterior distributions of generation orders based on a pre-defined generation order and adaptive orders obtained from beam-search, respectively.

Experimental results on word order recovery, machine translation, code generation, and image caption demonstrate that our algorithm can generate sequences with arbitrary orders, while achieving competitive or even better performance compared to the conventional left-to-right generation. Case studies show that the proposed method adopts adaptive orders based on input information. The code will be released as part of the official

[*]This work was completed while the author worked as an AI resident at Facebook AI Research.

Transactions of the Association for Computational Linguistics, vol. 7, pp. 661–676, 2019. https://doi.org/10.1162/tacl_a_00292
Action Editor: Alexandra Birch. Submission batch: 4/2019; Revision batch: 7/2019; Published: 11/2019.

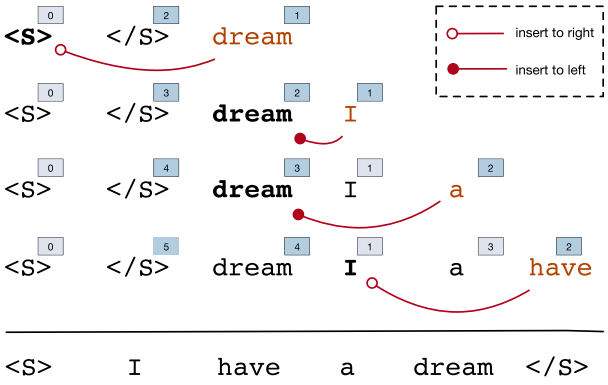

Figure 1: An example of InDIGO. At each step, we simultaneously predict the next token and its (relative) position to be inserted. The final output sequence is obtained by mapping the words based on their positions.

repository of Fairseq (`https://github.com/pytorch/fairseq`).

2 Neural Autoregressive Decoding

Let us consider the problem of generating a sequence $\boldsymbol{y} = (y_1, ..., y_T)$ conditioned on some inputs, e.g., a source sequence $\boldsymbol{x} = (x_1, ..., x_{T'})$. Our goal is to build a model parameterized by θ that models the conditional probability of $\boldsymbol{y}$ given $\boldsymbol{x}$, which is factorized as:

$$p_\theta(\boldsymbol{y}|\boldsymbol{x}) = \prod_{t=0}^{T} p_\theta(y_{t+1}|y_{0:t}, x_{1:T'}), \quad (1)$$

where y_0 and y_{T+1} are special tokens $\langle$s$\rangle$ and $\langle$/s$\rangle$, respectively. The model sequentially predicts the conditional probability of the next token at each step t, which can be implemented by any function approximator such as RNNs (Bahdanau et al., 2015) and Transformer (Vaswani et al., 2017).

Learning Neural autoregressive model is commonly learned by maximizing the conditional likelihood $\log p(\boldsymbol{y}|\boldsymbol{x}) = \sum_{t=0}^{T} \log p_\theta(y_{t+1}|y_{0:t}, x_{1:T'})$ given a set of parallel examples.

Decoding A common way to decode a sequence from a trained model is to make use of the autoregressive nature that allows us to predict one word at each step. Given any source $\boldsymbol{x}$, we essentially follow the order of factorization to generate tokens sequentially using some heuristic-based algorithms such as greedy decoding and beam-search.

3 Insertion-based Decoding with Inferred Generation Order (InDIGO)

Equation 1 explicitly assumes a *left-to-right* (L2R) generation order of the sequence $\boldsymbol{y}$. In principle, we can factorize the sequence probability in any permutation and train a model for each permutation separately. As long as we have an infinite amount of data with proper optimization performed, all these models are equivalent. Nevertheless, Vinyals et al. (2016) have shown that the generation order of a sequence actually matters in many real-world tasks, e.g., language modeling.

Although the L2R order is a strong inductive bias, as it is natural for most human beings to read and write sequences from left to right, L2R is not necessarily the optimal option for generating sequences. For instance, people sometimes tend to think of central phrases first before building up a whole sentence. For programming languages, it is beneficial to be generated based on abstract syntax trees (Yin and Neubig, 2017).

Therefore, a natural question arises, *how can we decode a sequence in its best order?*

3.1 Orders as Latent Variables

We address this question by modeling generation orders as latent variables. Similar to Vinyals et al. (2016), we rewrite the target sequence $\boldsymbol{y}$ in a particular order $\boldsymbol{\pi} = (z_2, ..., z_T, z_{T+1}) \in \mathcal{P}_T$[1] as a set $\boldsymbol{y}_{\boldsymbol{\pi}} = \{(y_2, z_2), ..., (y_{T+1}, z_{T+1})\}$, where (y_t, z_t) represents the t-th generated token and its absolute position, respectively. Different from the common notation, the target sequence is 2-step drifted because the two special tokens $(y_0, z_0) = (\langle s \rangle, 0)$ and $(y_1, z_1) = (\langle /s \rangle, T+1)$ are always prepended to represent the left and right boundaries, respectively. Then, we model the conditional probability as the joint distribution of words and positions by marginalizing all the orders:

$$p_\theta(\boldsymbol{y}|\boldsymbol{x}) = \sum_{\boldsymbol{\pi} \in \mathcal{P}_T} p_\theta(\boldsymbol{y}_{\boldsymbol{\pi}}|\boldsymbol{x}),$$

where for each element:

$$p_\theta(\boldsymbol{y}_{\boldsymbol{\pi}}|\boldsymbol{x}) = p_\theta(\underline{y_{T+2}}|y_{0:T+1}, z_{0:T+1}, x_{1:T'}) \cdot \prod_{t=1}^{T} p_\theta(\underline{y_{t+1}, z_{t+1}}|y_{0:t}, z_{0:t}, x_{1:T'}) \quad (2)$$

[1] $\mathcal{P}_T$ is the set of all the permutations of $(1, ..., T)$.

where the third special token $y_{T+2} = \langle\text{eod}\rangle$ is introduced to signal the end-of-decoding, and $p(y_{T+2}|\cdot)$ is the end-of-decoding probability.

At decoding time, the factorization allows us to decode autoregressively by predicting word y_{t+1} and its position z_{t+1} step by step. The generation order is *automatically inferred* during decoding.

3.2 Relative Representation of Positions

It is difficult and inefficient to predict the absolute positions z_t without knowing the actual length T. One solution is directly using the absolute positions $z_0^t, ..., z_t^t$ of the partial sequence $y_{0:t}$ at each autoregressive step t. For example, the absolute positions for the sequence ($\langle$s$\rangle$, $\langle$/s$\rangle$, dream, I) are $(z_0^t = 0, z_1^t = 3, z_2^t = 2, z_3^t = 1)$ in Figure 1 at step $t = 3$. It is, however, inefficient to model such explicit positions using a single neural network without recomputing the hidden states for the entire partial sequence, as some positions are changed at every step (as shown in Figure 1).

