Towards Stream Translation: Adaptive Computation Time for Simultaneous Machine Translation

Felix Schneider
Karlsruhe Institute of Technology
felix.schneider@kit.edu

Alexander Waibel
Karlsruhe Institute of Technology
alexander.waibel@kit.edu

Abstract

Simultaneous machine translation systems rely on a policy to schedule read and write operations in order to begin translating a source sentence before it is complete. In this paper, we demonstrate the use of Adaptive Computation Time (ACT) as an adaptive, learned policy for simultaneous machine translation using the transformer model and as a more numerically stable alternative to Monotonic Infinite Lookback Attention (MILk). We achieve state-of-the-art results in terms of latency-quality tradeoffs. We also propose a method to use our model on unsegmented input, i. e. without sentence boundaries, simulating the condition of translating output from automatic speech recognition. We present first benchmark results on this task.

1 Introduction

Simultaneous machine translation (MT) must accomplish two tasks: First, it must deliver correct translations on incomplete input as early as possible, i. e. before the source sentence is completely spoken. Second, in a realistic usage scenario, it must deal with unsegmented input, either speech directly or automatic transcriptions without punctuation or sentence boundaries. Until now, staged models (Niehues et al., 2016), which have a separate component to insert punctuation (Cho et al., 2012) achieved the best results in this task. In this paper, we will present the first step towards an end-to-end approach.

In recent years, a number of approaches for neural simultaneous machine translation have been proposed. They generally build on the common encoder-decoder framework (Sutskever et al., 2014), with the decoder deciding at each step whether to output a target language token based on the currently available information (WRITE) or to wait for one more encoder step in order to have more information available (READ).

In order to do this, the decoder relies on a *wait policy*. The published policies can be broadly divided into two categories:

- Fixed policies, which rely on pre-programmed rules to schedule the read and write operations, such as wait-k (Ma et al., 2019a) and wait-if (Cho and Esipova, 2016).
- Learned policies, which are trained either jointly with the translation model or separately. Examples include MILk (Arivazhagan et al., 2019) and the models of Satija and Pineau (2016) and Alinejad et al. (2018)

However, all of the above approaches train and evaluate their models on individual sentences. We want to work towards a translation system that can work on a continuous stream of input, such as text without punctuation and sentence segmentation. In a realistic usage scenario, segmentation information is not available and an end-to-end solution without a separate segmentation component is desirable. We therefore propose the use of Adaptive Computation Time (Graves, 2016) for simultaneous machine translation. This method achieves a better latency-quality trade-off than the previous best model, MILk, on segmented WMT 2014 German-to-English data. By extending this model with Transformer-XL-style memory (Dai et al., 2019), we are able to apply it directly to unsegmented text.

2 Background

As Arivazhagan et al. (2019) point out, most previous work in simultaneous machine translation focuses on segmenting continuous input into parts that can be translated, whether it is utterances speech or sentences for text (Cho et al., 2012, 2017;

Fügen et al., 2007; Oda et al., 2014; Yarmohammadi et al., 2013). For statistical machine translations, some approaches for stream translation without segmentation were known (Kolss et al., 2008). The more recent neural simultaneous MT approaches simply take this segmentation as given and focus on translating simultaneously within a sentence.

Several approaches (Grissom II et al., 2014; Niehues et al., 2018; Alinejad et al., 2018) try to predict the whole target sentence in advance, before the input is complete. It may be possible to extend such approaches to work on an input stream, but they have the undesirable property of overriding their old output, which can make reading the translation difficult to follow for a human.

Satija and Pineau (2016) train the wait policy as an agent with reinforcement learning, considering the pre-trained and fixed MT system as part of the environment. Such an agent could learn to also predict the end of sentences and thus extend to stream translation, but it would be effectively the same as an explicit segmentation.

Cho and Esipova (2016) and Ma et al. (2019a) each define their own fixed policy for simultaneous MT. Wait-k in particular is attractive because of its simplicity and ease of training. However, we believe that for very long input streams, an adaptive policy is necessary to make sure that the decoder never "falls behind" the input stream.

