Imputer: Sequence Modelling via Imputation and Dynamic Programming

William Chan [1] **Chitwan Saharia** [1 †] **Geoffrey Hinton** [1] **Mohammad Norouzi** [1] **Navdeep Jaitly** [2]

Abstract

This paper presents the Imputer, a neural sequence model that generates output sequences iteratively via imputations. The Imputer is an iterative generative model, requiring only a constant number of generation steps independent of the number of input or output tokens. The Imputer can be trained to approximately marginalize over all possible alignments between the input and output sequences, and all possible generation orders. We present a tractable dynamic programming training algorithm, which yields a lower bound on the log marginal likelihood. When applied to end-to-end speech recognition, the Imputer outperforms prior non-autoregressive models and achieves competitive results to autoregressive models. On LibriSpeech test-other, the Imputer achieves 11.1 WER, outperforming CTC at 13.0 WER and seq2seq at 12.5 WER.

1. Introduction

Neural sequence models have been widely successful across a wide range of applications, including machine translation (Bahdanau et al., 2015; Luong et al., 2015), speech recognition (Graves & Jaitly, 2014; Chan et al., 2016), speech synthesis (Oord et al., 2016; 2018) and image captioning (Vinyals et al., 2015; Xu et al., 2015). Autoregressive sequence models (e.g., Sutskever et al. (2014); Cho et al. (2014)) enable exact likelihood estimation, at the cost of requiring n generation steps to generate n tokens during inference. On the other hand, non-autoregressive models such as CTC (Graves et al., 2006) and NAT (Gu et al., 2018) can generate sequences in a single generation step, independent of the number of output tokens. However these non-autoregressive models typically make a strong conditional independence assumption between output tokens, often underperforming their autoregressive counterparts. Recently, there has been a growing interest in models that make a trade-off between the two extremes of fully autoregressive and fully non-autoregressive generation, such as the Insertion Transformer (Stern et al., 2019), Mask-Predict (Ghazvininejad et al., 2019), Levenstein Transformer (Gu et al., 2019b) and Multilingual KERMIT (Chan et al., 2019a). Such models sacrifice almost no performance, while requiring a logarithmic (Chan et al., 2019c) or a constant number of generation steps (Lee et al., 2018).

In this paper, we seek to extend prior work to achieve a desirable balance between fully autoregressive and fully non-autoregressive models. We are concerned with sequence problems in which a natural monotonic latent alignment exists between the source and target sequences (e.g., end-to-end speech recognition). Prior work typically does not take advantage of such useful inductive biases, which become increasingly important for problems with long output sequences. For instance, in speech recognition sequences are typically an order of magnitude longer than sequences seen in machine translation. The need to generate longer sequences highlights the importance of developing sequence generation frameworks with sub-linear inference time.

Three popular types of generative models have been developed for end-to-end speech recognition applications: CTC (Graves et al., 2006; Graves & Jaitly, 2014), RNN-T (Graves, 2012; Graves et al., 2013) and seq2seq (Chan et al., 2016; Bahdanau et al., 2016a). CTC is a non-autoregressive model and makes a conditional independence assumption between token frame predictions. Accordingly, CTC can make use of dynamic programming to marginalize over all possible alignments. CTC models exhibit fast inference but relatively poor performance (Battenberg et al., 2017). RNN-T is an autoregressive model that captures conditional dependencies between output tokens, but at the cost of n generation steps to generate n tokens. Similar to CTC, the RNN-T relies on dynamic programming to marginalize over all possible alignments for exact likelihood computation. However, the RNN-T dynamic programming framework typically relies on a linear bottleneck for tractability (e.g., the linear bottleneck between the prediction and transcription network (Graves, 2012)); this has typically limited the empirical

[†] Work done as part of the Google AI Residency. [1] Google Research, Brain Team, Toronto, Ontario, Canada. [2] Work done at Google; currently at The D. E. Shaw Group, New York, New York, USA. Correspondence to: W. Chan <williamchan@google.com>, C. Saharia <sahariac@google.com>, G. Hinton <geoffhinton@google.com>, M. Norouzi <mnorouzi@google.com>, N. Jaitly <navdeep.jaitly@deshaw.com>.

Proceedings of the 37^{th} International Conference on Machine Learning, Online, PMLR 119, 2020. Copyright 2020 by the author(s).

success of RNN-T (Prabhavalkar et al., 2017). Seq2seq is an autoregressive left-to-right model, which factorizes the probability with the chain rule resulting in exact likelihood estimates. Empirically, seq2seq often performs the best (Chiu et al., 2018). However, the seq2seq framework requires n generation steps to generate n tokens.

This paper presents the Imputer, with the following important characteristics:

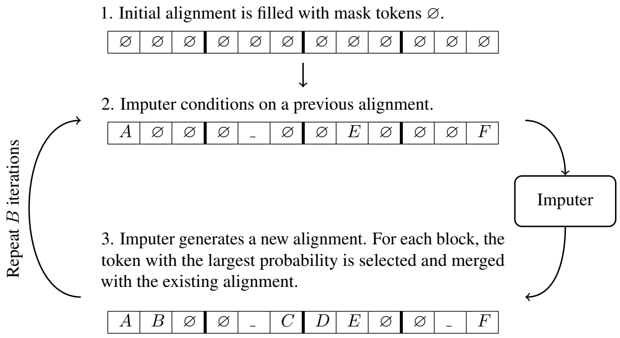
1. Imputer is an iterative generative model, requiring a constant number of generation steps independent of the number of output sequence tokens (see Figure 1).
2. Imputer approximately marginalizes over all possible alignments and generation orders. It is well suited for problems with a latent monotonic alignment such as speech recognition.
3. Imputer is not left-to-right. It can model language with bidirectional contextualization, and can model both local and global conditional dependencies.
4. Imputer superimposes the input and output sequences together, which allows the Imputer to simplify the architecture and avoid the cross-attention mechanism typically found in encoder-decoder sequence models.

2. The Imputation Framework

Notation. Let x denote the input sequence (e.g., audio) and y denote the output sequence (e.g., text). Let $y_i \in \mathcal{V}$ denote a token in the output sequence, where $\mathcal{V}$ is the vocabulary (e.g., alphabet). It is not required that x and y have an equal length, but x must be longer than or equal in length to y (i.e., $|x| \geq |y|$). Let a denote a latent alignment between x and y, with $a_i \in \mathcal{V}^+$ where $\mathcal{V}^+ = \mathcal{V} \cup \{``\_"\}$ and "_" is a special token called the blank token, which helps align x and y. This blank token is compatible with the definition from CTC (Graves et al., 2006). We assume there exists a monotonic alignment between x and y, and consequently $|a| = |x| \geq |y|$, such that when we remove the blank tokens from a, we recover y. We also introduce the function $\beta(y)$ that returns the set of all possible alignments for y (with length $|x|$). Let $\beta^{-1}(a)$ denote the collapse function which returns the y associated with a given a. Here is a concrete example: let $y = (A, B, C, D)$ and $|x| = 7$, then, one possible target alignment is $a = (\_, A, B, \_, C, \_, D)$ and $y = \beta^{-1}(a)$. We note to an astute reader well versed with CTC that the discussion of collapsing repetitions typically found in the CTC literature (Graves et al., 2006; Graves & Jaitly, 2014) is omitted, but CTC and Imputer can handle both cases, and this is simply a hyperparameter choice.

