Non-Autoregressive Machine Translation with Latent Alignments

Chitwan Saharia[*†]
Google Research, Brain Team
sahariac@google.com

William Chan[*]
Google Research, Brain Team
williamchan@google.com

Saurabh Saxena
Google Research, Brain Team
srbs@google.com

Mohammad Norouzi
Google Research, Brain Team
mnorouzi@google.com

Abstract

This paper investigates two latent alignment models for non-autoregressive machine translation, namely CTC and Imputer. CTC generates outputs in a single step, makes strong conditional independence assumptions about output variables, and marginalizes out latent alignments using dynamic programming. Imputer generates outputs in a constant number of steps, and approximately marginalizes out possible generation orders and latent alignments for training. These models are simpler than existing non-autoregressive methods, since they do not require output length prediction as a pre-process. In addition, our architecture is simpler than typical encoder-decoder architectures, since input-output cross attention is not used. On the competitive WMT'14 En→De task, our CTC model achieves 25.7 BLEU with a single generation step, while Imputer achieves 27.5 BLEU with 2 generation steps, and 28.0 BLEU with 4 generation steps. This compares favourably to the baseline autoregressive Transformer with 27.8 BLEU.

1 Introduction

Non-autoregressive neural machine translation (Gu et al., 2018) aims to enable parallel generation of output tokens without sacrificing translation quality. There has been a surge of recent interest in this family of efficient decoding models, resulting in the development of iterative refinement (Lee et al., 2018), CTC models (Libovicky and Helcl, 2018), insertion-based methods (Stern et al., 2019; Chan et al., 2019b), edit-based methods (Gu et al., 2019; Ruis et al., 2019), masked language models (Ghazvininejad et al., 2019, 2020b), and normalizing flow models (Ma et al., 2019). Some of these methods generate the output tokens in a constant number of steps (Gu et al., 2018; Libovicky and Helcl, 2018; Lee et al., 2018; Ghazvininejad et al., 2019, 2020b), while others require a logarithmic number of generation steps (Stern et al., 2019; Chan et al., 2019b,a; Li and Chan, 2019).

Recent progress has decreased the gap between autoregressive and non-autoregressive models' translation scores. However, non-autoregressive models often suffer from two main limitations:

1. First, most non-autoregressive models assume that the output tokens are conditionally independent given the input. This leads to the weakness of such models in generating multi-modal outputs (Gu et al., 2018), and materializes in the form of *token repetitions* in the decoded outputs. Addressing this limitation generally involves stochastic search algorithms like noisy parallel decoding (Gu et al., 2018), iterative decoding (Ghazvininejad et al., 2019, 2020b), or simple but less effective heuristic methods such as collapsing repetitions (Lee et al., 2018).
2. The second limitation of many prior non-autoregressive models is the requirement of *output length prediction* as a pre-process. Autoregressive models have the ability to dynamically adjust the output sequence length by emitting an <END> token at any generation step to stop. Many non-autoregressive models often require a fixed length decoder, thus they train a separate target length prediction module, and at inference time, first predict and condition on the target length, and then generate the output tokens (Gu et al., 2018). Since, the model needs to commit to a fixed predicted length, which cannot be changed dynamically, it is often required to use multiple length candidates and re-score them to produce the final translation (Ghazvininejad et al., 2019, 2020b).

[*] Equal contribution.
[†] Work done as part of the Google AI Residency.

This paper addresses the limitations of existing non-autoregressive machine translation models by using *latent alignment models*. Latent alignment models utilize a sequence of discrete latent alignment variables to monotonically align the contextualized embeddings of input tokens and output tokens. Such models use dynamic programming to marginalize out the alignment variables during training. Two instances of latent alignment models, used for speech recognition, include Connectionist Temporal Classification (CTC) (Graves et al., 2006, 2013; Graves and Jaitly, 2014) and Imputer (Chan et al., 2020). We find that these families of models, when properly adapted for non-autoregressive machine translation, advance the state-of-the-art on WMT'14 En↔De and WMT'16 En↔Ro.

The main contributions of this paper include:

1. We adapt latent alignment models based on CTC and Imputer for non-autoregressive machine translation.
2. We achieve a new state-of-the-art 25.8 BLEU on WMT'14 En→De for single step non-autoregressive machine translation.
3. We achieve 27.5 BLEU with 2 step generation, 28.0 BLEU with 4 step generation, and 28.2 BLEU with 8 step generation for WMT'14 En→De, setting a new state-of-the-art for non-autoregressive machine translation with a constant number of generation steps.

2 Latent Alignment Models

We will first begin by defining an *alignment*. Our definition of alignment has been greatly influenced by and conforms to the prior work in the CTC literature (Graves et al., 2006, 2013; Graves and Jaitly, 2014). We use the term alignment as defined in these works, and our usage of the term *alignment* should not be confused with the word alignment problem (Manning et al., 1999; Dyer et al., 2013). Loosely speaking, an alignment is a mapping between a source sequence and a target sequence; an alignment can be constructed by infilling a special "blank token" over the target sequence to match the source sequence length. The alignment is the same length as the source sequence, and collapsing the alignment(s' blank tokens) will recover the target sequence. We will more formally define an alignment in the next paragraph.