Relative Positions We propose using relative-position representations $\boldsymbol{r}_{0:t}^t$ instead of absolute positions $z_{0:t}^t$. We use a ternary vector $\boldsymbol{r}_i^t \in \{-1, 0, 1\}^{t+1}$ as the relative-position representation for z_i^t. The j-th element of $\boldsymbol{r}_i^t$ is defined as:

$$\boldsymbol{r}_{i,j}^t = \begin{cases} -1 & z_j^t > z_i^t \ \text{(left)} \\ 0 & z_j^t = z_i^t \ \text{(middle)} \\ 1 & z_j^t < z_i^t \ \text{(right)} \end{cases}, \quad (3)$$

where the elements of $\boldsymbol{r}_i^t$ show the relative positions with respect to all the other words in the partial sequence at step t. We use a matrix $R^t = \left[\boldsymbol{r}_0^t, \boldsymbol{r}_1^t, ..., \boldsymbol{r}_t^t\right]$ to show the relative-position representations of all the words in the sequence. The relative-position representation can always be mapped back to the absolute position z_i^t by:

$$z_i^t = \sum_{j=0}^{t} \max(0, \boldsymbol{r}_{i,j}^t) \quad (4)$$

One of the biggest advantages for using such vector-based representations is that at each step, updating the relative-position representations is simply *extending* the relative-position matrix R^t with the next predicted relative position, because the (left, middle, right) relations described in Equation (3) stay unchanged once they are created. Thus, we update R^t as follows:

$$R^{t+1} = \left[\begin{array}{ccc|c} & R^t & & \boldsymbol{r}_{t+1,0}^{t+1} \\ & & & \vdots \\ & & & \boldsymbol{r}_{t+1,t}^{t+1} \\ \hline -\boldsymbol{r}_{t+1,0}^{t+1} & \cdots & -\boldsymbol{r}_{t+1,t}^{t+1} & 0 \end{array}\right] \quad (5)$$

where we use $\boldsymbol{r}_{t+1}^{t+1}$ to represent the relative position at step $t + 1$. This append-only property enables our method to reuse the previous hidden states without recomputing the hidden states at each step. For simplicity, the superscript of $\boldsymbol{r}$ is omitted from now on without causing conflicts.

Algorithm 1 Insertion-based Decoding

Initialize: $\boldsymbol{y} = (\langle\text{s}\rangle, \langle\text{/s}\rangle)$, $R = \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix}$, $t = 1$
repeat
  Predict the next word y_{t+1} based on $\boldsymbol{y}, R$.
  if y_{t+1} is $\langle\text{eod}\rangle$ **then**
    break
  end if
  Choose an existing word $y_k \in \boldsymbol{y}$;
  Choose the left or right (s) of y_k to insert;
  Obtain the next position $\boldsymbol{r}_{t+1}$ with k, s (Eq. (6)).
  Update R by appending $\boldsymbol{r}_{t+1}$ (Eq. (5)).
  Update $\boldsymbol{y}$ by appending y_{t+1}
  Update $t = t + 1$
until Reach the maximum length
Map back to absolute positions $\boldsymbol{\pi}$ (Eq. (4))
Reorder $\boldsymbol{y}$: $y_{z_i} = y_i \quad \forall z_i \in \boldsymbol{\pi}, i \in [0, t]$

3.3 Insertion-based Decoding

Given a partial sequence $y_{0:t}$ and its corresponding relative-position representations $\boldsymbol{r}_{0:t}$, not all of the 3^{t+2} possible vectors are valid for the next relative-position representation, $\boldsymbol{r}_{t+1}$. Only these vectors corresponding to *insertion* operations satisfy Equation (4). In Algorithm 1, we describe an insertion-based decoding framework based on this observation. The next word y_{t+1} is predicted based on $y_{0:t}$ and $\boldsymbol{r}_{0:t}$. We then choose an existing word y_k ($0 \le k \le t$)) from $y_{0:t}$ and insert y_{t+1} to its left or right. As a result, the next position $\boldsymbol{r}_{t+1}$ is determined by

$$\boldsymbol{r}_{t+1,j} = \begin{cases} s & j = k \\ \boldsymbol{r}_{k,j} & j \neq k \end{cases}, \quad \forall j \in [0, t] \quad (6)$$

where $s = -1$ if y_{t+1} is on the left of y_k, and $s = 1$ otherwise. Finally, we use $\boldsymbol{r}_{t+1}$ to update the relative-position matrix R as shown in Equation (5).

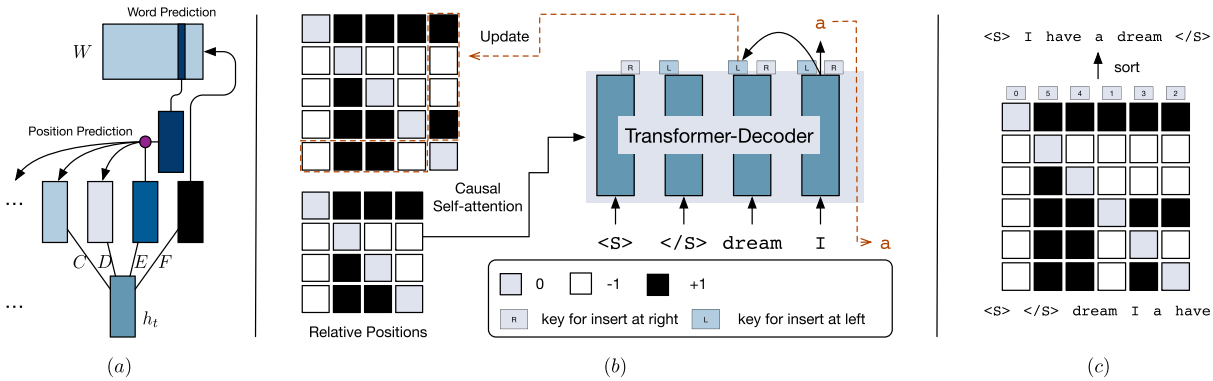

Figure 2: The overall framework of the proposed Transformer-InDIGO which includes (a) the word & position prediction module; (b) the one step decoding with position updating; (c) final decoding output by reordering. The black-white blocks represent the relative position matrix.

4 Model

We present Transformer-InDIGO, an extension of Transformer (Vaswani et al., 2017), supporting insertion-based decoding. The overall framework is shown in Figure 2.

4.1 Network Design

We extend the decoder of Transformer with relative-position-based self-attention, joint word and position prediction, and position updating modules.

Self-Attention One of the major challenges that prevents the vanilla Transformer from generating sequences following arbitrary orders is that the absolute-position-based positional encodings are inefficient (as mentioned in Section 3.2), in that absolute positions are changed during decoding, invalidating the previous hidden states. In contrast, we adapt Shaw et al. (2018) to use relative positions in self-attention. Different from Shaw et al. (2018), in which a clipping distance d (usually $d \geq 2$) is set for relative positions, our relative-position representations only preserve $d = 1$ relations (Equation (3)).

Each attention head in a multi-head self-attention module of Transformer-InDIGO takes the hidden states of a partial sequence $y_{0:t}$, denoted as $U = (\boldsymbol{u}_0, ..., \boldsymbol{u}_t)$, and its corresponding relative position matrix R^t as input, where each input state $\boldsymbol{u}_i \in \mathbb{R}^{d_{\text{model}}}$. The logit $e_{i,j}$ for attention is computed as:

$$e_{i,j} = \frac{\left(\boldsymbol{u}_i^\top Q\right) \cdot \left(\boldsymbol{u}_j^\top K + A_{[\boldsymbol{r}_{i,j}+1]}\right)^\top}{\sqrt{d_{\text{model}}}}, \quad (7)$$

where $Q, K \in \mathbb{R}^{d_{\text{model}} \times d_{\text{model}}}$ and $A \in \mathbb{R}^{3 \times d_{\text{model}}}$ are parameter matrices. $A_{[\boldsymbol{r}_{i,j}+1]}$ is the row vector indexed by $\boldsymbol{r}_{i,j} + 1$, which biases all the input keys based on the relative position, $\boldsymbol{r}_{i,j}$.

Word and Position Prediction Like the vanilla Transformer, we take the representations from the last layer of self-attention, $H = (\boldsymbol{h}_0, ..., \boldsymbol{h}_t)$ and $H \in \mathbb{R}^{d_{\text{model}} \times (t+1)}$, to predict both the next word y_{t+1} and its position vector $\boldsymbol{r}_{t+1}$ in two stages based on the following factorization:

$$p(y_{t+1}, \boldsymbol{r}_{t+1}|H) = p(y_{t+1}|H) \cdot p(\boldsymbol{r}_{t+1}|y_{t+1}, H)$$

It can also be factorized as predicting the position before predicting the next word, yet our preliminary experiments show that predicting the word first works slightly better. The prediction module for word and position prediction are shown in Figure 2(a).

