University of Zurich[UZH]

Master's Thesis

Eager Machine Translation

Markus Göckeritz
13-750-633

Advisor: Mathias Müller

Prof. Dr. Rico Sennrich
Prof. Dr. Martin Volk

Department of Computational Linguistics
Faculty of Arts and Social Sciences

Department of Informatics
Faculty of Business, Economics and Informatics

23.03.2020

Abstract

Eager machine translation is a recent approach to simultaneous machine translation introduced by [Press and Smith, 2018]. The eager machine translation model translates simultaneously. More precisely, the model produces one translation token for every source token that it consumes, immediately after consuming it.

To achieve this, the eager translation model needs to learn when and how to wait for relevant information from the source sentence. [Press and Smith, 2018] introduce a novel pre-processing step where they add padding tokens, or `WAIT` tokens, to the target sentences. With this, the eager model is supposed to learn to wait.

We present a detailed analysis of the eager translation model and its inference hyperparameters. We show that the eager model is prone to producing translations that are either too short or too long. That is, translations consistently exceed sentence boundaries or are cut off. We show that the inference hyperparameters can be used as proxy parameters to control the lengths of the translations that the model produces.

We show that, for a random subset of all translations, the eager translation model predominantly produces `WAIT` tokens to wait for relevant information from the source sentence. That is, the eager translation model is capable of capturing, learning, and applying some of the waiting mechanics induced by the use of `WAIT` tokens. We are unable to show the extent to which this generally applies to the model.

[Press and Smith, 2018] use beam search to improve the quality of their translation model. With beam search, the final translation cannot be returned before the entire source sentence was processed. Beam search also increases translation latency. This defeats the purpose of eager translation and effectively makes eager translation a non-simultaneous process.

We apply sequence-level knowledge distillation [Hinton et al., 2015; Kim and Rush, 2016] and show that we can eliminate the need for beam search for eager translation entirely. Using a Transformer [Vaswani et al., 2017] as the teacher model and training an eager student on the outputs from running beam search with the Transformer, we improve the translation quality of the eager translation model by 3.6 `BLEU` with greedy decoding. The student model outperforms the baseline eager translation model by more than 2 `BLEU` while translating at almost twice the speed. The student model also exhibits better waiting behavior. It produces significantly fewer sentences that are too short but has a strong tendency to produce sentences that are too long. The student model outperforms our baseline eager translation model in every

dimension by a substantial margin.

It is generally assumed that knowledge distillation makes the training data more deterministic and less noisy [Gu et al., 2016, 2017; Zhou et al., 2019]. Similarly, knowledge distillation might make the distilled data set less semantically divergent [Vyas et al., 2018], more parallel, and more monotonous. This could explain the improved translation quality. We show that the distilled data set is significantly more parallel and monotonous than the original training data, and demonstrate that the student model produces translations with much higher confidence than the teacher model. We also show that the increase in determinism, parallelism and confidence does not explain the superior translation quality of the student model.

Zusammenfassung

Eager Machine Translation ist ein Übersetzungsmodell, welches kürzlich von [Press and Smith, 2018] vorgestellt wurde. Das Modell ist ein simultanes Übersetzungsmodell. Genauer gesagt, übersetzt das Modell *eifrig*, indem es für jedes Wort, welches es vom Quellsatz liest, genau ein Wort produziert, direkt nachdem es das Quellwort konsumiert hat.

Damit dies möglich ist, muss das Modell warten können. [Press and Smith, 2018] stellen einen speziellen Vorverarbeitungsschritt vor, in welchem den Zielsätzen der Trainingsdaten ein spezielles `Warte`-Zeichen eingefügt wird. Das Modell wird dann mit diesen `Warte`-Zeichen trainiert und lernt damit im Idealfall, wann und wie es warten muss.

Wir haben das Übersetzungsmodell und seine Hyperparameter genau untersucht. Wir präsentieren diese Analyse und zeigen, dass das Übersetzungsmodell von [Press and Smith, 2018] dazu neigt, Übersetzungen zu produzieren, die entweder zu kurz oder zu lang sind. Wir zeigen, dass die Länge der Übersetzungen mit den Hyperparametern kontrolliert werden kann und wie sich diese auf die Übersetzungsqualität auswirken.

Wir zeigen weiterhin, dass das Eager Translation Model fähig ist richtig zu warten. Wir analysieren das Warteverhalten des Modells auf einem zufälligen Subset aller Übersetzungen und stellen eindeutig fest, dass das Modell warten kann.