2.1. Connectionist Temporal Classification (CTC)

We will first briefly discuss Connectionist Temporal Classification (CTC) (Graves et al., 2006). CTC models an alignment a with the conditional independence assumption between token frame predictions:


1. Initial alignment is filled with mask tokens ∅.

$\varnothing\ \varnothing\ \varnothing \mid \varnothing\ \varnothing\ \varnothing \mid \varnothing\ \varnothing\ \varnothing \mid \varnothing\ \varnothing\ \varnothing$

2. Imputer conditions on a previous alignment.

$A\ \varnothing\ \varnothing \mid \varnothing\ \_\ \varnothing \mid \varnothing\ E\ \varnothing \mid \varnothing\ \varnothing\ F$

3. Imputer generates a new alignment. For each block, the token with the largest probability is selected and merged with the existing alignment.

$A\ B\ \varnothing \mid \varnothing\ \_\ C \mid D\ E\ \varnothing \mid \varnothing\ \_\ F$

Figure 1. Visualization of the Imputer's decoding procedure. For this example, the alignment comprises 4 blocks with block size of $B = 3$ tokens each. The alignment "A B _ _ _ C D E _ _ _ F" is being imputed, with 1 token per block imputed at each decoding iteration. The decoding process takes exactly B iterations.

$$p_\theta(a|x) = \prod_i p(a_i|x; \theta) \tag{1}$$

CTC is typically trained to maximize the log-likelihood:

$$\log p_\theta(y|x) = \log \sum_{a \in \beta(y)} p_\theta(a|x) \tag{2}$$

Equation 2 marginalizes over all possible alignments $\beta(y)$ compatible with y. The conditional independence assumption warrants Equation 2 to be solved exactly and efficiently via dynamic programming (Graves et al., 2006), and permits for fully non-autoregressive generation in one inference step. However, the conditional independence assumption also means tokens are generated without dependencies on each other, consequently the model cannot easily model language dependencies and/or multimodal outputs.

2.2. The Imputer

Imputer is an iterative generative model. At each generative step, Imputer conditions on a previous partially generated alignment and emits a new alignment. Each generative step emits a complete alignment, but, during inference only a small subset is selected to generate the next partial alignment. This subset typically includes the previously emitted tokens, resulting in an iterative refinement process. This iterative process allows Imputer to model conditional dependencies across generation steps, which CTC lacks. The Imputer parameterizes the alignment distribution with a conditional independence assumption between token predictions:

$$p_\theta(a|\tilde{a}, x) = \prod_i p(a_i|\tilde{a}, x; \theta) \tag{3}$$

where $\tilde{a}$ is some previous alignment, and $\tilde{a}_i \in \mathcal{V}^{++}$ where $\mathcal{V}^{++} = \mathcal{V}^{+} \cup \{\varnothing\}$ where $\varnothing$ is the masked out token. The $\varnothing$ mask token is compatible with the mask token definition from BERT (Devlin et al., 2019). To be explicit, $\tilde{a}$ may be all $\varnothing$ (e.g., all tokens are masked out), and $a_i = \tilde{a}_i \forall \tilde{a}_i \neq \varnothing$ (i.e., once a token is committed in $\tilde{a}$, a must be consistent). Here is a concrete example, if $y = (A, B, C, D)$ and $|x| = 7$, given a previous alignment $\tilde{a} = (\varnothing, A, \varnothing, \varnothing, C, \_, D)$, one possible target alignment is $a = (\_, A, B, \_, C, \_, D)$. One key property of Imputer is that, it can model conditional dependencies through the conditioning on $\tilde{a}$ (e.g., across generation steps), and achieve parallel generation through the conditional independence assumption in-between new token predictions (e.g., within a generation step).

2.3. Inference

We will first describe inference of Imputer, then we will describe the training objective function afterwards. The structure of the Imputer framework allows for very flexible balance between autoregressive and non-autoregressive generation. On one extreme, Imputer can be fully non-autoregressive with parallel generation, here $\tilde{a}$ is simply all $\varnothing$ blank tokens (i.e., not conditioned on any information), and a and consequently y is generated in just 1 step. This would make full conditional independence assumption between tokens, falling back to CTC-like model (Graves et al., 2006) (depending on the loss chosen to train Imputer).

On the other extreme, Imputer can be fully autoregressive with serial generation: At each generation step, we sample one token from Imputer (e.g., Equation 3), join it with the previous hypothesis $\tilde{a}$, and repeat this process until $\tilde{a}$ has no more masked tokens. This autoregressive inference process will take exactly $|x|$ generation steps. This procedure will not make any conditional independence assumption between tokens, but at the cost of generation speed. Note, this autoregressive process need not be left-to-right, as there is no constraint on the generation order.

We present an alternative inference procedure that takes a balance between fully autoregressive generation and fully non-autoregressive generation. Our inference procedure will result in a constant number of generation steps, while still modelling conditional dependencies on both a local and global basis. Our inference strategy begins by segmenting the sequence a into equal sized blocks of size B. We initialize a previous alignment with all masked out tokens $\varnothing$. We then compute the model's new alignment prediction (conditioned on the source sequence and the previous alignment). For each block, we select the slot (and its token) with the maximum token-level probability prediction from the model (among slots that are masked out). We merge this prediction with the previous alignment. This partially predicted alignment is then used for inference in the next decoding iteration. Since we use fixed and equal sized blocks, and each decoding iteration ensures exactly one token imputation per block, the inference procedure will finish in exactly B iterations. Note that in the extreme case of $B = 1$, this will result in fully non-autoregressive generation, while the other extreme $B = |a|$ will result in fully autoregressive generation.

