How to Do Simultaneous Translation Better with Consecutive Neural Machine Translation?

Yun Chen, Liangyou Li, Xin Jiang, Xiao Chen, Qun Liu
Huawei Noah’s Ark Lab, Hong Kong, China
{chenyun.nlp,liliangyou,jiang.xin,chen.xiao2,qun.liu}@huawei.com

Abstract
Despite the success of neural machine translation (NMT), simultaneous neural machine translation (SNMT), the task of translating in real time before a full sentence has been observed, remains challenging due to the syntactic structure difference and simultaneity requirements. In this paper, we propose a general framework to improve simultaneous translation with a pretrained consecutive neural machine translation (CNMT) model. Our framework contains two parts: prefix translation that utilizes a pretrained CNMT model to better translate source prefixes and a stopping criterion that determines when to stop the prefix translation. Experiments on three translation corpora and two language pairs show the efficacy of the proposed framework on balancing the quality and latency in simultaneous translation.

1 Introduction
Simultaneous translation (Fügen et al., 2007; Oda et al., 2014; Grissom et al., 2014; Cho and Esipova, 2016; Gu et al., 2017; Ma et al., 2018), the task of producing a partial translation of a sentence before the whole input sentence ends, is useful in many scenarios including outbound tourism, international summit and multilateral negotiations. Different from the consecutive translation in which translation quality alone matters, simultaneous translation trades off between translation quality and latency. The syntactic structure difference between the source and target language makes simultaneous translation more challenging. For example, when translating from a verb-final (SOV) language (e.g., Japanese) to a verb-media (SVO) language (e.g., English), the verb appears much later in the source sequence than in the target language. Some premature translations can lead to significant loss in quality (Ma et al., 2018).

Recently, a number of researchers have endeavored to explore methods for simultaneous translation in the context of NMT (Sutskever et al., 2014; Bahdanau et al., 2015; Vaswani et al., 2017; Gehring et al., 2017). Some of them propose sophisticated training frameworks explicitly designed for simultaneous translation (Ma et al., 2018; Arivazhagan et al., 2019). These approaches are either memory inefficient during training (Ma et al., 2018) or hard to implement (Arivazhagan et al., 2019). Others utilize a full-sentence base model to perform simultaneous translation by modifications to the encoder and the decoding process. To match the incremental source context, they replace the bidirectional encoder with a left-to-right encoder (Cho and Esipova, 2016; Satija and Pineau, 2016; Gu et al., 2017; Alinejad et al., 2018) or recompute the encoder hidden states (Zheng et al., 2019). On top of that, heuristic algorithms (Cho and Esipova, 2016; Dalvi et al., 2018) or a READ/WRITE model trained with reinforcement learning (Satija and Pineau, 2016; Gu et al., 2017; Alinejad et al., 2018) or supervised learning (Zheng et al., 2019) are used to decide, at every step, whether to wait for the next source token or output a target token. However, these models either cannot directly use a pretrained vanilla CNMT model with bidirectional encoder as the base model or work in a sub-optimal way in the decoding stage.

In this paper, we study the problem of how to do simultaneous translation better with a pretrained vanilla CNMT model. We formulate simultaneous translation as two nested loops: an outer loop that updates input buffer with newly observed source tokens and an inner loop that translates source tokens in the buffer updated at each outer step. For the outer loop, the input buffer can be updated by an ASR system with an arbitrary update schedule. For the inner loop, we perform prefix translation using the pretrained CNMT model with dynami-
where $\phi$ is a set of model parameters. At inference time, the NMT model first encodes a source language sentence $x = \{x_1, ..., x_{T_x}\}$ with its encoder and passes the encoded representations $h = \{h_1, ..., h_{T_h}\}$ to a greedy decoder. Then the greedy decoder generates a translated sentence in the target language by sequentially choosing the most likely token at each step $t$:

$$y_t = \text{argmax}_y p(y|y_{<t}, x).$$  \tag{2}$$

The distribution of next target word is defined as:

$$p(y|y_{<t}, x) \propto \exp[\phi_{\text{OUT}}(z_t)]$$

$$z_t = \phi_{\text{DEC}}(y_{t-1}, z_{<t}, h),$$  \tag{3}$$

where $z_t$ is the decoder hidden state at position $t$. In consecutive NMT, once obtained, the encoder hidden states $h$ and the decoder hidden state $z_t$ are not updated anymore and will be reused during the entire decoding process.