Das Modell muss dennoch in seinem Warteverhalten limitiert werden und es ist nicht klar, wann und wie das Modell genau wartet. Diesen Zusammenhang konnten wir nicht aufklären.

[Press and Smith, 2018] verwenden Beam Search um die Übersetzungsqualität ihres Modells zu verbessern. Jedoch kann mit Beam Search die Übersetzung erst zurückgegeben werden, wenn das Modell den gesamten Quellsatz verarbeitet hat. Zusätzlich verlangsamt Beam Search den Übersetzungsprozess. Dies macht Eager Translation zu einem nicht-simultanen Prozess und widerspricht dem Sinn, eifrig zu übersetzen.

Wir haben ein Transformer Model [Vaswani et al., 2017] trainiert und damit Sequence-Level Knowledge Distillation [Hinton et al., 2015; Kim and Rush, 2016] auf das Eager Translation Model angewendet. Ohne Beam Search erreicht unser Student Model eine Verbesserung der Übersetzungsqualität von 3.6 `BLEU`, im Vergleich zum Eager Translation Model, ebenfalls ohne Beam Search. Das Student Model erreicht eine Verbesserung von über 2 `BLEU` ohne Beam Search im Vergleich zum Eager Translation Model in seiner besten Konfiguration, mit Beam Search. In diesem Vergleich übersetzt unser Student Model mit der beinahe doppelten Geschwindigkeit des Standardmodells. Wir konnten damit die Notwendigkeit für Beam Search für eifrige Übersetzung eliminieren.

Das Student Model zeigt zusätzlich das bessere Warteverhalten und produziert deutlich weniger Übersetzungen, die zu kurz sind. Jedoch hat das Student Model eine starke Tendenz dazu, zu lange Sätze zu produzieren.

Das Student Model übertrifft unser Vergleichsmodell bei Weitem und in jeder Hinsicht.

Wieso Knowledge Distillation zu einer Verbesserung der Übersetzungsqualität führt, ist jedoch unklar. Knowledge Distillation soll die Trainingsdaten deterministischer [Gu et al., 2016, 2017; Zhou et al., 2019], semantisch ähnlicher [Vyas et al., 2018] und möglicherweise paralleler und monotoner machen. Dadurch könnte sich die Verbesserung in der Übersetzungsqualität des Student Model erklären lassen. Wir konnten diese Eigenschaften im destillierten Datensatz nachweisen und zeigen, dass das Student Model seine Übersetzungen mit einer deutlich höheren Konfidenz produziert. Wir zeigen zusätzlich, dass diese Eigenschaften nicht zur Erklärung der verbesserten Übersetzungsqualität ausreichen.

Acknowledgement

I would like to express my gratitude and appreciation to my advisor, Mathias Müller, and Prof. Dr. Rico Sennrich and Prof. Dr. Martin Volk for giving me the opportunity to write this thesis at the Department of Computational Linguistics.

I wish to thank my advisor, Mathias Müller, for his valuable guidance, his great patience, especially during the early stages of this work, his cheerful enthusiasm and his continuous support throughout the process of writing this thesis.

I would also like to thank Gregory Germann and David Ackermann for proofreading my work and for their valuable suggestions.

Finally, I wish to thank my parents for their support and encouragement throughout my studies.

Contents

List of Figures

List of Tables

List of Acronyms

BLEU	Bilingual Evaluation Understudy
BPE	Byte Pair Encoding
CNN	Convolutional Neural Network
DE	Deutsch
EN	English
EOS	End Of Sentence
GRU	Gated Recurrent Unit
LSTM	Long Short-Term Memory
LBLEU	Latency Bilingual Evaluation Understudy
NMT	Neural Machine Translation
RNN	Recurrent Neural Network
SPI	Source Padding Injection
WMT	Workshop on Statistical Machine Translation

1 Introduction

1.1 Motivation

Neural Machine Translation (NMT) is currently dominated by models that exhibit the following properties:

- **Bipartite structure:** Virtually all methods use so-called *encoder-decoder* models with an *attention* mechanism. Recurrent, convolutional, and self-attentional neural networks have all been proposed as encoder and decoder blocks. That is, the bipartite structure itself, meaning the division of labour between encoder and decoder, is rarely questioned [Cho et al., 2014b; Bahdanau et al., 2014; Vaswani et al., 2017; Gehring et al., 2017].
- **Autoregressive inference:** Most NMT models generate translations word by word, conditioning each decision on previously generated words. It is, however, common to *train* non-autoregressively, for instance, with Transformer models [Vaswani et al., 2017].
- **Non-simultaneous translation:** During training and inference, the first word of the translation is produced only after processing the entire input sentence. It is impossible for most models to translate simultaneously. That is, generating the first word of the translation after seeing only a prefix of the input.

