Learning Coupled Policies for Simultaneous Machine Translation using Imitation Learning

Philip Arthur[†]
Department of
Data Science and AI
Monash University

Trevor Cohn[‡]
School of Computing and
Information Systems
University of Melbourne

Gholamreza Haffari[†]
Department of
Data Science and AI
Monash University

[†]{philip.arthur, gholamreza.haffari}@monash.edu
[‡]tcohn@unimelb.edu.au

Abstract

We present a novel approach to efficiently learn a simultaneous translation model with coupled programmer-interpreter policies. First, we present an algorithmic oracle to produce oracle READ/WRITE actions for training bilingual sentence-pairs using the notion of word alignments. This oracle actions are designed to capture enough information from the partial input before writing the output. Next, we perform a coupled scheduled sampling to effectively mitigate the exposure bias when learning both policies jointly with imitation learning. Experiments on six language-pairs show our method outperforms strong baselines in terms of translation quality while keeping the translation delay low.

1 Introduction

Simultaneous machine translation (SIMT) is a setting where the translator needs to incrementally generate the translation while the source utterance is being received. This is a challenging translation scenario as the SIMT model needs to trade off delaying translation output and the quality of the generated translation.

Recent research on SIMT relies on a strategy to decide when to read a word from the input or write a word to the output (Satija and Pineau, 2016; Gu et al., 2017). This is based on a sequential decision making formulation of SIMT, where the decision making about the next READ/WRITE action is made by an *agent*, interacting with the neural machine translation (NMT) *environment*. Current approaches are sub-optimal as they either fix the agent's policy to focus learning the NMT model (Ma et al., 2019; Dalvi et al., 2018) or learn adaptive agent policies while the NMT model is fixed (Gu et al., 2017; Alinejad et al., 2018). We argue that the interpreter should also learn to generate correct translation from incomplete input information. This is challenging as we need to optimize both programmer's and interpreter's policies to balance the tradeoff between quality and delay in the reward.

Previous research has considered the use of imitation learning (IL) to train the agent's policy (Zheng et al., 2019a,b), which is generally superior to reinforcement Learning (RL) in terms of the stability and sample complexity. However, the bottleneck of IL in SIMT is the *un*availability of the *oracle* sequence of actions. Designing *algorithmic* oracles to compute sequence of READ/WRITE actions with low translation latency and high translation quality is under-explored.

We present an IL approach to efficiently learn effective coupled programmer-interpreter policies in SIMT, based on the following contributions. First, we present a simple, fast, and effective algorithmic oracle to produce oracle actions from the training bilingual sentence-pairs based on statistical word alignments (Brown et al., 1993). Next, we design a framework that uses scheduled sampling on both programmer and interpreter. This is different from the typical IL scenarios, where there is only one policy to learn. As the two policies *collaborate*, their learning needs to be robust not only to their own incorrect predictions, but also to incorrect predictions of the other policy to mitigate this coupled *exposure bias*.

Experiments on six language pairs (translating to English from Arabic, Czech, German, Romanian, Hungarian, and Bulgarian) show the policies trained using our approach compares favorably with strong policies from the previous work. We attribute the effectiveness of the learned coupled policies to (i) the scheduled sampling, which handles the coupled exposure bias, resulting in up to 5-8 BLEU score improvements, and (ii) the quality of oracle actions generated by our algorithmic oracle, which balances translation quality and delay.

Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics, pages 2709–2719
April 19 - 23, 2021.

Algorithm 1 Generation in NPI-SIMT

$i, j \leftarrow 0$
while a stopping condition is not met **do**
  $t \leftarrow i + j$
  $\boldsymbol{s}_{t+1} \leftarrow \boldsymbol{f}_{\text{prog}}(\boldsymbol{s}_t, [a_t, \boldsymbol{g}_j, \boldsymbol{h}_i])$
  $P_{\text{prog}} \leftarrow \text{softmax}(\text{dense}_{\text{prog}}(\boldsymbol{s}_{t+1}))$
  $a_{t+1} \sim P_{\text{prog}}$
  if $a_{t+1} = \text{READ}$ **then**
    $i \leftarrow i + 1$
    $\boldsymbol{h}_i \leftarrow \boldsymbol{f}_{\text{enc}}(\boldsymbol{h}_{i-1}, x_i)$
  else
    $j \leftarrow j + 1$
    $\boldsymbol{g}_j \leftarrow \boldsymbol{f}_{\text{intp}}(\boldsymbol{g}_{j-1}, y_{j-1}, \boldsymbol{h}_{\leq i})$
    $P_{\text{intp}} \leftarrow \text{softmax}(\text{dense}_{\text{intp}}(\boldsymbol{g}_j))$
    $y_j \sim P_{\text{intp}}$
  end if
end while

2 NPI Approach to SIMT

We describe generation in our neural programmer-interpreter (NPI) approach to simultaneous machine translation (SIMT) in Algorithm 1. At each time step t, the *programmer* needs to decide whether to READ the next source word or to WRITE the next target word in the translation. The *interpreter* then immediately *executes* the action generated by the programmer. Both programmer and interpreter are modeled using Markov Decision Process (MDP) where prediction at particular timestep depends on the history of previous predictions. The indices i_t and j_t are the number of READ and WRITE actions in the program up to time step t.

The Programmer needs to *sequentially* decide about the next *action*, given the previous actions $\boldsymbol{a}_{<t}$ and the prefix of the source utterance read so far $\boldsymbol{x}_{\leq i_t}$ as well as the prefix of the target translation generated so far $\boldsymbol{y}_{\leq j_t}$. That is, our programmer is modeled as $P_{\text{prog}}(a|\boldsymbol{a}_{<t}, \boldsymbol{x}_{\leq i_t}, \boldsymbol{y}_{\leq j_t})$.