Let x be our source sequence and let y be our target sequence, where $y_i \in \mathcal{V}$ and $\mathcal{V}$ is our target vocabulary. We make two assumptions: 1) there exists a monotonic mapping between the model alignment predictions and the target sequence, and 2) the source sequence is at least as long as the target sequence i.e., $|x| \geq |y|$. We define an alignment a between x and y as a discrete sequence where $a_i \in \mathcal{V} \cup \{\text{"\_"}\}$ and $|a| = |x|$. "_" is a special "blank" token which is removed to recover the target sequence y, this blank token conforms with definition from prior CTC literature (Graves et al., 2006). We define a function $\beta(y)$ that returns all possible alignments for a sequence y of a particular length $|x|$. We also define the collapsing function $\beta^{-1}(a)$ where, $\beta^{-1}(a) = y$ if $a \in \beta(y)$. In practice, it is useful to define $\beta^{-1}(a)$ as a function which collapses all consecutive repeated tokens, then removes all blank tokens. This formulation follows the CTC alignment definition exactly (Graves et al., 2006). Here is a concrete example, given a source sequence x of length 10, and a target sequence $y = (A, A, B, C, D)$, one possible alignment a can be $(\_, A, A, \_, A, B, B, C, \_, D)$.

The log-likelihood of the target sequence is recovered by marginalizing out the latent alignments:

$$\log p_\theta(y|x) = \log \sum_{a \in \beta(y)} p_\theta(a|x) \tag{1}$$

Direct summation over Equation 1 is typically intractable, since there are a combinatorial number of alignment terms. In the next two subsections, we will briefly describe two variants of latent alignment models which leverage dynamic programming to tractably compute the log-likelihood, Connectionist Temporal Classification (CTC) (Graves et al., 2006) and Imputer (Chan et al., 2020).

2.1 Connectionist Temporal Classification

Connectionist Temporal Classification (CTC) (Graves et al., 2006, 2013; Graves and Jaitly, 2014) models the alignment distribution with strong conditional independence assumptions between tokens:

$$p_\theta(a|x) = \prod_i p(a_i|x;\theta) \tag{2}$$

Leveraging this strong conditional independence assumption enables CTC to use an efficient dynamic programming algorithm to exactly

marginalize out the latent alignments:

$$\log p_\theta(y|x) = \log \sum_{a \in \beta(y)} \prod_i p(a_i|x;\theta) \quad (3)$$

This allows us to tractably compute the log-likelihood (and its gradient). We refer the reader to Graves et al. (2006) for the exact details of the dynamic programming algorithm. During inference, CTC generates the alignment distribution in parallel with a single generation step, the output sequence can then be recovered by greedy or beam search (Graves et al., 2006).

2.2 Imputer

The CTC model makes strong conditional independence assumption between alignment token predictions. This assumption licenses CTC to generate the entire alignment in parallel, with a single generation step independent of number of source or target tokens. However, the strong conditional independence assumption limits its capacity to model complex multi-modal distributions. On the other hand, autoregressive models are capable of modelling such complex multi-modalities with the chain rule factorization, but requires n decoding steps to generate n tokens during inference. Imputer (Chan et al., 2020) aims to address these limitations.

Imputer is an iterative generative model needing only a constant number of generation steps for inference. It makes conditional independence assumptions within a generation step to achieve parallel generation, and models conditional dependencies across generation steps. This approach has been applied successfully in speech recognition (Chan et al., 2020), matching autoregressive models with only a constant number of generation steps. Imputer models the distribution of alignments $p_\theta(a|x)$ as:

$$p_\theta(a|x) = \sum_{\tilde{a} \in \gamma(a)} p_\theta(a|\tilde{a}, x) p(\tilde{a}|x) \quad (4)$$

where $\tilde{a}$ is a (partially masked out) alignment, and $\gamma(a)$ is the set of all possible masking permutations of a. Equation 4 marginalizes over all possible alignments between the input and output sequences, and all possible generation orders. Imputer models the next alignment a conditioned on the previous alignment $\tilde{a}$:

$$p_\theta(a|\tilde{a}, x) = \prod_i p(a_i|\tilde{a}, x; \theta) \quad (5)$$

The key insight to Imputer is that we can construct a log-likelihood lower-bound:

$$\log p_\theta(y|x) \geq \mathbb{E}_{a \sim \beta(y)} \left[\mathbb{E}_{\tilde{a} \sim \gamma(a)} \left[\log \sum_{a' \in \beta'(\tilde{a}, a)} p_\theta(a'|\tilde{a}, x) \right] \right] \quad (6)$$

where $a' \sim \beta'(\tilde{a}, a)$ captures all possible alignments a' consistent with $(\tilde{a}, a)$ (Chan et al., 2020). This equation can be solved efficiently via dynamic programming (Chan et al., 2020). This formulation licenses Imputer with an iterative generation process. Tokens are generated independently (and in parallel) within a generation step, but are conditioned on the partially predicted alignment $\tilde{a}$ of the last iteration (unlike CTC). In practice, Imputer uses a constant number of decoding iterations independent of the sequence length (Chan et al., 2020).