In this thesis, we are concerned with *eager* machine translation and investigate and analyze the model that was introduced by [Press and Smith, 2018], a recent model that breaks some of those established conventions. More precisely, the eager model consists of a single recurrent neural network (RNN), rather than two distinct modules, and translates simultaneously, or *eagerly*, in the sense that it generates an output word immediately every time it consumes a word from the input sentence.

Translating simultaneously comes at the cost of being unable to revert past decisions. Since an output word is generated immediately after an input word is consumed, subsequent tokens, and thereby potentially essential information, are not taken into

consideration for the translation of a word at any given step. This is particularly challenging for the translation of words whose real meaning only becomes evident subsequently in the input sentence. As an example, consider the following scenario:

Input:	`Der Schläger war wütend.`
Desired Output:	`The thug was furious.`
Simultaneous Translation Prefix:	`The racket was ___`

After seeing "Der Schläger war" as inputs and each time generating the next word of the translation, a simultaneous model would now be presented with "wütend" as the next word. This new evidence drastically reduces the likelihood of "racket" as the correct translation for "Schläger". Yet, a simultaneous model cannot change the word "racket", now a ridiculous translation, to "thug", now a more likely translation, as it has already generated and returned "racket" in a previous step. To avoid situations like these, simultaneous models are trusted to *anticipate* future words that they have not yet seen [Grissom II et al., 2014; Ma et al., 2018].

Naturally, anticipating future events may not always be possible. Another technique to counteract this apparent shortcoming of translation models is *beam search*. In the example above, beam search could be used to hold on to both "thug" and "racket" as the translations of "Schläger" and decide between them later, after seeing more words from the input sentence.

However, beam search arguably removes the simultaneity of the translation from the eager model. With beam search, the translation cannot be returned until the entire input sentence was read. This effectively makes eager translation a non-simultaneous procedure.

Additionally, using beam search increases translation latency. [Press and Smith, 2018] describe the eager model as low-latency, but latency increases with beam search and the beam sizes used in this model are unusually high.

Tuned on the development set, they are as high as 35 for some experiments. Common beam sizes range from 5 to 10. In conventional models, a beam size of 35 does not improve translation quality over beam size 5 or 10. On the contrary, if the training data is noisy, a beam size of 35 might even lower translation quality [Ott et al., 2018].

1.2 Research Questions

Lower Beam Size with Knowledge Distillation

We identify beam search as a key stumbling block for simultaneous and low-latency eager translation. The primary goal of this work is to lower beam size or remove beam search altogether, while not sacrificing translation quality in the eager translation model introduced by [Press and Smith, 2018].

We hypothesize that applying sequence-level knowledge distillation to the eager translation model may reduce the necessary beam size or even eliminate the need for beam search altogether.

Knowledge distillation was initially proposed by [Buciluǎ et al., 2006] and [Hinton et al., 2015] as a form of model compression where the knowledge of a large model, called the *teacher model*, is transferred to a smaller model, called the *student model*, by training the student model on the predictions of the teacher model. [Kim and Rush, 2016] show that knowledge distillation can be applied to neural machine translation systems and further demonstrate that sequence-level knowledge distillation may eliminate the need for beam search in NMT systems entirely. We anticipate similar results when applying sequence-level knowledge distillation to the eager translation model. In a similar context, [Gu et al., 2017] have used sequence-level knowledge distillation to improve the performance of non-autoregressive models by teaching them on the predictions of a conventional Transformer model.

Analyze Properties of Knowledge Distillation

The reasons for the improvement in translation quality are not entirely clear. [Gu et al., 2017] generally assumed that the predictions of the teacher model make the training data more deterministic and less noisy as models tend to translate consistently. That is, a neural translation model will consistently produce the same translation when given the same source sentence. [Zhou et al., 2019] later showed that sequence-level knowledge distillation does indeed make the training data more deterministic and that knowledge distillation is essentially equivalent to selecting "modes" over the original data set. Predictions of the teacher model may also make the training data less semantically divergent [Vyas et al., 2018] than the original training data.