Most recently, the best results are produced by monotonic attention approaches (Raffel et al., 2017; Chiu and Raffel, 2017), in particular Arivazhagan et al. (2019). Their approach uses RNNs, whereas we would like to use the state-of-the-art Transformer architecture (Vaswani et al., 2017). Unfortunately, we were unable to transfer their results to the Transformer, largely due to numerical instability problems. Ma et al. (2019b) claim to have done this, but we were unable to reproduce their results either. We therefore propose our own, more stable, architecture based on Adaptive Computation Time (ACT, Graves (2016))

3 Model

A machine translation model transforms a source sequence $\mathrm{x} = \{x_1, x_2, \dots x_{|\mathrm{x}|}\}$ into a target sequence $\mathrm{y} = \{y_1, y_2, \dots y_{|\mathrm{y}|}\}$, where, generally, $|\mathrm{x}| \neq |\mathrm{y}|$. Our model is based on the Transformer model (Vaswani et al., 2017), consisting of an encoder and a decoder. The encoder produces a vector representation for each input token, the decoder autoregressively produces the target sequence. The decoder makes use of the source information via an attention mechanism (Bahdanau et al., 2015), which calculates a context vector from the encoder hidden states.

$$h_{1\dots|\mathrm{x}|} = \textsc{Encoder}(x_{1\dots|\mathrm{x}|}) \quad (1)$$
$$c_i = \textsc{Attention}(y_{i-1}, h_{1\dots|\mathrm{x}|}) \quad (2)$$
$$y_i = \textsc{Decoder}(y_{i-1}, c_i) \quad (3)$$

In the offline case, the encoder has access to all inputs at once and the attention has access to all encoder hidden states. The standard soft attention calculates the context vector as a linear combination of all hidden states:

$$e_i^n = \textsc{Energy}(y_{i-1}, h_n) \quad (4)$$
$$w_i^n = \frac{\exp(e_i^n)}{\sum_{k=1}^{|\mathrm{x}|} \exp(e_i^k)} \quad (5)$$
$$c_i = \sum_{n=1}^{|\mathrm{x}|} w_i^k h_n \quad (6)$$

Here, Energy could be a multi-layer perceptron or, in the case of Transformer, a projection followed by a dot product.

In the simultaneous case, there are additional constraints: Each encoder state must only depend on the representations before it and the inputs up to the current one as input becomes available incrementally. In addition, we require a *wait policy* which decides in each step whether to READ another encoder state or to WRITE a decoder output. Each READ incurs a delay, but gives the decoder more information to work with. We denote the encoder step at which the policy decides to WRITE in decoder step i as $N(i)$.

$$h_j = \textsc{Encoder}(h_{j-1}, x_j) \quad (7)$$
$$a_i^n = \textsc{Policy}(y_{i-1}, h_n) \quad (8)$$
$$N(i) = \min\{n : a_i^n = \text{ WRITE }\} \quad (9)$$
$$c_i = \textsc{Attention}(y_{i-1}, h_{1\dots N(i)}) \quad (10)$$
$$y_i = \textsc{Decoder}(y_{i-1}, c_i) \quad (11)$$

Note that this kind of discrete decision-making process is not differentiable. Some approaches using reinforcement learning have been proposed

(Grissom II et al., 2014; Satija and Pineau, 2016), but we will focus on the monotonic attention approaches.

3.1 Monotonic Attention

In monotonic attention (Raffel et al., 2017), the context is exactly the encoder state at $N(i)$. Additionally, $N(i)$ increases monotonically. For each encoder and decoder step, the policy predicts p_i^n, the probability that we will WRITE at encoder step n. During inference, we simply follow this (non-differentiable) stochastic process[1]. During training, we instead train with the expected value of c_i. To that end, we calculate α_i^n, the probability that decoder step i will attend to encoder step n.

$$p_i^n = \sigma(\text{ENERGY}(s_{i-1}, h_n)) \tag{12}$$

$$a_i^n \sim \text{Bernoulli}(p_i^n) \quad \text{Inference only} \tag{13}$$

$$\alpha_i^n = p_i^n \left((1 - p_i^{n-1}) \frac{\alpha_i^{n-1}}{p_i^{n-1}} + \alpha_{i-1}^n \right) \tag{14}$$

$$c_i = \sum_{n=1}^{|\mathrm{x}|} \alpha_i^n h_n \tag{15}$$

This model needs no additional loss function besides the translation loss. It is not incentivised to READ any further than it has to because the model can only attend to one token at a time. At the same time, this is a weakness of the model, as it has access to only a very narrow portion of the input at a time.