First, we predict the next word y_{t+1} from the categorical distribution $p_{\text{word}}(y|H)$ as:

$$p_{\text{word}}(y|H) = \text{softmax}\left((\boldsymbol{h}_t^\top F) \cdot W^\top\right), \quad (8)$$

where $W \in \mathbb{R}^{d_{\text{V}} \times d_{\text{model}}}$ is the embedding matrix and d_{V} is the size of vocabulary. We linearly project the last representation $\boldsymbol{h}_t$ using $F \in \mathbb{R}^{d_{\text{model}} \times d_{\text{model}}}$ for querying W. Then, as shown in Equation (6), the prediction of the next position is done by performing insertion operations to existing words that can be modeled similarly to Pointer Networks (Vinyals et al., 2015). We predict a pointer $k_{t+1} \in [0, 2t+1]$ based on:

$$p_{\text{pointer}}(k|y_{t+1}, H) = \text{softmax}\left((\boldsymbol{h}_t^\top E + W_{[y_{t+1}]}) \cdot \begin{bmatrix} H^\top C \\ H^\top D \end{bmatrix}^\top\right), \quad (9)$$

where $C, D, E \in \mathbb{R}^{d_{\text{model}} \times d_{\text{model}}}$ and $W_{[y_{t+1}]}$ is the embedding of the predicted word. C, D are used to obtain the left and right keys, respectively, considering that each word has two ''keys'' (its left and right) for inserting the generated word. The query vector is obtained by adding up the word embedding $W_{[y_{t+1}]}$, and the linearly projected state, $\boldsymbol{h}_t^\top E$. The resulting relative-position vector, $\boldsymbol{r}_{t+1}$ is computed using k_{t+1} according to Equation (6). We manually set $p_{\text{pointer}}(0|\cdot) = p_{\text{pointer}}(2{+}t|\cdot) = 0$ to avoid any word from being inserted to the left of $\langle \text{s} \rangle$ and the right of $\langle \text{/s} \rangle$.

Position Updating As mentioned in Sec. 3.1, we update the relative position representation R^t with the predicted r_{t+1}. Because updating the relative positions will not change the pre-computed relative-position representations, Transformer-InDIGO can reuse the previous hidden states in the next decoding step the same as the vanilla Transformer.

4.2 Learning

Training requires maximizing the marginalized likelihood in Equation (2). Yet this is intractable since we need to enumerate all of the $T!$ permutations of tokens. Instead, we maximize the ELBO of the original objective by introducing an approximate posterior distribution of generation orders $q(\boldsymbol{\pi}|\boldsymbol{x}, \boldsymbol{y})$, which provides the probabilities of latent generation orders based on the ground-truth sequences $\boldsymbol{x}$ and $\boldsymbol{y}$:

$$\begin{aligned}\mathcal{L}_{\text{ELBO}} &= \mathop{\mathbb{E}}_{\boldsymbol{\pi}\sim q} \log p_\theta(\boldsymbol{y}_{\boldsymbol{\pi}}|\boldsymbol{x}) + \mathcal{H}(q) \\ &= \mathop{\mathbb{E}}_{\boldsymbol{r}_{2:T+1}\sim q} \Bigg(\sum_{t=1}^{T+1} \underbrace{\log p_\theta(y_{t+1}|y_{0:t}, \boldsymbol{r}_{0:t}, x_{1:T'})}_{\text{Word Prediction Loss}} \\ &+ \sum_{t=1}^{T} \underbrace{\log p_\theta(\boldsymbol{r}_{t+1}|y_{0:t+1}, \boldsymbol{r}_{0:t}, x_{1:T'})}_{\text{Position Prediction Loss}} \Bigg) + \mathcal{H}(q),\end{aligned} \tag{10}$$

where $\boldsymbol{\pi} = \boldsymbol{r}_{2:T+1}$, sampled from $q(\pi|\boldsymbol{x}, \boldsymbol{y})$, is represented as relative positions. $\mathcal{H}(q)$ is the entropy term which can be ignored if q is fixed during training. Equation (10) shows that given a sampled order, the learning objective is divided into word and position objectives. For calculating the position prediction loss, we aggregate the two probabilities corresponding to the same position by

$$p_\theta(\boldsymbol{r}_{t+1}|\cdot) = p_{\text{pointer}}(k^l|\cdot) + p_{\text{pointer}}(k^r|\cdot), \tag{11}$$

where $p_{\text{pointer}}(k^l|\cdot)$ and $p_{\text{pointer}}(k^r|\cdot)$ are calculated simultaneously from the same softmax function in Equation (9). $k^l, k^r (k^l \neq k^r)$ represent the keys corresponding to the same relative position.

Here, we study two types of $q(\boldsymbol{\pi}|\boldsymbol{x}, \boldsymbol{y})$:

Pre-defined Order If we already possess some prior knowledge about the sequence, e.g., the L2R order is proven to be a strong baseline in many scenarios, we assume a Dirac-delta distribution $q(\boldsymbol{\pi}|\boldsymbol{x}, \boldsymbol{y}) = \delta(\boldsymbol{\pi} = \boldsymbol{\pi}^*(\boldsymbol{x}, \boldsymbol{y}))$, where $\boldsymbol{\pi}^*(\boldsymbol{x}, \boldsymbol{y}))$ is a predefined order. In this work, we study a set of pre-defined orders, which can be found in Table 1, for evaluating their effect on generation.

Searched Adaptive Order (SAO) We choose the approximate posterior q as the point estimation that maximizes $\log p_\theta(\boldsymbol{y}_{\boldsymbol{\pi}}|\boldsymbol{x})$, which can also be seen as the maximum-a-posteriori (MAP) estimation on the latent order $\boldsymbol{\pi}$. In practice, we approximate these generation orders π through *beam-search* (Pal et al., 2006). Unlike the original beam-search for autoregressive decoding that searches in the sequence space to find the sequence maximizing the probability shown in Equation 1, we search in the space of all the permutations of the target sequence to find π maximising Equation 2, as all the target tokens are known in advance during training.

More specifically, we maintain B sub-sequences with the maximum probabilities using a set $\mathcal{B}$ at each step t. For every sub-sequence $y_{0:t}^{(b)} \in \mathcal{B}$, we evaluate the probabilities of every possible choice from the remaining words $y' \in \boldsymbol{y} \setminus y_{0:t}^{(b)}$ and its position $\boldsymbol{r}'$. We calculate the cumulative likelihood for each $y', \boldsymbol{r}'$, based on which we select top-B sub-sequences as the new set $\mathcal{B}$ for the next step. After obtaining the B generation orders, we optimize our objective as an average over these orders:

$$\mathcal{L}_{SAO} = \frac{1}{B} \sum_{\pi \in \mathcal{B}} \log p_\theta(\boldsymbol{y}_{\boldsymbol{\pi}}|\boldsymbol{x}) \tag{12}$$

where we assume $q(\boldsymbol{\pi}|\boldsymbol{x}, \boldsymbol{y}) = \begin{cases} 1/B & \boldsymbol{\pi} \in \mathcal{B} \\ 0 & \text{otherwise} \end{cases}$.

