Train Once, and Decode As You Like

Chao Tian[1] **Yifei Wang**[2] **Hao Cheng**[1] **Yijiang Lian**[3] **Zhihua Zhang**[2]
[1] Academy for Advanced Interdisciplinary Studies, Peking University
[2] School of Mathematical Sciences, Peking University
[3] Baidu Inc.
{tianchao, yifei_wang, hao.cheng}@pku.edu.cn
lianyijiang@baidu.com
zhzhang@math.pku.edu.cn

Abstract

In this paper we propose a unified approach for supporting different generation manners of machine translation, including autoregressive, semi-autoregressive, and refinement-based non-autoregressive models. Our approach works by repeatedly selecting positions and generating tokens at these selected positions. After being trained once, our approach achieves better or competitive translation performance compared with some strong task-specific baseline models in all the settings. This generalization ability benefits mainly from the new training objective that we propose. We validate our approach on the WMT'14 English-German and IWSLT'14 German-English translation tasks. The experimental results are encouraging.

1 Introduction

Neural sequence models (Sutskever et al., 2014; Bahdanau et al., 2014; Vaswani et al., 2017) have been successfully applied to machine translation. Previous work in this area is mostly based on the encoder-decoder framework, where an encoder first makes a source sentence into hidden states and then a decoder generates a corresponding target sentence from these hidden states. The generation process usually relies on the chain-rule factorization and is performed in an autoregressive manner, i.e., tokens by tokens from left to right. Although this manner brings some advantages in training and inference, high latency in inference is a grave disadvantage due to a linear number of generation steps (Gu et al., 2017).

In order to reduce the latency during inference, Gu et al. (2017) proposed the Non-Autoregressive Transformer (NAT) that can generate an entire translation in parallel given a source sentence. Lee et al. (2018) modified the initial non-autoregressive translation repeatedly through a denoising autoencoder. Although these non-autoregressive models can generate a corresponding translation at a constant number of generation steps, they are obviously worse than their autoregressive counterparts. Wang et al. (2018) devised the Semi-Autoregressive Transformer (SAT) that keeps the autoregressive property in global but relieves it in local to generate multiple successive tokens in parallel. By generating K tokens in parallel each time, SAT can reduce the number of generation steps to $1/K$ of the original quantity. A proper K can achieve a good trade-off between speed and latency. In addition to parallel generation, some previous work (Welleck et al., 2019; Stern et al., 2019; Gu et al., 2019) pays attention to non-monotonic sequence generation, e.g., a binary tree traversal. These models are mainly based on the insertion operation.

We can abstract the aforementioned different generation manners of machine translation into a more general process. One first determines a stopping criterion, then repeatedly selects positions and generates tokens at these selected positions until the stopping criterion is met. Different position selection strategies can simulate different generation manners. For example, if a model always predicts the token at the next position and stops translation when a specific terminator (EOS) is generated or the pre-specified length is met, this can be considered as an autoregressive model. Based on this idea, we propose a unified framework of different generation manners of machine translation, taking the models mentioned above as special cases. In other words, our framework supports different popular generation manners such as autoregressive, semi-autoregressive, non-autoregressive based on iterative refinement and non-monotonic

This work is licensed under a Creative Commons Attribution 4.0 International Licence. Licence details: http://creativecommons.org/licenses/by/4.0/.

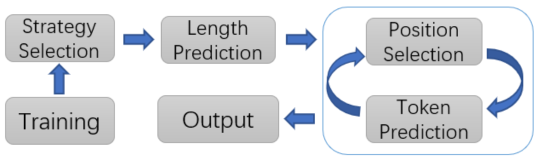

Figure 1: Flowchart of our approach. Strategy selection is to choose a generation manner, i.e., the number and position of tokens generated each time. Position selection and token prediction are performed repeatedly in turn until the entire sentence is generated.

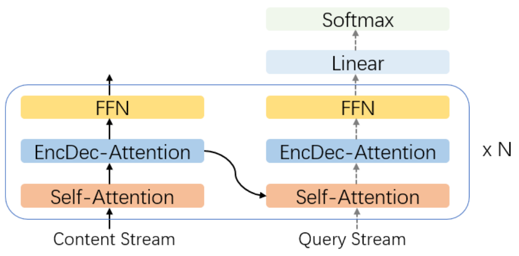

Figure 2: Illustration of the decoder. Two lines (a black solid line and a gray dashed line) represent two streams of attention and blocks of the same color share parameters in the same layer.

generation with one set of well-trained parameters. The benefit is obvious that deploying a flexible unified model is more space-efficient than deploying multiple task-specific models, assuming that users have different needs for the trade-off between translation speed and performance in different situations.

To implement this framework, our approach should have the ability to generate corresponding tokens after pre-assigned or adaptive position selection. A vanilla neural sequence model, which always predicts the next token given a source sentence and all previous generated tokens, cannot complete this task. We organically incorporate the two-stream self-attention mechanism (Yang et al., 2019) for target-aware representations into traditional machine translation models to solve this problem. In addition to a well-designed model structure, a matching training objective is also indispensable. Inspired by the permutation language modeling (Yang et al., 2019), we maximize the expectation of log conditional likelihood of all possible factorization orders of a pair of source and target sentences, rather than a traditional left-to-right objective. We experimentally validate that the new training objective significantly improves the robustness and performance of models.