The Interpreter needs to execute the action generated by the programmer. At time step t, if the generated action a_t is READ, we reveal the next input token. Otherwise, if WRITE, then we generate the next target word according to $P_{\text{intp}}(y|\boldsymbol{a}_{\leq t}, \boldsymbol{x}_{\leq i_t}, \boldsymbol{y}_{\leq j_t})$.[1]

The Probabilistic Model. The probability of simultaneously generating the translation $\boldsymbol{y}$ and the sequence of actions $\boldsymbol{a}$ for a source utterance $\boldsymbol{x}$ is,

$$P_{\text{SIMT}}(\boldsymbol{y}, \boldsymbol{a}|\boldsymbol{x}) = \prod_{t=1}^{|\boldsymbol{x}|+|\boldsymbol{y}|} P_{\text{prog}}(a|\boldsymbol{a}_{<t}, \boldsymbol{x}_{\leq i_t}, \boldsymbol{y}_{\leq j_t}) \times \prod_{t:a_t=\text{WRITE}} P_{\text{intp}}(y|\boldsymbol{a}_{\leq t}, \boldsymbol{x}_{\leq i_t}, \boldsymbol{y}_{\leq j_t}).$$

Training the Model. In SIMT, we are interested in not only producing a high quality translation, but also reducing the delay between the times of receiving the source words and generating their translations. Training of the model based on this hybrid training objective can be done by reinforcement learning (RL) or imitation learning (IL). The RL approach has been attempted by (Satija and Pineau, 2016; Gu et al., 2017; Alinejad et al., 2018) for training the programmer; however, it is unstable due to sparsity of the reward function and these works also assumed a fixed interpreter. We thus take the IL approach for a sample efficient, effective, and stable learning of policies in NPI-SIMT.

3 Deep Coupled Imitation Learning

Our goal is to learn a pair of policies for the programmer and interpreter using IL. §3.1 describes the method of learning of both policies where their learning inter-dependency needs to be taken into account. §3.2 describes our novel *oracle* program actions for each sentence pair in the training set, i.e., the program $\boldsymbol{a}$ which has been responsible for generating the translation $\boldsymbol{y}$ for a source utterance $\boldsymbol{x}$ with as low delay as possible. Our overall training algorithm is depicted in Algorithm 2. $\hat{\boldsymbol{X}} = [\hat{\boldsymbol{x}}_1, ..., \hat{\boldsymbol{x}}_{|\boldsymbol{x}|}]$ is the encoding of the input sequence and $\hat{\boldsymbol{Y}} = [\hat{\boldsymbol{y}}_1, ..., \hat{\boldsymbol{y}}_{|\boldsymbol{y}|}]$ is a list of interpreter hidden states for all predictions. During training, these values are calculated before calculating the loss of the programmer.

3.1 Learning Robust Coupled Policies

Assuming we have the oracle actions, we can learn the policies for both the programmer and interpreter using behavioural cloning in IL (Torabi et al., 2019). That is, the model parameters are learned by maximising the likelihood of the oracle actions for both the programmer and interpreter,

$$\theta^*_{\text{prog}}, \theta^*_{\text{intp}} := \arg\max_{\theta_{\text{prog}}, \theta_{\text{intp}}} \sum_{(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{a})} \sum_{t=1}^{|\boldsymbol{x}|+|\boldsymbol{y}|} \log P_{\text{prog}}(a|\boldsymbol{a}_{<t}, \boldsymbol{x}_{\leq i_t}, \boldsymbol{y}_{\leq j_t}; \theta_{\text{prog}}) + \sum_{t:a_t=\text{WRITE}} \log P_{\text{intp}}(y|\boldsymbol{x}_{\leq i_t}, \boldsymbol{y}_{\leq j_t}; \theta_{\text{intp}}).$$

[1] The counter i and j are also incremented according to the respective actions at time t.

This is akin to have the expectation, in the original training objective of NPI, under a point-mass distribution over the oracle actions.

IL with behavioural cloning does not lead to robust policies for unseen examples in the test time due to exposure bias (Bengio et al., 2015). That is, the agent is only exposed to situations resulting from the *correct* actions in the training time, leading to its inability to mitigate from propagation of errors faced due to incorrect actions in the test time. Scheduled sampling (Bengio et al., 2015; Ross et al., 2011) addresses this issue by exposing the agent to incorrect decisions in training time through perturbation of the oracle decisions, which we extend to learning policy *pairs*. Crucially, the programmer-interpreter policies need to be robust to incorrect decisions encountered not only in their own trajectories, but also to one anothers' trajectories.

Learning the Programmer. To train our programmer on a training example $(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{a})$ with scheduled sampling, we first create the perturbation $(\boldsymbol{a}', \boldsymbol{y}')$ of the ground truth program and interpreter decisions. The perturbed program $\boldsymbol{a}'$ and translation $\boldsymbol{y}'$ are only used as the input to the recurrent architectures of the programmer and interpreter's decoder. They are created by replacing some of the ground truth element by randomly selecting an action from the predictive distribution of each model. We then maximise the following training objective,

$$\theta^*_{\text{prog}} := \arg\max_{\theta_{\text{prog}}} \sum_{(\boldsymbol{x},\boldsymbol{y}',\boldsymbol{a},\boldsymbol{a}',\boldsymbol{a}'')} \sum_{t=1}^{|\boldsymbol{x}|+|\boldsymbol{y}'|} \log P_{\text{prog}}(a|\boldsymbol{a}'_{<t}, \boldsymbol{x}_{\leq i_t}, \boldsymbol{y}'_{\leq j_t}; \theta_{\text{prog}}). \quad (1)$$

Based on the generative process described in Algorithm 1, the programmer conditions the generation of actions in each time step on the current states of the NMT's encoder and decoder. Hence, while training the programmer, the valid READ/WRITE actions need to be communicated to the interpreter and be executed in order to provide NMT's encoder/decoder states to the programmer to condition upon. Crucially, the communicated program needs to be valid.

Valid Program A ground truth program $\boldsymbol{a}$ is a valid sequence of READ/WRITE actions if $|\text{READ} \in \boldsymbol{a}| = |\boldsymbol{x}|$ and $|\text{WRITE} \in \boldsymbol{a}| = |\boldsymbol{y}|$. This valid program ensures the NPI model to safely consume a pair of parallel sentence. We generate a valid perturbation $\boldsymbol{a}''$ by only permuting the READ/WRITE actions of the program $\boldsymbol{a}$ (Figure 1).