Beam-Search with Dropout The goal of beam-search is to approximately find the most likely generation orders, which limits learning from exploring other generation orders that may not be favourable currently but may ultimately be deemed better. Prior research (Vijayakumar et al.,

Pre-defined Order	Descriptions
Left-to-right (L2R)	Generate words from left to right. (Wu et al., 2018)
Right-to-left (R2L)	Generate words from right to left. (Wu et al., 2018)
Odd-Even (ODD)	Generate words at odd positions from left to right, then generate even positions. (Ford et al., 2018)
Balanced-tree (BLT)	Generate words with a top-down left-to-right order from a balanced binary tree. (Stern et al., 2019)
Syntax-tree (SYN)	Generate words with a top-down left-to-right order from the dependency tree. (Wang et al., 2018b)
Common-First (CF)	Generate all common words first from left to right, and then generate the others. (Ford et al., 2018)
Rare-First (RF)	Generate all rare words first from left to right, and then generate the remaining. (Ford et al., 2018)
Random (RND)	Generate words in a random order shuffled every time the example was loaded.

Table 1: Descriptions of the pre-defined orders used in this work. Major references that have explored these generation orders with different models and applications are also marked.

2016) also pointed out that the search space of the standard beam-search is restricted. We encourage exploration by injecting noise during beam-search (Cho, 2016). Particularly, we found it effective to keep the dropout on (e.g., dropout $= 0.1$).

Bootstrapping from a Pre-defined Order During preliminary experiments, sequences returned by beam-search were often degenerated by always predicting common or functional words (‘‘the’’, ‘‘,’’, etc.) as the first several tokens, leading to inferior performance. We conjecture that is due to the fact that the position prediction module learns much faster than the word prediction module, and it quickly captures spurious correlations induced by a poorly initialized model. It is essential to balance the learning progress of these modules. To do so, we bootstrap learning by pre-training the model with a pre-defined order (e.g., L2R), before training with beam-searched orders.

4.3 Decoding

As for decoding, we directly follow Algorithm 1 to sample or decode greedily from the proposed model. However, in practice beam-search is important to explore the output space for neural autoregressive models. In our implementation, we perform beam-search for InDIGO as a two-step search. Suppose the beam size B, at each step, we do beam-search for word prediction and then with the searched words, try out all possible positions and select the top-B sub-sequences. In preliminary experiments, we also tried doing beam-search for word and positions simultaneously with their joint probability. However, it did not seem helpful.

5 Experiments

We evaluate InDIGO extensively on four challenging sequence generation tasks: word order recovery, machine translation, natural language to code generation (NL2Code, Ling et al., 2016) and image captioning. We compare our model trained with the pre-defined orders and the adaptive orders obtained by beam-search. We use the same architecture for all orders including the standard L2R order.

5.1 Experimental Settings

Dataset The machine translation experiments are conducted on three language pairs for studying how the decoding order influences the translation quality of languages with diversified characteristics: WMT’16 Romanian-English (Ro-En),[2] WMT 18 English-Turkish (En-Tr),[3] and KFTT English-Japanese (En-Ja, Neubig, 2011).[4] The English part of the Ro-En dataset is used for the word order recovery task. For the NL2Code task, We use the Django dataset (Oda et al., 2015)[5] and the MS COCO (Lin et al., 2014) with the standard split (Karpathy and Fei-Fei, 2015) for the NL2Code task and image captioning, respectively. The dataset statistics are shown in Table 2.

Preprocessing We apply the Moses tokenization[6] and normalization on all the text datasets except for codes. We perform $32,000$ joint BPE (Sennrich et al., 2016) operations for the MT datasets, while using all the unique words as the vocabulary for NL2Code. For image captioning, we follow the same procedure as described by Lee et al. (2018), where we use 49 512-dimensional image feature vectors (extracted from a pretrained

[2] http://www.statmt.org/wmt16/translation-task.html

[3] http://www.statmt.org/wmt18/translation-task.html

[4] http://www.phontron.com/kftt/.

[5] https://github.com/odashi/ase15-django-dataset

Dataset	Train	Dev	Test	Length
WMT16 Ro-En	620k	2000	2000	26.48
WMT18 En-Tr	207k	3007	3000	25.81
KFTT En-Ja	405k	1166	1160	27.51
Django	16k	1000	1801	8.87
MS-COCO	567k	5000	5000	12.52

Table 2: Dataset statistics for the machine translation, code generation, and image captioning tasks. Length represents the average number of tokens for target sentences of the training set.

ResNet-18 [He et al., 2016]) as the input to the Transformer encoder. The image features are fixed during training.

Models We set $d_{\text{model}} = 512$, $d_{\text{hidden}} = 2048$, $n_{\text{heads}} = 8$, $n_{\text{layers}} = 6$, $\text{lr}_{\text{max}} = 0.0005$, warmup $= 4000$, and dropout $= 0.1$ throughout all the experiments. The source and target embedding matrices are shared except for En-Ja, as our preliminary experiments showed that keeping the embeddings not shared significantly improves the translation quality. Both the encoder and decoder use relative positions during self-attention except for the word order recovery experiments (where the position embedding is removed in the encoder, as there is no ground-truth position information in the input). We do not introduce task-specific modules such as copying mechanism (Gu et al., 2016).

Training When training with the pre-defined orders, we reorder words of each training sequence in advance accordingly, which provides supervision of the ground-truth positions that each word should be inserted. We test the pre-defined orders listed in Table 1. The SYN orders were generated according to the dependency parse obtained by a dependency parse parser from Spacy (Honnibal and Montani, 2017) following a parent-to-children left-to-right order. The CF & RF orders are obtained based on vocabulary cut-off so that the number of common words and the number of rare words are approximately the same (Ford et al., 2018). We also consider on-the-fly sampling a random order for each sentence as the baseline (RND). When using L2R as the pre-defined order, Transformer-InDIGO is almost equivalent to the vanilla Transformer, as the position prediction simply learns to predict the next position as the left of the $\langle s \rangle$ symbol. The only difference is that it enhances the vanilla Transformer with a small number of additional parameters for the position prediction.

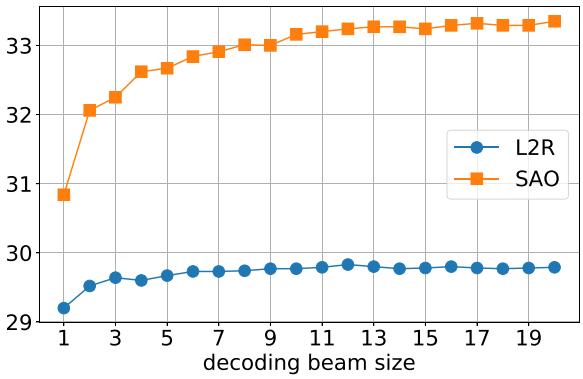

Figure 3: The BLEU scores on the test set for word order recovery with various decoding beam sizes.

We also train Transformer-InDIGO using the SAO where we set the beam size to 8. In default, models trained with SAO are bootstrapped from a slightly pre-trained (6,000 steps) model in L2R order.

Inference During the test time, we do beam-search as described in Sec. 4.3. We observe from our preliminary experiments that models trained with different orders (either pre-defined or SAO) have very different optimal beam sizes for decoding. Therefore, we perform sensitivity studies, in which the beam sizes vary from $1 \sim 20$ and pick the beam size with the highest BLEU score on the validation set for each particular model.

5.2 Results and Analysis

Word Order Recovery Word order recovery takes a bag of words as input and recovers its original word order, which is challenging as the search space is factorial. We do not restrict the vocabulary of the input words. We compare our model trained with the L2R order and eight SAO from beam-search for word order recovery. The BLEU scores over various beam sizes are shown in Figure 3. The model trained with SAO lead to higher BLEU scores over that trained with L2R with a gain up to 3 BLEU scores. Furthermore, increasing the beam size brings more improvements for SAO compared with L2R, suggesting that InDIGO produces more diversified predictions so that it has higher chances to recover the order.

