FlowSeq: Non-Autoregressive Conditional Sequence Generation with Generative Flow

Xuezhe Ma[*,1] **Chunting Zhou**[*,1] **Xian Li**[2] **Graham Neubig**[1] **Eduard Hovy**[1]
[1]Language Technologies Institute, Carnegie Mellon University
[2]Facebook AI
{xuezhem, chuntinz, gneubig, ehovy}@cs.cmu.edu xianl@fb.com

Abstract

Most sequence-to-sequence (seq2seq) models are *autoregressive*; they generate each token by conditioning on previously generated tokens. In contrast, non-autoregressive seq2seq models generate all tokens in one pass, which leads to increased efficiency through parallel processing on hardware such as GPUs. However, directly modeling the joint distribution of all tokens simultaneously is challenging, and even with increasingly complex model structures accuracy lags significantly behind autoregressive models. In this paper, we propose a simple, efficient, and effective model for non-autoregressive sequence generation using latent variable models. Specifically, we turn to generative flow, an elegant technique to model complex distributions using neural networks, and design several layers of flow tailored for modeling the conditional density of sequential latent variables. We evaluate this model on three neural machine translation (NMT) benchmark datasets, achieving comparable performance with state-of-the-art non-autoregressive NMT models and almost constant decoding time w.r.t the sequence length.[1]

1 Introduction

Neural sequence-to-sequence (seq2seq) models (Bahdanau et al., 2015; Rush et al., 2015; Vinyals et al., 2015; Vaswani et al., 2017) generate an output sequence $\mathbf{y} = \{y_1, \ldots, y_T\}$ given an input sequence $\mathbf{x} = \{x_1, \ldots, x_{T'}\}$ using conditional probabilities $P_\theta(\mathbf{y}|\mathbf{x})$ predicted by neural networks (parameterized by θ).

Most seq2seq models are *autoregressive*, meaning that they factorize the joint probability of the output sequence given the input sequence $P_\theta(\mathbf{y}|\mathbf{x})$ into the product of probabilities over the next token in the sequence given the input sequence and previously generated tokens:

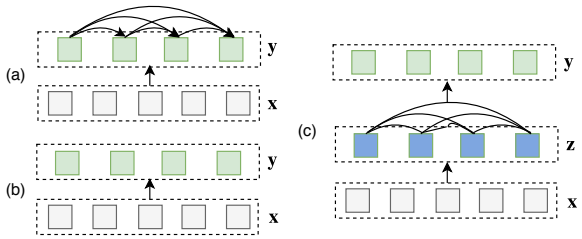


Figure 1: (a) Autoregressive (b) non-autoregressive and (c) our proposed sequence generation models. $\mathbf{x}$ is the source, $\mathbf{y}$ is the target, and $\mathbf{z}$ are latent variables.

$$P_\theta(\mathbf{y}|\mathbf{x}) = \prod_{t=1}^{T} P_\theta(y_t|y_{<t}, \mathbf{x}). \quad (1)$$

Each factor, $P_\theta(y_t|y_{<t}, \mathbf{x})$, can be implemented by function approximators such as RNNs (Bahdanau et al., 2015) and Transformers (Vaswani et al., 2017). This factorization takes the complicated problem of *joint estimation* over an exponentially large output space of outputs $\mathbf{y}$, and turns it into a *sequence of tractable multi-class classification problems* predicting y_t given the previous words, allowing for simple maximum log-likelihood training. However, this assumption of left-to-right factorization may be sub-optimal from a modeling perspective (Gu et al., 2019; Stern et al., 2019), and generation of outputs must be done through a linear left-to-right pass through the output tokens using beam search, which is not easily parallelizable on hardware such as GPUs.

Recently, there has been work on non-autoregressive sequence generation for neural machine translation (NMT; Gu et al. (2018); Lee et al. (2018); Ghazvininejad et al. (2019)) and language modeling (Ziegler and Rush, 2019). Non-autoregressive models attempt to model the joint distribution $P_\theta(\mathbf{y}|\mathbf{x})$ directly, decoupling the dependencies of decoding history during generation.

[*] Equal contribution, in alphabetical order.
[1] https://github.com/XuezheMax/flowseq

Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing, pages 4282–4292, Hong Kong, China, November 3–7, 2019.

A naïve solution is to assume that each token of the target sequence is independent given the input:

$$P_\theta(\mathbf{y}|\mathbf{x}) = \prod_{t=1}^{T} P_\theta(y_t|\mathbf{x}). \quad (2)$$

Unfortunately, the performance of this simple model falls far behind autoregressive models, as seq2seq tasks usually do have strong conditional dependencies between output variables (Gu et al., 2018). This problem can be mitigated by introducing a latent variable $\mathbf{z}$ to model these conditional dependencies:

$$P_\theta(\mathbf{y}|\mathbf{x}) = \int_{\mathbf{z}} P_\theta(\mathbf{y}|\mathbf{z},\mathbf{x}) p_\theta(\mathbf{z}|\mathbf{x}) d\mathbf{z}, \quad (3)$$

where $p_\theta(\mathbf{z}|\mathbf{x})$ is the prior distribution over latent $\mathbf{z}$ and $P_\theta(\mathbf{y}|\mathbf{z},\mathbf{x})$ is the "generative" distribution (a.k.a decoder). Non-autoregressive generation can be achieved by the following independence assumption in the decoding process:

$$P_\theta(\mathbf{y}|\mathbf{z},\mathbf{x}) = \prod_{t=1}^{T} P_\theta(y_t|\mathbf{z},\mathbf{x}). \quad (4)$$

Gu et al. (2018) proposed a $\mathbf{z}$ representing fertility scores specifying the number of output words each input word generates, significantly improving the performance over Eq. (2). But the performance still falls behind state-of-the-art autoregressive models due to the limited expressiveness of fertility to model the interdependence between words in $\mathbf{y}$.

In this paper, we propose a simple, effective, and efficient model, **FlowSeq**, which models expressive prior distribution $p_\theta(\mathbf{z}|\mathbf{x})$ using a powerful mathematical framework called generative flow (Rezende and Mohamed, 2015). This framework can elegantly model complex distributions, and has obtained remarkable success in modeling continuous data such as images and speech through efficient density estimation and sampling (Kingma and Dhariwal, 2018; Prenger et al., 2019; Ma and Hovy, 2019). Based on this, we posit that generative flow also has potential to introduce more meaningful latent variables $\mathbf{z}$ in the non-autoregressive generation in Eq. (3).