We evaluate our approach on the WMT’14 English-German and IWSLT’14 German-English translation tasks. Compared with some strong task-specific baseline models, our approach achieves better or comparable results in all generation manners with shared parameters. We also investigate the impact of different generation orders on translation performance. It is surprising to find that the left-to-right order achieves the best results among seven generation orders when generating tokens one by one, although our approach has no special preference for the generation order during training. This shows that the left-to-right order is still a simple but powerful way of language modeling, which may explain why most (conditional) language models are based on autoregressive decomposition.

To this end, we summarize our contributions as follows:

- We propose a unified approach that supports different popular generation manners of machine translation;
- We experimentally validate that conditional permutation language modeling objective can significantly improve the generalization of the model for different generation manners;
- We further support that the left-to-right order is still a pretty good choice for conditional language generation;
- Our approach achieves better or comparable results with shared parameters compared with some task-specific baseline models in all generation manners on the WMT’14 English-German and IWSLT’14 German-English translation tasks, respectively.

2 Related work

Neural sequence models based on autoregressive decomposition (Sutskever et al., 2014; Bahdanau et al., 2014; Vaswani et al., 2017) have become the benchmark for machine translation. Recently, some work (Gu et al., 2017; Lee et al., 2018; Wang et al., 2018; Stern et al., 2019) has found that it is unnecessary to generate tokens in an autoregressive manner. In other words, the generation process can be parallel or non-monotonic. However, the model structure and training strategy of these methods are task-specific, and cannot flexibly support different generation requirements for a better trade-off between speed and translation performance.

Our model is inspired by XLNet (Yang et al., 2019), a pre-trained language model. It achieved the state-of-the-art results on 18 different language understanding tasks, partly due to the novel permutation language modeling. We find that the permutation language modeling also has special benefits for conditional language models, which makes the generation process more robust to the order and manner.

We must also mention the work of Mansimov et al. (2019). They similarly defined a generalized framework of sequence generation, but there are some differences between these two frameworks. The main difference is that our model is based on the permutation language modeling (Yang et al., 2019) and selects the position by pre-specifying or adaptively according to the probability, while their model is based on the masked language modeling (Devlin et al., 2018; Lample and Conneau, 2019) and selects the position by more complex adaptive Gibbs sampling. Apart from this, they used a pre-trained model trained on a monolingual dataset and finetuned it on the translation task while we only use the parallel corpus and the training process does not require two stages.

3 Methodology

Given a pair of source sentence $X = \{x_1, x_2, \cdots, x_{T'}\}$ and corresponding target sentence $Y = \{y_1, y_2, \cdots, y_T\}$, vanilla neural sequence models, such as the Transformer (Vaswani et al., 2017), factorize the conditional probability in an autoregressive manner,

$$P(Y|X) = \prod_{t=1}^{T} p(y_t|X, \boldsymbol{y}_{<t}). \tag{1}$$

Models trained with this objective only learn to predict the next token given all previous generated tokens and are not able to predict tokens at any specified position as we expect. In order to equip our model with this ability, a new training objective, i.e., conditional probabilities for different factorization orders of target sentences is necessary. Borrowing ideas from XLNet (Yang et al., 2019), we propose the conditional permutation language modeling objective. Specifically, define $\boldsymbol{z}$ as a possible permutation of a length-T index sequence $\{1, 2, \cdots, T\}$. We use z_t and $\boldsymbol{z}_{<t}$ to represent the t-th element and the first $t-1$ elements of $\boldsymbol{z}$, respectively. We maximize the expectation of log conditional likelihood of all possible factorization orders of a pair of source and target sentences as follows [1]:

$$P(Y|X) = \mathbb{E}_{\boldsymbol{z} \in \mathcal{Z}_T} \Big[\prod_{t=1}^{T} p(y_{z_t}|X, \boldsymbol{y}_{\boldsymbol{z}_{<t}}) \Big], \tag{2}$$

where $\mathcal{Z}_T$ is the set of all possible permutations. Different from using an autoregressive factorization, a model trained with this new objective is expected to be able to predict tokens at any specified position given tokens generated at other positions, possibly to the left or the right of the current position. The model has access to bidirectional context (Yang et al., 2019) instead of always needing to know all the tokens on the left, which greatly improves the flexibility and robustness of the model. Different from models based on masked language model that only optimize the loss of masked tokens, our model can calculate and optimize the loss of all tokens simultaneously. In addition, (conditional) permutation language modeling is a correct probability decomposition while masked language modeling assumes that masked tokens are

[1]As with XLNet(Yang et al., 2019), the permutation mechanism is implemented by a proper mask matrix instead of really changing the input order.

independent given other tokens, which may lose the dependency between masked tokens (Yang et al., 2019).