2.3 Advantages

Both CTC and Imputer have seen much success in tasks like speech recognition (Graves and Jaitly, 2014; Chan et al., 2020). However, to the best of our knowledge, these latent alignment models have not been widely applied to machine translation, with the exception of Libovicky and Helcl (2018). These latent alignment models hold two key advantages over prior non-autoregressive machine translation work (Gu et al., 2018; Ghazvininejad et al., 2019), namely: the *token repetition problem* and the *target length prediction problem*. We will now discuss them briefly.

2.3.1 Token Repetition Problem

Non-autoregressive sequence models make conditional independence assumption between token predictions. This licenses them to parallel token generation during inference, however it makes it difficult to model complex multi-modal distributions. This is especially true for single step generation models which make strong conditional independence assumptions. During inference, this conditional independent generation often results in the *token repetition problem*, where tokens are erroneously repeated in the output sequence.

This issue has been discussed extensively in prior works (Gu et al., 2018; Lee et al., 2018; Ghazvininejad et al., 2019) in the context of machine translation, and has been shown to have a

negative impact on performance. To handle these repetitions, Gu et al. (2018) used Noisy Parallel Decoding, wherein they sample a large number of translation hypotheses and use an autoregressive teacher to re-score them in order to implicitly penalize translations with more erroneous repetitions. Lee et al. (2018) adopted a simple but less effective heuristic of simply removing all consecutive repetitions from the predicted target sequence. Ghazvininejad et al. (2019) hypothesized that iterative decoding can help remove repetitions by allowing the model to condition on parts of the input, thus collapsing the multi-modal distribution into a sharper uni-modal distribution. They empirically show that the first few decoding iterations are crucial for removing repetitions resulting in sharp increase in performance.

Like other non-autoregressive models, latent alignment models also perform conditionally independent generation, and hence face the issue of token repetitions. Although, they differ from the other models in that they do not generate the target sequence directly. Rather, the inference process involves generation of the target alignment, followed by collapsing the generated alignment into the target sequence using the collapsing function β^{-1}. Recall by construction, β^{-1} collapses repeated tokens (Graves et al., 2006), this inference process enables these models to implicitly handle erroneous repetitions by naturally collapsing them. In particular, for single step decoding, we show that our CTC based model implicitly removes most of the repetitions during collapsing the alignment into target sequence, resulting in a significant improvement in translation quality over prior single step generation models. Furthermore, we show that Imputer requires just 4 decoding iterations to achieve state-of-the-art translation score on WMT14 En→De, in comparison to 10 iterations used by Mask-Predict (Ghazvininejad et al., 2019).

2.3.2 Target Length Prediction Problem

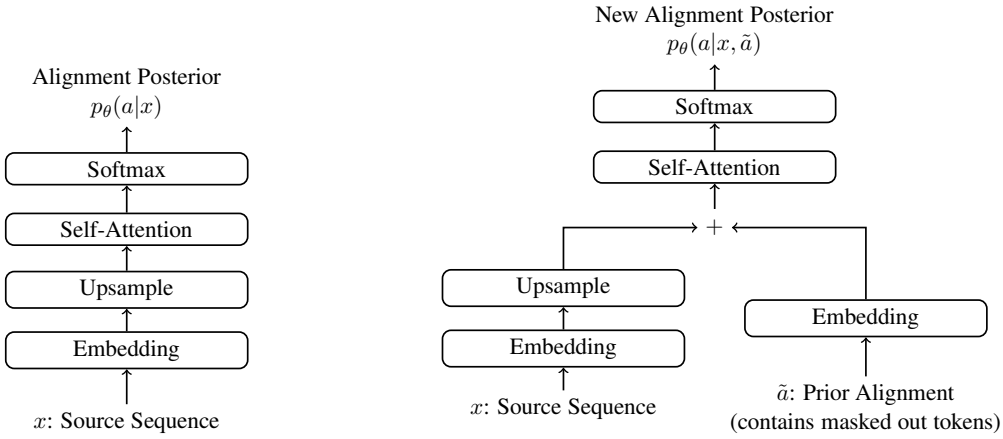
Many prior non-autoregressive models (Gu et al., 2018; Ghazvininejad et al., 2019) first predict the target length, then conditioned on the target length predict the target sequence. This is needed because these architectures utilize an encoder-decoder formulation, and the decoder requires a fixed canvas size to work with. The length is fixed, and it cannot be changed dynamically by the model during decoding. Due to this lack of flexibility, during inference, one typically samples multiple length candidates, and performs decoding for each length followed by re-ranking them to get a final translation. This not only requires tuning of a new hyperparameter for determining the number of length candidates to sample during inference, but also entails considerable amount of extra inference computation.