FlowSeq is a *flow-based sequence-to-sequence* model, which is (to our knowledge) the first non-autoregressive seq2seq model utilizing generative flows. It allows for efficient parallel decoding while modeling the joint distribution of the output sequence. Experimentally, on three benchmark datasets for machine translation – WMT2014, WMT2016 and IWSLT-2014, FlowSeq achieves comparable performance with state-of-the-art non-autoregressive models, and almost constant decoding time w.r.t. the sequence length compared to a typical left-to-right Transformer model, which is super-linear.

2 Background

As noted above, incorporating expressive latent variables $\mathbf{z}$ is essential to decouple the dependencies between tokens in the target sequence in non-autoregressive models. However, in order to model all of the complexities of sequence generation to the point that we can read off all of the words in the output in an independent fashion (as in Eq. (4)), the prior distribution $p_\theta(\mathbf{z}|\mathbf{x})$ will necessarily be quite complex. In this section, we describe generative flows (Rezende and Mohamed, 2015), an effective method for arbitrary modeling of complicated distributions, before describing how we apply them to sequence-to-sequence generation in §3.

2.1 Flow-based Generative Models

Put simply, flow-based generative models work by transforming a simple distribution (e.g. a simple Gaussian) into a complex one (e.g. the complex prior distribution over $\mathbf{z}$ that we want to model) through a chain of invertible transformations.

Formally, a set of latent variables $\boldsymbol{\upsilon} \in \Upsilon$ are introduced with a simple prior distribution $p_\Upsilon(\upsilon)$. We then define a bijection function $f : \mathcal{Z} \to \Upsilon$ (with $g = f^{-1}$), whereby we can define a generative process over variables $\mathbf{z}$:

$$\begin{aligned} \boldsymbol{\upsilon} &\sim p_\Upsilon(\boldsymbol{\upsilon}) \\ \mathbf{z} &= g_\theta(\boldsymbol{\upsilon}). \end{aligned} \quad (5)$$

An important insight behind flow-based models is that given this bijection function, the change of variable formula defines the model distribution on $\mathbf{z} \in \mathcal{Z}$ by:

$$p_\theta(\mathbf{z}) = p_\Upsilon(f_\theta(\mathbf{z})) \left| \det\left(\frac{\partial f_\theta(\mathbf{z})}{\partial \mathbf{z}}\right) \right|. \quad (6)$$

Here $\frac{\partial f_\theta(\mathbf{z})}{\partial \mathbf{z}}$ is the Jacobian matrix of f_θ at $\mathbf{z}$.

Eq. (6) provides a way to calculate the (complex) density of $\mathbf{z}$ by calculating the (simple) density of υ and the Jacobian of the transformation from $\mathbf{z}$ to υ. For efficiency purposes, flow-based models generally use certain types of transformations f_θ where both the inverse functions

g_θ and the Jacobian determinants are tractable to compute. A stacked sequence of such invertible transformations is also called a (normalizing) *flow* (Rezende and Mohamed, 2015):

$$\mathbf{z} \underset{g_1}{\overset{f_1}{\longleftrightarrow}} H_1 \underset{g_2}{\overset{f_2}{\longleftrightarrow}} H2 \underset{g_3}{\overset{f_3}{\longleftrightarrow}} \cdots \underset{g_K}{\overset{f_K}{\longleftrightarrow}} \boldsymbol{\upsilon},$$

where $f = f_1 \circ f_2 \circ \cdots \circ f_K$ is a flow of K transformations (omitting θs for brevity).

2.2 Variational Inference and Training

In the context of maximal likelihood estimation (MLE), we wish to minimize the negative log-likelihood of the parameters:

$$\min_{\theta \in \Theta} \frac{1}{N} \sum_{i=1}^{N} -\log P_\theta(\mathbf{y}^i|\mathbf{x}^i), \tag{7}$$

where $D = \{(\mathbf{x}^i, \mathbf{y}^i)\}_{i=1}^N$ is the set of training data. However, the likelihood $P_\theta(\mathbf{y}|\mathbf{x})$ after marginalizing out latent variables $\mathbf{z}$ (LHS in Eq. (3)) is intractable to compute or differentiate directly. Variational inference (Wainwright et al., 2008) provides a solution by introducing a parametric *inference model* $q_\phi(\mathbf{z}|\mathbf{y}, \mathbf{x})$ (a.k.a posterior) which is then used to approximate this integral by sampling individual examples of $\mathbf{z}$. These models then optimize the *evidence lower bound* (ELBO), which considers both the "reconstruction error" $\log P_\theta(\mathbf{y}|\mathbf{z}, \mathbf{x})$ and KL-divergence between the posterior and the prior:

$$\begin{aligned}\log P_\theta(\mathbf{y}|\mathbf{x}) \geq \mathrm{E}_{q_\phi(\mathbf{z}|\mathbf{y},\mathbf{x})}[\log P_\theta(\mathbf{y}|\mathbf{z}, \mathbf{x})] \\ - \mathrm{KL}(q_\phi(\mathbf{z}|\mathbf{y}, \mathbf{x})||p_\theta(\mathbf{z}|\mathbf{x})).\end{aligned} \tag{8}$$

Both inference model ϕ and decoder θ parameters are optimized according to this objective.

3 FlowSeq

We first overview FlowSeq's architecture (shown in Figure 2) and training process here before detailing each component in following sections. Similarly to classic seq2seq models, at both training and test time FlowSeq first reads the whole input sequence $\mathbf{x}$ and calculates a vector for each word in the sequence, the source encoding.

At training time, FlowSeq's parameters are learned using a variational training paradigm overviewed in §2.2. First, we draw samples of latent codes $\mathbf{z}$ from the current posterior $q_\phi(\mathbf{z}|\mathbf{y}, \mathbf{x})$. Next, we feed $\mathbf{z}$ together with source encodings into the decoder network and the prior flow to compute the probabilities of $P_\theta(\mathbf{y}|\mathbf{z}, \mathbf{x})$ and $p_\theta(\mathbf{z}|\mathbf{x})$ for optimizing the ELBO (Eq. (8)).