We hypothesize that this may also be the case for the eager translation model. We are further interested in the *monotonicity* in the distilled training data. Natural translations are not *eager-feasible*. The ith source word might be aligned to any word

in the translation. If knowledge distillation makes translations more monotonous, this could make eager translation substantially easier and could explain the improved translation quality.

1.3 Thesis Structure

In Chapter 2, we present an overview of the related scientific research that elaborates on neural machine translation, simultaneous translation, and knowledge distillation.

We describe the eager translation model as introduced by [Press and Smith, 2018] in Chapter 3 and create the context for this work.

We present a detailed quantitative analysis of the eager translation model and its inference hyperparameters in Chapter 4.

In Chapter 5, we present a qualitative analysis of the translations that the eager model produces and discuss its ability to wait for relevant information from the source sentence.

We apply sequence-level knowledge distillation and present our findings in Chapter 6. We also propose two measures to quantify determinism in our models and to compute parallelism in our data sets and show the results of our computations.

The thesis concludes with the general results of our evaluations as well as our thoughts on the contributions we were able to make.[1]

[1]Substantial parts of the introduction were taken from the thesis proposal.

2 Background

This thesis is concerned with *eager* machine translation, a recent approach to simultaneous machine translation introduced by [Press and Smith, 2018]. In this chapter, we present an overview of the relevant scientific literature.

We describe traditional and current approaches to neural machine translation in Chapter 2.1 to give the broader context to this work.

We elaborate on simultaneous machine translation in Chapter 2.2 and show traditional as well as more recent approaches to simultaneous machine translation. Eager translation is an approach to simultaneous machine translation. This gives us the detailed context to this work and puts eager translation in comparison with other approaches to simultaneous machine translation.

In Chapter 2.3, we present an overview of knowledge distillation and knowledge distillation in neural machine translation systems. This chapter is particularly relevant as the primary work of this thesis is to apply knowledge distillation to the eager machine translation model.

2.1 Neural Machine Translation

Neural machine translation systems are translation systems where the model components are neural networks. These translation models are commonly implemented as *encoder-decoder* models that subdivide the translation process into two consecutive steps. The encoder reads a source sentence and encodes it into a vector representation of fixed length. This vector is then passed to the decoder, and the decoder transforms it into a sentence in the target language [Kalchbrenner and Blunsom, 2013; Sutskever et al., 2014; Wu et al., 2016; Cho et al., 2014b; Bahdanau et al., 2014; Luong et al., 2015; Vaswani et al., 2017; Gehring et al., 2017].

This bipartite architecture, i.e., the division of labour between the encoder and decoder and the transformation of the source sentence into a single vector, enables the decoder to produce a translation whose length does not depend on the length of

the source sentence.

The objective of the decoder is to estimate the conditional probability $p(y|x)$ where $x = (x_1, ..., x_T)$ is a source sentence of length T and $y = (y_1, ..., y_{T'})$ is the target sentence with length T', such that T and T' may differ.

In neural machine translation, the encoder and decoder are two distinct neural networks. The three most popular types of neural networks that are used are Recurrent Neural Networks (RNNs), Convolutional Neural Networks (CNNs), and Transformer networks. We describe the use of RNNs and Transformers in neural machine translation in the following two sections but do not elaborate on CNNs in neural machine translations because they do not appear in our work.

2.1.1 RNN Encoder-Decoder

Traditionally, both the encoder and decoder are implemented as two distinct Recurrent Neural Networks. The specific implementation and type of RNNs differ between architectures. [Sutskever et al., 2014; Wu et al., 2016] and [Luong et al., 2015] used multi-layered RNNs with Long Short-Term Memory (LSTM) units [Hochreiter and Schmidhuber, 1997] and [Cho et al., 2014b] and [Bahdanau et al., 2014] used an LSTM-inspired hidden unit, also known as Gated Recurrent Unit (GRU).

In this RNN encoder-decoder architecture, the encoder reads the source sentence $x = (x_1, ..., x_T)$, one word at a time, and encodes it into a sequence of hidden states h_t

$$h_t = f(x_t, h_{t-1}) \tag{2.1}$$

where f is a nonlinear function, such as an LSTM unit.

The encoder produces a summary of the source sentence as a fixed-length vector representation c of the input sequence x.

$$c = q(h_1, ..., h_T) \tag{2.2}$$

Typically, this context vector c is the last hidden state h_T of the encoder. Another method is to use the average of all hidden states of the encoder.