Our latent alignment based models do not require target length prediction, but rather implicitly determine the target sequence length through the alignment. This is possible since the alignment is of the same length as the source sequence, thus eliminating the requirement of predicting target length in advance during inference. The caveat is that we can not generate a target sequence longer than the source sequence, which we address in Section 3. Libovicky and Helcl (2018), which also applied CTC to machine translation, made a similar argument, and we further extend this to Imputer. Our approach simplifies the architecture and decoding process, avoiding a need to build a target length prediction model and searching over it during inference.

3 Adapting Latent Alignment Models for Machine Translation

In this section, we will now discuss adapting the latent alignment models to machine translation. In Section 2, we identified two assumptions made between the source sequence x and the target sequence y: 1) there exists a monotonic mapping between the model alignment predictions and the target sequence, and 2) the length of target sequence is less than or equal to length of source sequence, i.e. $|y| \leq |x|$. We will now address these issues to adapt our models for machine translation.

Monotonic Assumption. A monotonic structure between model alignment predictions and the target sequence is desired for the dynamic programming algorithm to marginalize out the latent alignments in Equation 1. Unlike tasks such as speech recognition, there generally does not naturally exist a monotonic relationship between the model alignment predictions and the target sequence, (e.g., speech-to-text is inherently local, where as there is typically global word reordering in machine translation). We hypothesize that if we use a powerful deep neural network like the Transformer (Vaswani et al., 2017), the Transformer will have sufficient computational capacity



(a) CTC (b) Imputer

Figure 1: Visualization of the CTC (a) and Imputer (b) architecture for non-autoregressive machine translation.

to learn to reorder the contextual embeddings such that it is approximately monotonic with the target sequence. Libovicky and Helcl (2018) also made a similar assumption.

Length Assumption. By construction, our alignments are the same length as the source sequence, and consequently we can not generate a target sequence longer than the source sequence. This is not a problem for speech recognition, since the source sequence is generally much longer than the target sequence; however this is prohibitively restrictive for machine translation. This issue was also discussed in Libovicky and Helcl (2018), and they proposed a simple solution of up-sampling the source sequence to s times the original length. Choosing a sufficient canvas scale s, we can ensure the alignment is sufficiently long enough to model the target sequence across our training distribution. We use a very similar up-sampling method applied on the embedding matrix of the source sequence. Given a source sequence embedding $x \in \mathbb{R}^{|x| \times d}$ with d-dimension and length $|x|$, we simply up-sample $x' \in \mathbb{R}^{s \cdot |x| \times d}$ via an affine transform.

3.1 Model Architecture

Our neural architecture is simply a stack of self-attention layers (Vaswani et al., 2017). The source sequence is upsampled (to handle longer target sequences as described above). In the Imputer architecture, the input to our self-attention stack is simply the superpositioning of the upsampled source and the previous alignment. Our work differs from prior method, 1) our unified architecture does not have separate encoder decoders which require cross-attention mechanisms, 2) our architecture is bidirectional, and does not rely on casuality masks. Figure 1 visualizes our architecture.

4 Related Work

There has been significant prior work on non-autoregressive iterative methods for machine translation (Gu et al., 2018), some of which are: iterative refinement (Lee et al., 2018), insertion-based methods (Stern et al., 2019; Chan et al., 2019a; Li and Chan, 2019), and conditional masked language models (Ghazvininejad et al., 2019, 2020b). Like insertion-based models (Stern et al., 2019; Chan et al., 2019c), our work does not commit to a fixed target length; insertion-based models can dynamically grow the canvas size, whereas our work which relies on a latent alignment can only generate a target sequence up to a fixed maximum predetermined length. Compared to conditional masked languages models (Ghazvininejad et al., 2019, 2020b), key differences are: 1) our models do not require target length prediction, and 2) we eschew the encoder-decoder neural architecture formulation, but rather rely on the single simple decoder architecture. KERMIT (Chan et al., 2019b,a) also has a similar neural architecture as us, they also eschew the conventional encoder-decoder architecture and have a unified architecture. Our work relies on the superpositioning of the input and output sequences via the latent alignment, whereas KERMIT relies on concatenation to process the input and output sequences. Their work is also more focused on generative $p(x, y)$ modelling, whereas our work is focused on conditional modelling $p(y|x)$.

Input: Ein weiterer, besonders wichtiger Faktor sei die Vernetzung von Hochschulen und Unternehmen.

Output: Another particularly important factor is the networking of universities and businesses.

Imputer Decoding:

Another _ _ _ _ _ particularly _ important _ factor _ is _ the _ _ _ networking _ of _ universities and _ _ businesses _ .
Another _ _ _ _ _ particularly _ important _ factor _ is _ the _ _ _ networking _ of _ universities and _ _ businesses _ .
Another _ _ _ _ _ particularly _ important _ factor _ is _ the _ _ _ networking _ of _ universities and _ _ businesses _ .
Another _ _ _ _ _ particularly _ important _ factor _ is _ the _ _ _ networking _ of _ universities and _ _ businesses _ .
Another _ _ _ _ _ particularly _ important _ factor _ is _ the _ _ _ networking _ of _ universities and _ _ businesses _ .
Another _ _ _ _ _ particularly _ important _ factor _ is _ the _ _ _ networking _ of _ universities and _ _ businesses _ .
Another _ _ _ _ _ particularly _ important _ factor _ is _ the _ _ _ networking _ of _ universities and _ _ businesses _ .
Another _ _ _ _ _ particularly _ important _ factor _ is _ the _ _ _ networking _ of _ universities and _ _ businesses _ .