2.4. Marginalization

We now proceed to describe the objective function of Imputer. We start off by writing the marginalization over all possible alignments and generation orders:

$$\log p_\theta(y|x) = \log \sum_{a \in \beta(y)} \sum_{\tilde{a} \in \gamma(a)} p_\theta(a|\tilde{a}, x) p(\tilde{a}|x) \tag{4}$$

where $\beta(y)$ captures all possible alignments of y, $\gamma(a)$ captures all possible masking permutations, and $p(\tilde{a}|x)$ is some prior. The combination of $\beta \times \gamma$ will capture all possible alignments and masking permutations, and consequently it will capture all possible alignments and generation order. We can rewrite this under expectation of the importance sampling distributions q and r:

$$= \log \mathbb{E}_{a \sim q} \left[\frac{1}{q(a)} \mathbb{E}_{\tilde{a} \sim r} \left[\frac{1}{r(\tilde{a})} p_\theta(a|\tilde{a}, x) p(\tilde{a}|x) \right] \right] \tag{5}$$

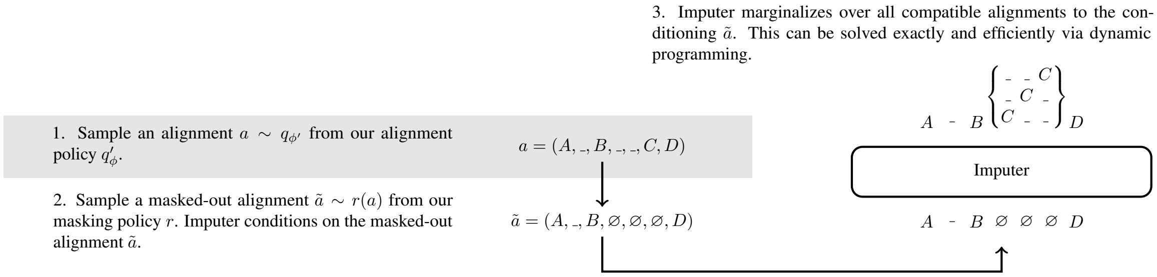
$$\geq \mathbb{E}_q \left[\mathbb{E}_r \left[\log p_\theta(a|\tilde{a}, x) + \log p(\tilde{a}|x) \right] + H(r) \right] + H(q) \tag{6}$$

$$\geq \mathbb{E}_q \left[\mathbb{E}_r \left[\log p_\theta(a|\tilde{a}, x) \right] + H(r) \right] + H(q) \tag{7}$$

where q replaces β to capture all possible alignments, and r replaces γ to capture all possible maskings. As long as there is support everywhere under q and r, this equality in Equation 5 holds. We can further use Jensen's inequality to create a lower bound in Equation 7. Assuming q and r are distributions that do not share parameters with p_θ, we can ignore the entropy terms during the gradient computation. We define our objective function based on Equation 7 without the constant entropy terms:

$$J(\theta) = \underbrace{\mathbb{E}_{a \sim q} [\mathbb{E}_{\tilde{a} \sim r}}_{\text{roll-in policy}} [\log p_\theta(a|\tilde{a}, x)]] \tag{8}$$

There are two main problems with Equation 8, 1) high variance due to the double sampling procedure, and 2) it is unclear how to choose r and perhaps more importantly how to choose q. Consequently, naively optimizing for Equation 8 directly is difficult in practice (Luo et al., 2016). Equation 8 can be interpreted as $q \times r$ being our roll-in policy to generate a $\tilde{a}$, and our model θ is taught to complete the alignment output given some sampled previous alignment $\tilde{a}$. One way to reinterpret the training problem is, 1) what


3. Imputer marginalizes over all compatible alignments to the conditioning $\tilde{a}$. This can be solved exactly and efficiently via dynamic programming.

1. Sample an alignment $a \sim q_{\phi'}$ from our alignment policy q'_{ϕ}.

2. Sample a masked-out alignment $\tilde{a} \sim r(a)$ from our masking policy r. Imputer conditions on the masked-out alignment $\tilde{a}$.

$a = (A, \_, B, \_, \_, C, D)$
$\tilde{a} = (A, \_, B, \varnothing, \varnothing, \varnothing, D)$
Imputer
A - B $\varnothing$ $\varnothing$ $\varnothing$ D
A - B $\left\{\begin{matrix} \_ \; \_ \; C \\ \_ \; C \; \_ \\ C \; \_ \; \_ \end{matrix}\right\}$ D

Figure 2. Visualization of the Imputer dynamic programming training procedure. A roll-in policy is used to sample a masked-out alignment $\tilde{a}$. Imputer marginalizes over all compatible alignments with $\tilde{a}$ over the masked-out regions.

should the roll-in policy used to sample $\tilde{a}$ be? and 2) given a partially completed alignment $\tilde{a}$, what should the target policy be to complete the alignment? We will discuss the choice of roll-in policy in more detail in Section 2.5. We take two approaches for our target policy, 1) we use Imitation Learning and follow an expert, and 2) we use Dynamic Programming to marginalize over all possible alignments compatible with $\tilde{a}$. We will discuss these target policies in detail in Section 2.6 and 2.7.

2.5. Roll-in Policy

The ideal choice of the roll-in policy is obvious, we want to align our training and inference conditions, thus the ideal choice of the roll-in should be drawing $\tilde{a}$ samples from our model's policy p_θ. However, this would be prohibitively expensive as it would involve sampling/decoding on-the-fly during training. Our strategy is to sample from some other distribution, which is inexpensive yet close to our model θ.

Alignment Policy β. We will first discuss how to sample alignments. Let us assume we had access to an expert ϕ (e.g., existing CTC or Imputer model). We could sample q_ϕ and use this as our alignment sampling distribution. We could in theory compute the logits for all the examples in the training distribution, store them offline, and sample from it during training. However, this would be prohibitively expensive as it would require large amounts of memory. We found it convenient and working well in practice to simply create a pseudo-expert policy $q_{\phi'}$ as a distribution we can easily sample from. We first search for the best empirical alignment $\hat{a}^*_\phi$ under the expert ϕ:

$$\hat{a}^*_\phi = \arg\max_a q_\phi(a|x, y) \tag{9}$$

For certain classes of models (e.g., CTC), Equation 9 can be solved efficiently and exactly via dynamic programming (Graves et al., 2006). We then create $q_{\phi'}$ from $\hat{a}^*_\phi$ by adding some small noise:

$$q_{\phi'} = \hat{a}^*_\phi + \mathcal{N} \tag{10}$$

where $\mathcal{N}$ is some noise distribution (e.g., randomly shifting the alignment left-or-right). Sampling from $q_{\phi'}$ is especially convenient because we can store $\hat{a}^*_\phi$ offline, and sample the noise distribution $\mathcal{N}$ on-the-fly during training (which is cheap), and there would be no need to store or sample from q_ϕ directly during training.

An alternative approach is to draw samples from $q_{\theta'}$, a stationary distribution created from a stale copy of the model θ'. This approach would more closely align the training and inference conditions. This approach is also convenient since we will not need to do live decoding or sampling during training, and since $q_{\theta'}$ is stationary, we also would not need to backpropagate through it. We experiment with both approaches in our experimental section.