3.1 Architecture

Due to position-independent predictions, the conventional Transformer does not support our new objective mentioned above. To solve this problem, we integrate the two-stream self-attention mechanism (Yang et al., 2019) into the Transformer decoder. Specifically, our model has two sets of hidden states. A content representation $h(\boldsymbol{x}_{\boldsymbol{z}_{\leq t}})$, abbreviated as h_{z_t}, extracts and encodes information up to z_t-th position ($\boldsymbol{x}_{\boldsymbol{z}_{\leq t}}$) into hidden states, and a query representation $g(\boldsymbol{x}_{\boldsymbol{z}_{<t}}, z_t)$, abbreviated as g_{z_t}, only has access to the information before $\boldsymbol{z}_t$-th position ($\boldsymbol{x}_{\boldsymbol{z}_{<t}}$) and is used to predict token at position z_t. The two sets of hidden states are necessary, otherwise contradictory requirements will arise (Yang et al., 2019). Since our model is for translation tasks, proper attention to the source sentence representations, i.e., the top-level hidden states of the encoder $\boldsymbol{S} = \{s_i\}_{i=1}^{T'}$, is useful. So we intersperse the two-stream self-attention with the encoder-decoder attention as follows:

- $h_{z_t}^{(m)} \leftarrow$ Self-Attention$(Q = h_{z_t}^{(m-1)}, KV = h_{\boldsymbol{z}_{\leq t}}^{(m-1)})$;
- $h_{z_t}^{(m)} \leftarrow$ EncDec-Attention$(Q = h_{z_t}^{(m)}, KV = \boldsymbol{S})$;
- $g_{z_t}^{(m)} \leftarrow$ Self-Attention$(Q = g_{z_t}^{(m-1)}, KV = h_{\boldsymbol{z}_{<t}}^{(m)})$;
- $g_{z_t}^{(m)} \leftarrow$ EncDec-Attention$(Q = g_{z_t}^{(m)}, KV = \boldsymbol{S})$;

where Q, K and V denote the query, key and value in the attention mechanism (Vaswani et al., 2017) and superscript $m \in \{1, 2, \ldots, N\}$ indicates different layers. Functions with the same name in each layer, i.e., Self-Attention, EncDec-Attention and FFN in Figure 2, will share parameters to reduce the number of parameters and improve translation performance[2]. It is worth mentioning that our attention operation does not introduce additional parameters compared with the Transformer. The layer normalization (Ba et al., 2016), residual connection (He et al., 2016), position-wise feed-forward networks and multi-head attention mechanism (Vaswani et al., 2017) are also applied as the Transformer, although the expression is not shown here for simplicity. The query stream is initialized with a set of trainable vector, i.e., $g_i^{(0)} = \omega_i$, while the content stream is set to corresponding word embedding plus trainable position embedding, i.e., $h_i^{(0)} = e(x_i) + p_i$. Two stream attentions are updated in turns.

3.2 The Encoder & Decoder

The encoder and decoder are composed of a stack of N identical layers, respectively. The encoder remains the same as the Transformer encoder, while the decoder applies the architecture in section 3.1 instead. The top-level query stream $g_{z_t}^{(N)}$ will be used to predict the token at position z_t after a linear layer and softmax operation. The specific decoder structure is shown in Figure 2.

3.3 Target length prediction

Non-autoregressive models usually require determining the target length in advance due to the parallel generation process. Some previous work (Ma et al., 2019; Shu et al., 2019) used multilayer perceptrons as an extra length predictor. Wang et al. (2019) proposed a simple yet effective method that determine the target length by adding a constant bias term, i.e. $T = T' + \triangle T$. We adopted the first method, namely MLP. One difference is that we model the length prediction as a regression problem rather than a classification problem from previous work. Specifically, we mean-pool the encoder output $\boldsymbol{S}$ into a single vector, followed by a two-layer perceptron. The perceptron outputs a real number to predict the length difference between the source sentence and the target sentence. During training, we optimize the L_2 norm of the difference of the prediction output and the ground-truth. When inferring, the prediction output will be rounded to an integer. The detailed description and comparison of several methods are in the appendix.

[2]We find that sharing parameters can slightly improve translation performance.

3.4 Decoding methods

Given a predicted length, our model repeatedly selects positions, followed by generating tokens at selected positions. The position selection strategy can be specified in advance based on the absolute position, as mentioned in Section 4.1. To predict the token at t-th position, we only need to feed the query stream w_t, and the corresponding content stream can be set to any vector with the same dimension as w_t, which cannot affect the output due to causal mask. The model will output the probability distribution on the vocabulary at t-th position. Probabilistic adaptive selection is also supported. Specifically, the model first calculates the probability of tokens in the vocabulary at all empty positions in parallel, and then only retains the token with the highest probability at each position. The confidence of each position is defined as the probability of the corresponding token. Finally, the most certain K positions and corresponding tokens will be retained as a step of generation. We have also considered the difference between the highest probability and the second-highest probability as the confidence level for each position but found that it slightly hurts translation performance.

A more advanced decoding method is to select the predicted length $\hat{l}$ and its $2B$ neighbors, i.e $\{\hat{l} \pm i\}, i \in \{0, 1, \ldots, B\}$, then generate translations with these different lengths in parallel (Wang et al., 2019; Ghazvininejad et al., 2019). Then we select the translation with the highest average log-probability as the candidate result like Ghazvininejad et al. (2019):

$$\frac{1}{N_j} \sum_{i=1}^{N_j} \log p_i^j.$$

Our model does not need a well-trained autoregressive model for ranking (Gu et al., 2017; Ma et al., 2019), although it may lead to better performance. Since repeated translation is a common and serious problem (Wang et al., 2019), consecutive redundant tokens will be deleted automatically from the candidate result to get the final result.