Figure 2: Example of top-k decoding using Imputer with 8 decoding steps. For a sentence of length N, the model imputes $\lceil \frac{N}{8} \rceil$ tokens at every decoding step. In each row, blue underlined tokens are the ones being imputed. Tokens that are not generated yet are colored gray.

Our CTC work is closely related to and inspired heavily by Libovicky and Helcl (2018), which applied CTC single step generation models. The key difference is that our work used data distillation for training, and we find that distillation provides significant boost in performance for our CTC models.

Finally, our work is closely related to the concurrent work of Ghazvininejad et al. (2020a) on AXE CMLM. Similar to our work, they also assume a latent alignment and use dynamic programming for learning. Their work focused on single step generation and demonstrated strong results, while we apply our models to both single step and iterative generation.

5 Experiments

Dataset. We perform experiments on WMT'14 En↔De, using newstest2013 as the development set, and report newstest2014 as the test set. We also report our performance on WMT'16 En-Ro. We use SentencePiece (Kudo and Richardson, 2018) to generate a shared subword vocabulary. We evaluate the performance of our models with BLEU (Papineni et al., 2002).

Hyperparameters. We follow the base Transformer (Vaswani et al., 2017) for our experiments. However, since our architecture does not contain an encoder, we simply double the number of layers in our decoder to maintain the same number of parameters. Our models consists of 12 self-attention layers, with 512 hidden size, 2048 filter size, and 8 attention heads per layer. We use 0.1 dropout for regularization. We batch sequences of approximately same lengths together, with approximately 2048 tokens per batch. We use Adam optimizer (Kingma and Ba, 2015) with $\beta = (0.9, 0.997)$ and $\epsilon = 10^{-9}$. The learning rate warms up to 10^{-3} in the first 10k steps, and then decays with the inverse square root schedule following the Tensor2Tensor implementation (Vaswani et al., 2018). We train all our models for 2M steps. We train the Imputer using CTC loss (all masked prior alignment) for 1M steps, followed by bernoulli masking policy (Chan et al., 2020) for next 1M steps. We average the 5 checkpoints with the best performance on the development set to get the final model. For Imputer, we use top-k decoding during inference. We use canvas scale $s = 2$ for all our experiments, meaning we upsample the source sequence by a factor of 2.

Distillation. We follow prior work (Gu et al., 2018; Lee et al., 2018; Stern et al., 2019; Ghazvininejad et al., 2019) and data distilled from an autoregressive teacher for training our models. We use autoregressive base Transformers for generating distilled data. For iterative generation, we also report performance of Imputer model trained on data distilled from autoregressive big Transformers to compare our results with (Ghazvininejad et al., 2019, 2020b) which distilled from a big Transformer. For WMT'16 En-Ro, we use the distilled dataset provided by (Ghazvininejad et al., 2019) [1]. We will also analyze the impact of distillation on the performance of our models in Section 6.2.

[1] https://github.com/facebookresearch/Mask-Predict

Table 1: BLEU comparison for various single step generation models. Our simple CTC model is able to outperform all prior single step generation models. †The main difference between our CTC model and prior work (Libovicky and Helcl, 2018) is that we use data distillation.

Method	Iterations	WMT'14 En→De	WMT'14 De→En	WMT'16 En→Ro	WMT'16 Ro→En
Non-Autoregressive					
Iterative Refinement (Lee et al., 2018)	1	13.9	16.7	24.5	25.7
NAT with Fertility (Gu et al., 2018)	1	17.7	21.5	27.3	29.1
CTC† (Libovicky and Helcl, 2018)	1	17.7	19.8	19.9	24.7
Mask-Predict (Ghazvininejad et al., 2019)	1	18.0	19.3	27.3	28.2
SMART (Ghazvininejad et al., 2020b)	1	18.6	23.8	-	-
Auxiliary Regularization (Wang et al., 2019)	1	20.7	24.8	-	-
Bag-of-ngrams Loss (Shao et al., 2020)	1	20.9	24.6	28.3	29.3
Hint-based Training (Li et al., 2019)	1	21.1	25.2	-	-
FlowSeq (Ma et al., 2019)	1	21.5	26.2	29.3	30.4
Bigram CRF (Sun et al., 2019)	1	23.4	27.2	-	-
AXE CMLM (Ghazvininejad et al., 2020a)	1	23.5	27.9	30.8	31.5
Our Work					
CTC†	1	25.7	28.1	32.2	31.6
Imputer	1	**25.8**	**28.4**	**32.3**	**31.7**

Table 2: WMT'14 En↔De repeated token percentage comparison for single step generation models.