To address this, two extensions to monotonic attention have been proposed: Monotonic Chunkwise Attention (MoChA, Chiu and Raffel (2017)) and Monotonic Infinite Lookback Attention (MILk, Arivazhagan et al. (2019)), which we will look at in more detail here.

3.2 Monotonic Infinite Lookback Attention

Monotonic Infinite Lookback Attention (MILk) combines soft and monotonic attention. The attention can look at all hidden states from the start of the input up to $N(i)$, which is determined by a monotonic attention module. The model is once again trained in expectation, with p_i^n and α_i^n calculated as in eqs. (12) and (14). The attention energies e_i^n are calculated as in equation (4).

[1] Although we encourage the model to make clear decisions by adding noise in the policy, see the original paper for more details.

$$\beta_i^n = \sum_{k=n}^{|\mathrm{x}|} \left(\frac{\alpha_i^k \exp(e_i^n)}{\sum_{l=1}^{k} \exp(e_i^l)} \right) \tag{16}$$

$$c_i = \sum_{n=1}^{|\mathrm{x}|} \beta_i^n h_n \tag{17}$$

This method does however introduce the need for a second loss function, as the monotonic attention head can simply always decide to advance to the end of the input where the soft attention can attend to the whole sequence. Therefore, in addition to the typical log-likelihood loss, the authors introduce a loss derived from $\mathrm{n} = \{N(1), \ldots N(|\mathrm{y}|)\}$, weighted by a hyperparameter λ:

$$L(\theta) = -\sum_{(\mathrm{x},\mathrm{y})} \log p(\mathrm{y}|\mathrm{x};\theta) + \lambda \mathcal{C}(\mathrm{n}) \tag{18}$$

Unfortunately, despite following all advice from Raffel et al. (2017), applying gradient clipping and different energy functions from Arivazhagan et al. (2019), we were not able to adapt MILk for use with the transformer model, largely due to the numerical instability of calculating α_i^n (see Raffel et al. (2017) for more details on this problem). We therefore turn to a different method which has so far not been applied to simultaneous machine translation, namely Adaptive Computation Time (ACT, (Graves, 2016)).

3.3 Adaptive Computation Time

Originally formulated for RNNs without the encoder-decoder framework, Adaptive Computation Time is a method that allows the RNN to "ponder" the same input for several timesteps, effectively creating sub-timesteps. We will first go over the original use-case, although we intentionally match the notation above. At each timestep i, we determine $N(i)$, the number of timesteps spent pondering the current input. We do so by predicting a probability at each sub-timestep s_i^n. We stop once the sum of these probabilities exceeds a threshold. We also calculate a *remainder* $R(i)$. Eqns. (19) through (22) are adapted from Graves (2016) and apply to RNNs:

$$p_i^n = \sigma(\textsc{Energy}(s_i^n)) \tag{19}$$

$$N(i) = \min\{n' : \sum_{n=1}^{n'} p_i^n \geq 1 - \epsilon\} \tag{20}$$

$$R(i) = 1 - \sum_{n=1}^{N(i)-1} p_i^n \tag{21}$$

$$\alpha_i^n = \begin{cases} R(i) & \text{if } n = N(i) \\ p_i^n & \text{otherwise} \end{cases} \tag{22}$$

It follows directly from the definition that α_i is a valid probability distribution. Compared to monotonic attention, ACT directly predicts the expected value for the amount of steps that the model takes, rather than calculating it from stopping probabilites. As-is, the model has no incentive to keep the ponder times short, so we introduce an additional loss:

$$\mathcal{C} = \sum_{i=1}^{|x|} N(i) + R(i) \tag{23}$$

Note that the computation for $N(i)$ is not differentiable so it is treated as a constant and the loss is equivalent to just summing the remainders.