Masking Policy γ**.** Until now, we have only discussed how to select the q part of the roll-in alignment policy, we will now discuss our masking policy r. There are obvious choices for r, for example Bernoulli (Devlin et al., 2019) or Uniform distributions (Stern et al., 2019). We also designed a masking policy inline with our inference procedure described in Section 2.3. During inference, we $\arg\max$ one token from each block, parallel across all blocks. This means at any given point, there is an equal number of tokens per block. Following this observation, we design our Block sampling strategy to follow this inference condition. We first sample a number $b \in [0, B)$, where B is the size of our block. We then simply randomly mask out b number of tokens per block.

2.6. Imitation Learning

Let us assume we had access to the same (psuedo-)expert ϕ' (as described in Section 2.5). One simple objective is to simply imitate the expert:

$$J_{\text{IM}}(\theta) = \mathbb{E}_{a \sim q_{\phi'}} \left[\mathbb{E}_{\tilde{a} \sim r} \left[\log p_\theta(a|\tilde{a}, x) \right] \right] \tag{11}$$

Equation 11 can be interpreted as $q_{\phi'} \times r$ being our roll-in policy, and our model θ is taught to imitate the pseudo-expert ϕ'.

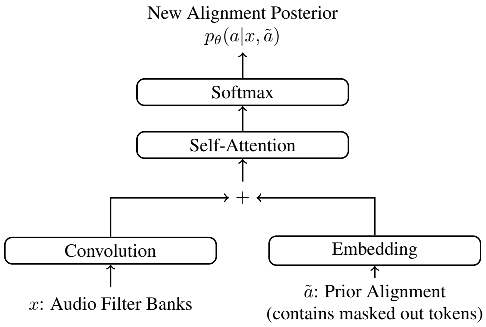


Figure 3. Visualization of the Imputer architecture. Imputer conditions on the previous alignment $\tilde{a}$, a convolutional network processes the audio features x, and a Transformer self-attention stack consolidates all available context to generate a new alignment.

2.7. Dynamic Programming

One limitation of the Imitation Learning approach above is that when we distill $q_{\phi'}$ as our target distribution, we are constrained to the knowledge (and errors) of ϕ'. This is especially concerning if ϕ' does not capture the best possible alignment for our model θ. We present a new objective function which decouples the role of $q_{\phi'}$ as both the roll-in distribution and target policy. We keep $q_{\phi'} \times r$ as our roll-in distribution, but for our target policy we marginalize over all possible compatible alignments:

$$J_{\text{DP}}(\theta) = \mathbb{E}_{a\sim q_{\phi'}}\left[\mathbb{E}_{\tilde{a}\sim r}\left[\log \sum_{a'\in\beta'(\tilde{a},a)} p_\theta(a'|\tilde{a},x)\right]\right] \tag{12}$$

where $\beta'(\tilde{a},a)$ returns the set of all possible alignments compatible with $\tilde{a}$ drawn from our roll-in distribution $q_{\phi'} \times r$. The key difference between Equation 11 and Equation 12 is the extra summation $\sum_{a'\in\beta'(\tilde{a},a)}$. The summation says, under the roll-in distribution of $q_{\phi'} \times r$, let us marginalize over all compatible alignments with $\tilde{a}$. Conveniently, the $\log\sum$ term in Equation 12 can be computed exactly and efficiently via dynamic programming (Graves et al., 2006). The dynamic programming to solve Equation 12 is exactly the same as the CTC dynamic programming (Graves et al., 2006), except we force emit symbols. Equation 12 also has the added side-effect of having a much lower variance gradient due to the summation term (which can be seen as smoothing out the loss). Figure 2 visualizes the training procedure.

2.8. Lower bound

We can actually show our Imitation Learning algorithm (Equation 11) is a lower bound for our Dynamic Programming (Equation 12), which in turn is a lower bound for the log probability (up to a constant, which can be dropped during the gradient computation). We start from Equation 4:

$$\begin{aligned}
&\log p_\theta(y|x)\\
&\quad= \log \sum_{a\in\beta(y)} \sum_{\tilde{a}\in\gamma(a)} p_\theta(a|\tilde{a},x) && (13)\\
&\quad= \log \sum_{a\in\beta(y)} \sum_{\tilde{a}\in\gamma(a)} \sum_{a'\in\beta'(\tilde{a},a)} \frac{1}{c(\tilde{a},a)} p_\theta(a'|\tilde{a},x) && (14)\\
&\quad= \log \sum_{a\in\beta(y)} \sum_{\tilde{a}\in\gamma(a)} \frac{1}{c(\tilde{a},a)} \sum_{a'\in\beta'(\tilde{a},a)} p_\theta(a'|\tilde{a},x) && (15)\\
&\quad\geq \mathbb{E}_{a\sim q_{\phi'}}\left[\mathbb{E}_{\tilde{a}\sim r}\left[\log \sum_{a'\in\beta'(\tilde{a},a)} p_\theta(a'|\tilde{a},x)\right]\right] + C && (16)
\end{aligned}$$

Equation 13 writes out the marginalization over all possible alignments $a \in \beta(y)$, and over all possible maskings $\tilde{a} \in \gamma(a)$. We can rewrite Equation 13 with an additional summation over $\beta'(\tilde{a},a)$, where $\beta'(\tilde{a},a)$ captures all compatible alignments with (partially) masked-out alignment $\tilde{a}$. This will inherently result in repetition of alignments, and the constant $c(\tilde{a},a)$ is the number of times each alignment is repeated. We note that the repetition constant $c(\tilde{a},a)$ is simply a function of $\tilde{a}$ and a:

$$c(\tilde{a},a) = \prod_{s\in\text{masked\_segments}(\tilde{a})} \binom{s_n}{s_k} \tag{17}$$

where $\text{masked\_segments}(\tilde{a})$ captures each contiguous masked out segment of $\tilde{a}$, and s_n is the number of mask tokens in the segment, and s_k is the number of target tokens in the segment. Equation 15 re-arranges the constant, and Equation 16 applies importance sampling, Jensen's inequality and collapses the constants into $C = \mathbb{E}_{q_{\phi'}}\left[H(r) - \mathbb{E}_r\left[\log(c(\tilde{a},a))\right]\right] + H(q_{\phi'})$, since they do not affect the gradient and optimization. Equation 16 recovers our dynamic programming algorithm, which is a lower bound up to constant C.

It is further obvious that the our Imitation Learning algorithm 11) is a lower bound of Equation 16. We replace $a' \in \beta'(\tilde{a},a)$ with $a' = a$, which means we are simply dropping terms in the summation; since all the terms dropped are positive, the Imitation Learning algorithm is a (weak) lower bound to our Dynamic Programming algorithm. Note, that both our Imitation Learning and Dynamic Programming algorithm uses the same roll-in policy $\phi' \times \gamma$, which allows us ignore the importance sampling correction terms.