4 Special Cases

Thanks to our model's flexibility and robustness, different popular generation manners can be simulated by a specific position selection strategy given a predicted target length.

4.1 Monotonic & Non-monotonic autoregressive model

Monotonic autoregressive generation is the most popular generation manner, which relies on the chain rule in an autoregressive manner (1). Any model that can calculate the conditional probability on the right side of Equation 1 is expected to be able to generate tokens autoregressively. Since our model optimizes the expectation of log conditional likelihood of all possible factorization orders (2), it is naturally expected to support a particular factorization order,

$$P(Y|X) = \prod_{t=1}^{T} p(y_{z_t}|X, \boldsymbol{y}_{\boldsymbol{z}_{<t}}), \forall \boldsymbol{z} \in \mathcal{Z}_T. \tag{3}$$

If we choose $\boldsymbol{z}$ to be an identity map, i.e., $\boldsymbol{z}_i = i, \forall i \in \{1, 2, \cdots, T\}$, it is easy to find that Equation 1 is exactly the same as Equation 3. Our model can simulate this situation easily by only predicting the token at position t in the t-th generation step. This is the same as a standard autoregressive model except for the stopping criterion. Our model predicts the target length in advance, while theirs predicts the EOS token during generation.

In addition to this, our model supports generation in completely different position-based specified orders, i.e., $\forall \boldsymbol{z} \in \mathcal{Z}_T$. In theory, we can support all $T!$ generation orders, but this is too much to verify one by one. We verified six typical generation orders in experiments, but this does not mean that our model only supports these orders.

4.2 Semi-autoregressive model

A semi-autoregressive model factorizes the conditional probability according to the group-level chain rule (Wang et al., 2018),

$$P(Y|X) = \prod_{t=1}^{[(T-1)/K]+1} p(G_t|X, G_1, \cdots, G_{t-1}), \tag{4}$$

where

$$G_1, G_2, \cdots, G_{[(T-1)/K]+1} = y_1 \cdots y_K, y_{K+1}, \cdots, y_{2K}, \cdots, y_{[(T-1)/K]\times K+1}, \cdots, y_T$$

are consecutive groups from the segmentation of Y with group size K.

Our model can select several positions and predict tokens at these selected positions in parallel to simulate this generation manner. The generation order does not have to be left-to-right like SAT (Wang et al., 2018). All position-based specified orders mentioned in Section 4.1 and the adaptive order in Section 3.4 can be implemented in parallel. We find that the adaptive order achieves better results than position-based orders.

4.3 Non-autoregressive model

Generation with adaptive order of our model can also be viewed as a refinement-based non-autoregressive manner. Specifically, we predict tokens at all empty positions, and then keep only a few positions with the highest probability, which is equivalent to masking tokens with low probability. This is similar to the idea of CMLM (Ghazvininejad et al., 2019), except that they will have an artificial symbol MASK, but ours will not. Different from the semi-autoregressive manner, the number of generation steps for non-autoregressive manner is a pre-specified constant, instead of being related to the target length.

5 Experimental settings

Data and preprocessing We evaluate our framework on the WMT'14 English-German (around 4.5M sentences pairs) and IWSLT'14 German-English (around $153k$ sentences pairs) translation tasks. The implementation is based on Pytorch (Paszke et al., 2019). We use scripts from fairseq (Ott et al., 2019) to preprocess the dataset, which first tokenize each sentence using a script from Moses (Koehn et al., 2007) and then segment each word into subword units using BPE (Sennrich et al., 2015) with a joint vocabulary. The sizes of vocabulary for WMT dataset and IWSLT dataset are $37k$ and $10k$, respectively.

Hyperparameters and optimization We follow the standard hyperparameters for transformers in the experiments. For WMT dataset, we use 6 blocks, 8 attention heads, 512 model dimensions and 2048 hidden state dimensions. For a smaller IWSLT dataset, we use 5 blocks, 2 attention heads, 278 model dimensions and 507 hidden state dimensions as Gu et al. (2017). We choose GELU (Hendrycks and Gimpel, 2016) as the activation function and apply dropout (Srivastava et al., 2014) with $p_{drop} = 0.1$ to the output of each sub-layer for regularization. Sentence pairs are batched together by approximate length and each training batch contains approximately 10000 tokens for WMT dataset (1000 tokens for IWSLT dataset). Label smoothing (Szegedy et al., 2016) is employed with value $\epsilon_{ls} = 0.1$ to improve accuracy and BLEU score (Vaswani et al., 2017). We use the Adam optimizer (Kingma and Ba, 2014) with $\beta_1 = 0.9, \beta_2 = 0.98$ and $\epsilon = 10^{-9}$. The learning rate warms up to 7×10^{-4} from 1×10^{-7} within 4000 steps for WMT dataset (1000 steps for IWSLT dataset) and then decays with the inverse square-root schedule. If the perplexity on the development set does not drop within 15 consecutive epochs, the training will be stopped early. The 5 checkpoints with the lowest perplexity on the development set will be averaged as the model parameters. The number of B is set to 2 in the non-autoregressive setting on WMT dataset for a fair comparison with CMLM, and 1 in other cases which is a good trade-off between computing consumption and translation performance.