At test time, generation is performed by first sampling a latent code $\mathbf{z}$ from the prior flow by executing the generative process defined in Eq. (5). In this step, the source encodings produced from the encoder are used as conditional inputs. Then the decoder receives both the sampled latent code $\mathbf{z}$ and the source encoder outputs to generate the target sequence $\mathbf{y}$ from $P_\theta(\mathbf{y}|\mathbf{z}, \mathbf{x})$.

3.1 Source Encoder

The source encoder encodes the source sequences into hidden representations, which are used in computing attention when generating latent variables in the posterior network and prior network as well as the cross-attention with decoder. Any standard neural sequence model can be used as its encoder, including RNNs (Bahdanau et al., 2015) or Transformers (Vaswani et al., 2017).

3.2 Posterior

Generation of Latent Variables. The latent variables $\mathbf{z}$ are represented as a sequence of continuous random vectors $\mathbf{z} = \{\mathbf{z}_1, \ldots, \mathbf{z}_T\}$ with the same length as the target sequence $\mathbf{y}$. Each $\mathbf{z}_t$ is a d_z-dimensional vector, where d_z is the dimension of the latent space. The posterior distribution $q_\phi(\mathbf{z}|\mathbf{y}, \mathbf{x})$ models each $\mathbf{z}_t$ as a diagonal Gaussian with learned mean and variance:

$$q_\phi(\mathbf{z}|\mathbf{y}, \mathbf{x}) = \prod_{t=1}^{T} \mathcal{N}(\mathbf{z}_t|\mu_t(\mathbf{x}, \mathbf{y}), \sigma_t^2(\mathbf{x}, \mathbf{y})) \tag{9}$$

where $\mu_t(\cdot)$ and $\sigma_t(\cdot)$ are neural networks such as RNNs or Transformers.

Zero initialization. While we perform standard random initialization for most layers of the network, we initialize the last linear transforms that generate the μ and $\log \sigma^2$ values with zeros. This ensures that the posterior distribution as a simple normal distribution, which we found helps train very deep generative flows more stably.

Token Dropout. The motivation of introducing the latent variable $\mathbf{z}$ into the model is to model the uncertainty in the generative process. Thus, it is preferable that $\mathbf{z}$ capture contextual interdependence between tokens in $\mathbf{y}$. However, there is an obvious local optimum where the posterior network generates a latent vector $\mathbf{z}_t$ that only encodes the information about the corresponding target token y_t, and the decoder simply generates the

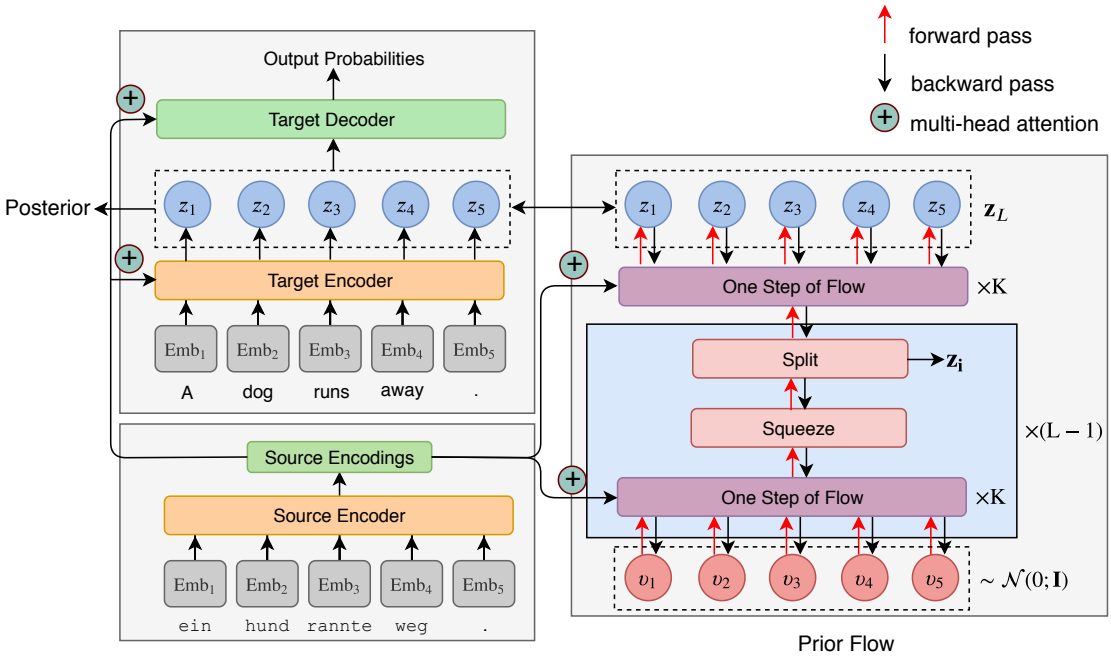


Figure 2: Neural architecture of FlowSeq, including the encoder, the decoder and the posterior networks, together with the multi-scale architecture of the prior flow. The architecture of each flow step is in Figure 3.

"correct" token at each step t with $\mathbf{z}_t$ as input. In this case, FlowSeq reduces to the baseline model in Eq. (2). To escape this undesired local optimum, we apply token-level dropout to randomly drop an entire token when calculating the posterior, to ensure the model also has to learn how to use contextual information. This technique is similar to the "masked language model" in previous studies (Melamud et al., 2016; Devlin et al., 2018; Ma et al., 2018).

3.3 Decoder

As the decoder, we take the latent sequence $\mathbf{z}$ as input, run it through several layers of a neural sequence model such as a Transformer, then directly predict the output tokens in $\mathbf{y}$ individually and independently. Notably, unlike standard seq2seq decoders, we do not perform causal masking to prevent attending to future tokens, making the model fully non-autoregressive.

3.4 Flow Architecture for Prior

The flow architecture is based on Glow (Kingma and Dhariwal, 2018). It consists of a series of steps of flow, combined in a multi-scale architecture (see Figure 2.) Each step of flow consists three types of elementary flows – actnorm, invertible multi-head linear, and coupling. Note that all three functions are invertible and conducive to calculation of log determinants (details in Appendix A).