The decoder is initialized with the context vector c from the encoder and decodes c back into a variable-length sequence in the target language. Similar to the encoder,

the decoder generates a translation word by word and conditions its prediction at time step t on the prediction of the previous time step y_{t-1}. This recurrent form of translation is referred to as *autoregressive* inference. Specifically, at each time step t, using the context vector c from the encoder, the current hidden state s_t of the decoder and the prediction from the previous time step y_{t-1}, the decoder computes the conditional probability

$$p(y_t|y_1, ..., y_{t-1}, c) = g(y_{t-1}, s_t, c) \tag{2.3}$$

where g is a nonlinear function that produces a probability distribution over all the words in the target vocabulary for the next symbol y_t.

The result of the decoder is the conditional probability

$$p(y_1, ..., y_{T'}|x_1, ..., x_T) = \prod_{t=1}^{T} p(y_t|y_1, ..., y_t) \tag{2.4}$$

such that $p(y|x)$ is a distribution of class probabilities represented with a *softmax* over all the words in the target vocabulary and the target sentence $\hat{y} = \text{argmax}_y\, p(y|x)$ is the most likely translation.

The encoder and decoder are trained jointly to maximize the conditional probability of producing a correct translation y given a source sentence x. The model is trained on multi-class predictions at the word-level with the objective to minimize the cross-entropy between the predictions of the model and the training data at each iteration in the decoder.

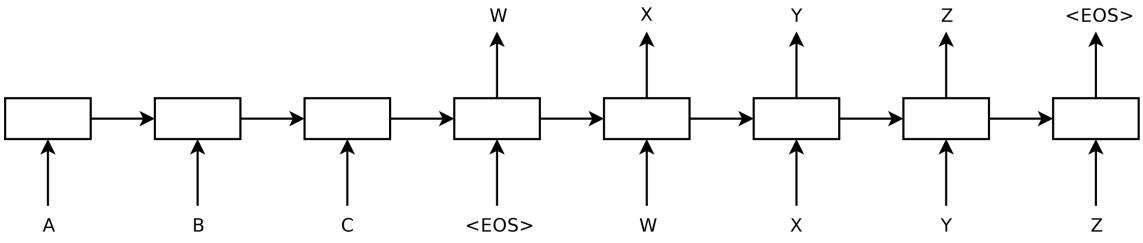


Figure 1: Single-layer RNN encoder-decoder model translating "ABC" into "WXYZ". Figure taken from [Sutskever et al., 2014].

2.1.2 Attention-Based Neural Machine Translation

In the traditional RNN encoder-decoder approach, the decoder uses the same context vector c in every iteration and does not change or update the context vector as the translation process progresses. This requires the encoder to compress all relevant information from the source sentence into a single vector of fixed length, which makes it difficult for basic encoder-decoder models to translate long source sentences [Cho et al., 2014a].

[Bahdanau et al., 2014] extend this approach with an alignment model that allows the decoder to search for relevant information from the source sentence during decoding. More precisely, in their architecture, the encoder produces a *sequence* of hidden states $(h_1, ..., h_T)$ from a bidirectional RNN. The encoder consists of two distinct RNNs that process the source sentence in both directions. A forward RNN processes the source sentence in its original order and produces a set of forward hidden states $(\overrightarrow{h_1}, ..., \overrightarrow{h_T})$. A backward RNN processes the source sentence in reverse order and produces a set of backward hidden states $(\overleftarrow{h_T}, ..., \overleftarrow{h_1})$. These hidden states are then concatenated such that the encoder state at time step h_j contains both hidden states $h_j = [\overrightarrow{h_j^T}; \overleftarrow{h_j^T}]$. The motivation is to include information about both the preceding as well as the following words in a single encoder state.

Then, rather than conditioning the translation on a summary of all encoder states, the decoder computes a new context vector c_i during every time step i. This context vector c_i is computed as a weighted sum of all encoder states it receives

$$c_i = \sum_{j=1}^{T} a_{ij} h_j \tag{2.5}$$

where the weight a_{ij} is a "softmax" output

$$a_{ij} = \frac{\exp(e_{ij})}{\sum_{k=1}^{T} \exp(e_{ik})} \tag{2.6}$$

of an alignment model e_{ij} that computes a score for how well an input token at position j matches an output token at position i

$$e_{ij} = a(s_{i-1}, h_j) \tag{2.7}$$

where h_j is the encoder state at position j, and s_{i-1} is the hidden state of the decoder after the previous prediction. The weight a_{ij} defines how much of the encoder state

h_j should be considered for the generation of the ith output token.