3 Simultaneous NMT

In SNMT, we receive streaming input tokens, and learn to translate them in real-time. We formulate simultaneous translation as two nested loops: the outer loop that updates an input buffer with newly observed source tokens and the inner loop that translates source tokens in the buffer updated at each outer step.

More precisely, suppose at the end of an outer step $s-1$, the input buffer is $x^{s-1} = \{x_1, ..., x_{\eta[s-1]}\}$, and the output buffer is $y^{s-1} = \{y_1, ..., y_{\tau[s-1]}\}$. Then at outer step $s$, the system translates with the following steps:

1. The system observes $c_s > 0$ new source tokens and updates the input buffer to be $x^s = \{x_1, ..., x_{\eta[s]}\}$ where $\eta[s] = \eta[s-1] + c_s$.

2. Then, the system starts inner loop translation and writes $w_s >= 0$ target tokens to the output buffer. The output buffer is updated to be $y^s = \{y_1, ..., y_{\tau[s]}\}$ where $\tau[s] = \tau[s-1] + w_s$.

The simultaneous decoding process continues until no more source tokens are added in the outer loop. We define the last outer step as the terminal outer step $S$, and other outer steps as non-terminal outer steps.

For the outer loop, we make no assumption about the value of $c_s$, while all previous work assumes $c_s = 1$. This setting is more realistic because a) increasing $c_s$ can reduce the number of outer steps, thus reducing computation cost; b) in a real speech translation application, an ASR system may generate multiple tokens at a time.

For the inner loop, we adapt a pretrained vanilla CNMT model to perform partial translation with two important concerns:

1. Prefix translation: given a source prefix $x^s = \{x_1, ..., x_{\eta[s]}\}$ and a target prefix $y^s_{\tau[s-1]} = \{y_1, ..., y_{\tau[s-1]}\}$, how to predict the remaining target tokens?

2. Stopping criterion: since the NMT model is trained with full sentences, how to design the stopping criterion for it when translating partial source sentences?

3.1 Prefix Translation

At an outer step $s$, given encoder hidden states $h^s$ for source prefix $x^s = \{x_1, ..., x_{\eta[s]}\}$ and decoder hidden states $z^s_{\tau[s-1]}$ for target prefix $y^s_{\tau[s-1]} = \{y_1, ..., y_{\tau[s-1]}\}$, we perform prefix translation sequentially with a greedy decoder:

$$z_t^s = \phi_{\text{DEC}}(y_{t-1}, z_{<t}, h^s)$$

$$p(y|y_{<t}, x^s) \propto \exp[\phi_{\text{OUT}}(z_t^s)]$$

$$y_t = \text{argmax}_y p(y|y_{<t}, x^s),$$  \tag{4}$$
where \( t \) starts from \( t = \tau \cdot [s - 1] + 1 \). The prefix translation terminates when a stopping criterion meets, yielding a translation \( Y^s = \{ y_1, ..., y_{\tau[s]} \} \).

However, a major problem comes from the above translation method: how can we obtain the encoder hidden states \( h^s \) and decoder hidden states \( z_{\tau[s-1]} \) at the beginning of prefix translation? In CNMT, the encoder hidden states and previous decoder hidden states are reused at each decoding time step. Different from CNMT, SNMT is fed with an incremental source side context. On the encoder side, we can address this by either reusing previous encoder hidden states (Cho and Esipova, 2016; Gu et al., 2017; Dalvi et al., 2018; Alinejad et al., 2018):

\[
h^s = \phi_{\text{ENC}} (\{ x_{\eta[s-1]} + t \}_{t=1}^v, h^{s-1}) \tag{5}
\]
or dynamically re-building all encoder hidden states (Ma et al., 2018):

\[
h^s = \phi_{\text{ENC}} (x^s). \tag{6}
\]