Actnorm. The activation normalization layer (actnorm; Kingma and Dhariwal (2018)) is an alternative for batch normalization (Ioffe and Szegedy, 2015), that has mainly been used in the context of image data to alleviate problems in model training. Actnorm performs an affine transformation of the activations using a scale and bias parameter per feature for sequences:

$$\mathbf{z}'_t = \mathbf{s} \odot \mathbf{z}_t + \mathbf{b}. \tag{10}$$

Both $\mathbf{z}$ and $\mathbf{z}'$ are tensors of shape $[T \times d_z]$ with time dimension t and feature dimension d_z. The parameters are initialized such that over each feature $\mathbf{z}'_t$ has zero mean and unit variance given an initial mini-batch of data.

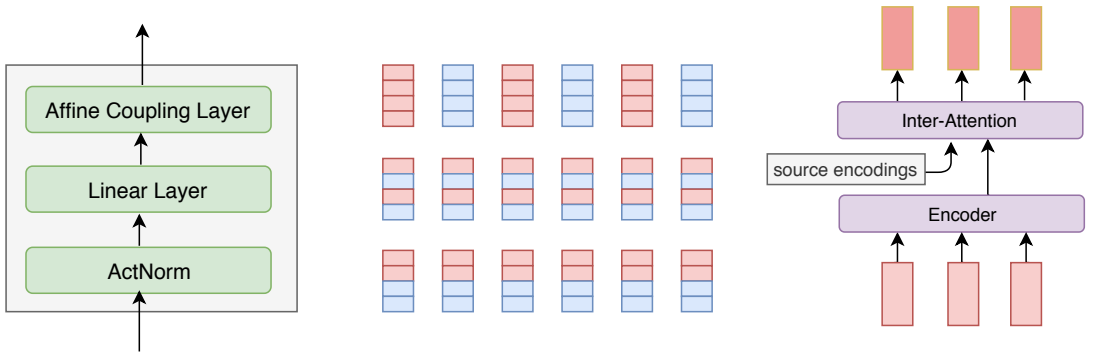
Invertible Multi-head Linear Layers. To incorporate general permutations of variables along the feature dimension to ensure that each dimension can affect every other ones after a sufficient number of steps of flow, Kingma and Dhariwal (2018) proposed a trainable invertible 1×1 convolution layer for 2D images. It is straightforward to apply similar transformations to sequential data:

$$\mathbf{z}'_t = \mathbf{z}_t \mathbf{W}, \tag{11}$$

where $\mathbf{W}$ is the weight matrix of shape $[d_z \times d_z]$. The log-determinant of this transformation is:

$$\log \left| \det \left(\frac{\partial \text{linear}(\mathbf{z}; \mathbf{W})}{\partial \mathbf{z}} \right) \right| = T \cdot \log |\det(\mathbf{W})|$$

The cost of computing $\det(\mathbf{W})$ is $O(d_z^3)$.



(a) One step of flow. (b) Coupling layer splits. (c) NN function on the split of the coupling layer.

Figure 3: (a) The architecture of one step of our flow. (b) The visualization of three split pattern for coupling layers, where the red color denotes $\mathbf{z}_a$ and the blue color denotes zv_b. (c) The attention-based architecture of the NN function in coupling layers.

Unfortunately, d_z in Seq2Seq generation is commonly large, e.g. 512, significantly slowing down the model for computing $\det(\mathbf{W})$. To apply this to sequence generation, we propose a multi-head invertible linear layer, which first splits each d_z-dimensional feature vector into h heads with dimension $d_h = d_z/h$. Then the linear transformation in (11) is applied to each head, with $d_h \times d_h$ weight matrix $\mathbf{W}$, significantly reducing the dimension. For splitting of heads, one step of flow contains one linear layer with either row-major or column-major splitting format, and these steps with different linear layers are composed in an alternating pattern.

Affine Coupling Layers. To model interdependence across time steps, we use affine coupling layers (Dinh et al., 2016):

$$\begin{aligned} \mathbf{z}_a, \mathbf{z}_b &= \text{split}(\mathbf{z}) \\ \mathbf{z}'_a &= \mathbf{z}_a \\ \mathbf{z}'_b &= \text{s}(\mathbf{z}_a, \mathbf{x}) \odot \mathbf{z}_b + \text{b}(\mathbf{z}_a, \mathbf{x}) \\ \mathbf{z}' &= \text{concat}(\mathbf{z}'_a, \mathbf{z}'_b), \end{aligned}$$

where $\text{s}(\mathbf{z}_a, \mathbf{x})$ and $\text{b}(\mathbf{z}_a, \mathbf{x})$ are outputs of two neural networks with $\mathbf{z}_a$ and $\mathbf{x}$ as input. These are shown in Figure 3 (c). In experiments, we implement $\text{s}(\cdot)$ and $\text{b}(\cdot)$ with one Transformer decoder layer (Vaswani et al., 2017): multi-head self-attention over $\mathbf{z}_a$, followed by multi-head inter-attention over $\mathbf{x}$, followed by a position-wise feed-forward network. The input $\mathbf{z}_a$ is fed into this layer in one pass, without causal masking.

As in Dinh et al. (2016), the $\text{split}()$ function splits $\mathbf{z}$ the input tensor into two halves, while the concat operation performs the corresponding reverse concatenation operation. In our architecture, three types of split functions are used, based on the split dimension and pattern. Figure 3 (b) illustrates the three splitting types. The first type of split groups $\mathbf{z}$ along the time dimension on alternate indices. In this case, FlowSeq mainly models the interactions between time-steps. The second and third types of splits perform on the feature dimension, with continuous and alternate patterns, respectively. For each type of split, we alternate $\mathbf{z}_a$ and $\mathbf{z}_b$ to increase the flexibility of the split function. Different types of affine coupling layers alternate in the flow, similar to the linear layers.