We now go on to transfer ACT to the encoder-decoder domain. Now, instead of pondering the input to an RNN, like in original ACT, the decoder ponders over zero or more encoder steps. The encoder still works as in eq. (7) and does not use ACT. Instead, we apply the ACT ponder mechanism to the monotonic encoder-decoder attention. Let $N(i)$ denote the last encoder step to which we can attend. We make sure that $N(i)$ advances monotonically:

$$p_i^n = \sigma(\textsc{Energy}(y_{i-1}, h_n)) \tag{24}$$

$$N(i) = \min\{n' : \sum_{n=N(i-1)}^{n'} p_i^n \geq 1 - \epsilon\} \tag{25}$$

$$\alpha_i^n = \begin{cases} R(i) & \text{if } n = N(i) \\ p_i^n & \text{if } N(i-1) \leq n < N(i) \\ 0 & \text{otherwise} \end{cases} \tag{26}$$

Then we proceed as in equations (16) and (17). Note that in this formulation, it is possible that $N(i) = N(i-1)$ (i. e. the model pondering for zero steps), indicating consecutive WRITEs. In original ACT, it is not possible to ponder the input for zero steps. Also, similar to MILk, we consider $p_i^{|x|}$ to be 1 always. See figure 2 for a visualisation of α_i^n on a concrete example.

3.4 Transformer XL

Finally, we introduce two aspects of the Transformer XL language model (Dai et al., 2019) into our model: Relative attention and memory.

We replace the Transformer self-attention in both encoder and decoder with relative attention. In relative self-attention, we calculate Energy as follows:

$$\begin{aligned} \textsc{Energy}(x_i, x_j) = {} & x_i^\top W_q^\top W_E\, x_j \\ & + x_i^\top W_q^\top W_R\, R_{i-j} \\ & + u^\top W_E\, x_j \\ & + v^\top W_R\, R_{i-j} \end{aligned} \tag{27}$$

Where W_q, W_e, W_R, u, v are learnable parameters and R are relative position encodings. Afterwards, we proceed as in equation (16) and (17) for simultaneous models or eqautions (5) and (6) for offline models.

For our streaming model, we also use Transformer XL-style memory during training. This means that we keep the hidden states of both encoder and decoder from the previous training step during training. Both self-attention and encoder-decoder attention are able to attend to these states as well as the current input sentence. However, no gradients can flow through the old states to the model parameters.

3.5 Stream Translation

Our stream translation model should not rely on any segmentation information of the input and must be able to translate a test set as a single, continuous sequence. To achieve this, we extend the standard transformer model in the following ways:

- We use ACT monotonic attention to constrain the encoder-decoder attention. The position of the monotonic attention head also gives us a pointer to the model's current read position in the input stream that advances token by token, and not sentence by sentence and therefore requires no sentence segmentation.

- We change all self-attentions to relative attention, as well as removing absolute position encodings. We could encode positions as absolute since the beginning of the stream. However, Neishi and Yoshinaga (2019) showed that Transformer with absolute position encodings generalizes poorly to unseen sequence

lengths. In a continuous stream, relative encodings are the more logical choice.

- We add Transformer XL-style history to the model so that even the first positions of a sample have a history buffer for self-attention. This simulates the evaluation condition where we don't restart the model each sentence.

- During inference, we cannot cut off the history at sentence boundaries (such as keeping exactly the last sentence) because this information is not available. Instead, we adopt a rolling history buffer approach, keeping n_h previous positions for the self-attention. To simulate this condition in training, we apply a mask to the self-attention, masking out positions more than n_h positions in the past.

- During training, we concatenate multiple samples to a length of at least n_h tokens. This is to allow the model to READ past the end of an input sentence into the next one. Normally, this is prevented by setting $p_i^{|\mathrm{x}|} = 1$. However during inference, $|\mathrm{x}|$ is not available and therefore the model should learn to stop READing at appropriate times even across sentence boundaries.

- We use the ponder loss of equation (23) in addition to the cross-entropy translation loss with a weighting parameter λ as in equation (18).

4 Experiments

4.1 Segmented Translation

In our first set of experiments, we demonstrate the ability of ACT to produce state-of-the art results in sentence-based simultaneous machine translation. For comparison to Arivazhagan et al. (2019), we choose the same dataset: WMT2014 German-to-English (4.5M sentences). As they report their delay metrics on tokenized data, we also use the same tokenization and vocabulary.

All models follow the Transformer "base" configuration (Vaswani et al., 2017) and are implemented in fairseq (Ott et al., 2019). In addition to the simultaneous models, we train a baseline Transformer model. All models except the baseline use relative self-attention. We pre-train an offline model with future-masking in the encoder as a common basis for all simultaneous models.