Knowledge distillation Previous work has shown that sequence-level knowledge distillation (Kim and Rush, 2016) can improve non-autoregressive model performance (Gu et al., 2017; Lee et al., 2018). As a result, we train a transformer with basic configuration (Vaswani et al., 2017) on the raw dataset, and then use beam search with beam size equal to 4 to translate the source sentences in the raw training set

Models	Order	WMT14EN-DE	IWSLT14DE-EN
Transformer (Vaswani et al., 2017)	left-to-right	-/27.30	-/-
Transformer (our implementation)	left-to-right	26.36/27.11	32.11/33.2
Mansimov et al. (2019)	left-to-right	24.27/25.15	-/-
	Least2most	23.08/23.81	-/-
	Uniform	21.01/22.16	-/-
	Easy first	23.73/24.13	-/-
Our model	left-to-right	26.98/27.23	31.16/31.9
	right-to-left	25.77/26.33	30.79/31.77
	Uniform	25.45/26.02	29.04/30.18
	Odd first	25.93/26.45	30.68/31.37
	Even first	26.19/26.58	30.66/31.33
	Binary tree	25.68/26.19	28.76/29.88
	Adaptive	26.64/-	31.8/-

Table 1: BLEU scores of autoregressive setting. Each pair of results separated by '/' represents greedy decoding and beam search with beam size equal to 4, respectively. Our model uses an MLP-based length predictor without multiple length candidates. The result of Mansimov et al. (2019) didn't use knowledge distillation but was pre-trained on an additional 5M monolingual sentences.

to obtain the corresponding target sentence to build a new training set. The new dataset is used as the training set instead of the raw dataset.

6 Results and analysis

It is necessary to mention that different from other task-specific models, our model shares a set of well-trained parameters in all settings, namely the results of Table 1, 2 and 3. The unified model structure and training mechanism provides a flexible trade-off between performance and latency, but may slightly sacrifice the translation performance of specific tasks. Fortunately, our model achieves better performance than most baseline models.

6.1 Autoregressive setting

We first generate translation tokens by tokens in monotonic (left-to-right) and non-monotonic (other five manners) order. We also consider the adaptive manner mentioned in section 3.4 as a special generation order. The seven orders are summarized as follows:

- left-to-right: generate tokens from left to right;
- right-to-left: generate tokens from right to left;
- uniform: select an empty position with a uniform probability to fill in token each time;
- odd first: generate tokens at the position with odd index from left to right first and then tokens at the position with even index from left to right;
- even first: generate tokens at the position with even index from left to right first and then tokens at the position with odd index from left to right;
- binary tree: generate tokens according to the in-order traversal of a balanced binary tree translated from the index sequence;
- adaptive: generate tokens according to probabilistic adaptive position selection.

We observe that the results of all seven generation orders are comparable to the Transformer in Table 1. In the case of greed, the worst result also reaches approximately 97% (25.45/26.36) of the baseline on WMT dataset and 90% (28.76/32.11) on IWSLT dataset. If beam search is used, the ratio slightly drops to

	Models	WMT14EN-DE	IWSLT14DE-EN
$K = 1$	Transformer	26.36/27.11	32.11/33.2
$K = 2$	SAT	26.09/26.90	-/-
	Our model	26.48/26.70	31.39/31.58
$K = 4$	SAT	24.67/25.71	-/-
	Our model	26.18/26.41	30.19/30.49
$K = 6$	SAT	23.93/24.83	-/-
	Our model	25.94/26.17	29.16/29.56

Table 2: BLEU scores of semi-autoregressive setting. K represents the number of tokens generated in parallel each time. $K = 1$ denotes corresponding autoregressive counterpart Transformer (Vaswani et al., 2017). Each pair of results separated by '/' represents greedy decoding and advanced decoding method, i.e., beam search with beam size equal to 4 for SAT (Wang et al., 2018) and length candidate with $B = 1$ for our model. Our model uses the adaptive order.

96% (26.02/27.11) on WMT dataset but maintains at 90% (29.88/33.2) on IWSLT dataset. This shows that our model is robust to the generation order. We also find that left-to-right achieves the best translation performance on WMT dataset and second-best translation performance on IWSLT dataset, respectively. Note that our model has no special preference for generation order due to the conditional permutation language modeling object during training, this phenomenon may suggest that the left-to-right order is a simple but effective order for language modeling. There is also an interesting phenomenon that the gap between greed and beam search of our model is not as large as the baseline model in most cases. We explain that our model can only try to replace certain tokens given a predicted length while the baseline is more flexible and can stop translation dynamically during beam search. So the baseline model is likely to improve more scores than ours when using beam search.

6.2 Semi-autoregressive setting

We compare the results of our model with SAT (Wang et al., 2018) in Table 2. If using greedy decoding, our model gets better results than SAT under all different K. And this gap increases as K increases. With $K = 6$, our model is 2.01 BLEU higher than SAT. If using advanced decoding methods, our model is still better than SAT with $K = 4/6$. With $K = 6$, the gap between the two models reaches 1.34 BLEU. Our model can maintain 97% (26.17/27.11) translation quality on WMT dataset and 89% (29.56/33.2) translation quality on IWSLT dataset with only $1/6$ of the original number of generation steps. It is worth mentioning that SAT will retrain the model from scratch for different K, but our model shares the same parameters in all settings, including settings in other sections. We also observe that our model is more robust to K, which means that the increase in K leads to less BLEU reduction than SAT. We explain that the generation task is more difficult with a larger K. SAT forces the model to generate in parallel from left to right, while our model allows the model to choose its most confident positions to fill in tokens. So the difficulty of a large K is smaller for our model, which leads to a more robust translation performance.