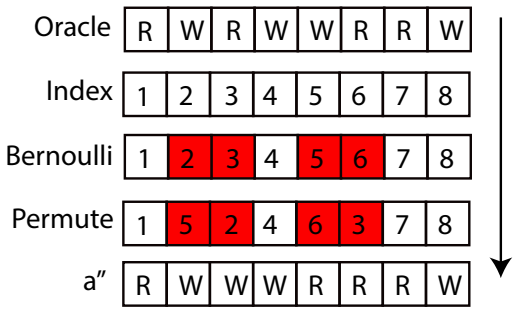

Figure 1: Creating valid perturbation from oracle program. We use a combination of Bernoulli sample and permutation function.

Algorithm 2 Training NPI-SIMT

Require: $\mathcal{D}$: Sentence pairs with oracle actions, $\beta_1, \beta_2, \beta_3$: scheduled sampling probabilities for $\boldsymbol{y}', \boldsymbol{a}', \boldsymbol{a}''$.

while a stopping condition is not met **do**
  randomly pick $(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{a}) \in \mathcal{D}$
  $\boldsymbol{y}' \leftarrow \text{perturbSeq}(\boldsymbol{y}, \beta_1, \theta_{\text{intp}})$
  $\boldsymbol{a}' \leftarrow \text{perturbSeq}(\boldsymbol{a}, \beta_2, \theta_{\text{prog}})$
  $\boldsymbol{a}'' \leftarrow \text{perturbProgValid}(\boldsymbol{a}, \beta_3)$
  $\hat{\boldsymbol{y}}, \hat{\boldsymbol{X}}, \hat{\boldsymbol{Y}} \leftarrow \text{forward\_intp}(\theta_{\text{intp}}, \boldsymbol{x}, \boldsymbol{y}', \boldsymbol{a}'')$
  $\hat{\boldsymbol{a}} \leftarrow \text{forward\_prog}(\theta_{\text{prog}}, \boldsymbol{a}', \hat{\boldsymbol{X}}, \hat{\boldsymbol{Y}})$
  $\theta_{\text{intp}} \leftarrow \theta_{\text{intp}} - \alpha_1 \nabla\delta(\hat{\boldsymbol{y}}, \boldsymbol{y})$
  $\theta_{\text{prog}} \leftarrow \theta_{\text{prog}} - \alpha_2 \nabla\delta(\hat{\boldsymbol{a}}, \boldsymbol{a})$
end while

We further extend the definition of a valid program with respect to the domain knowledge of translation so that: (i) no WRITE at the beginning, and (ii) no READ at the end of the program.

Learning the Interpreter. The interpreter needs to be robust to the incorrect actions in the previously generated words in the translations as well as the READ/WRITE actions generated by the programmer. This is done by communicating $\boldsymbol{a}''$ to the intepreter during training. Thus, the training objective for the interpreter is,

$$\theta^*_{\text{intp}} := \arg\max_{\theta_{\text{intp}}} \sum_{(\boldsymbol{x},\boldsymbol{y},\boldsymbol{y}',\boldsymbol{a}'')} \sum_{t:a''_t=\text{WRITE}} \log P_{\text{intp}}(y|\boldsymbol{x}_{\leq i_t}, \boldsymbol{y}'_{\leq j_t}; \theta_{\text{intp}}). \quad (2)$$

3.2 Oracle Program Actions

Our proposed oracle should measure the appropriate amount of inputs needed for translating a particular target word $\boldsymbol{y}_{j_t}$. This is done by determining the *key* word or phrases δ_j which contain important information of $\boldsymbol{y}_{j_t}$, and therefore guiding the programmer to read until δ_j before writing $\boldsymbol{y}_{j_t}$. Algorithm 3 outlines our oracle generation

Algorithm 3 Oracle Generation

Require: $\boldsymbol{a}$: Symmetrized alignment of $\boldsymbol{x}$ and $\boldsymbol{y}$ in forms of $\boldsymbol{a}_{i,j}$, which means that $\boldsymbol{x}_i$ is aligned to $\boldsymbol{y}_j$. Index starts with 0.

$\delta_{\text{READ}} := -1$
for $j \in$ **range**$(0, |\boldsymbol{y}|)$ **do**
  $\delta_j \leftarrow \max[\{i \in a_{i,j}\}]$
  for $(\delta_j - \delta_{\text{READ}})$ times **do**
    emit READ
  end for
  $\delta_{\text{READ}} \leftarrow \max(\delta_j, \delta_{\text{READ}})$
  emit WRITE
end for

procedure. No READ operation is emitted if the $\delta_j \leq \delta_{\text{READ}}$ or $\{i \in a_{i,j}\} = \emptyset$.

δ_j can be heuristically determined by using word alignment (Brown et al., 1993; Koehn et al., 2003), which captures strong relationship between tokens. In the case of many to one alignment of source to target, we choose the furthest source word. In the case of no alignment, nothing is done as it means the target word can be induced merely from the decoder without needing to read additional inputs. This oracle can generally generate a valid program. Caution is needed to ensure that no WRITE at the beginning and no READ at the end of the generated oracle. This can be done by aligning the first words of the parallel sentence. Similarly we also need to align the last words of the parallel sentence. [2]

4 Experiments

Our experiments aim to measure the effectiveness of our proposed method versus a strong wait-k baseline, over a range of languages of varying difficulty, syntactic complexity, and lexical complexity.

4.1 Settings

Datasets. Our main experiment will be performed in higher quality corpus which is designed for spoken dialogue and carefully edited dataset. Additionally, we perform a single large scale experiment using crawled corpus such as WMT to show that our method also scales to a large dataset.