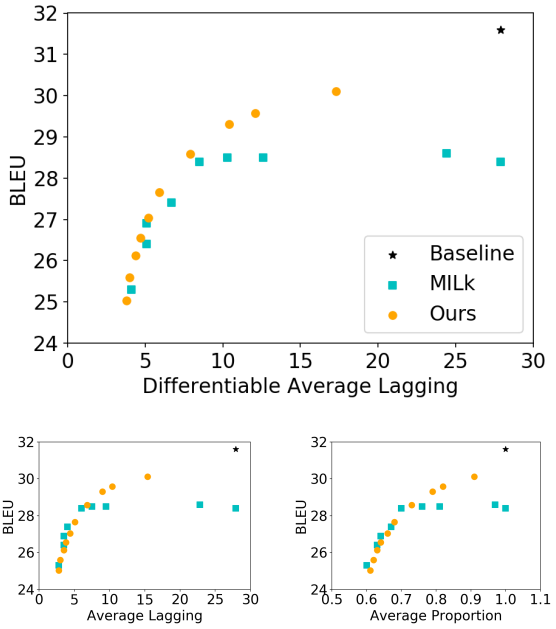

Figure 1: Quality-Latency comparison for German-to-English `newstest2015` in tokenized DAL (top), AL (bottom left) and AP (bottom right)

For the simultaneous models, we vary the value of λ and initialize the parameters from the pre-trained model. We found that training from the start with the latency loss can cause extreme latency behaviour, where the model either reads no input from the source at all or always waits until the end. We theorize that the best strategy would be to introduce the latency loss gradually during training, but leave that experiment for future work.

All models are trained using the Adam Optimizer (Kingma and Ba, 2015). For the pre-training model, we vary the learning rate using a cosine schedule from $2.5 \cdot 10^{-4}$ to 0 over 200k steps. For the ACT model, we start the learning rate at $4 \cdot 10^{-5}$ and use inverse square root decay (Vaswani et al., 2017) for 1000 steps.

We measure translation quality in detokenized, cased BLEU using `sacrebleu`[2] (Post, 2018). We measure latency in Average Lagging (Ma et al., 2019a), Differentiable Average Lagging (Arivazhagan et al., 2019) and Average Proportion (Cho and Esipova, 2016). For direct comparison, we report the tokenized latency metrics, but we provide the detokenized metrics in the appendix.

Figure 1 shows our results for this task. We generally achieve a better quality-latency tradeoff

[2] `BLEU+case.mixed+lang.de-en+numrefs.1 +smooth.exp+test.wmt15+tok.13a +version.1.4.3`

as measured by DAL, and a comparable one as measured by AP and AL. We note also that the ceiling for quality of ACT is higher than that of MILk. Whereas MILk loses two BLEU points to their baseline model even when given full attention ($\lambda = 0.0$), our model would seem to get closer to the performance of the baseline with decreasing λ.

4.2 Stream Translation

In this set of experiments, we demonstrate our model's ability to translate continuous streams of input with no sentence segmentation. For training, we use the IWSLT 2020 simultaneous translation data (which includes all WMT2019 data) with 37.6M sentences total. We choose this dataset because of a larger amount of document-level data (3.8M sentences). Because we will use Transformer XL-style memory, we depend on as much contextual data as possible. We evaluate on the IWSLT tst2010 test set in German to English. On the source side, we convert to lower case and remove all punctuation.

In addition to the baseline normal Transformer model, we train our model in three steps: First an offline, sentence-based relative self-attention Transformer, then the Transformer XL and finally the ACT+XL model, each one initializing its parameters on the last one. Both the relative model and the Transformer XL use the cosine schedule starting at $2.5 \cdot 10^{-4}$ and training for 200k and 40k steps, respectively. The ACT+XL model uses inverse square root decay, starting at $4 \cdot 10^{-5}$ as above and trains for 1000 steps. We also experiment with training ACT+XL directly from the relative model.

We evaluate as before[3], treating the test set as a single sequence. BLEU scores are calculated by re-segmenting the output according to the original reference based on Word Error Rate (Matusov et al., 2005). All reported metrics are detokenized. The baseline and relative models use beam search, the others use greedy decoding.

Unfortunately, the range of the λ parameter that produces sensible results is much more restricted than for the sentence-based model (see "Analysis", below). We report results with $\lambda = 0.25$ and 0.3.