On the decoder side, since the encoder hidden states have been updated from \( h^{s-1} \) to \( h^s \), we can choose to reuse previous decoder hidden states (Cho and Esipova, 2016; Gu et al., 2017; Dalvi et al., 2018; Ma et al., 2018):

\[
z_{\tau[s-1]}^s = z_{s-1} \tag{7}
\]
or rebuild all previous decoder hidden states from current encoder hidden states \( h^s \) with force decoding:

\[
z_{\tau[s-1]}^s = \phi_{\text{DEC}} (y_{\tau[s-1]}^s, h^s). \tag{8}
\]

To better predict the remaining target tokens, we rebuild all encoder and decoder hidden states following Eq. 6 and 8 at the beginning of prefix translation. This strategy ensures that all encoder and decoder hidden states are obtained by attending to the same source tokens, which is consistent with how encoder and decoder hidden states are computed at training time. Besides, these attainable source tokens are all available source context at current time. Compared with using Eq. 5 or 7, our method can potentially better utilize the available source context.

### 3.2 Stopping Criterion

In consecutive NMT, the decoding algorithm such as greedy decoding or beam search terminates when the translator predicts an EOS token or the length of the translation meets a predefined threshold:

\[
\min \{ \text{maxlen}, u \cdot \|x\| + v \}, \tag{9}
\]

where \( \text{maxlen} \), \( u \) and \( v \) are all hyper-parameters. In fairseq-py\(^1\), they set \( \text{maxlen} = +\infty \), \( u = 0 \) and \( v = 200 \) at inference time by default. The decoding for most source sentences terminates when the translator predicts the EOS token.\(^2\) In simultaneous decoding, since we use a NMT model pretrained on full sentences to translate partial source sentences, it tends to predict EOS when the source context has been fully translated. However, such strategy could be too aggressive for simultaneous translation. Fig. 1 shows such an example. At outer step 2, the translator predicts “you EOS”, emitting target token “you”. However, “you” is not the expected translation for “you”, in the context of “你好”。

The right decision is that prefix translation at outer step 2 should stop without emitting any words.

To alleviate such problems and do better simultaneous translation with pretrained CNMT model, we propose two novel stopping criteria for prefix translation.

#### 3.2.1 Length and EOS Control

In consecutive translation, the decoding process stops mainly when predicting EOS. In contrast, for prefix translation at non-terminal outer step, we use both length and EOS to stop the prefix translation process. We achieve this by setting the hyper-parameters in Eq. 9 as \( \text{maxlen} = +\infty \), \( u = 1 \) and \( v = -d \), where \( d \) is a non-negative integer. The hyper-parameter \( d \) determines the translation latency of the system.

More specifically, before prefix translation at outer step \( s \), we have source prefix \( x^s = \{ x_1, ..., x_{\eta[s]} \} \) and target prefix \( y_{\tau[s-1]}^s = \{ y_1, ..., y_{\tau[s-1]} \} \). Prefix translation terminates at inner step \( w_s \) when predicting an EOS token or

\(^1\)https://github.com/pytorch/fairseq
\(^2\)We conduct greedy decoding on the validation set of WMT15 EN→DE translation with fairseq-py, and find that 100% translation terminates with EOS predicted.
satisfying:

\[ w_s = \begin{cases} \max(0, \eta [s] - \tau [s - 1] - d) & s < S \\ 200 - \tau [s - 1] & s = S \end{cases} \cdot (10) \]

We call this stopping criterion as Length and EOS (LE) stopping controller.

### 3.2.2 Learning When to Stop

Although simple and easy to implement, LE controller lacks the capability to learn the optimal timing with which to stop prefix translation. Therefore, we design a small trainable network called Trainable (TN) stopping controller to learn when to stop prefix translation for non-terminal outer step. Fig. 2 shows the illustration.

At each inner decoding step \( k \) for non-terminal outer step \( s \), the TN controller utilizes a stochastic policy \( \pi_\theta \) parameterized by a neural network to make the binary decision on whether to stop translation at current stage:

\[
\pi_\theta(a_{τ[s-1]+k} | z^s_{τ[s-1]+k}) = f_\theta(z^s_{τ[s-1]+k}),
\]

(11)

where \( z^s_{τ[s-1]+k} \) is the current decoder hidden state. The prefix translation stops if the TN controller predicts \( a_{τ[s-1]+k} = 1 \). The controller function \( f_\theta \) can take on a variety of forms, and for simplicity we implement with a feedforward network with two hidden layers, followed by a softmax layer.