Multi-scale Architecture. We follow Dinh et al. (2016) in implementing a multi-scale architecture using the squeezing operation on the feature dimension, which has been demonstrated helpful for training deep flows. Formally, each scale is a combination of several steps of the flow (see Figure 3 (a)). After each scale, the model drops half of the dimensions with the third type of split in Figure 3 (b) to reduce computational and memory cost, outputting the tensor with shape $[T \times \frac{d}{2}]$. Then the squeezing operation transforms the $T \times \frac{d}{2}$ tensor into an $\frac{T}{2} \times d$ one as the input of the next scale. We pad each sentence with EOS tokens to ensure T is divisible by 2. The right component of Figure 2 illustrates the multi-scale architecture.

3.5 Predicting Target Sequence Length

In autoregressive seq2seq models, it is natural to determine the length of the sequence dynamically by simply predicting a special EOS token. However, for FlowSeq to predict the entire sequence in parallel, it needs to know its length in advance to generate the latent sequence $\mathbf{z}$. Instead of predicting the absolute length of the target sequence, we predict the length difference between source

and target sequences using a classifier with a range of $[-20, 20]$. Numbers in this range are predicted by max-pooling the source encodings into a single vector,[2] running this through a linear layer, and taking a softmax. This classifier is learned jointly with the rest of the model.

3.6 Decoding Process

At inference time, the model needs to identify the sequence with the highest conditional probability by marginalizing over all possible latent variables (see Eq. (3)), which is intractable in practice. We propose three approximating decoding algorithms to reduce the search space.

Argmax Decoding. Following Gu et al. (2018), one simple and effective method is to select the best sequence by choosing the highest-probability latent sequence $\mathbf{z}$:

$$\begin{aligned}\mathbf{z}^* &= \operatorname*{argmax}_{\mathbf{z}\in\mathcal{Z}} p_\theta(\mathbf{z}|\mathbf{x}) \\ \mathbf{y}^* &= \operatorname*{argmax}_{\mathbf{y}} P_\theta(\mathbf{y}|\mathbf{z}^*, \mathbf{x})\end{aligned}$$

where identifying $\mathbf{y}^*$ only requires independently maximizing the local probability for each output position (see Eq. 4).

Noisy Parallel Decoding (NPD). A more accurate approximation of decoding, proposed in Gu et al. (2018), is to draw samples from the latent space and compute the best output for each latent sequence. Then, a pre-trained autoregressive model is adopted to rank these sequences. In FlowSeq, different candidates can be generated by sampling different target lengths or different samples from the prior, and both of the strategies can be batched via masks during decoding. In our experiments, we first select the top l length candidates from the length predictor in §3.5. Then, for each length candidate we use r random samples from the prior network to generate output sequences, yielding a total of $l \times r$ candidates.

Importance Weighted Decoding (IWD) The third approximating method is based on the *lower bound of importance weighted estimation* (Burda et al., 2015). Similarly to NPD, IWD first draws samples from the latent space and computes the best output for each latent sequence. Then, IWD ranks these candidate sequences with K importance samples:

$$\begin{aligned}\mathbf{z}_i &\sim p_\theta(\mathbf{z}|\mathbf{x}), \forall i = 1, \ldots, N \\ \hat{\mathbf{y}}_i &= \operatorname*{argmax}_{\mathbf{y}} P_\theta(\mathbf{y}|\mathbf{z}_i, \mathbf{x}) \\ \mathbf{z}_i^{(k)} &\sim q_\phi(\mathbf{z}|\hat{\mathbf{y}}_i, \mathbf{x}), \forall k = 1, \ldots, K \\ P(\hat{\mathbf{y}}_i|\mathbf{x}) &\approx \frac{1}{K}\sum_{k=1}^{K} \frac{P_\theta(\hat{\mathbf{y}}_i|\mathbf{z}_i^{(k)}, \mathbf{x}) p_\theta(\mathbf{z}_i^{(k)}|\mathbf{x})}{q_\phi(\mathbf{z}_i^{(k)}|\hat{\mathbf{y}}_i, \mathbf{x})}\end{aligned}$$

IWD does not rely on a separate pre-trained model, though it significantly slows down the decoding speed. The detailed comparison of these three decoding methods is provided in §4.2.

3.7 Discussion

Different from the architecture proposed in Ziegler and Rush (2019), the architecture of FlowSeq is not using any autoregressive flow (Kingma et al., 2016; Papamakarios et al., 2017), yielding a truly non-autoregressive model with efficient generation. Note that the FlowSeq remains non-autoregressive even if we use an RNN in the architecture because RNN is only used to encode a complete sequence of codes and all the input tokens can be fed into the RNN in parallel. This makes it possible to use highly-optimized implementations of RNNs such as those provided by cuDNN.[3] Thus while RNNs do experience some drop in speed, it is less extreme than that experienced when using autoregressive models.

4 Experiments

4.1 Experimental Setups

Translation Datasets We evaluate FlowSeq on three machine translation benchmark datasets: WMT2014 DE-EN (around 4.5M sentence pairs), WMT2016 RO-EN (around 610K sentence pairs) and a smaller dataset IWSLT2014 DE-EN (around 150K sentence pairs). We use scripts from fairseq (Ott et al., 2019) to preprocess WMT2014 and IWSLT2014, where the preprocessing steps follow Vaswani et al. (2017) for WMT2014. We use the data provided in Lee et al. (2018) for WMT2016. For both WMT datasets, the source and target languages share the same set of BPE embeddings while for IWSLT2014 we use separate embeddings. During training, we filter out sentences longer than 80 for WMT dataset and 60 for IWSLT, respectively.

[2] We experimented with other methods such as mean-pooling or taking the last hidden state and found no major difference in our experiments

[3] https://devblogs.nvidia.com/optimizing-recurrent-neural-networks-cudnn-5/

Models	WMT2014 EN-DE	WMT2014 DE-EN	WMT2016 EN-RO	WMT2016 RO-EN	IWSLT2014 DE-EN
Raw Data					
CMLM-base	10.88	–	20.24	–	–
LV NAR	11.80	–	–	–	–
FlowSeq-base	18.55	23.36	29.26	30.16	**24.75**
FlowSeq-large	**20.85**	**25.40**	**29.86**	**30.69**	–
Knowledge Distillation					
NAT-IR	13.91	16.77	24.45	25.73	21.86
CTC Loss	17.68	19.80	19.93	24.71	–
NAT w/ FT	17.69	21.47	27.29	29.06	20.32
NAT-REG	20.65	24.77	–	–	23.89
CMLM-small	15.06	19.26	20.12	20.36	–
CMLM-base	18.12	22.26	23.65	22.78	–
FlowSeq-base	21.45	26.16	29.34	30.44	**27.55**
FlowSeq-large	**23.72**	**28.39**	**29.73**	**30.72**	–

Table 1: BLEU scores on three MT benchmark datasets for FlowSeq with argmax decoding and baselines with purely non-autoregressive decoding method. The first and second block are results of models trained w/w.o. knowledge distillation, respectively.