3. Model

In this section, we describe the neural network architecture of our Imputer implementation. Figure 3 visualizes

Table 1. Wall Street Journal Character Error Rate (CER) and Word Error Rate (WER).

Model	CER	WER	Iterations
seq2seq			
Bahdanau et al. (2016a)	6.4	18.6	n
Bahdanau et al. (2016b)	5.9	18.0	n
Chorowski & Jaitly (2017)	-	10.6	n
Zhang et al. (2017)	-	10.5	n
Chan et al. (2017)	-	9.6	n
Kim et al. (2017)	7.4	-	n
Serdyuk et al. (2018)	6.2	-	n
Tjandra et al. (2018)	6.1	-	n
Sabour et al. (2019)	3.1	9.3	n
CTC			
Graves & Jaitly (2014)	8.4	27.3	1
Liu et al. (2017)	-	16.7	1
CTC (Our Work)	5.6	15.2	1
Imputer (IM)	6.2	16.5	8
Imputer (DP)	4.9	12.7	8

Table 2. LibriSpeech test-clean and test-other Word Error Rate (WER).

Method	clean	other	Iterations
seq2seq			
Zeyer et al. (2018a)	4.9	15.4	n
Zeyer et al. (2018b)	4.7	15.2	n
Irie et al. (2019)	4.7	13.4	n
Sabour et al. (2019)	4.5	13.3	n
Luscher et al. (2019)	4.4	13.5	n
Park et al. (2019)	4.1	12.5	n
ASG/CTC			
Collobert et al. (2016)	7.2	-	1
Liptchinsky et al. (2017)	6.7	-	1
CTC (Our Work)	4.6	13.0	1
Imputer (IM)	5.5	14.6	8
Imputer (DP)	4.0	11.1	8

the Imputer neural network. The Imputer processes the x audio features (e.g., filter banks) with a stack of convolutional layers. These features are then added to alignment embeddings (from a previous alignment $\tilde{a}$), which are then passed through a stack of Transformer self-attention layers (Vaswani et al., 2017). Finally, a softmax layer models the new alignment a. We describe the exact hyperparameters used in Section 4.

4. Experiments

We experiment with two competitive speech tasks, the 82 hours Wall Street Journal (WSJ) (Paul & Baker, 1992) dataset and the 960 hours LibriSpeech (Panayotov et al., 2015) dataset. We use Kaldi (Povey et al., 2011) to generate 80-dimensional filter banks, with delta and delta-delta. We use SentencePiece (Kudo & Richardson, 2018) to generate a 400 BPE subword vocabulary. We compare our Imputer models to other end-to-end speech recognition networks (i.e., without relying on an external language model). We also opt to compare results without the use of data augmentation, since different authors use different data augmentation methods (e.g., speed perturbation (Li et al., 2019), tempo/volume perturbation (Tuske et al., 2019), SpecAugment (Park et al., 2019), etc...) which makes fair comparison difficult.

Our neural network uses 2 layers of convolution each with 11×3 filter size and stride of 2×1. For our WSJ experiments, we use 8 Transformer self-attention layers with 4 attention heads, 512 hidden size, 2048 filter size, dropout rate 0.2 and train for 300k steps. For our LibriSpeech experiments, we use 16 Transformer self-attention layers with 4 attention heads, 512 hidden size, 4096 filter size, dropout rate 0.3 and train for 1M steps. We perform no model selection and simply report the CER/WER at the end of training.

We built strong CTC baselines with the architecture described above, which outperform prior non-autoregressive work. We also use our CTC models as our expert alignment model ϕ for our alignment roll-in. We add a small amount of noise by shuffling the alignment by up to 1 frame to the left-or-right, however this was not critical and only resulted in very minor improvements ($<$0.2) in WER. Our Imputer results are reported with the Block masking policy.

Table 1 and Table 2 summarizes our WSJ and LibriSpeech experiments respectively, along with prior work. We find our Imputer model to outperform prior non-autoregressive work. For WSJ, our Imputer model achieves 12.7 WER, while our non-autoregressive CTC baseline achieves 15.2 WER. We find our Imputer model competitive to prior work of autoregresive seq2seq models which achieve 9.3 WER (Sabour et al., 2019). For LibriSpeech, our CTC baseline achieves 13.0 WER, a strong autoregressive seq2seq baseline achieves 12.5 WER (Park et al., 2019), and our Imputer model achieves 11.1 WER outperforming both works. Figure 4 gives an example decoding path from LibriSpeech.

We find our Dynamic Programming (DP) algorithm to outperform our Imitation Learning (IL) algorithm in both WSJ and LibriSpeech. Our IL algorithm achieves 14.6 WER while our DP algorithm achieves 11.1 WER for LibriSpeech. This is expected since our DP algorithm is a tighter lower bound for the likelihood. However, we also found our IL algorithm to perform worse than our CTC baseline. One hypothesis is that the variance of the IL gradient may be much higher, and the usage of dynamic programming may help in both generalization and optimization.

4.1. Analysis

In this section we will analyze different decoding strategies and training/inference block sizes.

Decoding Strategy. During inference, we want to avoid

b	Alignment																															
0	∅	∅	∅	∅	∅	∅	∅	∅	∅	∅	∅	∅	∅	∅	∅	∅	∅	∅	∅	∅	∅	∅	∅	∅	∅	∅	∅	∅	∅	∅	∅	∅
1	∅	∅	∅	[HAVE	∅	∅	∅	∅	∅	[L	∅	∅	∅	∅	∅	∅	∅	∅	∅	Y	∅	∅	∅	∅	∅	∅	L	∅	∅	∅	∅	∅
2	∅	∅	∅	[HAVE	∅	∅	[TO	∅	∅	[L	∅	∅	IVE	∅	∅	∅	∅	∅	∅	Y	∅	∅	[SE	∅	∅	∅	L	∅	F	∅	∅	∅
3	∅	∅	_	[HAVE	∅	∅	[TO	∅	_	[L	∅	∅	IVE	∅	∅	∅	∅	[M	∅	Y	∅	∅	[SE	∅	_	∅	L	∅	F	∅	∅	∅
4	∅	∅	_	[HAVE	∅	∅	[TO	_	_	[L	∅	∅	IVE	_	∅	∅	∅	[M	∅	Y	∅	∅	[SE	_	_	∅	L	∅	F	∅	∅	_
5	_	∅	_	[HAVE	∅	∅	[TO	_	_	[L	∅	_	IVE	_	∅	∅	∅	[M	_	Y	∅	∅	[SE	_	_	_	L	∅	F	∅	∅	_
6	_	∅	_	[HAVE	∅	_	[TO	_	_	[L	∅	_	IVE	_	[WITH	∅	∅	[M	_	Y	∅	_	[SE	_	_	_	L	∅	F	∅	_	_
7	_	_	_	[HAVE	∅	_	[TO	_	_	[L	_	_	IVE	_	[WITH	∅	_	[M	_	Y	∅	_	[SE	_	_	_	L	_	F	∅	_	_
8	_	_	_	[HAVE	_	_	[TO	_	_	[L	_	_	IVE	_	[WITH	_	_	[M	_	Y	_	_	[SE	_	_	_	L	_	F	_	_	_