Order	WMT16 Ro → En				WMT18 En → Tr				KFTT En → Ja			
	BLEU	Ribes	Meteor	TER	BLEU	Ribes	Meteor	TER	BLEU	Ribes	Meteor	TER
RND	20.20	79.35	41.00	63.20	03.04	55.45	19.12	90.60	17.09	70.89	35.24	70.11
L2R	31.82	83.37	52.19	50.62	14.85	69.20	33.90	**71.56**	30.87	77.72	48.57	59.92
R2L	31.62	83.18	52.09	50.20	14.38	68.87	33.33	71.91	30.44	**77.95**	47.91	61.09
ODD	30.11	83.09	50.68	50.79	13.64	68.85	32.48	72.84	28.59	77.01	46.28	60.12
BLT	24.38	81.70	45.67	55.38	08.72	65.70	27.40	77.76	21.50	73.97	40.23	64.39
SYN	29.62	82.65	50.25	52.14	–				–			
CF	30.25	83.22	50.71	50.72	12.04	67.61	31.18	74.75	28.91	77.06	46.46	61.56
RF	30.23	83.29	50.72	51.73	12.10	67.44	30.72	73.40	27.35	76.40	45.15	62.14
SAO	**32.47**	**84.10**	**53.00**	**49.02**	**15.18**	**70.06**	**34.60**	**71.56**	**31.91**	77.56	**49.66**	**59.80**

Table 3: Results of translation experiments for three language pairs in different decoding orders. Scores are reported on the test set with four widely used evaluation metrics (BLEU↑, Meteor↑, TER↓, and Ribes↑). We do not report models trained with SYN order on En-Tr and En-Ja due to the lack of reliable dependency parsers. The statistical significance analysis[6] between the outputs of SAO and L2R are conducted using BLEU score as the metric, and the p-values are ≤ 0.001 for all three language pairs.

Machine Translation As shown in Table 3, we compare our model trained with pre-defined orders and the SAO with varying setups. We use four evaluation metrics including BLEU (Papineni et al., 2002), Ribes (Isozaki et al., 2010), Meteor (Banerjee and Lavie, 2005), and TER (Snover et al., 2006) to avoid using a single metric that might be in favor of a particular generation order. Most of the pre-defined orders (except for the random order and the balanced tree [BLT] order) perform reasonably well with InDIGO on the three language pairs. The best score with a predefined word ordering is reached by the L2R order among the pre-defined orders except for En-Ja, where the R2L order works slightly better according to Ribes. This indicates that in machine translation, the monotonic orders are reasonable and reflect the languages. ODD, CF, and RF show similar performance, which is below the L2R and R2L orders by around 2 BLEU scores. The tree-based orders, such as the SYN and BLT orders, do not perform well, indicating that predicting words following a syntactic path is not preferable. On the other hand, Table 3 shows that the model with SAO achieves competitive and even statistically significant improvements over the L2R order. The improvements are larger for Turkish and Japanese, indicating that a flexible generation order may improve the translation quality for languages with different syntactic structures from English.

[6] https://github.com/moses-smt/mosesdecoder

Model	Django		MS-COCO	
	BLEU	Accuracy	BLEU	CIDEr-D
L2R	36.74	13.6%	22.12	68.88
SAO	**42.33**	**16.3%**	**22.58**	**69.42**

Table 4: Results on the official test sets for both code generation and image captioning tasks.

Code Generation The goal of this task is to generate Python code based on a natural language description, which can be achieved by using a standard sequence-to-sequence generation framework such as the proposed Transformer-InDIGO. As shown in Table 4, SAO works significantly better than the L2R order in terms of both BLEU and accuracy. This shows that flexible generation orders are more preferable in code generation.

Image Captioning For the captioning task, one caption is generated per image and is compared against five human-created captions during testing. As show in Table 4, we observe that SAO obtains higher BLEU and CIDEr-D (Vedantam et al., 2015) compared to the L2R order, and it implies that better captions are generated with different orders.

5.3 Ablation Study

Model Variants Table 5 shows the results of the ablation studies using the machine translation task. SAO without bootstrapping nor beam-search degenerates by approximate 1 BLEU score on Ro-En, demonstrating the effectiveness of these

Model Variants	dev	test
Baseline L2R	32.53	31.82
SAO default	**33.60**	**32.47**
no bootstrap	32.86	31.88
no bootstrap, no noise	32.64	31.72
bootstrap from R2L order	33.12	32.02
bootstrap from SYN order	33.09	31.93
Stern et al. (2019) - Uniform	29.99	28.52
Stern et al. (2019) - Binary	32.27	30.66

Table 5: Ablation study for machine translation on WMT16 Ro-En. Results of Stern et al. (2019) are based on greedy decoding with the EOS penalty.

Model	Training (b/s)	Decoding (ms/s)
L2R	4.21	12.3
SAO ($b = 1$)	1.12	12.5
SAO ($b = 8$)	0.58	12.8

Table 6: Comparison of the L2R order with SAO on running time, where b/s is batches per second and ms/s is ms per sentence. All experiments are conducted on 8 Nvidia V100 GPUs with 2000 tokens per GPU. We also compare beam sizes of 1 and 8 for SAO to search the best orders during training. We report the decoding speed of all three models based on greedy decoding.

two methods. We also test SAO by bootstrapping from a model trained with a R2L order as well as a SYN order, which obtains slightly worse yet comparable results compared to bootstrapping from L2R. This suggests that the SAO algorithm is quite robust with different bootstrapping methods, and L2R bootstrapping performs the best. In addition, we re-implement a recent work (Stern et al., 2019) that adopts a similar idea of generating sequences through insertion operations for machine translation. We use the best settings of their algorithm, i.e., training with binary-tree/ uniform slot-losses and slot-termination, while removing the knowledge distillation for a fair comparison with ours. Our model obtains better performance compared with Stern et al. (2019) on WMT16 Ro-En.

Running Time As shown in Table 6, InDIGO decodes sentences as efficient as the standard L2R autoregressive models. However, it is slower in terms of training time using SAO as the supervision, as additional efforts are needed to search the generation orders, and it is difficult to parallelize the SAO. SAO with beam sizes 1 and 8 are 3.8 and 7.2 times slower than L2R, respectively. Note that enlarging the beam size during training won't affect the decoding time as searching the best orders only happen in the training time. We will investigate off-line searching methods to speed up SAO training and make InDIGO more scalable in the future.

5.4 Visualization

Relative-Position Matrix In Figure 4, we show an instantiated example produced by InDIGO, which is randomly sampled from the validation set of the KFTT En-Ja dataset. The relative-position matrices (R^t) and their corresponding absolute positions (z^t) are shown at each step. We argue that relative-position matrices are flexible to encode position information, and its append-only property enables InDIGO to reuse previous hidden states.

Case Study We demonstrate how InDIGO works by uniformly sampling examples from the validation sets for machine translation (Ro-En), image captioning, and code generation. As shown in Figure 5, the proposed model generates sequences in different orders based on the order used for learning (either pre-defined or SAO). For instance, the model generates tokens approximately following the dependency parse when we used the SYN order for the machine translation task. On the other hand, the model trained using the RF order learns to first produce verbs and nouns first, before filling up the sequence with remaining functional words.

We observe several key characteristics about the inferred orders of SAO by analyzing the model's output for each task: (1) For machine translation, the generation order of an output sequence does not deviate too much from L2R. Instead, the sequences are shuffled with chunks, and words within each chunk are generated in a L2R order; (2) In the examples of image captioning and code generation, the model tends to generate most of the words in the L2R order and insert a few words afterward in certain locations. Moreover, we provide more examples in the appendix.