The alignment model is a feedforward neural network that is jointly trained with the encoder and decoder to maximize the conditional probability of producing a correct translation.

This targeted selection of hidden states from the encoder is called *attention*. It enables the encoder to produce more than a single vector for the entire source sentence and allows the decoder to selectively scan the result of the encoder for relevant information. [Bahdanau et al., 2014] and [Luong et al., 2015] show that encoder-decoder models with attention perform much better on longer sequences.

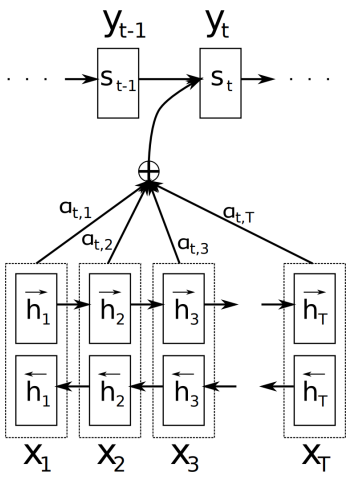


Figure 2: Attention-based encoder-decoder model. Bidirectional RNN computes hidden states which are used to compute context vectors. Figure taken from [Bahdanau et al., 2014].

2.1.3 Transformer

[Vaswani et al., 2017] introduced a novel sequence transduction model, the Transformer, that is entirely built on self-attention and does not use any recurrent neural networks.

In traditional RNN encoder-decoder architectures [Bahdanau et al., 2014], the attention mechanism computes scores for how well each source token matches the current output token. During every step of the decoder, the attention function, or alignment function, computes a weight between the current hidden state of the decoder and every hidden state of the encoder. Then, using these weights, the attention mechanism computes a new context vector as a proportional selection from the hidden states of the encoder. Given a source sentence of length n and a target sentence of length m, this computation is performed $n * m$ times in total.

In self-attention, the attention mechanism computes attention scores for all inputs and between all inputs. The motivation is to capture dependencies between tokens and to capture the internal structure of the sentence, and thus, everything attends to everything.

[Bahdanau et al., 2014] use a feedforward network to compute the alignment between inputs, whereas [Vaswani et al., 2017] use the scaled dot-product. [Vaswani et al., 2017] use the scaled dot-product because, with it, self-attention can be implemented with a single matrix multiplication, which makes the operation very fast and space-efficient.

$$Attention(Q, K, V) = softmax(\frac{QK^T}{\sqrt{d_k}})V \tag{2.8}$$

Q, K and V in Equation 2.8 are all matrices.

The Transformer consists of an encoder and a decoder module. Both encoder and decoder consist of a stack of Transformer blocks where each Transformer block consists of a self-attention mechanism and a feedforward neural network. In the encoder, the attention mechanism computes self-attention for and between all source tokens. In the decoder, the attention mechanism computes self-attention for and between all target tokens. The attention results and the original tokens are added together, normalized, and linearly transformed.

The decoder contains an additional attention component that computes self-attention between the outputs of the encoder and the previous results of the decoder. It computes self-attention where the queries are decoder states, and the keys and values are encoder states.

All attention blocks in [Vaswani et al., 2017] are multi-head attention mechanisms. That is, the attention computation is performed multiple times in parallel, once for every head. The results of all computations are then simply concatenated and linearly transformed into the expected dimensions. This allows the model to consider multiple different representations at different positions, almost like using an ensemble of attention mechanisms.

2.1.4 Beam Search

During the decoding process, at every time step t, the decoder produces a probability distribution for the next token y_t in the translation given the current translation context. The default strategy might be to select the most likely token $\hat{y}_t$ and proceed

with the translation process. However, since an NMT system produces an entire sentence, the sequence distribution is more important than the likelihood of a single token at some time step t. It might very well be that the most likely *sequence* $\hat{y}$ is different to the sequence of the most likely tokens $y = \{\hat{y}_1, ..., \hat{y}_T\}$. Essentially, to find the best translation for a source sentence x, the NMT system needs to find the most likely sequence $\hat{y}$ among all possible sequences y. However, this is not feasible. The sequence distribution over the sample space of all possible sequences grows exponentially large with every token in the sequence, and computing this sequence is intractable.