Model	En→De	De→En
Gold Test Set	0.04%	0.02%
Mask-Predict (Ghazvininejad et al., 2019)	16.72%	12.31%
AXE CMLM (Ghazvininejad et al., 2020a)	1.41%	1.03%
CTC (Our Work)	**0.17%**	**0.23%**

5.1 Single Step Decoding

We first analyse the performance of latent alignment models in single step decoding. CTC makes full conditional independence assumption allowing generation of entire target sequence in a single step. We can also perform non-autoregressive single step generation with Imputer by imputing all the tokens at once. Table 1 summarizes the performance of our models. We compare our model's performance with other non-autoregressive single step generation models. Our CTC model achieves 25.7 BLEU, and our Imputer model achieves 25.8 BLEU for WMT'14 En→De. We also find our single step generation models to outperform the autoregressive GNMT (Wu et al., 2016) on En→De with 24.6 BLEU. To the best of our knowledge, our CTC and Imputer models outperform all prior work on single step generation on WMT'14 En↔De and WMT'16 En↔Ro.

We compare the repetition rate of our CTC model with single step Mask-Predict (Ghazvininejad et al., 2019) and the concurrent work AXE CMLM (Ghazvininejad et al., 2020a) in Table 2. We also report the percentage of repetitions in the original test set for reference. We observe a significantly lower rate of token repetitions in our CTC model compared to both the models. This empirical observation supports our hypothesis that β^{-1} helps remove spurious token repetitions.

5.2 Iterative Decoding

We now analyze the performance of Imputer. Imputer uses a constant number of decoding iterations independent of sequence length. We compare our performance with other sub-linear non-autoregressive models, ranging from models requiring logarithmic to constant number of decoding iterations. Table 3 summarizes the results of Imputer model.

Our Imputer model using 8 decoding iterations achieves 28.2 BLEU on En→De, slightly outperforming the autoregressive Transformer of 27.8 BLEU. On De→En, we achieve 31.3 BLEU, on par with the autoregressive Transformer model. Similarly, on En↔Ro, Imputer matches the performance of the autoregressive teacher using just 4 decoding iterations. We also observe robustness of our Imputer model when we reduce the number of decoding iterations from 8 to 2. Using just 2 iterations, we obtain 27.5 BLEU on En→De and 30.2 BLEU on De→En. These results were trained with distillation from a big Transformer model. However, even when we distill from the base Transformer as shown in Table 5

Table 3: BLEU comparison for various autoregressive and non-autoregressive models. Imputer is able to match the autoregressive Transformer baseline with just 4 generation steps.

Method	Iterations	WMT'14 En→De	WMT'14 De→En	WMT'16 En→Ro	WMT'16 Ro→En
Autoregressive					
Base Transformer[2]	n	27.8	31.2	34.3	34.0
Non-Autoregressive					
Insertion Transformer (Stern et al., 2019)	$\approx \log_2 n$	27.4	-	-	-
KERMIT (Chan et al., 2019b)	$\approx \log_2 n$	27.8	30.7	-	-
Iterative Refinement (Lee et al., 2018)	10	21.6	25.5	29.3	30.2
Mask-Predict (Ghazvininejad et al., 2019)	4	25.9	29.9	32.5	33.2
	10	27.0	30.5	33.1	33.3
SMART (Ghazvininejad et al., 2020b)	4	27.0	30.9	-	-
	10	27.7	**31.3**	-	-
Our Work					
Imputer	2	27.5	30.2	33.7	33.4
	4	28.0	31.0	34.3	34.0
	8	**28.2**	**31.3**	**34.4**	**34.1**

, Imputer still performs on par with the autoregressive Transformer achieving 27.9 and 31.1 BLEU on En→De and De→En respectively.

6 Analysis

In this section, we perform further analysis on our latent alignment models. We analyze the impact of the number of decoding iterations on Imputer, and the impact of distillation on our models.

6.1 Impact of Number of Decoding Iterations

The number of decoding iterations is an important hyperparameter in iterative models, providing a tunable trade-off between quality and inference speed. The ideal parallel decoding model should be robust to the number of decoding iterations, i.e. reducing the number of iterations should have minimal impact on performance. To analyse this capability of our Imputer model, we study the impact of number of decoding iterations vs. BLEU. We use the Imputer trained with data distilled from the autoregressive base Transformer teacher for this analysis. Imputer controls the number of decoding iterations through the top-k hyperparameter, which imputes k tokens per step. On one end, imputing all the tokens ($k = \infty$) in one step results in single step decoding, while on the other end, imputing 1 token per step ($k = 1$) results in linear autoregressive decoding (but not necessarily left-to-right).

[2]En↔De base Transformer is our work, and En↔Ro is from (Ghazvininejad et al., 2019).

Figure 3 shows the BLEU score vs T for WMT'14 En→De test set, where T is the number of decoding iterations. As expected, the performance consistently increases with increase in T. We find that Imputer is robust to T, sacrificing just 0.6 BLEU points when reducing T from 8 to 2. We can match the performance of its autoregressive teacher using just 4 decoding iterations. Interestingly, the performance keeps increasing consistently beyond 8 iterations, and even outperforming the autoregressive teacher itself slightly. In the extreme case of autoregressive $O(n)$ decoding, we obtain 28.3 BLEU score, exceeding the teacher's performance by 0.5 BLEU points.