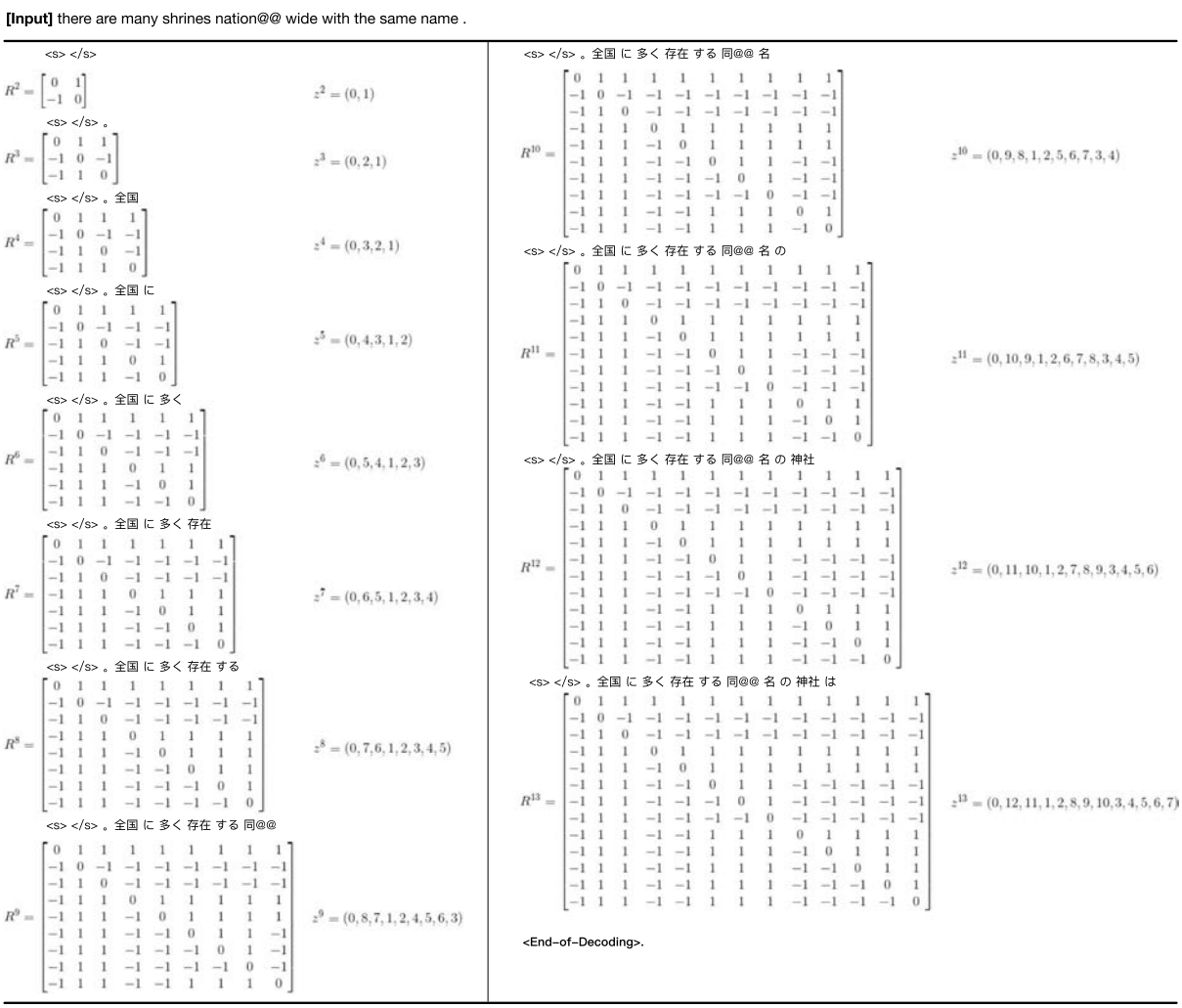

Figure 4: An instantiated concrete example of the decoding process using InDIGO sampled from the En-Ja translation datset. The final output is reordered based on the predicted relative-position matrix.

6 Related Work

Decoding for Neural Models Neural autoregressive modelling has become one of the most successful approaches for generating sequences (Sutskever et al., 2011; Mikolov, 2012), which has been widely used in a range of applications, such as machine translation (Sutskever et al., 2014), dialogue response generation (Vinyals and Le, 2015), image captioning (Karpathy and Fei-Fei, 2015), and speech recognition (Chorowski et al., 2015). Another stream of work focuses on generating a sequence of tokens in a non-autoregressive fashion (Gu et al., 2018; Lee et al., 2018; van den Oord et al., 2018), in which the discrete tokens are generated in parallel. Semi-autoregressive modelling (Stern et al., 2018; Wang et al., 2018a) is a mixture of the two approaches, while largely adhering to left-to-right generation. Our method is different from these approaches as we support flexible generation orders, while decoding autoregressively.

Non-L2R Orders Previous studies on generation order of sequences mostly resort to a fixed set of generation orders. Wu et al. (2018) empirically show that R2L generation outperforms its L2R counterpart in a few tasks. Ford et al. (2018) devise a two-pass approach that produces partially-filled sentence ‘‘templates’’ and then fills in missing tokens. Zhu et al. (2019) also propose to generate tokens by first predicting a text template and infill the sentence afterwards while in a more general way. Mehri and Sigal (2018) propose a

WMT16 Ro-En

[source] acestia dezvaluie secre@@ tele pant@@ om@@ im@@ ei lor la bbc break@@ fast .
[target] they reveal their pan@@ to secre@@ ts to bbc break@@ fast .

– InDIGO learned from Pre-defined L2R Order

they	
they reveal	they reveal the secre@@ ts of their sho@@ es
they reveal the	they reveal the secre@@ ts of their sho@@ es to
they reveal the secre@@	they reveal the secre@@ ts of their sho@@ es to the
they reveal the secre@@ ts	they reveal the secre@@ ts of their sho@@ es to the bbc
they reveal the secre@@ ts of	they reveal the secre@@ ts of their sho@@ es to the bbc break@@
they reveal the secre@@ ts of their	they reveal the secre@@ ts of their sho@@ es to the bbc break@@ fast
they reveal the secre@@ ts of their sho@@	they reveal the secre@@ ts of their sho@@ es to the bbc break@@ fast .

– InDIGO learned from Pre-defined R2L Order

.	
fast .	ts at the bbc break@@ fast .
break@@ fast .	secre@@ ts at the bbc break@@ fast .
bbc break@@ fast .	their secre@@ ts at the bbc break@@ fast .
the bbc break@@ fast .	reveal their secre@@ ts at the bbc break@@ fast .
at the bbc break@@ fast .	they reveal their secre@@ ts at the bbc break@@ fast .

– InDIGO learned from Pre-defined SYN Order

reveal	they reveal the secre@@ ts of sho@@ .
they reveal	they reveal the secre@@ ts of sho@@ e .
they reveal secre@@	they reveal the secre@@ ts of their sho@@ e .
they reveal secre@@ ts	they reveal the secre@@ ts of their sho@@ e in .
they reveal secre@@ ts .	they reveal the secre@@ ts of their sho@@ e in break@@ .
they reveal the secre@@ ts .	they reveal the secre@@ ts of their sho@@ e in break@@ fast .
they reveal the secre@@ ts of .	they reveal the secre@@ ts of their sho@@ e in bbc break@@ fast .

– InDIGO learned from Pre-defined RF Order

reveal	they reveal secrecy sho@@ es bbc break@@ fast
reveal secrecy	they reveal the secrecy sho@@ es bbc break@@ fast
reveal secrecy sho@@	they reveal the secrecy of sho@@ es bbc break@@ fast
reveal secrecy sho@@ es	they reveal the secrecy of their sho@@ es bbc break@@ fast
reveal secrecy sho@@ es bbc	they reveal the secrecy of their sho@@ es at bbc break@@ fast
reveal secrecy sho@@ es bbc break@@	they reveal the secrecy of their sho@@ es at the bbc break@@ fast
reveal secrecy sho@@ es bbc break@@ fast	they reveal the secrecy of their sho@@ es at the bbc break@@ fast .