We evaluate our proposed method on 6 language pairs, in all cases translating into English, with the source languages chosen to cover a wide range of language families and syntax. We use German (DE), Czech (CS) and Arabic (AR) from the IWSLT 2016 translation dataset (Cettolo et al., 2012). We use the provided training and development sets as-is, and concatenate all provided test sets to create our test set. We also evaluate Hungarian (HR), Bulgarian (BG), and Romanian (RO) from the SETIMES corpus (Tyers and Alperen, 2010). As this corpus is not partitioned, we use the majority of the data for training, holding out 2000 random sentence pairs for development and another 2000 sentence pairs for testing. Together these languages are representative of Germanic, West Slavic, Arabic, Uralic, East Slavic, and Italic language families, respectively.

We use sentencepiece (Kudo and Richardson, 2018) to build and tokenize our training data with 16k vocabulary size. Then we generate our oracle program actions based on the segmented tokens. We use fast_align (Dyer et al., 2013) to generate symmetrized alignments between tokens. Unless otherwise specified, we use the default settings of the mentioned toolkit.

Evaluation. We evaluate the SIMT systems based on its translation quality and delay. Translation quality can be measured by case sensitive BLEU (Papineni et al., 2002).[3] We adopt three delay measurements by previous studies. First, average proportion (AP) (Gu et al., 2017) is a fraction of read source words per emited target words. Second, average lagging (AL) (Ma et al., 2019) is an average number of lagged source words until all inputs are read. Finally the differentiable-AL (DAL) (Arivazhagan et al., 2019) is a refinement of AL which also accumulates the cost of writing output tokens after inputs are fully read.

Baseline. We compare against the wait-k baseline (Ma et al., 2019) where the programmer's policy begins with k numbers of READ, and is followed by switching WRITE and READ, until the source sentence is exhausted or end of sentence (EOS) symbol is written. If the source sentence is exhausted, the programmer will only emit WRITE actions. This baseline was shown to be superior compared to the reinforcement learning approach (Zheng et al., 2019a), and k can be tuned for the desired delay. Arivazhagan et al. (2019)'s approach is superior than the wait-k baseline. However there

[2] Our oracle algorithm's code is released in https://github.com/Monash-NLP-ML-Group/arthur-eacl2021.

[3] Calculated using sacrebleu (Post, 2018). BLEU+case.mixed+numrefs.1+smooth.exp+tok.13a+version.1.4.4 is our `sacreblue`'s signature

	DE→EN				CS→EN				AR→EN				HR→EN				BG→EN				RO→EN			
	BLEU	DAL	AL	AP	BLEU	DAL	AL	AP	BLEU	DAL	AL	AP	BLEU	DAL	AL	AP	BLEU	DAL	AL	AP	BLEU	DAL	AL	AP
wait-k																								
$k=1$	14.40	3.74	2.22	0.61	11.87	4.05	2.60	0.63	16.08	4.04	2.88	0.64	24.78	3.72	2.15	0.58	22.02	3.58	1.58	0.56	22.76	3.51	1.43	0.55
$k=2$	18.67	4.01	2.93	0.64	15.90	4.21	3.20	0.66	19.46	4.35	3.44	0.68	28.50	3.99	2.80	0.60	25.37	3.99	2.27	0.59	28.31	3.82	2.13	0.58
$k=3$	21.13	4.59	3.71	0.68	17.27	4.82	3.92	0.70	22.11	4.91	4.21	0.72	32.04	4.55	3.52	0.63	26.79	4.90	3.35	0.62	32.13	4.31	2.88	0.61
$k=4$	24.21	5.03	4.32	0.71	18.54	5.49	4.72	0.74	22.60	5.67	5.04	0.75	35.16	5.20	4.36	0.66	31.42	5.37	4.10	0.65	34.74	5.12	3.83	0.64
$k=5$	25.63	5.81	5.19	0.75	19.77	6.23	5.59	0.77	23.22	6.50	5.90	0.79	37.42	5.91	5.12	0.69	34.83	5.85	4.73	0.67	38.70	5.63	4.37	0.66
$k=6$	26.19	6.68	6.11	0.78	20.19	7.00	6.41	0.80	23.09	7.47	6.90	0.82	38.41	6.90	6.14	0.72	37.26	6.64	5.56	0.70	39.90	6.63	5.59	0.69
$k=7$	26.89	7.51	7.01	0.81	20.12	7.95	7.33	0.83	23.51	8.20	7.67	0.84	39.77	7.69	6.97	0.74	38.42	7.53	6.54	0.72	40.35	7.54	6.46	0.72
$k=\infty$	28.05	21.67	21.67	1.00	20.85	20.91	20.91	1.00	23.54	19.98	19.98	1.00	41.86	27.56	27.56	1.00	41.98	29.06	29.06	1.00	46.22	29.88	29.88	1.00
NPI-SIMT	17.53	5.05	1.96	0.57	13.58	3.23	1.16	0.55	15.78	3.65	1.34	0.57	25.72	4.30	1.82	0.55	25.42	5.75	2.61	0.57	24.60	5.81	2.46	0.57
$+\boldsymbol{a}'$	20.89	3.98	1.65	0.56	16.17	3.02	1.22	0.55	16.12	2.88	1.14	0.56	30.66	4.05	1.83	0.55	30.73	4.25	1.96	0.55	30.47	5.07	2.25	0.56
$+\boldsymbol{a}''$	20.83	5.14	2.04	0.58	16.96	4.07	1.56	0.57	17.18	3.44	0.96	0.56	33.45	5.31	2.35	0.58	32.97	6.80	3.25	0.60	34.05	7.45	3.43	0.61
$+\boldsymbol{a}', \boldsymbol{a}''$	**22.37**	**4.11**	**1.83**	**0.57**	**18.69**	**3.32**	**1.43**	**0.56**	19.33	3.20	1.31	0.57	**33.69**	**4.23**	**2.00**	**0.56**	**33.78**	**4.69**	**2.18**	**0.56**	**35.97**	**5.32**	**2.55**	**0.58**
$+\boldsymbol{a}', \boldsymbol{a}'', \boldsymbol{y}'$	**22.38**	**4.04**	**1.80**	**0.57**	**18.97**	**3.24**	**1.32**	**0.56**	**20.47**	**2.89**	**1.15**	**0.55**	**35.38**	**3.96**	**1.84**	**0.56**	**34.67**	**4.72**	**2.26**	**0.56**	**37.92**	**4.98**	**2.38**	**0.57**
Oracle-at-test	30.53	3.66	1.72	0.57	23.17	3.17	1.50	0.56	25.63	3.10	1.49	0.57	44.98	3.84	1.84	0.55	46.68	3.98	1.92	0.56	50.31	4.32	2.10	0.56

Table 1: Full results on IWSLT and SETIMES datasets. Boldface indicates better translation quality versus wait-k are about the same delay (relevant systems indicated using underline within same column). Oracle-at-test is the system where the correct program is given during testing, serving as an upper bound on translation quality at a given delay.

is currently no open source code available and their end-to-end approach is not using an oracle policy. As our goal is not to beat the state-of-the-art, we leave this comparison as a future work.