To train the TN controller, we freeze the NMT model with pretrained parameters, and optimize the TN network with policy gradient for reward maximization \( J = \mathbb{E}_{\pi_\theta} \left( \sum_{t=1}^{T_e} r_t \right) \). With a trained TN controller, prefix translation stops at inner decoding step \( w_s \) when predicting an EOS token or satisfying:

\[
\begin{cases}
  a_{τ[s-1]+w_s} = 1 & s < S \\
  w_s = 200 - \tau [s - 1] & s = S
\end{cases}
\]

(12)

In the following, we talk about the details of the reward function and the training detail with policy gradient.

#### Reward

To trade-off between translation quality and latency, we define the reward function at inner decoding step \( k \) of outer step \( s \) as:

\[
r_t = r^Q_t + \alpha \cdot r^P_t,
\]

(13)

where \( t = τ [s - 1]+k \), and \( r^Q_t \) and \( r^P_t \) are rewards related to quality and delay, respectively. \( \alpha \geq 0 \) is a hyper-parameter that we adjust to balance the trade-off between translation quality and delay. Similar to Gu et al. (2017), we utilize sentence-level BLEU (Papineni et al., 2002; Lin and Och, 2004) with reward shaping (Ng et al., 1999) as the reward for quality:

\[
r^Q_t = \begin{cases}
  \Delta \text{BLEU}(y^*, y, t) & k \neq w_s \text{ or } s \neq S \\
  \text{BLEU}(y^*, y) & k = w_s \text{ and } s = S
\end{cases}
\]

(14)

where

\[
\Delta \text{BLEU}(y^*, y, t) = \text{BLEU}(y^*, y_t) - \text{BLEU}(y^*, y_{t-1})
\]

(15)

is the intermediate reward. Note that the higher the values of BLEU are, the more rewards the TN controller receives.

Following Gu et al. (2017); Ma et al. (2018), we use average lagging (AL) as the reward for latency:

\[
r^P_t = \begin{cases}
  0 & k \neq w_s \text{ or } s \neq S \\
  -|d^{AL}(x, y) - d^*|_+ & k = w_s \text{ and } s = S
\end{cases}
\]

(16)

where

\[
d^{AL}(x, y) = \frac{1}{t_e} \sum_{t=1}^{T_e} l(t) - \frac{t - 1}{\lambda}
\]

(17)

\( l(t) \) is the number of observed source tokens when generating the \( t \)-th target token, \( t_e = \arg\min_t (l(t) = |x|) \) denotes the earliest point when the system observes the full source sentence, \( \lambda = \frac{|y|}{|x|} \) represents the target-to-source length ratio and \( d^* \geq 0 \) is a hyper-parameter called target delay that indicates the desired system latency. Note that the lower the values of AL are, the more rewards the TN controller receives.

#### Policy Gradient

We train the TN controller with policy gradient(Sutton et al., 1999), and the gradients are:

\[
\nabla_\theta J = \mathbb{E}_{\pi_\theta} \left[ \sum_{t=1}^{T_e} R_t \nabla_\theta \log \pi_\theta(a_t | \cdot) \right]
\]

(18)
Table 1: # sentences in each dataset.

| Dataset   | Train | Validation | Test |
|-----------|-------|------------|------|
| IWSLT16   | 193,591 | 993 | 1,305 |
| WMT15     | 3,745,796 | 3,003 | 2,169 |
| NIST      | 1,252,977 | 878 | 4,103 |

where $R_t = \sum_{i=1}^{T_t} r_i$ is the cumulative future rewards for the current decision. We can adopt any sampling approach to estimate the expected gradient. In our experiments, we randomly sample multiple action trajectories from the current policy $\pi_\theta$ and estimate the gradient with the collected accumulated reward. We try the variance reduction techniques by subtracting a baseline average reward estimated by a linear regression model from $R_t$ and find that it does not help to improve the performance. Therefore, we just normalize the reward in each mini batch without using baseline reward for simplicity.