Modules and Hyperparameters We implement the encoder, decoder and posterior networks with standard (unmasked) Transformer layers (Vaswani et al., 2017). For WMT datasets, the encoder consists of 6 layers, and the decoder and posterior are composed of 4 layers, and 8 attention heads. and for IWSLT, the encoder has 5 layers, and decoder and posterior have 3 layers, and 4 attention heads. The prior flow consists of 3 scales with the number of steps $[48, 48, 16]$ from bottom to top. To dissect the impact of model dimension on translation quality and speed, we perform experiments on two versions of FlowSeq with $d_{model}/d_{hidden} = 256/512$ (base) and $d_{model}/d_{hidden} = 512/1024$ (large). More model details are provided in Appendix B.

Optimization Parameter optimization is performed with the Adam optimizer (Kingma and Ba, 2014) with $\beta = (0.9, 0.999)$ and $\epsilon = 1e-6$. Each mini-batch consist of 2048 sentences. The learning rate is initialized to $5e-4$, and exponentially decays with rate 0.999995. The gradient clipping cutoff is 1.0. For all the FlowSeq models, we apply 0.1 label smoothing and averaged the 5 best checkpoints to create the final model.

At the beginning of training, the posterior network is randomly initialized, producing noisy supervision to the prior. To mitigate this issue, we first set the weight of the KL term in ELBO to zero for 30,000 updates to train the encoder, decoder and posterior networks. Then the KL weight linearly increases to one for another 10,000 updates, which we found essential to accelerate training and achieve stable performance.

Models	WMT2014 EN-DE	WMT2014 DE-EN	WMT2016 EN-RO	WMT2016 RO-EN
Autoregressive Methods				
Transformer-base	27.30	–	–	–
Our Implementation	27.16	31.44	32.92	33.09
Raw Data				
CMLM-base (refinement 4)	22.06	–	30.89	–
CMLM-base (refinement 10)	**24.65**	–	**32.53**	–
FlowSeq-base (IWD $n = 15$)	20.20	24.63	30.61	31.50
FlowSeq-base (NPD $n = 15$)	20.81	25.76	31.38	32.01
FlowSeq-base (NPD $n = 30$)	21.15	26.04	31.74	32.45
FlowSeq-large (IWD $n = 15$)	22.94	27.16	31.08	32.03
FlowSeq-large (NPD $n = 15$)	23.14	27.71	31.97	32.46
FlowSeq-large (NPD $n = 30$)	23.64	**28.29**	32.35	**32.91**
Knowledge Distillation				
NAT-IR (refinement 10)	21.61	25.48	29.32	30.19
NAT w/ FT (NPD $n = 10$)	18.66	22.42	29.02	31.44
NAT-REG (NPD $n = 9$)	24.61	28.90	–	–
LV NAR (refinement 4)	24.20	–	–	–
CMLM-small (refinement 10)	25.51	29.47	31.65	32.27
CMLM-base (refinement 10)	**26.92**	**30.86**	**32.42**	**33.06**
FlowSeq-base (IWD $n = 15$)	22.49	27.40	30.59	31.58
FlowSeq-base (NPD $n = 15$)	23.08	28.07	31.35	32.11
FlowSeq-base (NPD $n = 30$)	23.48	28.40	31.75	32.49
FlowSeq-large (IWD $n = 15$)	24.70	29.44	31.02	31.97
FlowSeq-large (NPD $n = 15$)	25.03	30.48	31.89	32.43
FlowSeq-large (NPD $n = 30$)	25.31	30.68	32.20	32.84

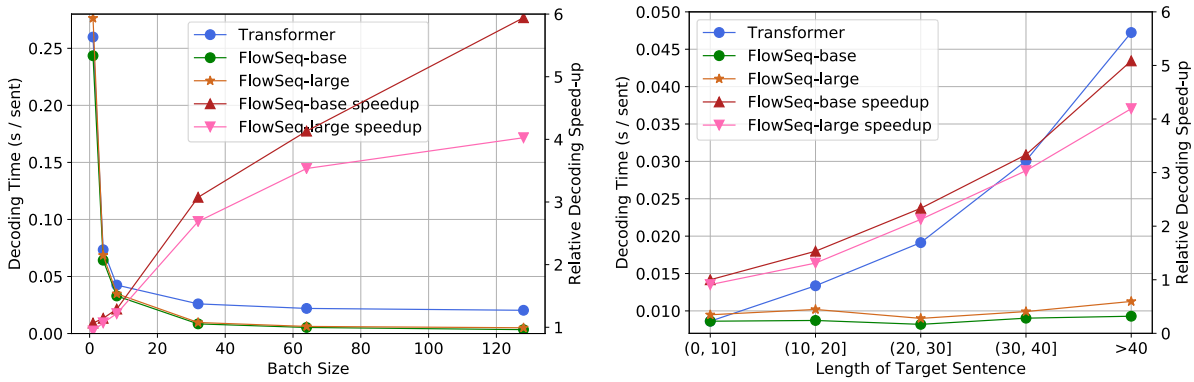
Table 2: BLEU scores on two WMT datasets of models using advanced decoding methods. The first block are Transformer-base (Vaswani et al., 2017). The second and the third block are results of models trained w/w.o. knowledge distillation, respectively. $n = l \times r$ is the total number of candidates for rescoring.

Knowledge Distillation Previous work on non-autoregressive generation (Gu et al., 2018; Ghazvininejad et al., 2019) has used translations produced by a pre-trained autoregressive NMT model as the training data, noting that this can significantly improve the performance. We analyze the impact of distillation in § 4.2.