Beam search is a search algorithm to explore the sample space of all possible sequences by holding on to multiple hypotheses during the translation process. The number of hypotheses that are held on to during the translation process is called the *beam size*. At the first time step, the translation system selects the k most likely tokens as k different hypotheses. At every subsequent step, the NMT system expands all hypotheses and considers the k most likely tokens for every hypothesis. Then, the relative probabilities of all expanded hypotheses are compared, and only the k most likely sequences are kept. This happens at every time step t and produces k translation candidates at the end of the translation process, from which the NMT system selects the most likely sequence.

2.2 Simultaneous Machine Translation

Simultaneous translation systems differ from traditional neural machine translation systems in that a simultaneous translation system produces a partial translation of the source sentence before it has processed the entire input. For each token that is read from the source sentence, a simultaneous translation system may return any number of translation tokens before it consumes the next token from the input. It may also emit zero tokens and therefore *wait* for more input. Translating simultaneously means sequentially deciding between reading the next word or returning another translation. It should be noted that "translation" in this context means one or more tokens, not an entire target sentence.

Reading more input words gives the translation system more evidence and context, but it also delays the translation. In a setting in which it is crucial to receive meaningful partial translations in a timely manner, such as diplomatic meetings or in real-time speech translation, the objective of a translation system is usually a well-balanced trade-off between the quality of the translation and the delay at which it is returned [Grissom II et al., 2014; Cho and Esipova, 2016; Gu et al., 2016; Satija

and Pineau, 2016].

Simultaneous translation is particularly challenging because essential information may be unavailable at a given translation step and be hidden in subsequent tokens. The true meaning of a token may only become evident after subsequent tokens are processed. Consider the following two sentences:

> She passed out the tests.
> She passed out from shock.

The first three words of both sentences are identical. Reading only "She passed out", a simultaneous translation system would produce the same translation prefix for both sentences, because the true meaning of the words "passed out" only becomes evident after reading subsequent tokens.

Additionally, a word may appear in the translated sentence before it appears in the source sentence because different languages have different word orders. This makes simultaneous translation substantially more difficult [Grissom II et al., 2014].

A simultaneous translation system must not only produce a high-quality translation, given a sentence in the source language; it must also perform a segmentation of the source sentence that enables the translation of sub-sentence sequences, without sacrificing translation quality and while retaining the original semantics.

Traditional statistical simultaneous translation systems divide the translation process into two distinct steps. First, a phrase-based segmentation model reads the input sentence, one word at a time, and constructs phrases. Then, these phrases are sent to a machine translation model. The simultaneity in these types of systems is essentially achieved by segmenting the input sentence into sub-sentence chunks and then translating these chunks separately [Bangalore et al., 2012; Fujita et al., 2013; Sridhar et al., 2013; Yarmohammadi et al., 2013; Oda et al., 2014].

These two stages are largely independent of each other, and the translation model may be of any reasonable choice, including, with the recent successful application of neural networks in machine translation tasks, a neural machine translation system. However, a more holistic approach might be more suitable. That is, instead of deciding how to segment the input sequence into chunks and then translating these chunks in an isolated fashion, the segmentation steps could be informed by how previous chunks have been translated such that the segmentation and translation are performed by a single architecture.

2.2.1 Simultaneous Greedy Decoding

Recently, architectures where the segmentation is performed by a subcomponent of the translation model have been proposed [Cho and Esipova, 2016; Dalvi et al., 2018]. These architectures produce translations in chunks by evaluating heuristics during the decoding process and returning partial translations if particular conditions are met. Among those papers, we have selected [Cho and Esipova, 2016] to illustrate such a heuristic approach.

[Cho and Esipova, 2016] propose a neural machine translation model that translates simultaneously. In their approach, [Cho and Esipova, 2016] use an existing attention-based neural machine translation model [Sutskever et al., 2014; Bahdanau et al., 2014] and modify the decoder such that during every iteration, the decoder decides between writing and reading tokens. The authors call their modified decoding process *simultaneous greedy decoding*.

Simultaneous greedy decoding defines two hyperparameters, *step size* δ and *initially-read source symbols* s_0. These two parameters control the trade-off between translation quality and translation latency. The step size δ defines how many source tokens are read during every read operation, and the number of initially-read source symbols s_0 defines the number of source tokens that are consumed in the first iteration of the decoder.