Table 1 shows our results. There is a drop of 4 BLEU points when moving to simultaneous translation, which is similar to our experiments on segmented text. While there is room for improvement, these are promising results, and, to the best of our knowledge, the first demonstration of unsegmented end-to-end stream translation.

Model	**AP**	**AL**	**DAL**	**BLEU**
Baseline	—	—	—	32.0
Relative	—	—	—	33.1
XL	—	—	—	34.4
ACT+XL				
$\lambda = 0.25$	0.5	206	329	30.2
$\lambda = 0.3$	0.5	107	180	30.3
ACT+XL	*directly from relative*			
$\lambda = 0.25$	0.5	222	394	26.4

Table 1: Results for the stream translation experiment

4.3 Analysis

For the segmented translation, we compare two different latency schedules in figure 2. Both schedules advance relatively homogenously. This may indicate that the ACT attention layer needs to be expanded to extract more grammatical information and make more informed decisions on waiting. Nevertheless, the model produces good results and we even observe implicit verb prediction as in Ma et al. (2019a). We also note that the high latency models' latency graph tends to describe a curve, whereas the low latency models tend to uniformly advance by one token per output token.

This behaviour can be explained by the properties of Differentiable Average Lagging. The ponder loss objective that ACT is trained on may seem very different, but actually produces somewhat similar gradients to DAL [4], so the model incidentally also learns a behaviour that optimizes DAL.

DAL is monotonically increasing, i. e. the model can never "catch up" any delay by WRITing multiple tokens without READing (assuming $|\mathrm{y}| = |\mathrm{x}|$). It achieves the same DAL but with better translation by always READing one token when it WRITEs. Therefore, to achieve DAL $= k$ for a given k, the ideal waiting strategy is wait-k.

In the case of stream translation, we make two important observations: First, that systems with $\lambda < 0.25$ do not produce acceptable results (BLEU scores < 10). This is because they fall behind the input by waiting too much and have to skip sentences to catch back up. Once an input word is more than n_h tokens behind, it is removed from

[3] `BLEU+case.mixed+lang.de-en+numrefs.1 +smooth.exp+iwslt17/tst2010+tok.13a +version.1.4.3`

[4] $\frac{\partial \mathrm{DAL}}{\partial \alpha_i^n} = i - N(i) - 1$, $\frac{\partial \mathrm{ACT}}{\partial \alpha_i^n} = -1$ for $N(i-1) \leq i \leq N(i)$, else 0

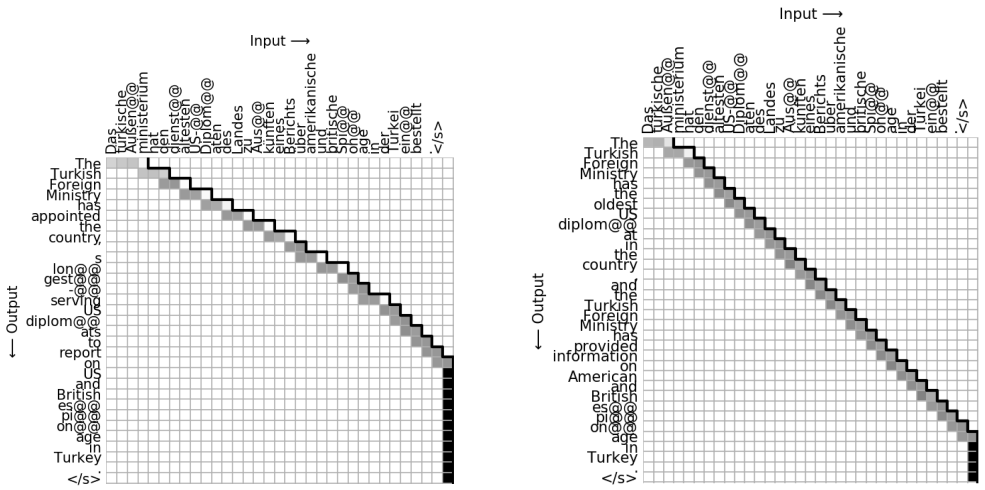

Figure 2: The same sentence from `newstest2015` translated by an ACT system with $\lambda = 0.1$ (left) and $\lambda = 0.4$ (right). The shading indicates the α_i^n as predicted by the ACT attention module (darker = higher probability), the black line indicates the hard attention cutoff. The low-latency model approaches the behaviour of a wait-4 model. Note the (incorrect) attempt of the left model to predict the verb "einbestellt" = "summons", whereas the right model takes the first half of the sentence as complete, leaving out the verb.

the memory and if it is not translated by then, it may be forgotten. Therefore, we found it essential to train more aggressive latency regimes. On the other hand, systems with $\lambda > 0.3$ sometimes read too little source information or stop reading new source words altogether.