4.2 Main Results

We first conduct experiments to compare the performance of FlowSeq with strong baseline models, including NAT w/ Fertility (Gu et al., 2018), NAT-IR (Lee et al., 2018), NAT-REG (Wang et al., 2019), LV NAR (Shu et al., 2019), CTC Loss (Libovický and Helcl, 2018), and CMLM (Ghazvininejad et al., 2019).

Table 1 provides the BLEU scores of FlowSeq with argmax decoding, together with baselines with purely non-autoregressive decoding methods that generate output sequence in one parallel pass. The first block lists results of models trained on raw data, while the second block are results using



(a) batch size

(b) target length

Figure 4: The decoding speed of Transformer (batched, beam size 5) and FlowSeq on WMT14 EN-DE test set (a) w.r.t different batch sizes (b) bucketed by different target sentence lengths (batch size 32).

knowledge distillation. Without using knowledge distillation, FlowSeq base model achieves significant improvement (more than 9 BLEU points) over CMLM-base and LV NAR. It demonstrates the effectiveness of FlowSeq on modeling the complex interdependence in target languages.

Towards the effect of knowledge distillation, we can mainly obtain two observations: i) Similar to the findings in previous work, knowledge distillation still benefits the translation quality of FlowSeq. ii) Compared to previous models, the benefit of knowledge distillation on FlowSeq is less significant, yielding less than 3 BLEU improvement on WMT2014 DE-EN corpus, and even no improvement on WMT2016 RO-EN corpus. The reason might be that FlowSeq does not rely much on knowledge distillation to alleviate the multi-modality problem.

Table 2 illustrates the BLEU scores of FlowSeq and baselines with advanced decoding methods such as iterative refinement, IWD and NPD rescoring. The first block in Table 2 includes the baseline results from autoregressive Transformer. For the sampling procedure in IWD and NPD, we sampled from a reduced-temperature model (Kingma and Dhariwal, 2018) to obtain high-quality samples. We vary the temperature within $\{0.1, 0.2, 0.3, 0.4, 0.5, 1.0\}$ and select the best temperature based on the performance on development sets. The analysis of the impact of sampling temperature and other hyper-parameters on samples is in § 4.4. For FlowSeq, NPD obtains better results than IWD, showing that FlowSeq still falls behind auto-regressive Transformer on model data distributions. Comparing with CMLM (Ghazvininejad et al., 2019) with 10 iterations of refinement, which is a contemporaneous work that achieves state-of-the-art translation performance, FlowSeq obtains competitive performance on both WMT2014 and WMT2016 corpora, with only slight degradation in translation quality. Leveraging iterative refinement to further improve the performance of FlowSeq has been left to future work.

4.3 Analysis on Decoding Speed

In this section, we compare the decoding speed (measured in average time in seconds required to decode one sentence) of FlowSeq at test time with that of the autoregressive Transformer model. We use the test set of WMT14 EN-DE for evaluation and all experiments are conducted on a single NVIDIA TITAN X GPU.

How does batch size affect the decoding speed? First, we investigate how different decoding batch size can affect the decoding speed. We vary the decoding batch size within $\{1, 4, 8, 32, 64, 128\}$. Figure. 4a shows that for both FlowSeq and Transformer decoding is faster when using a larger batch size. However, FlowSeq has much larger gains in the decoding speed w.r.t. the increase in batch size, gaining a speed up of 594% of base model and 403% of large model when using a batch size of 128. We hypothesize that this is because the operations in FlowSeq are more friendly to batching while the Transformer model with beam search at test time is less efficient in benefiting from batching.

How does sentence length affect the decoding speed? Next, we examine if sentence length is a major factor affecting the decoding speed. We bucket the test data by the target sentence length.

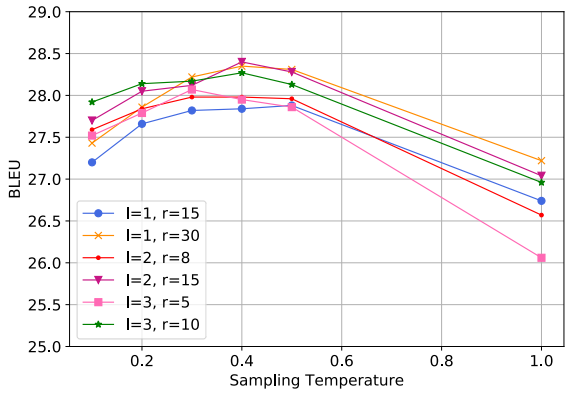


Figure 5: Impact of sampling hyperparameters on the rescoring BLEU on the dev set of WMT14 DE-EN. Experiments are performed with FlowSeq-base trained with distillation data. l is the number of length candidates. r is the number of samples for each length.

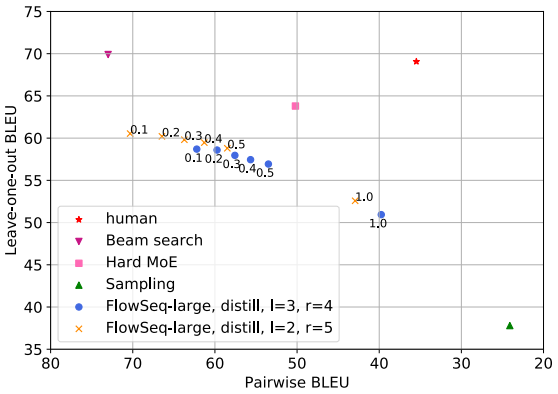


Figure 6: Comparisons of FlowSeq with human translations, beam search and sampling results of Transformer-base, and mixture-of-experts model (Hard MoE (Shen et al., 2019)) on the averaged leave-one-out BLEU score v.s pairwise-BLEU in descending order.

From Fig. 4b, we can see that as the sentence length increases, FlowSeq achieves almost constant decoding time while Transformer has a linearly increasing decoding time. The relative decoding speed up of FlowSeq versus Transformer linearly increases as the sequence length increases. The potential of decoding long sequences with constant time is an attractive property of FlowSeq.

4.4 Analysis of Rescoring Candidates

In Fig. 5, we analyze how different sampling hyperparameters affect the performance of rescoring. First, we observe that the number of samples r for each length is the most important factor. The performance is always improved with a larger sample size. Second, a larger number of length candidates does not necessarily increase the rescoring performance. Third, we find that a larger sampling temperature (0.3 - 0.5) can increase the diversity of translations and leads to better rescoring BLEU. However, the latent samples become noisy when a large temperature (1.0) is used.