Figure 4 also depicts impact of number of decoding iterations bucketed by the target sequence length N. We use the `compare-mt` (Neubig et al., 2019) package to bucket test set examples based on target sentence length, and compute BLEU score for each bucket using different number of decoding iterations. Increase in the number of decoding iterations provides consistent gain across all buckets.

6.2 Impact of Distillation

We analyze the impact of distillation on our models by comparing them to original training data versus training data from a base Transformer teacher on the WMT'14 En→De dataset. Table 4 summarizes the results. In all cases, models trained with the distilled data perform significantly better than the model trained with the original data. We observe that the performance gap is

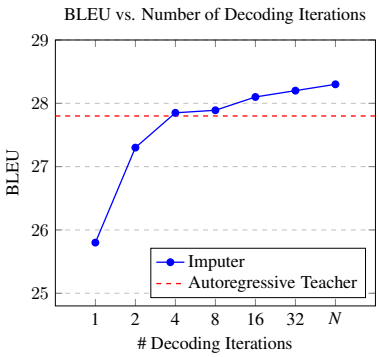


Figure 3: WMT'14 En→De BLEU comparison for different number of decoding iterations for Imputer.

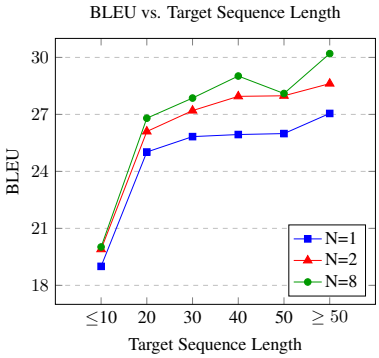


Figure 4: WMT'14 En→De BLEU comparison for sentences binned by target sequence length for Imputer; N is the number of decoding iterations.

largest in the case of CTC model, and decreases with increase in number of decoding iterations. This is consistent to prior work finding distillation to improve model quality (Gu et al., 2018; Zhou et al., 2020).

7 Conclusion

In this paper, we investigated two latent alignments models, CTC and Imputer, for non-autoregressive machine translation. CTC is a single step generation model, while Imputer is an iterative generative model requiring only a constant number of generation steps. Our models rely on dynamic programming to marginalize out the latent alignment. Unlike many prior works, our models do not need to perform target length prediction, and our models use a simplified neural architecture without the need of cross-attention mechanism found in many prior encoder-decoder architectures. Our CTC model achieves 25.7 BLEU with single step generation, while our Imputer achieves 27.5 BLEU with 2-step generation, 28.0 BLEU with 4-step generation, and 28.2 BLEU with 8-step generation; this compares favourably to the autoregressive Transformer of 27.8 BLEU on the competitive WMT'14 En→De task.

Table 4: WMT'14 En→De BLEU comparison on the impact of distillation.

Method	**Iterations**	**Original**	**Distillation**
CTC	1	15.6	25.4
Imputer	4	24.7	27.9
	8	25.0	27.9

Table 5: WMT'14 En-De BLEU comparison for distillation base vs big Transformer, and number of decoding iterations.

Model	Iterations	**En→De**	**De→En**
Transformer *(Base)*	n	27.8	31.2
Transformer *(Big)*	n	29.5	32.2
Imputer *(Base Distillation)*	2	27.3	30.3
	4	27.9	30.9
	8	27.9	31.1
	n	28.3	31.2
Imputer *(Big Distillation)*	2	27.5	30.2
	4	28.0	31.0
	8	28.2	31.3
	n	28.4	31.4

Acknowledgments

We give thanks to Colin Cherry, Geoffrey Hinton, George Foster, Jakob Uszkoreit, Jamie Kiros, Julia Kreutzer, Samy Bengio, and Sara Sabour for research discussions and technical assistance.

References

Harris Chan, Jamie Kiros, and William Chan. 2019a. Multilingual KERMIT: Its Not Easy Being Generative. In *NeurIPS: Workshop on Perception as Generative Reasoning*.

William Chan, Nikita Kitaev, Kelvin Guu, Mitchell Stern, and Jakob Uszkoreit. 2019b. KERMIT: Generative Insertion-Based Modeling for Sequences. In *arXiv*.

William Chan, Chitwan Saharia, Geoffrey Hinton, Mohammad Norouzi, and Navdeep Jaitly. 2020. Imputer: Sequence modelling via imputation and dynamic programming. In *arXiv*.

William Chan, Mitchell Stern, Jamie Kiros, and Jakob Uszkoreit. 2019c. An Empirical Study of Generation Order for Machine Translation. In *arXiv*.

Chris Dyer, Victor Chahuneau, and Noah Smith. 2013. A Simple, Fast, and Effective Reparameterization of IBM Model 2. In *NAACL*.

Marjan Ghazvininejad, Vladimir Karpukhin, Luke Zettlemoyer, and Omer Levy. 2020a. Aligned Cross Entropy for Non-Autoregressive Machine Translation. In *arXiv*.

Marjan Ghazvininejad, Omer Levy, Yinhan Liu, and Luke Zettlemoyer. 2019. Mask-Predict: Parallel Decoding of Conditional Masked Language Models. In *EMNLP*.