Figure 4. Example Imputer inference from the LibriSpeech dev set with block size $B = 8$ for the target sequence “[HAVE [TO [LIVE [WITH [MYSELF” with generated alignment “_ _ _ [HAVE _ _ [TO _ _ [L _ _ IVE _ [WITH _ _ [M _ Y _ _ [SE _ _ _ L _ F _ _ _”. Imputer takes exactly block size number of generation steps ($B = 8$) to generate the entire sequence, independent of the output sequence length. At each b iteration, one token is filled in within each block and with parallel generation across blocks.

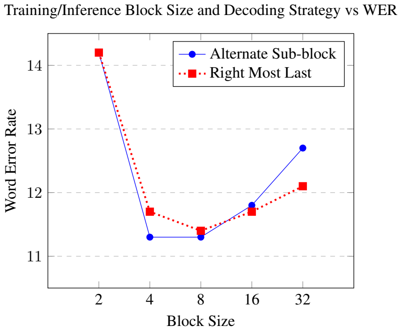


Figure 5. LibriSpeech dev-other WER with different block size used for training/inference and decoding strategy.

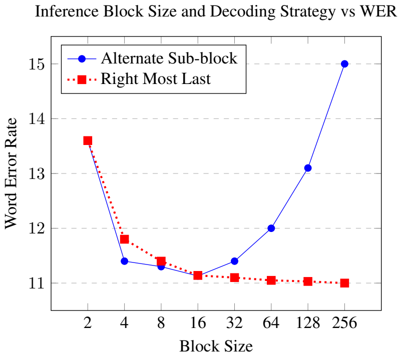


Figure 6. LibriSpeech dev-other WER with different block size used for inference and decoding strategy trained with block size 8.

making local conditional independence assumptions. We want to avoid neighbouring tokens to be generated in parallel. Our inference strategy to arg max only 1 token per block per generation iteration mostly accomplishes this. However, there is the edge case between the block boundaries which can result in two neighbouring tokens generated in parallel at the same time. We present two decoding strategies that will alleviate the model from generating parallel neighbouring tokens at this boundary condition:

1. **Alternate Sub-block**. Each block is divided into two sub-blocks, left sub-block and right sub-block. On even iterations, only the left sub-block is allowed to be imputed, and during odd iterations, only the right sub-block is allowed to be imputed.
2. **Right Most Last**. The right-most token in each block is only permitted to be imputed at the last iteration.

Block Size. The block size is an important hyperparameter for the Imputer model, since it trades-off between inference speed and model contextualization. A small block size will result in a smaller number of generation iterations, while a large block size will result in fewer conditional independence assumptions. We analyze the effect of block sizes in two ways:

1. We train our model to various block sizes $B \in \{2, 4, 8, 16, 32\}$, and use the same block size for inference.
2. We train our model to a fixed block size $B = 8$, and use different block sizes for inference.

We report both the decoding strategy and block size experiments for the LibriSpeech dev-other split in Figure 5 and Figure 6. Figure 5 compares the performance of the models trained with different block sizes (with inference using the same block size they were trained on), along with the two different decoding strategies. In case of both decoding strategies, block size $B = 8$ gives the lowest WER. As we decrease the block size further, WER increases because of more conditionally independent (parallel) generations. Interestingly, we find that models trained and decoded with larger block sizes also yield worse WER, even though they allow more conditional dependencies. We hypothesize this could be an optimization issue, as we found it more difficult to train larger block size models (e.g., the $\beta \times \gamma$ space is larger for larger block sizes).

Figure 6 compares the performance where the model is trained with a block size of $B = 8$, but decoded with various block sizes. When using Right Most Last decoding strategy, we find a monotonic decrease in WER with increase in block size resonating with the hypothesis that more conditional dependency should result in better WERs at the cost of inference speed. However, when using Alternate Sub-block decoding strategy, we observe a local minima at around $B =$

Table 3. LibriSpeech dev-other WER for various masking policies.

Masking Policy	WER
Bernoulli	11.2
Uniform	11.4
Block	11.4

16. We hypothesize that this behaviour occurs due to the over-constraining nature of Alternate Sub-block decoding strategy, as it forces the model to predict tokens for slots in specific sub-blocks at each iteration.

4.2. Masking Policy

As discussed in Section 2.5, we can use different masking policy γ. Table 3 reports the results over 3 different masking policies: Bernoulli, Uniform and Block. We find Imputer to be robust to the choice of masking policy.

4.3. Training with Model Samples

In Section 2.5, we discussed sampling from a (stale) copy of our model for the alignments in the roll-in policy. We trained models where we started off with roll-in alignments from a pre-trained CTC model (with noise), then decoded our Imputer model every 50k steps for our new alignments. However, we did not find improvements in performance, suggesting that using the CTC alignment is a sufficient for the roll-in alignments to train Imputer.

4.4. Top-k Decoding

We can ignore the block structure during inference, and decode via greedy top-k. This will once again result in a model requiring only constant B generation steps if we choose $k = {}^{|x|}/_{B}$. Once again, care is taken such that we do not impute neighbouring tokens in parallel. We find our models to perform slightly worse under this condition, suggesting the block inference strategy to reasonable, especially since speech is relatively local. We decode our Bernoulli model on LibriSpeech dev-other and it achieves 11.6 WER, compared to 11.2 WER using our Block inference strategy.

4.5. Simulated Annealing Decoding

One alternative view of Imputer is to view it as a transition operator in an energy-based system. We can use a simulated annealing like procedure for decoding. We tried various strategies where we deleted token commitments via a random sampling policy or based on the token-level likelihood (similar to Mask-Predict (Ghazvininejad et al., 2019)). We found improvements for unconverged models, however we did not find improvements on our converged models.