4.5 Analysis of Translation Diversity

Following (Shen et al., 2019), we analyze the output diversity of FlowSeq. Shen et al. (2019) proposed pairwise-BLEU and BLEU computed in a leave-one-out manner to calibrate the diversity and quality of translation hypotheses. A lower pairwise-BLEU score implies a more diverse hypothesis set. And a higher BLEU score implies a better translation quality. We experiment on a subset of test set of WMT14-ENDE with ten references each sentence (Ott et al., 2018). In Fig. 6, we compare FlowSeq with other multi-hypothesis generation methods (ten hypotheses each sentence) to analyze how well the generation outputs of FlowSeq are in terms of diversity and quality. The right corner area of the figure indicates the ideal generations: high diversity and high quality. While FlowSeq still lags behind the autoregressive generations, by increasing the sampling temperature it provides a way of generating more diverse outputs while keeping the translation quality almost unchanged. More analysis of translation outputs and detailed results are provided in the Appendix D and E.

5 Conclusion

We propose FlowSeq, an efficient and effective model for non-autoregressive sequence generation by using generative flows. One potential direction for future work is to leverage iterative refinement techniques such as masked language models to further improve translation quality. Another exciting direction is to, theoretically and empirically, investigate the latent space in FlowSeq, hence providing deep insights of the model, even enhancing controllable text generation.

Acknowledgments

This work was supported in part by DARPA grant FA8750-18-2-0018 funded under the AIDA program and grant HR0011-15-C-0114 funded under the LORELEI program. Any opinions, findings, and conclusions expressed in this material are those of the authors and do not necessarily reflect the views of DARPA. The authors thank Amazon for their gift of AWS cloud credits and anonymous reviewers for their helpful suggestions.

References

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2015. Neural machine translation by jointly learning to align and translate. *In International Conference on Learning Representations (ICLR)*.

Samuel R Bowman, Luke Vilnis, Oriol Vinyals, Andrew M Dai, Rafal Jozefowicz, and Samy Bengio. 2015. Generating sentences from a continuous space. *arXiv preprint arXiv:1511.06349*.

Yuri Burda, Roger Grosse, and Ruslan Salakhutdinov. 2015. Importance weighted autoencoders. *arXiv preprint arXiv:1509.00519*.

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*.

Laurent Dinh, Jascha Sohl-Dickstein, and Samy Bengio. 2016. Density estimation using real nvp. *arXiv preprint arXiv:1605.08803*.

Marjan Ghazvininejad, Omer Levy, Yinhan Liu, and Luke Zettlemoyer. 2019. Constant-time machine translation with conditional masked language models. *arXiv preprint arXiv:1904.09324*.

Jiatao Gu, James Bradbury, Caiming Xiong, Victor OK Li, and Richard Socher. 2018. Non-autoregressive neural machine translation. *In International Conference on Learning Representations (ICLR)*.

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

Sergey Ioffe and Christian Szegedy. 2015. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International Conference on Machine Learning*, pages 448–456.

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

Diederik P Kingma, Tim Salimans, Rafal Jozefowicz, Xi Chen, Ilya Sutskever, and Max Welling. 2016. Improving variational inference with inverse autoregressive flow. *The 29th Conference on Neural Information Processing Systems*.

Durk P Kingma and Prafulla Dhariwal. 2018. Glow: Generative flow with invertible 1x1 convolutions. In *Advances in Neural Information Processing Systems*, pages 10215–10224.

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*, pages 1173–1182.

Jindřich Libovický and Jindřich Helcl. 2018. End-to-end non-autoregressive neural machine translation with connectionist temporal classification. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 3016–3021.

Xuezhe Ma and Eduard Hovy. 2019. Macow: Masked convolutional generative flow. *arXiv preprint arXiv:1902.04208*.

Xuezhe Ma, Zecong Hu, Jingzhou Liu, Nanyun Peng, Graham Neubig, and Eduard Hovy. 2018. Stack-pointer networks for dependency parsing. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1403–1414.

Xuezhe Ma, Chunting Zhou, and Eduard Hovy. 2019. Mae: Mutual posterior-divergence regularization for variational autoencoders. In *Proceedings of the 7th International Conference on Learning Representations (ICLR-2019)*, New Orleans, Louisiana, USA.

Oren Melamud, Jacob Goldberger, and Ido Dagan. 2016. context2vec: Learning generic context embedding with bidirectional LSTM. In *Proceedings of The 20th SIGNLL Conference on Computational Natural Language Learning*, pages 51–61, Berlin, Germany. Association for Computational Linguistics.

Myle Ott, Michael Auli, David Grangier, et al. 2018. Analyzing uncertainty in neural machine translation. In *International Conference on Machine Learning*, pages 3953–3962.

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. In *Proceedings of NAACL-HLT 2019: Demonstrations*.

George Papamakarios, Theo Pavlakou, and Iain Murray. 2017. Masked autoregressive flow for density estimation. In *Advances in Neural Information Processing Systems*, pages 2338–2347.

Ryan Prenger, Rafael Valle, and Bryan Catanzaro. 2019. Waveglow: A flow-based generative network for speech synthesis. In *ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 3617–3621. IEEE.

Danilo Jimenez Rezende and Shakir Mohamed. 2015. Variational inference with normalizing flows. In *Proceedings of the 32nd International Conference on International Conference on Machine Learning-Volume 37*, pages 1530–1538. JMLR. org.

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.

Tianxiao Shen, Myle Ott, Michael Auli, et al. 2019. Mixture models for diverse machine translation: Tricks of the trade. In *International Conference on Machine Learning*, pages 5719–5728.

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*.

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

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.

Oriol Vinyals, Alexander Toshev, Samy Bengio, and Dumitru Erhan. 2015. Show and tell: A neural image caption generator. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3156–3164.

Martin J Wainwright, Michael I Jordan, et al. 2008. Graphical models, exponential families, and variational inference. *Foundations and Trends® in Machine Learning*, 1(1–2):1–305.

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*.

Zachary Ziegler and Alexander Rush. 2019. Latent normalizing flows for discrete sequences. In *International Conference on Machine Learning*, pages 7673–7682.