### 4 Experiments

#### 4.1 Settings

**Dataset** We compare our approach with the baselines on WMT15 German-English\(^3\) (DE-EN) translation in both directions. This is also the most widely used dataset to evaluate SNMT’s performance (Cho and Esipova, 2016; Gu et al., 2017; Ma et al., 2018; Arivazhagan et al., 2019; Zheng et al., 2019). To further evaluate our approach’s efficacy in trading off translation quality and latency on other language pair and spoken language, we also conduct experiments with the proposed LE and TN method on NIST Chinese-to-English\(^4\) (ZH→EN) translation and IWSLT16 German-English\(^5\) (DE-EN) translation in both directions. For WMT15, we use newstest2014 for validation and newstest2015 for test. For NIST, we use MT02 for validation, and MT05, MT06, MT08 for test. For IWSLT16, we use tst13 for validation and tst14 for test. Table 1 shows the details. All the data is tokenized and segmented into subword symbols using byte-pair encoding (Sennrich et al., 2016) to restrict the size of the vocabulary. We use 40,000 joint merge operations on WMT15, and 24,000 on IWSLT16. For NIST, we use 30,000 merge operations for source and target side separately. Without explicitly mention, we simulate simultaneous translation scenario at inference time with these datasets by assuming that the system observes one new source token at each outer step, i.e., $c_o = 1$.

**Pretrained NMT Model** We use Transformer (Vaswani et al., 2017) trained with maximum likelihood estimation as the pretrained CNMT model and implement our method based on fairseq-py.\(^6\) We follow the setting in transformer\_iwslt\_de\_en for IWSLT16 dataset, and transformer\_wmt\_en\_de for WMT15 and NIST dataset. Fairseq-py adds an EOS token for all source sentences during training and inference. Therefore, to be consistent with the CNMT model implemented with fairseq-py, we also add an EOS token at the end of the source prefix for prefix translation.

**TN Controller** To train the TN controller, we use a mini-batch size of 8, 16, 16 and sample 5, 10, 10 trajectories for each sentence pair in a batch for IWSLT16, WMT15 and NIST, respectively. We set the number of newly observed source tokens at each outer step to be 1 during the training for simplicity. We set $\alpha$ to be 0.04, and $d^*$ to be 2, 5, 8. All our TN controllers are trained with policy gradient using Adam optimizer (Kingma and Ba, 2015) with 30,000 updates. We select the last model as our final TN controller.

**Baseline** We compare our model against three baselines that utilize a pretrained CNMT model to perform simultaneous translation:

- **test\_time\_waitk:** the test-time waitk simultaneous decoding algorithm proposed in Ma et al. (2018), i.e., using a full-sentence model but decoding it with a waitk policy. We report the results when $k = 1, 3, 5, 7, 9$.

- **SL:** the SL model proposed in Zheng et al. (2019), which learns an adaptive READ/WRITE policy from oracle READ/WRITE sequences generated with heuristics. We report the results $\rho = 0.65, 0.6, 0.55, 0.5, 0.45, 0.4$.

- **Gu et al. (2017):** the adaptation of Gu et al. (2017)’s two-staged full-sentence model + reinforcement learning on Transformer by Ma

---

3http://www.statmt.org/wmt15/

4These sentence pairs are mainly extracted from LDC2002E18, LDC2003E07, LDC2003E14, Hansards portion of LDC2004T07, LDC2004T08 and LDC2005T06

5https://workshop2016.iwslt.org/

6https://github.com/pytorch/fairseq
Figure 3: Comparison with the baselines on the test set of WMT15 EN→DE translation and WMT15 DE→EN translation. The shown points from left to right on the same line are the results of simultaneous greedy decoding by increasing $d^*$ following $2, 5, 8$ for TN, $d$ following $0, 2, 4, 6, 8$ for LE, $\rho$ following $0.65, 0.6, 0.55, 0.5, 0.45, 0.4$, $k$ following $1, 3, 5, 7, 9$ for test_time_wait, and $CW$ following $2, 5, 8$ for Gu et al. (2017). The scores of Greedy decoding: $BLEU = 25.16, AL = 28.10$ for WMT15 EN→DE translation and $BLEU = 26.17, AL = 31.20$ for WMT15 DE→EN translation.

Figure 4: Performance on the test set of IWSLT16 EN→DE translation, IWSLT16 DE→EN translation and NIST ZH→EN translation. The shown points from left to right on the same line are the results of simultaneous greedy decoding by increasing $d^*$ for TN and $d$ for LE. $\star \star$: full-sentence (greedy and beam-search).