5. Related Work

Non-Autoregressive Transformer (NAT) (Gu et al., 2018) and Mask-Predict (Ghazvininejad et al., 2019) are closely related to our Imputer model. Mask-Predict was also recently adapted for end-to-end speech recognition (Chen et al., 2019). There are several key differences, 1) these models require predicting the target sequence length first, and 2) these models follow an encoder-decoder formulation which requires a cross-attention mechanism that may not be well suited for long monotonic sequence problems like speech recognition. Imputer does not rely on length prediction first but relies on a fixed size canvas, and the local prediction structure of the Imputer architecture is perhaps more suited for monotonic problems like speech recognition.

The Insertion Transformer (Stern et al., 2019) and other insertion-based models (Gu et al., 2019a; Welleck et al., 2019; Chan et al., 2019b; Li & Chan, 2019; Ruis et al., 2019) generates sequences through a series of insertions. The Insertion Transformer can dynamically grow its canvas size, whereas the Imputer has a fixed canvas size. Insertion Transformer has been shown to generate sequences in logarithmic $\log n$ iterations (Stern et al., 2019), where as the Imputer requires constant generation steps. Additionally, the Insertion Transformer with its vastly non-monotonic generation order (Chan et al., 2019c) lacks the monoticity that the Imputer dynamic programming endows, consequently we found the Insertion Transformer difficult to apply to speech recognition applications.

Our neural network architecture was also inspired by past work in convolutions (Sainath et al., 2013) and Transformers (Dong et al., 2018) for speech recognition. Mohamed et al. (2019) also combined convolutions with self-attention, where they also use convolutions to model local acoustic features, our work also uses self-attention to capture global signals from both the acoustics and the previous alignment, which licenses powerful bidirectional language modelling. Finally, we also mention the concurrent work of Synnaeve et al. (2019), where they also apply a convolutional Transformer architecture similiar to ours, showing strong empirical results on both CTC and seq2seq models. However, their work involved substantially larger models with data augmentation and focus on semi-supervised learning.

6. Conclusion

In this paper, we presented the Imputer, a neural sequence model which generates sequences iteratively via imputations. Imputer iteratively conditions on a previous partial alignment, and generates a new alignment. Imputer requires only a fixed constant number of generative iterations independent of the number of input or output tokens. Imputer can be trained to approximately marginalize over all possible alignments and generation order. We presented a tractable dynamic programming training algorithm. Our dynamic program marginalizes over all possible completions given any previous alignment, and we show it to be a lower bound for the likelihood. We apply Imputer to end-to-end speech recognition tasks, and find Imputer outperforming prior non-autoregressive approaches and achieving comparable performance to autoregressive models. On LibriSpeech test-other, Imputer achieves 11.1 WER, outperforming CTC at 13.0 WER and seq2seq at 12.5 WER.

Acknowledgements

We are grateful to Simon Kornblith, Sara Sabour, Saurabh Saxena, Yu Zhang and the Google Brain team for technical assistance and discussions.

References

Bahdanau, D., Cho, K., and Bengio, Y. Neural Machine Translation by Jointly Learning to Align and Translate. In *ICLR*, 2015.

Bahdanau, D., Chorowski, J., Serdyuk, D., Brakel, P., and Bengio, Y. End-to-End Attention-based Large Vocabulary Speech Recognition. In *ICASSP*, 2016a.

Bahdanau, D., Serdyuk, D., Brakel, P., Ke, N. R., Chorowski, J., Courville, A., and Bengio, Y. Task Loss Estimation for Sequence Prediction. In *ICLR Workshop*, 2016b.

Battenberg, E., Chen, J., Child, R., Coates, A., Gaur, Y., Li, Y., Liu, H., Satheesh, S., Seetapun, D., Sriram, A., and Zhu, Z. Exploring Neural Transducers for End-to-End Speech Recognition. In *ASRU*, 2017.

Chan, H., Kiros, J., and Chan, W. Multilingual KERMIT: Its Not Easy Being Generative. In *NeurIPS: Workshop on Perception as Generative Reasoning*, 2019a.

Chan, W., Jaitly, N., Le, Q., and Vinyals, O. Listen, Attend and Spell: A Neural Network for Large Vocabulary Conversational Speech Recognition. In *ICASSP*, 2016.

Chan, W., Zhang, Y., and Jaitly, N. Latent Sequence Decompositions. In *ICLR*, 2017.

Chan, W., Kitaev, N., Guu, K., Stern, M., and Uszkoreit, J. KERMIT: Generative Insertion-Based Modeling for Sequences. In *arXiv*, 2019b.

Chan, W., Stern, M., Kiros, J., and Uszkoreit, J. An Empirical Study of Generation Order for Machine Translation. In *arXiv*, 2019c.

Chen, N., Watanabe, S., Villalba, J., and Dehak, N. Non-Autoregressive Transformer Automatic Speech Recognition. In *arXiv*, 2019.

Chiu, C.-C., Sainath, T., Wu, Y., Prabhavalkar, R., Nguyen, P., Chen, Z., Kannan, A., Weiss, R. J., Rao, K., Gonina, E., Jaitly, N., Li, B., Chorowski, J., and Bacchiani, M. State-of-the-art Speech Recognition With Sequence-to-Sequence Models. In *ICASSP*, 2018.

Cho, K., van Merrienboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., and Bengio, Y. Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation. In *EMNLP*, 2014.

Chorowski, J. and Jaitly, N. Towards better decoding and language model integration in sequence to sequence models. In *INTERSPEECH*, 2017.

Collobert, R., Puhrsch, C., and Synnaeve, G. Wav2Letter: an End-to-End ConvNet-based Speech Recognition System. In *arXiv*, 2016.

Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *NAACL*, 2019.

Dong, L., Xu, S., and Xu, B. Speech-Transformer: A No-Recurrence Sequence-to-Sequence Model for Speech Recognition. In *ICASSP*, 2018.

Ghazvininejad, M., Levy, O., Liu, Y., and Zettlemoyer, L. Mask-Predict: Parallel Decoding of Conditional Masked Language Models. In *EMNLP*, 2019.

Graves, A. Sequence Transduction with Recurrent Neural Networks. In *ICML Representation Learning Workshop*, 2012.

Graves, A. and Jaitly, N. Towards End-to-End Speech Recognition with Recurrent Neural Networks. In *ICML*, 2014.

Graves, A., Fernandez, S., Gomez, F., and Schmidhuber, J. Connectionist Temporal Classification: Labelling Unsegmented Sequence Data with Recurrent Neural Networks. In *ICML*, 2006.

Graves, A., Mohamed, A., and Hinton, G. Speech Recognition with Deep Recurrent Neural Networks. In *ICASSP*, 2013.

Gu, J., Bradbury, J., Xiong, C., Li, V. O., and Socher, R. Non-Autoregressive Neural Machine Translation. In *ICLR*, 2018.

Gu, J., Liu, Q., and Cho, K. Insertion-based Decoding with Automatically Inferred Generation Order. In *arXiv*, 2019a.