Marjan Ghazvininejad, Omer Levy, and Luke Zettlemoyer. 2020b. Semi-Autoregressive Training Improves Mask-Predict Decoding. In *arXiv*.

Alex Graves, Santiago Fernandez, Faustino Gomez, and Jurgen Schmidhuber. 2006. Connectionist Temporal Classification: Labelling Unsegmented Sequence Data with Recurrent Neural Networks. In *ICML*.

Alex Graves and Navdeep Jaitly. 2014. Towards End-to-End Speech Recognition with Recurrent Neural Networks. In *ICML*.

Alex Graves, Abdel-rahman Mohamed, and Geoffrey Hinton. 2013. Speech Recognition with Deep Recurrent Neural Networks. In *ICASSP*.

Jiatao Gu, James Bradbury, Caiming Xiong, Victor O.K. Li, and Richard Socher. 2018. Non-Autoregressive Neural Machine Translation. In *ICLR*.

Jiatao Gu, Changhan Wang, and Jake Zhao. 2019. Levenshtein Transformer. In *NeurIPS*.

Diederik Kingma and Jimmy Ba. 2015. Adam: A Method for Stochastic Optimization. In *ICLR*.

Taku Kudo and John Richardson. 2018. SentencePiece: A simple and language independent subword tokenizer and detokenizer for Neural Text Processing. In *EMNLP*.

Jason Lee, Elman Mansimov, and Kyunghyun Cho. 2018. Deterministic Non-Autoregressive Neural Sequence Modeling by Iterative Refinement. In *EMNLP*.

Lala Li and William Chan. 2019. Big Bidirectional Insertion Representations for Documents. In *EMNLP: Workshop of Neural Generation and Translation*.

Zhuohan Li, Zi Lin, Di He, Fei Tian, Tao Qin, Liwei Wang, and Tie-Yan Liu. 2019. Hint-Based Training for Non-Autoregressive Machine Translation . In *EMNLP*.

Jindrich Libovicky and Jindrich Helcl. 2018. End-to-End Non-Autoregressive Neural Machine Translation with Connectionist Temporal Classification. In *EMNLP*.

Xuezhe Ma, Chunting Zhou, Xian Li, Graham Neubig, and Eduard Hovy. 2019. FlowSeq: Non-Autoregressive Conditional Sequence Generation with Generative Flow. In *EMNLP*.

Christopher D Manning, Christopher D Manning, and Hinrich Schütze. 1999. *Foundations of statistical natural language processing*. MIT press.

Graham Neubig, Zi-Yi Dou, Junjie Hu, Paul Michel, Danish Pruthi, Xinyi Wang, and John Wieting. 2019. compare-mt: A Tool for Holistic Comparison of Language Generation Systems. *CoRR*.

Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. BLEU: a Method for Automatic Evaluation of Machine Translation. In *ACL*.

Laura Ruis, Mitchell Stern, Julia Proskurnia, and William Chan. 2019. Insertion-Deletion Transformer. In *EMNLP: Workshop of Neural Generation and Translation*.

Chenze Shao, Jinchao Zhang, Yang Feng, Fandong Meng, and Jie Zhou. 2020. Minimizing the Bag-of-Ngrams Difference for Non-Autoregressive Neural Machine Translation. In *AAAI*.

Mitchell Stern, William Chan, Jamie Kiros, and Jakob Uszkoreit. 2019. Insertion Transformer: Flexible Sequence Generation via Insertion Operations. In *ICML*.

Zhiqing Sun, Zhuohan Li, Haoqing Wang, Di He, Zi Lin, and Zhihong Deng. 2019. Fast Structured Decoding for Sequence Models. In *NeurIPS*.

Ashish Vaswani, Samy Bengio, Eugene Brevdo, Francois Chollet, Aidan N. Gomez, Stephan Gouws, Llion Jones, Łukasz Kaiser, Nal Kalchbrenner, Niki Parmar, Ryan Sepassi, Noam Shazeer, and Jakob Uszkoreit. 2018. Tensor2Tensor for Neural Machine Translation. In *AMTA*.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention Is All You Need. In *NIPS*.

Yiren Wang, Fei Tian, Di He, Tao Qin, ChengXiang Zhai, and Tie-Yan Liu. 2019. Non-Autoregressive Machine Translation with Auxiliary Regularization. In *AAAI*.

Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V. Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, Jeff Klingner, Apurva Shah, Melvin Johnson, Xiaobing Liu, ukasz Kaiser, Stephan Gouws, Yoshikiyo Kato, Taku Kudo, Hideto Kazawa, Keith Stevens, George Kurian, Nishant Patil, Wei Wang, Cliff Young, Jason Smith, Jason Riesa, Alex Rudnick, Oriol Vinyals, Greg Corrado, Macduff Hughes, and Jeffrey Dean. 2016. Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation. In *arXiv*.

Chunting Zhou, Graham Neubig, and Jiatao Gu. 2020. Understanding Knowledge Distillation in Non-autoregressive Machine Translation. In *ICLR*.