Figure 5: Comparison of whether to reuse previous encoder or decoder hidden states on WMT15 EN→DE test set with the LE controller. The left Y axis is the BLEU score and the right Y axis is the length ratio: the translation length divided by the reference length. none: rebuild all encoder/decoder hidden states (Eq. 6+Eq. 8); decoder: reuse decoder hidden states and rebuild all encoder hidden states (Eq. 6+Eq. 7); encoder: reuse previous encoder hidden states and rebuild all decoder hidden states (Eq. 5+Eq. 8). The points from left to right on the same line are the results by increasing $d$ following $0, 2, 4, 6, 8$. 
et al. (2018). We report the results when using $CW = 2, 5, 8$ as the target delay.

We report the results when $d = 0, 2, 4, 6, 8$ for our proposed LE method and $d^* = 2, 5, 8$ for our proposed TN method. For all baselines, we cite the results reported in Zheng et al. (2019). Since they did not mention the details of data preprocessing, we cannot compare the BLEU and AL scores directly with theirs. Therefore, we normalize the BLEU and AL scores with its corresponding upper bound, i.e. the BLEU and AL scores obtained when the pretrained Transformer performs standard greedy decoding (Greedy).

### 4.2 Results

We compare our method with the baselines on the test set of WMT15 EN→DE and DE→EN translation tasks. Fig. 3 shows the result. The points closer to the upper left corner indicate better overall performance, namely low latency and high quality. In all these figures, we observe that, as latency increases, all methods improve in quality. The TN stopping controller significantly outperforms all the baseline systems in both translation tasks, demonstrating that it indeed learns the appropriate timing to stop prefix translation. The LE controller outperforms the baselines on WMT15 EN→DE translation at high latency region and performs similarly or worse on other cases.

We show the model’s efficacy in trading off quality and latency on other language pair and spoken language in Fig. 4. The TN controller obtains better performance on all translation tasks, especially at the low latency region. For example, on IWSLT16 EN→DE translation, it is +2.5 to +3.3 BLEU ahead of the LE method. TN also obtains promising translation quality with acceptable latency: with a lag of < 7 tokens, TN obtains 96.95%, 97.20% and 94.03% BLEU with respect to consecutive greedy decoding for IWSLT16 EN→DE, IWSLT16 DE→EN and NIST ZH→EN translation, respectively.

### 4.3 Analyze

We analyze the effect of different ways (Eq. 5-8) to obtain the encoder and decoder hidden states at the beginning of prefix translation with the LE controller. Fig. 5 shows the result. We try three variants: a) dynamically rebuild all encoder/decoder hidden states (none); b) reuse decoder hidden states and rebuild all encoder hidden states (decoder); c) reuse previous encoder hidden states and rebuild all decoder hidden states (encoder). The left Y axis and X axis show BLEU-vs-AL curve. We observe that if reusing previous encoder hidden states (encoder), the translation fails. We ascribe this to the discrepancy between training and decoding for the encoder. We also observe that when $d = 0, 2$, reusing decoder hidden states (decoder) obtain negative AL. To analyze this, we plot the translation to reference length ratio versus AL curve with the right Y axis and X axis. It shows that with decoder, the decoding process stops too early and generates
too short translations. Therefore, to avoid such problem and to be consistent with the training process of the CNMT model, it is important to dynamically rebuild all encoder/decoder hidden states for prefix translation.

Since we make no assumption about the \( c_s \), i.e., the number of newly observed source tokens at each outer step, we test the effect of different \( c_s \) at this section. Fig. 6 shows the result with the LE and TN controllers on the test set of WMT15 EN→DE translation. We observe that as \( c_s \) increases, both LE and TN trend to improve in quality and worsen in latency. When \( c_s = 1 \), LE controller obtains the best balance between quality and latency. In contrast, TN controller obtains similar quality and latency balance with different \( c_s \), demonstrating that TN controller successfully learns the right timing to stop regardless of the input update schedule. We also analyze the TN controller’s adaptability by monitoring the initial delay, i.e., the number of observed source tokens before emitting the first target token, on the test set of WMT15 EN→DE translation, as shown in Fig. 7. \( d^* \) is the target delay measured with AL (used in Eq. 16). It demonstrates that the TN controller has a lot of variance in its initial delay. The distribution of initial delay changes with different target delay: with higher target delay, the average initial delay is larger. For most sentences, the initial delay is within 1 – 7.