NPI-SIMT. Both the programmer and interpreter are modelled using a unidirectional recurrent neural network (RNN) with a long short term memory cell (LSTM). In particular, we follow the architecture of Luong et al. (2015) with the multi-layer perceptron attention of Bahdanau et al. (2015). Both the programmer and interpreter employ 20% dropout to the network output and 10% dropout to the embedding vector, and use a single layered LSTM with 512 hidden units. For the large scale experiment, we are using the transformer architecture (described in §4.5).

Training. We use `Adam` optimizer (Kingma and Ba, 2015) to train this framework. We track the learning rate of programmer and interpreter separately. We start with 0.001 learning rate, and start halving it whenever perplexity increase on development set. We use a fixed perturbation probability of 5%, 15%, and 15% for $\boldsymbol{y}'$, $\boldsymbol{a}'$, and $\boldsymbol{a}''$ respectively. Early stopping is executed at the fourth learning rate decay.

Testing. We use a beam search algorithm with a beam size of 5 and length normalization algorithm that divides hypothesis score by its length during search (Murray and Chiang, 2018).

4.2 Empirical Results

Scheduled Sampling. Our first experiment tests the effect of scheduled sampling (SS) in learning coupled policies in our NPI-SIMT method. For this purpose, we train four versions of our models where apply SS to both the programmer and the interpreter, only programmer, only interpreter, or neither. Table 1 shows the results, comparing against policies trained using the baseline wait-k method where $k \in \{1, 2, \ldots, 7, \infty\}$. The NPI-SIMT system that is trained using our proposed oracle (NPI SIMT) is able to learn from the low delay oracle as their natural delays (DAL, AL, AP) are generally as low as the delay of the oracle during training. However, it is clearly difficult to perfectly predict the oracle during test-time, and these programmer prediction errors resulted in mistakes in interpreter decisions.

Next, we consider the effect of perturbation and scheduled sampling on the proposed method. Table 1 shows that applying valid perturbation (a'') is more important than doing normal scheduled sampling (a'). This perturbation is directly correlated with the training of the interpreter, as such the noisy program make the interpreter resilient to the exposure bias. Applying both scheduled sampling further increased our proposed method accuracy. The schedule sampling on programmer ($+a', a''$) during training ameliorate this; as it increased up to 10 points of BLEU score in case of Romanian and Hungarian, 8 points in case of Bulgarian and, 5 points for German, Czech and Arabic. Additionally applying scheduled sampling on the interpreter ($+y'$) further improves translation accuracy while slightly decreasing the delay, both effects being consistent across all language pairs.

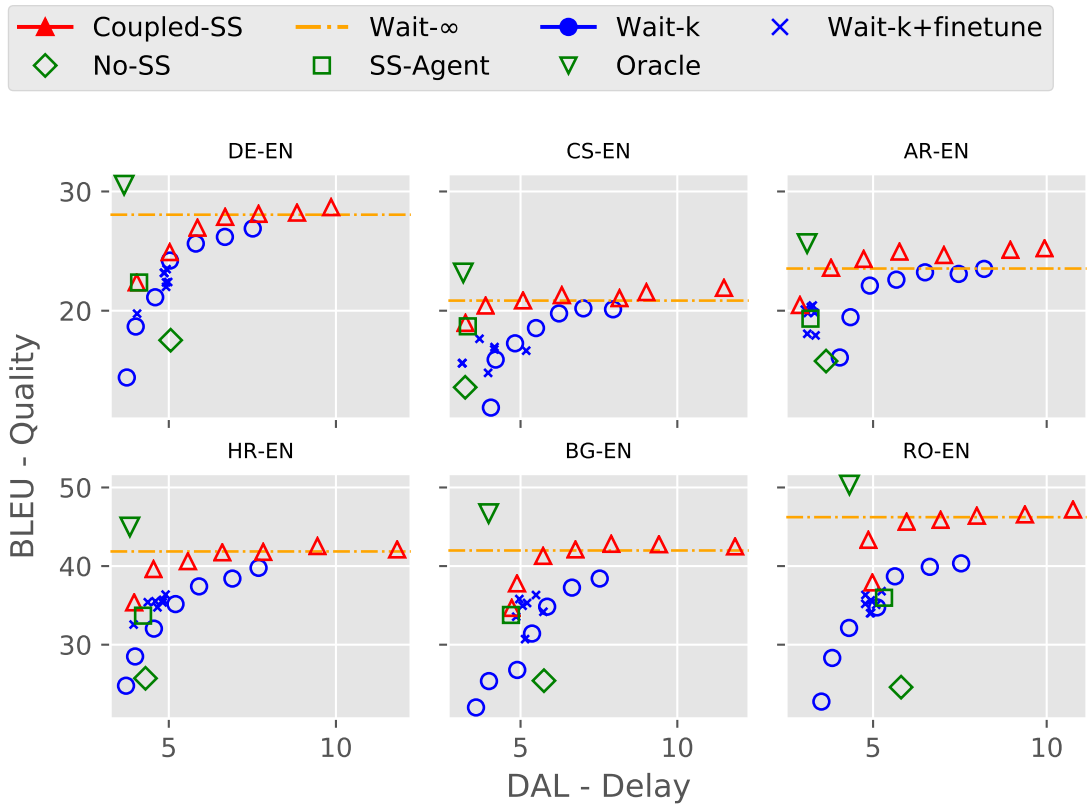

Figure 2: BLEU score versus delay using added delay and finetuning. Our proposed method coupled scheduled-sampling (Coupled-SS) method performs better than the wait-k baseline in all settings. The leftmost $\triangle$ is the result of our proposed method without added delay, while $\triangle$s to the right include delay (see description in text). The $\diamond$ and $\square$ are an ablation study considering not doing SS, and doing SS only on the agent, respectively. The $\times$ report finetuning various pretrained wait-k models with oracle+SS.