– InDIGO learned from Searched Adaptive Order (SAO)

.	
the .	they the secre@@ ts the bbc break@@ fast .
the bbc .	they the secre@@ ts of the bbc break@@ fast .
the bbc break@@ .	they the secre@@ ts of their the bbc break@@ fast .
the bbc break@@ fast .	they the secre@@ ts of their sho@@ the bbc break@@ fast .
they the bbc break@@ fast .	they the secre@@ ts of their sho@@ e the bbc break@@ fast .
they the the bbc break@@ fast .	they the secre@@ ts of their sho@@ e at the bbc break@@ fast .
they the secre@@ the bbc break@@ fast .	they reveal the secre@@ ts of their sho@@ e at the bbc break@@ fast .

MS-COCO

[caption-1] a tall woman is standing in a small kitchen
[caption-2] a girl is standing in a kitchen with a mug in her hands .
[caption-3] woman in knitted jump pants and yellow slee@@ v@@ eless top , in kitchen scene with matching yellow tone area .
[caption-4] a woman standing in a kitchen near a refrigerator and a stove
[caption-5] a woman with pi@@ g@@ tails is standing in a kitchen .

– InDIGO learned from Pre-defined L2R Order

a	a man standing in a kitchen
a man	a man standing in a kitchen holding
a man standing	a man standing in a kitchen holding a
a man standing in	a man standing in a kitchen holding a banana
a man standing in a	a man standing in a kitchen holding a banana .

– InDIGO learned from Searched Adaptive Order (SAO)

a	a woman standing in front of a
a woman	a woman standing in front of a refrigerator
a woman standing	a woman standing in front of a refrigerator .
a woman standing in	a woman in standing in front of a refrigerator .
a woman standing in front	a woman in a standing in front of a refrigerator .
a woman standing in front of	a woman in a kitchen standing in front of a refrigerator .

Django

[source instruction] if tok starts with _STR:0_ ,
[target Python code] if tok . startswith (' _STR:0_ ') :

– InDIGO learned from Searched Adaptive Order (SAO)

if	if parser . ('	if parser . (' _STR:0_ ') :
if parser	if parser . (' '	if parser . startswith (' _STR:0_ ') :
if parser .	if parser . (' ')	
if parser . (	if parser . (' ') :	

Figure 5: Examples randomly sampled from three tasks that are instructed to decode using InDIGO with various learned generation order. Words in red and underlined are the inserted token at each step. For visual convenience, we reordered all the partial sequences to its correct positions at each decoding step.

middle-out decoder that firstly predicts a middle-word and simultaneously expands the sequence in both directions afterwards. Previous studies also focused on decoding in a bidirectional fashion such as (Sun et al., 2017; Zhou et al., 2019a,b). Another line of work models sequence generation based on syntax structures (Yamada and Knight, 2001; Charniak et al., 2003; Chiang, 2005; Emami and Jelinek, 2005; Zhang et al., 2016; Dyer et al., 2016; Aharoni and Goldberg, 2017; Wang et al., 2018b; Eriguchi et al., 2017). In contrast, Transformer-InDIGO supports fully flexible generation orders during decoding.

There are two concurrent papers (Welleck et al., 2019; Stern et al., 2019) that study sequence generation in a non-L2R order. Welleck et al. (2019) propose a tree-like generation algorithm. Unlike this work, the tree-based generation order only produces a subset of all possible generation orders compared to our insertion-based models. Further, Welleck et al. (2019) find L2R is superior to their learned orders on machine translation tasks, while transformer-InDIGO with searched adaptive orders achieves better performance. Stern et al. (2019) propose a very similar idea of using insertion operations in Transformer for machine translation. The major difference is that they directly use absolute positions, whereas ours utilizes relative positions. As a result, their model needs to re-encode the partial sequence at every step, which is computationally more expensive. In contrast, our approach does not necessitate re-encoding the entire sentence during generation. In addition, knowledge distillation was necessary to achieve good performance in Stern et al. (2019), while our model is able to match the performance of L2R even without bootstrapping.

7 Conclusion

We have presented a novel approach—InDIGO—that supports flexible sequence generation. Our model was trained with either pre-defined orders or searched adaptive orders. In contrast to conventional neural autoregressive models that often generate from left to right, our model can flexibly generate a sequence following an arbitrary order. Experiments show that our method achieved competitive or even better performance compared with the conventional left-to-right generation on four tasks, including machine translation, word order recovery, code generation and image captioning.

For future work, it is worth exploring a trainable inference model to directly predict the permutation (Mena et al., 2018) instead of beam-search. Also, the proposed InDIGO could be extended for post-editing tasks such as automatic post-editing for machine translation and grammatical error correction by introducing additional operations such as ‘‘deletion’’ and ‘‘substitution’’.

Acknowledgments

We specially thank our action editor Alexandra Birch and all the reviewers for their great efforts to review the draft. We also would like to thank Douwe Kiela, Marc'Aurelio Ranzato, Jake Zhao, and our colleagues at FAIR for the valuable feedback, discussions, and technical assistance. This work was partly supported by Samsung Advanced Institute of Technology (Next Generation Deep Learning: From Pattern Recognition to AI) and Samsung Electronics (Improving Deep Learning Using Latent Structure). KC thanks for the support of eBay and Nvidia.

References

Roee Aharoni and Yoav Goldberg. 2017. Towards string-to-tree neural machine translation. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, volume 2, pages 132–140.

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2015. 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*.

Satanjeev Banerjee and Alon Lavie. 2005. 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*, pages 65–72.

Eugene Charniak, Kevin Knight, and Kenji Yamada. 2003. Syntax-based language models for statistical machine translation. In *MT Summit IX. Intl. Assoc. for Machine Translation*. Citeseer.

David Chiang. 2005. A hierarchical phrase-based model for statistical machine translation. In *Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics*, pages 263–270. Association for Computational Linguistics.

Kyunghyun Cho. 2016. Noisy parallel approximate decoding for conditional recurrent language model. *arXiv preprint arXiv:1605.03835*.

Jan K. Chorowski, Dzmitry Bahdanau, Dmitriy Serdyuk, Kyunghyun Cho, and Yoshua Bengio. 2015. Attention-based models for speech recognition. In *NIPS*, pages 577–585.

Chris Dyer, Adhiguna Kuncoro, Miguel Ballesteros, and Noah A. Smith. 2016. Recurrent neural network grammars. In *NAACL HLT 2016, The 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, San Diego California, USA, June 12-17, 2016*, pages 199–209.

Ahmad Emami and Frederick Jelinek. 2005. A neural syntactic language model. *Machine Learning*, 60(1-3):195–227.

Akiko Eriguchi, Yoshimasa Tsuruoka, and Kyunghyun Cho. 2017. Learning to parse and translate improves neural machine translation. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics, ACL 2017, Vancouver, Canada, July 30 - August 4, Volume 2: Short Papers*, pages 72–78.

Nicolas Ford, Daniel Duckworth, Mohammad Norouzi, and George E. Dahl. 2018. The importance of generation order in language modeling. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing, Brussels, Belgium, October 31 - November 4, 2018*, pages 2942–2946.

Jiatao Gu, James Bradbury, Caiming Xiong, Victor O. K. Li, and Richard Socher. 2018. Non-autoregressive neural machine translation. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, Canada, April 30-May 3, 2018, Conference Track Proceedings*.

Jiatao Gu, Zhengdong Lu, Hang Li, and Victor O. K. Li. 2016. Incorporating copying mechanism in sequence-to-sequence learning. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics, ACL 2016, August 7-12, 2016, Berlin, Germany, Volume 1: Long Papers*.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 770–778.

Matthew Honnibal and Ines Montani. 2017. spaCy 2: Natural language understanding with Bloom embeddings, convolutional neural networks and incremental parsing. To appear.