In speech translation, listeners are also concerned with long silences during which no translation occurs. Following Gu et al. (2017); Ma et al. (2018), we use Consecutive Wait (CW) to measure this:

\[
CW(x, y) = \frac{\sum_{s=1}^{S} c_s}{\sum_{s=1}^{S} \sum_{w_s > 0} w_s}.
\]

Fig. 8 shows the BLEU-vs-CW plots for our proposed two algorithms. The TN controller has higher CW than the LE controller. This is because TN controller prefers consecutive updating output buffer (e.g., it often produces \( w_s \) as 0 0 0 3 0 0 0 5 0 0 0 4 ...) while the LE controller often updates its output buffer following the input buffer (e.g., it often produces \( w_s \) as 0 0 0 1 1 1 1 1 ... when \( d = 4 \)). Although larger than LE, the CW for TN (< 6) is acceptable for most speech translation scenarios.

4.4 Translation Examples

Fig. 9 shows three translation examples with the LE and TN controllers on the test set of NIST ZH→EN and WMT15 EN→DE translation. In manual inspection of these examples and others, we find that the TN controller learns a conservative timing for stopping prefix translation. For example, in example 2, our method outputs translation “wu bangguo attended the signing ceremony” when observing “吴邦国出席签字仪式并”，instead of a more radical translation “wu bangguo attended the signing ceremony and”. Such strategy helps to alleviate the problem of premature translation, i.e., translating before observing enough future context.

5 Related Work

A number of works in simultaneous translation divide the translation process into two stages. A segmentation component first divides the incoming text into segments, and then each segment is
translated by a translator independently or with previous context. The segmentation boundaries can be predicted by prosodic pauses detected in speech (Fügen et al., 2007; Bangalore et al., 2012), linguistic cues (Sridhar et al., 2013; Matusov et al., 2007), or a classifier based on alignment information (Yarmohammadi et al., 2013; Siahbani et al., 2014) and translation accuracy (Oda et al., 2014; Grissom et al., 2014; Siahbani et al., 2018).

Some authors have recently endeavored to perform simultaneous translation in the context of NMT. Cho and Esipova (2016); Dalvi et al. (2018); Ma et al. (2018) introduce a manually designed criterion to control when to translate. Satija and Pineau (2016); Gu et al. (2017); Alinejad et al. (2018) extend the criterion into a trainable agent in a reinforcement learning framework. However, these work either develop sophisticated training frameworks explicitly designed for simultaneous translation (Ma et al., 2018) or fail to use a pretrained consecutive NMT model in an optimal way (Cho and Esipova, 2016; Dalvi et al., 2018; Satija and Pineau, 2016; Gu et al., 2017; Alinejad et al., 2018; Zheng et al., 2019). In contrast, our work is significantly different from theirs in the way of using pretrained consecutive NMT model to perform simultaneous translation and the design of the two stopping criteria.

6 Conclusion

We have presented a novel framework for improving simultaneous translation with a pretrained consecutive NMT model. The basic idea is to translate partial source sentence with the pretrained consecutive NMT model and stops the translation with two novel stopping criteria. Extensive experiments demonstrate that our method outperforms the state-of-the-art baselines in balancing between translation quality and latency.

References

Ashkan Alinejad, Maryam Siahbani, and Anoop Sarkar. 2018. Prediction improves simultaneous neural machine translation. In EMNLP.

Naveen Arivazhagan, Colin Cherry, Wolfgang Macherey, Chung-Cheng Chiu, Semih Yavuz, Ruoming Pang, Wei Li, and Colin Raffel. 2019. Monotonic infinite lookback attention for simultaneous machine translation. ArXiv, abs/1906.05218.

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2015. Neural machine translation by jointly learning to align and translate. In Proceedings of ICLR.

Srinivas Bangalore, Vivek Kumar Rangarajan Sridhar, Prakash Kolan, Ladan Goliipour, and Aura Jimenez. 2012. Real-time incremental speech-to-speech translation of dialogs. In HLT-NAACL.