Gu, J., Wang, C., and Zhao, J. Levenshtein Transformer. In *NeurIPS*, 2019b.

Irie, K., Prabhavalkar, R., Kannan, A., Bruguier, A., Rybach, D., and Nguyen, P. Model Unit Exploration for Sequence-to-Sequence Speech Recognition. In *arXiv*, 2019.

Kim, S., Hori, T., and Watanabe, S. Joint CTC-Attention based End-to-End Speech Recognition using Multi-task Learning. In *ICASSP*, 2017.

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

Lee, J., Mansimov, E., and Cho, K. Deterministic Non-Autoregressive Neural Sequence Modeling by Iterative Refinement. In *EMNLP*, 2018.

Li, J., Lavrukhin, V., Ginsburg, B., Leary, R., Kuchaiev, O., Cohen, J. M., Nguyen, H., and Gadde, R. T. Jasper: An End-to-End Convolutional Neural Acoustic Model. In *INTERSPEECH*, 2019.

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

Liptchinsky, V., Synnaeve, G., and Collobert, R. Letter-Based Speech Recognition with Gated ConvNets. In *arXiv*, 2017.

Liu, H., Zhu, Z., Li, X., and Satheesh, S. Gram-CTC: Automatic Unit Selection and Target Decomposition for Sequence Labelling. In *ICML*, 2017.

Luo, Y., Chiu, C.-C., Jaitly, N., and Sutskever, I. Learning Online Alignments with Continuous Rewards Policy Gradient. In *ICASSP*, 2016.

Luong, M.-T., Pham, H., and Manning, C. D. Effective Approaches to Attention-based Neural Machine Translation. In *EMNLP*, 2015.

Luscher, C., Beck, E., Irie, K., Kitza, M., Michel, W., Zeyer, A., Schluter, R., and Ney, H. RWTH ASR Systems for LibriSpeech: Hybrid vs Attention – w/o Data Augmentation. In *INTERSPEECH*, 2019.

Mohamed, A., Okhonko, D., and Zettlemoyer, L. Transformers with convolutional context for ASR. In *arXiv*, 2019.

Oord, A., Dieleman, S., Zen, H., Simonyan, K., Vinyals, O., Graves, A., Kalchbrenner, N., Senior, A., and Kavukcuoglu, K. WaveNet: A Generative Model for Raw Audio. In *arXiv*, 2016.

Oord, A., Li, Y., Babuschkin, I., Simonyan, K., Vinyals, O., Kavukcuoglu, K., van den Driessche, G., Lockhart, E., Cobo, L. C., Stimberg, F., Casagrande, N., Grewe, D., Noury, S., Dieleman, S., Elsen, E., Kalchbrenner, N., Zen, H., Graves, A., King, H., Walters, T., Belov, D., and Hassabis, D. Parallel WaveNet: Fast High-Fidelity Speech Synthesis. In *ICML*, 2018.

Panayotov, V., Chen, G., Povey, D., and Khudanpur, S. Librispeech: An ASR corpus based on public domain audio books. In *ICASSP*, 2015.

Park, D., Chan, W., Zhang, Y., Chiu, C.-C., Zoph, B., Cubuk, E., and Le, Q. SpecAugment: A Simple Data Augmentation Method for Automatic Speech Recognition. In *INTERSPEECH*, 2019.

Paul, D. and Baker, J. The Design for the Wall Street Journal-based CSR Corpus. In *Speech and Natural Language*, 1992.

Povey, D., Ghoshal, A., Boulianne, G., Burget, L., Glembek, O., Goel, N., Hannemann, M., Motlicek, P., Qian, Y., Schwarz, P., Silovsky, J., Stemmer, G., and Vesely, K. The Kaldi Speech Recognition Toolkit. In *ASRU*, 2011.

Prabhavalkar, R., Rao, K., Sainath, T. N., Li, B., Johnson, L., and Jaitly, N. A Comparison of Sequence-to-Sequence Models for Speech Recognition. In *INTERSPEECH*, 2017.

Ruis, L., Stern, M., Proskurnia, J., and Chan, W. Insertion-Deletion Transformer. In *EMNLP: Workshop of Neural Generation and Translation*, 2019.

Sabour, S., Chan, W., and Norouzi, M. Optimal Completion Distillation for Sequence Learning. In *ICLR*, 2019.

Sainath, T., Mohamed, A., Kingsbury, B., and Ramabhadran, B. Deep Convolutional Neural Networks for LVCSR. In *ICASSP*, 2013.

Serdyuk, D., Ke, N. R., Sordoni, A., Trischler, A., Pal, C., and Bengio, Y. Twin Networks: Matching the Future for Sequence Generation. In *ICLR*, 2018.

Stern, M., Chan, W., Kiros, J., and Uszkoreit, J. Insertion Transformer: Flexible Sequence Generation via Insertion Operations. In *ICML*, 2019.

Sutskever, I., Vinyals, O., and Le, Q. Sequence to Sequence Learning with Neural Networks. In *NIPS*, 2014.

Synnaeve, G., Xu, Q., Kahn, J., Grave, E., Likhomanenko, T., Pratap, V., Sriram, A., Liptchinsky, V., and Collobert, R. End-to-end ASR: from Supervised to Semi-Supervised Learning with Modern Architectures. In *arXiv*, 2019.

Tjandra, A., Sakti, S., and Nakamura, S. Sequence-to-Sequence ASR Optimization via Reinforcement Learning. In *ICASSP*, 2018.

Tuske, Z., Audhkasi, K., and Saon, G. Advancing Sequence-to-Sequence Based Speech Recognition. In *INTERSPEECH*, 2019.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., and Polosukhin, I. Attention Is All You Need. In *NIPS*, 2017.

Vinyals, O., Toshev, A., Bengio, S., and Erhan, D. Show and Tell: A Neural Image Caption Generator. In *CVPR*, 2015.

Welleck, S., Brantley, K., Daume, H., and Cho, K. Non-Monotonic Sequential Text Generation. In *ICML*, 2019.

Xu, K., Ba, J., Kiros, R., Cho, K., Courville, A., Salakhutdinov, R., Zemel, R., and Bengio, Y. Show, Attend and Tell: Neural Image Caption Generation with Visual Attention. In *ICML*, 2015.

Zeyer, A., Irie, K., Schlüter, R., and Ney, H. Improved training of end-to-end attention models for speech recognition. In *INTERSPEECH*, 2018a.

Zeyer, A., Merboldt, A., Schlüter, R., and Ney, H. A comprehensive analysis on attention models. In *NIPS: Workshop IRASL*, 2018b.

Zhang, Y., Chan, W., and Jaitly, N. Very Deep Convolutional Networks for End-to-End Speech Recognition. In *ICASSP*, 2017.