6.3 Non-autoregressive setting

We follow the linear annealing strategy of CMLM (Ghazvininejad et al., 2019). Specifically, in order to generate a sentence of length T in L iterations, our model simply keeps T/L tokens of all empty positions with the highest confidence levels each time.

We compare the performances of our model with three refinement-based non-autoregressive models: NAT-IR (Lee et al., 2018), Latent-Variable NAR (Shu et al., 2019), CMLM (Ghazvininejad et al., 2019) and three other popular non-autoregressive models: NAT (Gu et al., 2017), FlowSeq (Ma et al., 2019), NAT-REG (Wang et al., 2019) in Table 3. When using a vanilla generation method, i.e. only one translation candidate, our model achieves 20.77 BLEU with 2 iterations on WMT dataset. The BLEU rises to 25.10 with 4 iterations, which is 4.84 BLEU higher that NAT-IR with 6 iterations and 3.49 BLEU higher that NAT-IR with 11 iterations. The BLEU rises further to 26.24 on WMT dataset and 30.73 on IWSLT dataset

Models	Iterations	Candidates	AR Score	WMT14EN-DE	IWSLT14DE-EN
NAT(+FT)	-	1	-	17.69	20.32†
NAT(+FT+NPD)	-	100	True	19.17	24.21†
FlowSeq-base(+IWD)	-	15	False	22.49	-
FlowSeq-base(+NPD)	-	15	True	23.08	-
FlowSeq-base(+NPD)	-	30	True	23.48	-
NAT-REG	-	1	-	20.65	23.89
	-	9	True	24.61	28.04
NAT-IR	2*	1	-	13.91	21.86†
	6*	1	-	20.26	-
	11*	1	-	21.61	23.94†
	Adaptive*	1	-	21.54	24.63†
Latent-Variable NAR	2*	1	-	24.10	-
	2*	50	True	25.10	-
CMLM-base	2	5	False	22.91	-
	4	5	False	25.94	-
	10	5	False	27.03	-
Our model	2	1	-	20.77	23.84
	4	1	-	25.10	28.39
	10	1	-	26.24	30.73
	2	5/3	False	21.32	24.07
	4	5/3	False	25.50	28.60
	10	5/3	False	26.35	30.80

Table 3: BLEU scores of non-autoregressive setting. *Iterations* represents the number of iterations during generation. '*' indicates that the number contains a step of initial generation. *Candidates* represents the number of translation candidates. *AR Score* represents whether the score of these translation candidates depends on an additional autoregressive model. '†' denotes the reproduction of Wang et al. (2019). B is set to 2 (5 candidates) for WMT dataset and 1 (3 candidates) for IWSLT dataset.

with 10 iterations. Considering that we do not rely on multiple candidates and an autoregressive model for scoring, this result is encouraging. If using an advanced generation method, i.e., multiple length candidates for our model, the BLEU rises to 21.32 with 2 iterations, 25.50 with 4 iterations and 26.35 with 10 iterations. This result is higher than the other models except for CMLM, a strong non-autoregressive model with state-of-the-art translation performer. The gap is less than 0.7 BLEU in the case of 10 and reduces to 0.44 in the case of 4. Although sacrificing some translation quality, our model is more flexible and robust to support multiple translation manners with one training process which other models didn't show.

6.4 Training strategies

We compare the following four training objective under the same model structure in Table C,

- AR: an autoregressive manner (1), same as the Transformer;
- PAR: an autoregressive manner with permutation mechanism (2);
- SAR: a semi-autoregressive manner (4), same as SAT;
- PSAR: a semi-autoregressive manner with permutation mechanism,

$$P(Y|X) = \mathbb{E}_{z \in \mathcal{Z}_T}[\prod_{t=1}^{[(T-1)/K]+1} p(G_t|X, G_1 \cdots G_{t-1})] \quad (5)$$

where $G_1, \cdots, G_{[(T-1)/K]+1}$ are consecutive groups from the segmentation of a permutation of Y.

We observe that the model trained with AR performs normally with left-to-right order and $\tilde{K} = 1$ but performs poorly in other situations. The reason is that the model only learns to predict the next token given all previous generated tokens. It works fine if there is no gap between training and inference, but cannot generalize to other situations. The model trained with SAR shows similar properties. These two models lack generalization ability and are only suitable for the same generation manner as training.