Kyunghyun Cho and Masha Esipova. 2016. Can neural machine translation do simultaneous translation. CoRR, abs/1606.02012.

Fahim Dalvi, Nadir Durranı, Hassan Sajjad, and Stephan Vogel. 2018. Incremental decoding and training methods for simultaneous translation in neural machine translation. In NAACL-HLT.

Christian Fügen, Alexander H. Waibel, and Muntsin Kolss. 2007. Simultaneous translation of lectures and speeches. Machine Translation, 21:209–252.

Jonas Gehring, Michael Auli, David Grangier, Denis Yarats, and Yann Dauphin. 2017. Convolutional sequence to sequence learning. In ICML.

Alvin Grissom, He He, Jordan L. Boyd-Graber, John Morgan, and Hal Daumé. 2014. Don’t until the final verb wait: Reinforcement learning for simultaneous machine translation. In EMNLP.
Jiatao Gu, Graham Neubig, Kyunghyun Cho, and Victor O. K. Li. 2017. Learning to translate in real-time with neural machine translation. In EACL.

Diederik P. Kingma and Jimmy Ba. 2015. Adam: A method for stochastic optimization. CoRR, abs/1412.6980.

Chin-Yew Lin and Franz Josef Och. 2004. Automatic evaluation of machine translation quality using longest common subsequence and skip-bigram statistics. In ACL.

Mingbo Ma, Liang Huang, Hao Xiong, Kaibo Liu, Chuanqiang Zhang, Zhongjun He, Hairong Liu, Xing Li, and Haifeng Wang. 2018. Stacl: Simultaneous translation with integrated anticipation and controllable latency. CoRR, abs/1810.08398.

Evgeny Matusov, Dustin Hillard, Mathew Magimai-Doss, Dilek Z. Hakkani-Tür, Mari Ostendorf, and Hermann Ney. 2007. Improving speech translation with automatic boundary prediction. In INTERSPEECH.

Andrew Y. Ng, Daishi Harada, and Stuart J. Russell. 1999. Policy invariance under reward transformations: Theory and application to reward shaping. In ICML.

Yusuke Oda, Graham Neubig, Sakriani Sakti, Tomoki Toda, and Satoshi Nakamura. 2014. Optimizing segmentation strategies for simultaneous speech translation. In ACL.

Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. Bleu: a method for automatic evaluation of machine translation. In Proceedings of 40th Annual Meeting of the Association for Computational Linguistics, pages 311–318, Philadelphia, Pennsylvania, USA. Association for Computational Linguistics.

Harsh Satija and Joelle Pineau. 2016. Simultaneous machine translation using deep reinforcement learning. Abstraction in Reinforcement Learning Workshop, ICML.

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

Maryam Siahbani, Ramtin Mehdizadeh Seraj, Baskaran Sankaran, and Anoop Sarkar. 2014. Incremental translation using hierarchichal phrase-based translation system. 2014 IEEE Spoken Language Technology Workshop (SLT), pages 71–76.

Maryam Siahbani, Hassan Shavarani, Ashkan Alinejad, and Anoop Sarkar. 2018. Simultaneous translation using optimized segmentation. In AMTA.

Vivek Kumar Rangarajan Sridhar, John Chen, Srinivas Bangalore, Andrej Ljolje, and Rathinavelu Chengalvarayan. 2013. Segmentation strategies for streaming speech translation. In HLT-NAACL.

Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. 2014. Sequence to sequence learning with neural networks. In NIPS.

Richard S. Sutton, David A. McAllester, Satinder P. Singh, and Yishay Mansour. 1999. Policy gradient methods for reinforcement learning with function approximation. In NIPS.

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.

Mahsa Yarmohammadi, Vivek Kumar Rangarajan Sridhar, Srinivas Bangalore, and Baskaran Sankaran. 2013. Incremental segmentation and decoding strategies for simultaneous translation. In IJCNLP.

Baigong Zheng, Renjie Zheng, Mingbo Ma, and Liang Huang. 2019. Simpler and faster learning of adaptive policies for simultaneous translation. In Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP), pages 1349–1354, Hong Kong, China. Association for Computational Linguistics.