Oracle Policy vs Wait-k Policy. Figure 2 compares the policies trained by our algorithmic oracle vs those trained using the wait-k policy starting from $k = 1$. In each of these six plots, the policy trained using the oracle actions corresponds to the leftmost triangle point on the figure. Observe that the policy trained using the oracle actions compares favorably with those trained using the wait-k method in terms of translation quality (higher is better) and translation delay (lower is better).

Next we investigate the effect of increasing the delay of the oracle policy in a controlled manner onto the translation quality of the trained systems. As such, we increase the delay of the oracle policy by moving the last READ action in the oracle program to the beginning of the program, and thus increasing the delay of the oracle artificially. For additional delay, we repeat this process. We expect that the delayed oracle programs lead to trained policies with better translation quality at increased delay. The triangles in Figure 2 correspond to policies trained using the versions of the oracle program, where we added delays $\{0-5\}$. Observe that policies trained with the delayed versions of the oracle program consistently outperform the wait-k policies, across all languages.

The quality of the oracle is shown as the green triangle in Figure 2. This system is provided the oracle program at test time, unlike the other systems that allow errors to propagate from the interpreter into subsequent decisions of the programmer. Note both the low delay of the oracle, and also the fact that the BLEU score outperforms offline translation (wait-∞). This seemingly surprising finding can be explained by the oracle providing key information to the interpreter in the form of word and phrase segmentation of the inputs.

Achieving oracle level quality with a learned programmer is particularly difficult, which can be attributed to exposure bias. However, our coupled SS method manages to bridge much of the gap between the learned program and the oracle program.

_Aber _	_wenn B	_wir ut	_die _if	_Zusammensetzung	_des _we	_Erd **_look**	boden _at	s _the	**_nicht** _composition	**_ändern** _of	, _	_werden E	_wir ar	_das th	_nie ’	_tun s	. _ground	</s> , _we _never _will .	
_Aber _	_wenn B	_wir ut	_die _Zusammensetzung _des _Erd boden s **_nicht _ändern** , _if _we **_don ’ t _change** _the _composition _of									_werden _wir _the		_das _nie _soil ,		_tun . _we	</s> _will _never _do _that .		
_Aber _But	_wenn _if	_wir _we	_die _Zusammensetzung _des _Erd boden s **_nicht** **_don ’ t**						**_ändern** , **_change** _the _composition _of _the			_werden _wir _soil , _we _will			_das _nie _never		_tun . </s> _do _this .		

Table 2: The comparison of our wait-k, our coupled-SS (Co-SS) and the oracle trajectory. A column shows a sequence of consecutive READ and WRITE. Here our proposed method is able to imitate the oracle well, by patiently waiting for sufficient input to produce a good translation. Red texts indicate place of translation error.

Finetuning Wait-k. Next we consider warm-starting training using the wait-k model, and finetuning with the oracle program and SS training method. This method of training is cheaper when a wait-k system is available, as training converges in few iterations. In this setting, we allow retraining of the interpreter, as fixing the interpreter yielded poor BLEU scores. The $\times$ points in Figure 2 show the results of this experiment, where each point was warm-started with a different wait-k system. As it can be seen, all of these runs achieve similar results, but with inferior delay and quality to our proposed method (leftmost $\triangle$). One explanation to this result is that the interpreter already converged to the wait-k policy and retraining it results in an inferior model compared to training it jointly from scratch.[4]

4.3 Qualitative Analysis

Gu et al. (2017) address the difficulty of translating sentences in subject-object-verb order when translating from German to English. We show a typical example in Table 2 where the wait-k systems are forced to make difficult decisions with insufficient evidence, in this case of predicting negation of a verb which appears latter in the input. We compare systems with $AL \approx 2$ and $DAL \approx 4$ which is achieved by the wait-2, our coupled-SS system, and the oracle system. Consider first the wait-2 system. From the state shown at bottom right corner of the smaller red box, the system next generates a poor choice of verb (“look”), which was done without access to the verb in the German input. Instead the rightmost context word was “Zusammensetzung” (“composition”), which gives little information about the verb. One way around this problem is to use a specialized classifier which predicts the final verb (Grissom II et al., 2014). However, this is often onerous or impossible. In this example, the model must also predict the negation “nicht” which appears immediately before the final verb. In a real interpretation scenario, the only way to ensure we output a correct translation is to wait for the matrix verb and negation token.

In this example the oracle trajectory breaks down the input sentence into coherent chunks, and this leads to excellent translation of each segment, and with low delay. This is because the word alignment oracle includes crossing alignment inside a phrase, thus producing sequence of READ and WRITE that do not break phrase translations. We posit that this oracle provides the minimal context needed for SIMT to translate on the fly, with sufficient context to generate each output token.

Here our proposed system closely imitates the oracle trajectory. Our proposed method is more conservative in waiting for the input, waiting until the final verb to make precise prediction. This can be explained by the uncertainty over breaking the phrases by the programmer, and thus is incurs additional delay for the sake of better translation quality. Such behaviour that is observed in the output of human interpreters, who will often wait when they are unsure what the main speaker is talking about.

4.4 Oracle Behaviour towards Alignment

Section 4.3 has partly shown our oracle behavior in translating from a final-verb language into English. Here we discuss the oracle’s action when translating into a final verb language (English-Dutch). In English, the past participle is usually found right after the auxiliary verb. In this case, our oracle actions are conservative when waiting for inputs on the target side.