	ORDER	AR	PAR	SAR	PSAR
$\tilde{K} = 1$	LEFT-TO-RIGHT	25.13	26.62	11.53	25.90
	ADAPTIVE	1.68	26.04	1.12	25.40
$\tilde{K} = 2$	LEFT-TO-RIGHT	12.13	25.96	23.78	25.01
	ADAPTIVE	1.71	26.02	1.70	25.15
$\tilde{K} = 4$	LEFT-TO-RIGHT	2.22	24.67	9.49	23.09
	ADAPTIVE	0.52	25.74	1.65	24.74
$\tilde{K} = 6$	LEFT-TO-RIGHT	0.85	23.14	5.98	21.20
	ADAPTIVE	0.33	25.40	1.59	24.13

Table 4: BLEU scores with different training strategies on WMT14 EN-DE. K is set to 2 for SAR and PSAR. $\tilde{K}$ represents the number of tokens generated in parallel each time. Left-to-right and Adaptive denote two different generation orders. The constant bias term method with $\Delta T = 1$ is used for simplification.

Although all generation manners have gaps with their training objectives, models trained with PAR or PSAR achieve better results in all settings. This phenomenon shows that it is very useful to incorporate the permutation mechanism in a conditional language model to improve the model's generalization for different generation manners. They get even better performances than the model trained with AR when generating from left to right with $\tilde{K} = 1$, although there is no gap in the latter model between training and inference. If generating tokens in a semi-autoregressive manner, they also defeated the task-specific model with SAR. This phenomenon shows that the permutation mechanism also improves the performance of the models. We explain that permutation can be considered as a way of data augmentation, which enriches the number of examples and makes the training process more difficult. Models can certainly achieve better results if they can learn the knowledge correctly. We also observe that the model trained with PAR achieves better results than PSAR in all situations, which shows that it is unnecessary to incorporate parallelism into the permutation mechanism. We explain that PAR is a correct probability decomposition while PSAR assumes that tokens are independent at each step of generation to simplify modeling, which doesn't hold in a real corpus (Yang et al., 2019).

7 Conclusion

We have proposed a unified approach for different generation manners of machine translation. After being trained with the conditional permutation language modeling objective, our model is able to support different generation manners. We have validated our model on the WMT'14 English-German and IWSLT'14 German-English translation tasks. Compared with some strong baseline models, our model achieves better or competitive translation performance in all the settings without task-specific training.

Although we have only verified our ideas on translation tasks, it is easy to find that our approach can be naturally applied to other conditional sequence tasks. We would like to investigate the application of our approach to other tasks in the future.

Acknowledgments

Tian, Cheng and Zhang have been supported by the Beijing Natural Science Foundation (Z190001), National Key Research and Development Project of China (No. 2018AAA0101004), and Beijing Academy of Artificial Intelligence (BAAI).

References

Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. 2016. Layer normalization. *arXiv preprint arXiv:1607.06450*.

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2014. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

Marjan Ghazvininejad, Omer Levy, Yinhan Liu, and Luke Zettlemoyer. 2019. Mask-predict: Parallel decoding of conditional masked language models. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 6114–6123.

Jiatao Gu, James Bradbury, Caiming Xiong, Victor OK Li, and Richard Socher. 2017. Non-autoregressive neural machine translation. *arXiv preprint arXiv:1711.02281*.

Jiatao Gu, Qi Liu, and Kyunghyun Cho. 2019. Insertion-based decoding with automatically inferred generation order. *arXiv preprint arXiv:1902.01370*.

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.

Dan Hendrycks and Kevin Gimpel. 2016. Gaussian error linear units (gelus). *arXiv preprint arXiv:1606.08415*.

Yoon Kim and Alexander M Rush. 2016. Sequence-level knowledge distillation. *arXiv preprint arXiv:1606.07947*.

Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Philipp Koehn, Hieu Hoang, Alexandra Birch, Chris Callison-Burch, Marcello Federico, Nicola Bertoldi, Brooke Cowan, Wade Shen, Christine Moran, Richard Zens, et al. 2007. Moses: Open source toolkit for statistical machine translation. In *Proceedings of the 45th annual meeting of the association for computational linguistics companion volume proceedings of the demo and poster sessions*, pages 177–180.

Guillaume Lample and Alexis Conneau. 2019. Cross-lingual language model pretraining. *arXiv preprint arXiv:1901.07291*.

Jason Lee, Elman Mansimov, and Kyunghyun Cho. 2018. Deterministic non-autoregressive neural sequence modeling by iterative refinement. *arXiv preprint arXiv:1802.06901*.

Xuezhe Ma, Chunting Zhou, Xian Li, Graham Neubig, and Eduard Hovy. 2019. Flowseq: Non-autoregressive conditional sequence generation with generative flow. *arXiv preprint arXiv:1909.02480*.

Elman Mansimov, Alex Wang, and Kyunghyun Cho. 2019. A generalized framework of sequence generation with application to undirected sequence models. *arXiv preprint arXiv:1905.12790*.

Myle Ott, Sergey Edunov, Alexei Baevski, Angela Fan, Sam Gross, Nathan Ng, David Grangier, and Michael Auli. 2019. fairseq: A fast, extensible toolkit for sequence modeling. *arXiv preprint arXiv:1904.01038*.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. 2019. Pytorch: An imperative style, high-performance deep learning library. In *Advances in neural information processing systems*, pages 8026–8037.

Rico Sennrich, Barry Haddow, and Alexandra Birch. 2015. Neural machine translation of rare words with subword units. *arXiv preprint arXiv:1508.07909*.