Hideki Isozaki, Tsutomu Hirao, Kevin Duh, Katsuhito Sudoh, and Hajime Tsukada. 2010. 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.

Andrej Karpathy and Li Fei-Fei. 2015. Deep visual-semantic alignments for generating image descriptions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3128–3137.

Jason Lee, Elman Mansimov, and Kyunghyun Cho. 2018. Deterministic non-autoregressive neural sequence modeling by iterative refinement. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing, Brussels, Belgium, October 31 - November 4, 2018*, pages 1173–1182.

Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C. Lawrence Zitnick. 2014. Microsoft COCO: Common objects in context. In *European Conference on Computer Vision*, pages 740–755. Springer.

Wang Ling, Phil Blunsom, Edward Grefenstette, Karl Moritz Hermann, Tomás Kociský, Fumin Wang, and Andrew W. Senior. 2016. Latent predictor networks for code generation. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics,*

ACL 2016, August 7-12, 2016, Berlin, Germany, Volume 1: Long Papers.

Shikib Mehri and Leonid Sigal. 2018, Middle-out decoding, S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems 31*, pages 5523–5534. Curran Associates, Inc.

Gonzalo Mena, David Belanger, Scott Linderman, and Jasper Snoek. 2018. Learning latent permutations with Gumbel-Sinkhorn networks. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, Canada, April 30-May 3, 2018, Conference Track Proceedings*.

Tomáš Mikolov. 2012. Statistical language models based on neural networks. Presentation at Google, Mountain View, 2 April.

Graham Neubig. 2011. The Kyoto free translation task. http://www.phontron.com/kftt.

Yusuke Oda, Hiroyuki Fudaba, Graham Neubig, Hideaki Hata, Sakriani Sakti, Tomoki Toda, and Satoshi Nakamura. 2015. Learning to generate pseudo-code from source code using statistical machine translation. In *Proceedings of the 2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, ASE '15, pages 574–584, Lincoln, Nebraska, USA. IEEE Computer Society.

Aaron van den Oord, Yazhe Li, Igor Babuschkin, Karen Simonyan, Oriol Vinyals, Koray Kavukcuoglu, George van den Driessche, Edward Lockhart, Luis Cobo, Florian Stimberg, Norman Casagrande, Dominik Grewe, Seb Noury, Sander Dieleman, Erich Elsen, Nal Kalchbrenner, Heiga Zen, Alex Graves, Helen King, Tom Walters, Dan Belov, and Demis Hassabis. 2018. Parallel WaveNet: Fast high-fidelity speech synthesis. In *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 3918–3926, Stockholm, Sweden. PMLR.

Chris Pal, Charles Sutton, and Andrew McCallum. 2006. Sparse forward-backward using minimum divergence beams for fast training of conditional random fields. In *Acoustics, Speech and Signal Processing, 2006. ICASSP 2006 Proceedings. 2006 IEEE International Conference on*, volume 5, pages V–V. IEEE.

Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. BLEU: A method for automatic evaluation of machine translation. In *Proceedings of the 40th annual meeting on association for computational linguistics*, pages 311–318. Association for Computational Linguistics.

Alexander M. Rush, Sumit Chopra, and Jason Weston. 2015. A neural attention model for abstractive sentence summarization. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 379–389, Lisbon, Portugal. Association for Computational Linguistics.

Rico Sennrich, Barry Haddow, and Alexandra Birch. 2016. Neural machine translation of rare words with subword units. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1715–1725, Berlin, Germany. Association for Computational Linguistics.

Peter Shaw, Jakob Uszkoreit, and Ashish Vaswani. 2018. Self-attention with relative position representations. 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 464–468, New Orleans, Louisiana. Association for Computational Linguistics.

Matthew Snover, Bonnie Dorr, Richard Schwartz, Linnea Micciulla, and John Makhoul. 2006. A study of translation edit rate with targeted human annotation. In *Proceedings of Association for Machine Translation in the Americas*, pages 223–231.

Mitchell Stern, William Chan, Jamie Kiros, and Jakob Uszkoreit. 2019. Insertion transformer: Flexible sequence generation via insertion operations. *arXiv preprint arXiv:1902.03249*.

Mitchell Stern, Noam Shazeer, and Jakob Uszkoreit. 2018. Blockwise parallel decoding for deep autoregressive models. In *Advances in Neural Information Processing Systems*, pages 10107–10116.

Qing Sun, Stefan Lee, and Dhruv Batra. 2017. Bidirectional beam search: Forward-backward inference in neural sequence models for fill-in-the-blank image captioning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 6961–6969.

Ilya Sutskever, James Martens, and Geoffrey E Hinton. 2011. Generating text with recurrent neural networks. In *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, pages 1017–1024.

Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. 2014. Sequence to sequence learning with neural networks. In *Advances in Neural Information Processing Systems 27: Annual Conference on Neural Information Processing Systems 2014, December 8-13 2014, Montreal, Quebec, Canada*, pages 3104–3112.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Proceedings of the Annual Conference on Neural Information Processing Systems (NIPS)*.

Ramakrishna Vedantam, C Lawrence Zitnick, and Devi Parikh. 2015. Cider: Consensus-based image description evaluation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4566–4575.

Ashwin K Vijayakumar, Michael Cogswell, Ramprasath R Selvaraju, Qing Sun, Stefan Lee, David Crandall, and Dhruv Batra. 2016. Diverse beam search: Decoding diverse solutions from neural sequence models. *arXiv preprint arXiv:1610.02424*.

Oriol Vinyals, Samy Bengio, and Manjunath Kudlur. 2016. Order matters: Sequence to sequence for sets. In *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*.

Oriol Vinyals, Meire Fortunato, and Navdeep Jaitly. 2015. Pointer networks. In *Advances in Neural Information Processing Systems*, pages 2692–2700.

Oriol Vinyals and Quoc Le. 2015. A neural conversational model. *arXiv preprint arXiv:1506.05869*.

Chunqi Wang, Ji Zhang, and Haiqing Chen. 2018a. Semi-autoregressive neural machine translation. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 479–488, Brussels, Belgium. Association for Computational Linguistics.

Xinyi Wang, Hieu Pham, Pengcheng Yin, and Graham Neubig. 2018b. A tree-based decoder for neural machine translation. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 4772–4777, Brussels, Belgium. Association for Computational Linguistics.

Sean Welleck, Kianté Brantley, Hal Daumé III, and Kyunghyun Cho. 2019. Non-monotonic sequential text generation. *arXiv preprint arXiv:1902.02192*.

Lijun Wu, Xu Tan, Di He, Fei Tian, Tao Qin, Jianhuang Lai, and Tie-Yan Liu. 2018. Beyond error propagation in neural machine translation: Characteristics of language also matter. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 3602–3611, Brussels, Belgium. Association for Computational Linguistics.

Kenji Yamada and Kevin Knight. 2001. A syntax-based statistical translation model. In *Proceedings of the 39th Annual Meeting of the Association for Computational Linguistics*.

Pengcheng Yin and Graham Neubig. 2017. A syntactic neural model for general-purpose code generation. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 440–450, Vancouver, Canada. Association for Computational Linguistics.

Xingxing Zhang, Liang Lu, and Mirella Lapata. 2016. Top-down tree long short-term memory networks. In *Proceedings of the 2016*

Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, pages 310–320, San Diego, California. Association for Compu tational Linguistics.

Long Zhou, Jiajun Zhang, and Chengqing Zong. 2019a. Synchronous bidirectional neural machine translation. *Transactions of the Association for Computational Linguistics*, 7:91–105.

Long Zhou, Jiajun Zhang, Chengqing Zong, and Heng Yu. 2019b. Sequence generation: From both sides to the middle. *IJCAI*.

Wanrong Zhu, Zhiting Hu, and Eric Xing. 2019. Text Infilling. *arXiv*, *arXiv:1901.00158*.