The example is shown in Figure 3 in which “have worked” does not produce a crossing alignment. First, the generated oracle will READ “have” and WRITE “heb”. Then it will examine the next word, “jaren”, and determine whether it needs to

[4]Note that we also try to finetune the wait-k system without scheduled sampling but it yields far worse performance than the one with scheduled sampling. This finding is similar with the main experiment.

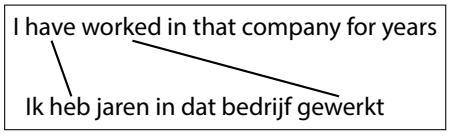

Figure 3: Translation example from English to Dutch. In this case "have worked" produce a non crossing alignment with "heb ... gewerkt" in Dutch.

READ more up until the word it is aligned. When it decided to WRITE "gewerkt"; the word "worked" would have been scanned in the past; so it should WRITE without an additional READ.

Next, it is also inevitable that our produced alignments are noisy and do not align all words correctly. It is currently not clear how much this will hurt the performance of the systems in terms of quality and delay due to alignment errors. In the worst case, our oracle will misguide the interpreter to guess target words without appropriate context (lowering quality, lowering delay) or wait for too many words (increasing quality, increasing delay). Both scenarios resulting from this noisy alignment are not catastrophic as it still depends on the interpreter's ability to guess translation outputs without appropriate input context.

4.5 Transformer and Large Scale Experiments

To show that the proposed method also extends to the transformer and larger parallel data, we conduct two experiments. The first experiment is changing the LSTM architecture with the transformer architecture similar to Ma et al. (2019). The second experiment uses 4.5 millions DE to EN parallel sentences from WMT 2015.

The interpreter is a standard 6 layers encoder-decoder NMT transformer similar to Vaswani et al. (2017). The programmer consists of single 6 layers encoder transformer with a binary classifier. For both networks, we allow attention to attend only to the previous timesteps. Then we use the program to mask out unseen inputs at each timestep in the interpreter. Unless otherwise specified, we use the default settings of training transformer as in Vaswani et al. (2017). We employ 30% dropout to all transformers and 10% to the embeddings, 16k vocabulary size, an average batch size of 4k, 8k steps learning rate warmup, 50 tokens maximum per sentence during training, for a total of 200k steps. We use a single pass of parallel scheduled sampling (Duckworth et al., 2019) for the transformer to generate $\boldsymbol{a}'$ and $\boldsymbol{y}'$ and set the $\boldsymbol{y}', \boldsymbol{a}', \boldsymbol{a}''$ perturbation rate to be 10%, 15%, and 25%. Training is completed within 20 hours on a single V100 GPU.[5]

	IWSLT			WMT		
	BLEU	AL	AP	BLEU	AL	AP
wait-k						
$k=1$	11.12	2.24	0.60	13.65	1.70	0.56
$k=2$	17.08	2.61	0.62	16.77	2.27	0.59
$k=3$	19.52	3.33	0.66	18.30	2.95	0.62
$k=4$	21.14	4.10	0.70	19.09	3.66	0.64
$k=5$	22.68	4.93	0.73	19.80	4.42	0.67
$k=6$	24.16	5.68	0.76	20.70	5.28	0.70
$k=7$	26.60	6.64	0.79	21.08	6.16	0.72
$k=\infty$	29.53	22.46	1.00	22.51	29.49	1.00
NPI-SIMT	22.06	1.73	0.56	17.57	3.25	0.59
$+\boldsymbol{a}'$	21.40	1.69	0.56	18.18	3.17	0.59
$+\boldsymbol{a}''$	23.28	1.85	0.57	18.70	3.33	0.60
$+\boldsymbol{a}', \boldsymbol{a}''$	23.82	1.83	0.57	18.78	3.54	0.60
$+\boldsymbol{a}', \boldsymbol{a}'', \boldsymbol{y}'$	24.54	1.78	0.57	19.07	3.27	0.59
Oracle-at-test	30.68	1.79	0.57	27.09	2.75	0.58

Table 3: Results using transformer on IWSLT and WMT corpora.

Table 3 presents the Transformer results on IWSLT and WMT. First we see that our transformer results are competitive or better than the LSTM on IWSLT dataset (compare with Table 1). These results are similar to the Ma et al. (2019) when comparing LSTM and transformer based architectures in the SIMT settings. Second, we see that our coupled scheduled sampling approach is also able to increase BLEU by up to 1.5 points compared to vanilla NPI-SIMT approach while also keeping the delay low (3.27 AL). The higher AL of the SIMT model in WMT compared to IWSLT is likely due to the higher AL of the oracle (2.75 vs. 1.79), which we attribute to the nature of the dataset. Arguably, this crawled corpus is less suitable for SIMT in general, because it contains considerably longer parallel sentences; moreover, the text is less reflective of a real simultaneous interpretation setting as it was built by only matching offline and post-edited texts. We are able to see similar improvements using coupled scheduled sampling over vanilla NPI-SIMT approach, showing the scalability of our approach.

5 Related Work

Satija and Pineau (2016); Gu et al. (2017) and Alinejad et al. (2018) formulate simultane-

[5]Because of the limitation of our computational resources, we are unable to use multiple GPUs for larger batch size. Using smaller batch size is known to reduce the overall performance of the transformer.

ous NMT as sequential decision making problem where an agent interacts with the environment (i.e. the underlying NMT model) through READ/WRITE actions. They pre-train the NMT system, while the agent's policy is trained using deep-RL.

Arivazhagan et al. (2020) highlights the poor performance of finetuned offline translation model when translating prefixes of input, which is the case of SIMT. Their approach uses retranslation strategy where every READ is performed, a new translation is generated from scratch, allowing revising translation on the fly and mitigating error propagation on the decoder that was attributed to the insufficient evidence when generating past output words. Their approach uses a stability metric which takes number of suffixes revisions made to produce latest translation. This approach involves wait-k inference, which limits number of words that can be emitted by the interpreter during one writing and thus limiting number of suffix revisions at the next writing. This wait-k inference is a heuristic that can be replaced by learning from the oracle.