Second, that the established latency metrics do not perform well on the very long sequence (with our tokenization, the source is 29 317 tokens long). While on single sentences, an AL score of 4 might indicate quite consistently a lag of around 4 tokens, a manual analysis of the output of our $\lambda = 0.3$ system shows a delay of between 40 and 60 words, quite far away from the automatic metrics of AL=107 and DAL=180. Average proportion in particular breaks down under these conditions and always reports 0.5.[5]

5 Conclusion and Future work

We have presented Adaptive Compuation Time (ACT) for simultaneous machine translation and demonstrated its ability to translate continuous, unsegmented streams of input text. To the best of our knowledge, this is the first end-to-end NMT model to do so. While stream translation model still loses a lot of performance compared to the sentence-based models, we see this as an important step towards end-to-end simultaneous stream translation.

We see several possibilites for future work on this model: Training the whole model in one training rather than the multiple rounds of pre-training may be possible by gradually introducing the latency loss during training. Perhaps the latency decisions can be improved by adding extra layers to the ACT attention module.

But most importantly, we believe the model must be adapted to the speech domain. Recently (see e. g. Di Gangi et al. (2019)), the Transformer has shown promising results for speech translation. For a realistic application we believe that a simultaneous translation model must work with speech input.

[5]The full output of the $\lambda = 0.3$ model can be found here: https://gist.github.com/felix-schneider/1462d855808e582aa19307f6b0d576e1

Acknowledgments

The work leading to these results has received funding from the European Union under grant agreement Nº 825460 and the Federal Ministry of Education and Research (Germany) / DLR Projektträger Bereich Gesundheit under grant agreement. Nº 01EF1803B.

References

Ashkan Alinejad, Maryam Siahbani, and Anoop Sarkar. 2018. Prediction improves simultaneous neural machine translation. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 3022–3027.

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. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 1313–1323.

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2015. Neural machine translation by jointly learning to align and translate. In *3rd International Conference on Learning Representations, ICLR 2015*.

Chung-Cheng Chiu and Colin Raffel. 2017. Monotonic chunkwise attention. *arXiv preprint arXiv:1712.05382*.

Eunah Cho, Jan Niehues, and Alex Waibel. 2012. Segmentation and punctuation prediction in speech language translation using a monolingual translation system. In *International Workshop on Spoken Language Translation (IWSLT) 2012*.

Eunah Cho, Jan Niehues, and Alex Waibel. 2017. Nmt-based segmentation and punctuation insertion for real-time spoken language translation. In *INTERSPEECH*, pages 2645–2649.

Kyunghyun Cho and Masha Esipova. 2016. Can neural machine translation do simultaneous translation? *arXiv preprint arXiv:1606.02012*.

Zihang Dai, Zhilin Yang, Yiming Yang, Jaime G Carbonell, Quoc Le, and Ruslan Salakhutdinov. 2019. Transformer-xl: Attentive language models beyond a fixed-length context. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 2978–2988.

Mattia A Di Gangi, Matteo Negri, and Marco Turchi. 2019. Adapting transformer to end-to-end spoken language translation. In *INTERSPEECH 2019*, pages 1133–1137. International Speech Communication Association (ISCA).

Christian Fügen, Alex Waibel, and Muntsin Kolss. 2007. Simultaneous translation of lectures and speeches. *Machine translation*, 21(4):209–252.

Alex Graves. 2016. Adaptive computation time for recurrent neural networks. *arXiv preprint arXiv:1603.08983*.

Alvin Grissom II, He He, Jordan Boyd-Graber, John Morgan, and Hal Daumé III. 2014. Don't until the final verb wait: Reinforcement learning for simultaneous machine translation. In *Proceedings of the 2014 Conference on empirical methods in natural language processing (EMNLP)*, pages 1342–1352.

Diederik P. Kingma and Jimmy Ba. 2015. Adam: A method for stochastic optimization. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*.