Raphael Shu, Jason Lee, Hideki Nakayama, and Kyunghyun Cho. 2019. Latent-variable non-autoregressive neural machine translation with deterministic inference using a delta posterior. *arXiv preprint arXiv:1908.07181*.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2014. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958.

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

I Sutskever, O Vinyals, and QV Le. 2014. Sequence to sequence learning with neural networks. *Advances in NIPS*.

Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. 2016. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2818–2826.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008.

Chunqi Wang, Ji Zhang, and Haiqing Chen. 2018. Semi-autoregressive neural machine translation. *arXiv preprint arXiv:1808.08583*.

Yiren Wang, Fei Tian, Di He, Tao Qin, ChengXiang Zhai, and Tie-Yan Liu. 2019. Non-autoregressive machine translation with auxiliary regularization. *arXiv preprint arXiv:1902.10245*.

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

Zhilin Yang, Zihang Dai, Yiming Yang, Jaime Carbonell, Ruslan Salakhutdinov, and Quoc V Le. 2019. Xlnet: Generalized autoregressive pretraining for language understanding. *arXiv preprint arXiv:1906.08237*.

A Length prediction

We compare the translation performance of the three length predictors on the WMT'14 English-German translation task. Specifically,

- Constant Bias Term (CBT): the constant bias ΔT is set to 1 according to the length statistics of the training set;
- Regression-based MLP (R-MLP): length prediction is modeled as a regression problem;
- Classification-based MLP (C-MLP): length prediction is modeled as a classification problem.

We mean-pool the encoder output $\boldsymbol{S}$ into a single vector, followed by a two-layer perceptron with middle hidden state dimension equal to half of the input dimension to predict the length difference. C-MLP has a similar structure to R-MLP, except that C-MLP outputs the classification probability of length difference in range [-35, 35] instead of a real value. In the classification scenario, the length predictor only cares about the classification accuracy and cannot realize the order relationship between different length categories; but in the regression scenario, the predicted length would not deviate much from the true length even if it is not correctly predicted due to a proper optimization objective. For a fair comparison, the two predictors use the well-trained encoder with fixed parameters as feature extraction, and filter out data pairs with length difference greater than 35.

ORDER	MANNER	CBT	R-MLP	C-MLP
LEFT-TO-RIGHT	$K = 1$	26.62	26.98	26.99
	$K = 2$	25.96	26.49	26.44
	$K = 4$	24.67	25.24	25.20
	$K = 6$	23.14	23.73	23.69
	$L = 2$	16.56	17.00	16.95
	$L = 4$	20.75	21.30	21.23
	$L = 10$	24.83	25.50	25.42
ADAPTIVE	$K = 1$	26.04	26.64	26.64
	$K = 2$	26.02	26.48	26.55
	$K = 4$	25.74	26.18	26.25
	$K = 6$	25.40	25.94	25.98
	$L = 2$	20.14	20.77	20.66
	$L = 4$	24.38	25.10	24.96
	$L = 10$	25.80	26.24	26.34

Table 5: BLEU scores of three length predictors on WMT dataset. *Order* indicates two generation orders. K represents the number of tokens generated each time in semi-autoregressive setting. L represents the number of iterations in non-autoregressive setting.

We observe that the two MLP-based methods performe similarly and are significantly better than CBT. But the number of parameters of R-MLP (0.13M) is less than that of C-MLP (0.15M).

B Comparison with XLNet

XLNet uses the partial prediction trick which only predict the last tokens in a factorization order to reduce the optimization difficulty, because the permutation language modeling objective is a much more challenging optimization problem due to the permutation. However, our model is for conditional generation rather than representation learning and the correct generation of the preceding tokens is quite important due to exposure bias. If using partial training, the model always generates tokens when it has seen multiple tokens during training, while it has to generate from scratch without any previous tokens during inference. The gap between training and inference may hurt the performance. We think that training with all objective is necessary for our model. We find that good initialization is helpful to solve this problem. We train this model until it reaches the stopping criterion, and then we use the obtained parameters to initialize the entire network to retrain it. Training-initialization-retraining can significantly improve results.

C Training strategies

We also provide the translation performance of models trained with SAR ($K = 4/6$) and PSAR ($K = 4/6$). We can find the same properties as mentioned in the paper.

	ORDER	$K = 4$		$K = 6$	
		SAR	PSAR	SAR	PSAR
$\tilde{K} = 1$	LEFT-TO-RIGHT	4.42	25.93	3.52	25.51
	ADAPTIVE	0.77	25.25	0.72	24.57
$\tilde{K} = 2$	LEFT-TO-RIGHT	9.33	25.05	5.77	24.52
	ADAPTIVE	1.11	25.03	1.03	24.47
$\tilde{K} = 4$	LEFT-TO-RIGHT	22.55	23.47	9.13	22.56
	ADAPTIVE	2.82	24.91	2.80	24.31
$\tilde{K} = 6$	LEFT-TO-RIGHT	11.46	21.45	20.56	20.23
	ADAPTIVE	3.36	24.54	5.21	23.83

Table 6: BLEU scores with different training strategies on WMT14 EN-DE. K is set to 4/6 for SAR and PSAR. $\tilde{K}$ represents the number of tokens generated in parallel each time. Left-to-right and Adaptive denote two different generation orders. The constant bias term method with $\Delta T = 1$ is used for simplification.