Ma et al. (2019); Dalvi et al. (2018) introduced the fixed wait-k policy, which allows the integrated training of the NMT model wrt the fixed policy, as opposed to the adaptive policy of Gu et al. (2017); Arivazhagan et al. (2019) jointly trains an adaptive policy and re-trains the underlying NMT system. Arivazhagan et al. (2019); Zheng et al. (2019b) produces oracle READ/WRITE actions using a pre-trained NMT model, which is then used to train an adaptive agent based on supervised learning, i.e. behavioural cloning in imitation learning. Compared to our oracle which is produced merely from word alignment, their method requires a full decoding of training corpus, which is computationally expensive. These works are different from ours in that: (i) they do not use word alignment to produce the oracle actions, and (ii) they do not use of scheduled sampling.

6 Conclusion

This paper proposes a simple and effective way to train a simultaneous translation system to produce low delay translations. Our central contribution is to determine a sufficient, if not minimum, amount of inputs to translate each target token. This is achieved using word-alignment to create an oracle, which is then used as part of a training algorithm based on imitation learning to learn coupled policies, for a "programmer" which decides when to wait for more input producing translation tokens, and an "interpreter" which generates the translation. We show the importance of scheduled sampling during learning, which is crucial to combat exposure bias. Overall we show improvements in BLEU score over naively trained systems with modest translation delays.

Future work is needed to better understand the effect of various alignment models and symmetrization methods on the generated oracle. Beyond this, other opportunities include applying the model to the real speech input and applying more sophisticated imitation learning techniques that involves the generated trajectories of both "interpreter" and "programmer".

Acknowledgment

We thank anonymous reviewers for their valuable comments that helped to make this paper considerably better. This work is supported by the Australian Research Council (ARC DP160102686) and an Amazon Research Award.

References

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

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*.

Naveen Arivazhagan, Colin Cherry, Wolfgang Macherey, and George Foster. 2020. Re-translation versus streaming for simultaneous translation. *arXiv preprint arXiv:2004.03643*.

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, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*.

Samy Bengio, Oriol Vinyals, Navdeep Jaitly, and Noam Shazeer. 2015. Scheduled sampling for sequence prediction with recurrent neural networks. In C. Cortes,

N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems 28*, pages 1171–1179. Curran Associates, Inc.

Peter F. Brown, Stephen Della Pietra, Vincent J. Della Pietra, and Robert L. Mercer. 1993. The mathematics of statistical machine translation: Parameter estimation. *Computational Linguistics*, 19(2):263–311.

Mauro Cettolo, Christian Girardi, and Marcello Federico. 2012. Wit^3: Web inventory of transcribed and translated talks. In *Proceedings of the 16th Conference of the European Association for Machine Translation (EAMT)*, pages 261–268, Trento, Italy.

Fahim Dalvi, Nadir Durrani, Hassan Sajjad, and Stephan Vogel. 2018. Incremental decoding and training methods for simultaneous translation in neural machine translation. In *Meeting of the North American Chapter of the Association for Computational Linguistics (NAACL)*, pages 493–499.

Daniel Duckworth, Arvind Neelakantan, Ben Goodrich, Lukasz Kaiser, and Samy Bengio. 2019. Parallel scheduled sampling. *arXiv preprint arXiv:1906.04331*.

Chris Dyer, Victor Chahuneau, and Noah A. Smith. 2013. A simple, fast, and effective reparameterization of IBM model 2. In *Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 644–648.

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.

Jiatao Gu, Graham Neubig, Kyunghyun Cho, and Victor O.K. Li. 2017. Learning to translate in real-time with neural machine translation. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 1, Long Papers*, pages 1053–1062, Valencia, Spain. Association for Computational Linguistics.

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*.

Philipp Koehn, Franz J. Och, and Daniel Marcu. 2003. Statistical phrase-based translation. In *Proceedings of the 2003 Human Language Technology Conference of the North American Chapter of the Association for Computational Linguistics*, pages 127–133.

Taku Kudo and John Richardson. 2018. Sentencepiece: A simple and language independent subword tokenizer and detokenizer for neural text processing. *arXiv preprint arXiv:1808.06226*.

Thang Luong, Hieu Pham, and Christopher D. Manning. 2015. Effective approaches to attention-based neural machine translation. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1412–1421, Lisbon, Portugal. Association for Computational Linguistics.

Mingbo Ma, Liang Huang, Hao Xiong, Renjie Zheng, Kaibo Liu, Baigong Zheng, Chuanqiang Zhang, Zhongjun He, Hairong Liu, Xing Li, Hua Wu, and Haifeng Wang. 2019. 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, Florence, Italy. Association for Computational Linguistics.

Kenton Murray and David Chiang. 2018. Correcting length bias in neural machine translation. In *Proceedings of the Third Conference on Machine Translation: Research Papers*, pages 212–223, Brussels, Belgium. Association for Computational Linguistics.

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

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, Brussels, Belgium. Association for Computational Linguistics.

Stephane Ross, Geoffrey Gordon, and Drew Bagnell. 2011. A reduction of imitation learning and structured prediction to no-regret online learning. In *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, pages 627–635.

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

Faraz Torabi, Garrett Warnell, and Peter Stone. 2019. Recent advances in imitation learning from observation. In *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI-19*, pages 6325–6331. International Joint Conferences on Artificial Intelligence Organization.

Francis M Tyers and Murat Serdar Alperen. 2010. South-east european times: A parallel corpus of balkan languages. In *Proceedings of the LREC Workshop on Exploitation of Multilingual Resources and Tools for Central and (South-) Eastern European Languages*, pages 49–53.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 5998–6008. Curran Associates, Inc.

Baigong Zheng, Renjie Zheng, Mingbo Ma, and Liang Huang. 2019a. 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*, pages 1349–1354.

Baigong Zheng, Renjie Zheng, Mingbo Ma, and Liang Huang. 2019b. Simultaneous translation with flexible policy via restricted imitation learning. In *Proceedings of the 57th Conference of the Association for Computational Linguistics, ACL 2019, Florence, Italy, July 28- August 2, 2019, Volume 1: Long Papers*, pages 5816–5822.