Muntsin Kolss, Stephan Vogel, and Alex Waibel. 2008. Stream decoding for simultaneous spoken language translation. In *Ninth Annual Conference of the International Speech Communication Association*.

Mingbo Ma, Liang Huang, Hao Xiong, Renjie Zheng, Kaibo Liu, Baigong Zheng, Chuanqiang Zhang, Zhongjun He, Hairong Liu, Xing Li, et al. 2019a. Stacl: Simultaneous translation with implicit anticipation and controllable latency using prefix-to-prefix framework. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 3025–3036.

Xutai Ma, Juan Pino, James Cross, Liezl Puzon, and Jiatao Gu. 2019b. Monotonic multihead attention. *arXiv preprint arXiv:1909.12406*.

Evgeny Matusov, Gregor Leusch, Oliver Bender, and Hermann Ney. 2005. Evaluating machine translation output with automatic sentence segmentation. In *International Workshop on Spoken Language Translation (IWSLT) 2005*.

Masato Neishi and Naoki Yoshinaga. 2019. On the relation between position information and sentence length in neural machine translation. In *Proceedings of the 23rd Conference on Computational Natural Language Learning (CoNLL)*, pages 328–338.

Jan Niehues, Thai Son Nguyen, Eunah Cho, Thanh-Le Ha, Kevin Kilgour, Markus Müller, Matthias Sperber, Sebastian Stüker, and Alex Waibel. 2016. Dynamic transcription for low-latency speech translation. In *Interspeech*, pages 2513–2517.

Jan Niehues, Ngoc-Quan Pham, Thanh-Le Ha, Matthias Sperber, and Alex Waibel. 2018. Low-latency neural speech translation. In *Proc. Interspeech 2018*, pages 1293–1297.

Yusuke Oda, Graham Neubig, Sakriani Sakti, Tomoki Toda, and Satoshi Nakamura. 2014. Optimizing segmentation strategies for simultaneous speech translation. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 551–556.

Myle Ott, Sergey Edunov, Alexei Baevski, Angela Fan, Sam Gross, Nathan Ng, David Grangier, and Michael Auli. 2019. fairseq: A fast, extensible toolkit for sequence modeling. In *Proceedings of NAACL-HLT 2019: Demonstrations*.

Matt Post. 2018. A call for clarity in reporting bleu scores. In *Proceedings of the Third Conference on Machine Translation: Research Papers*, pages 186–191.

Colin Raffel, Minh-Thang Luong, Peter J Liu, Ron J Weiss, and Douglas Eck. 2017. Online and linear-time attention by enforcing monotonic alignments. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 2837–2846. JMLR. org.

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

Ilya Sutskever, Oriol Vinyals, and Quoc V Le. 2014. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pages 3104–3112.

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

Mahsa Yarmohammadi, Vivek Kumar Rangarajan Sridhar, Srinivas Bangalore, and Baskaran Sankaran. 2013. Incremental segmentation and decoding strategies for simultaneous translation. In *Proceedings of the Sixth International Joint Conference on Natural Language Processing*, pages 1032–1036.

A Segmented Translation Results

λ	Tokenized		
	AP	**AL**	**DAL**
Baseline	1.0	27.9	27.9
0.0	0.91	15.4	17.3
0.01	0.82	10.4	12.1
0.05	0.79	9.0	10.4
0.1	0.73	6.8	7.9
0.15	0.68	5.1	5.9
0.2	0.66	4.4	5.2
0.25	0.64	3.8	4.7
0.3	0.63	3.5	4.4
0.4	0.62	3.0	4.0
0.5	0.61	2.8	3.8

Table 2: Tokenized metrics for `newstest2015` backing figure 1

λ	Detokenized			
	AP	**AL**	**DAL**	**BLEU**
Baseline	1.0	18.6	18.6	31.6
0.0	0.93	10.4	11.7	30.1
0.01	0.84	7.2	8.5	29.6
0.05	0.81	6.3	7.5	29.3
0.1	0.76	4.9	6.0	28.6
0.15	0.71	3.8	4.8	27.7
0.2	0.69	3.4	4.4	27.0
0.25	0.68	3.0	4.1	26.6
0.3	0.67	2.9	3.9	26.1
0.4	0.66	2.6	3.7	25.6
0.5	0.65	2.4	3.6	25.0

Table 3: Detokenized metrics for `newstest2015`