After reading the initial s_0 source tokens, the decoder produces a translation candidate y_t given the current translation context C. Then, during every following iteration, the decoder reads δ tokens and produces an updated translation context $C^{'}$ and decides between committing the translation candidate y_t and waiting for an additional batch of δ source tokens. This decision is based on a waiting criterion Λ that compares the translation contexts C and $C^{'}$. Whenever the criterion Λ is met, the decoder updates the current translation context C with the new translation context $C^{'}$ as well as the current source symbols s with the previous source symbols s, or s_0 for the first iteration, and the new source symbols δ.

[Cho and Esipova, 2016] introduce two waiting criteria, *wait-if-worse* and *wait-if-diff*. The criterion *wait-if-worse* compares the confidence of a prediction y_t given the previous set of source tokens s and the previous translation context C and the new set of source tokens $s + \delta$ and the new translation context $C^{'}$.

$$\Lambda(C, C \cup C^{'}) : \log p(y_t|y_{<t}, C) < \log p(y_t|y_{<t}, C \cup C^{'}) \tag{2.9}$$

If the confidence in the prediction y_t decreases with additional source tokens, the translation y_t is committed.

Similarly, *wait-if-diff* compares the most likely predictions between the previous and the current iteration

$$y_t = \underset{y}{\operatorname{argmax}}\, p(y|y_{<t}, C) \tag{2.10}$$

and

$$y_t^{'} = \underset{y}{\operatorname{argmax}}\, p(y|y_{<t}, C \cup C^{'}) \tag{2.11}$$

If the two predictions are different, the waiting criterion *wait-if-diff* is met. If the predictions are equal, the translation token y_t is committed.

This approach to simultaneous machine translation is *neural* in the sense that [Cho and Esipova, 2016] use a neural network to perform the translation. However, the segmentation in their model is done using heuristic modifications to the decoding process, and the waiting criteria that [Cho and Esipova, 2016] propose are not trainable.

2.2.2 Rewarding Waiting Behavior in Simultaneous Translation Systems

Heuristic approaches like simultaneous greedy decoding, as well as traditional approaches to using separate segmentation and translation models, lack a holistic approach to simultaneous translation. Ideally, a simultaneous translation system can be trained as one single architecture that performs both translation and segmentation.

More recent approaches to simultaneous neural machine translation employ reinforcement learning where an agent is trained with a reward function that considers both quality and delay [Grissom II et al., 2014; Satija and Pineau, 2016; Gu et al., 2016].

However, this is exceptionally challenging, and it is not apparent how to best measure the trade-off between latency and accuracy and how to appropriately capture "translation quality" within these two metrics.

[Grissom II et al., 2014] propose a modified version of BLEU as a measurement of the quality and performance of simultaneous translation systems. Particularly, [Grissom II et al., 2014] introduce *latency-bleu* or LBLEU as a word-by-word integral over partial translations $(y_1, ..., y_{T'})$ and the source sentence as the reference sequence r

$$\text{LBLEU}(x, y) = \frac{1}{T} \sum_{t \in T} \text{BLEU}(y_t, r) + T * \text{BLEU}(y_T, r) \tag{2.12}$$

where $x = (x_1, ..., x_T)$ is the source sentence, $y = (y_1, ..., y_{T'})$ is the target sentence, and y_t is the partial translation at time step t.

Whenever a new source word is revealed, LBLEU compares the current partial translation y_t with the source sentence x and computes the BLEU score for these two inputs. These scores are added together and normalized with the source sentence length. This produces a bias towards low latency because y_t grows over time, and the earlier $\text{BLEU}(y_t, x) > 0$, the longer the model is rewarded for it. Thus, [Grissom II et al., 2014] add the BLEU score of the final translation multiplied by the source sentence length to the score. Essentially, T could be replaced with any parameter β to bias towards different levels of latency. The bias towards latency decreases as $\beta \to \infty$.

[Satija and Pineau, 2016] extend LBLEU as proposed by [Grissom II et al., 2014] with a decay in $\text{BLEU}(y_t, r)$ whenever y_t is not updated between two time steps and add a penalty to LBLEU to punish the model whenever the translation quality between two partial translations y_t and $y_{t'}$ decreases. Also, rather than the length of the source sentence T, [Satija and Pineau, 2016] use the maximum permissible target sentence length as β.

[Grissom II et al., 2014; Satija and Pineau, 2016] and [Gu et al., 2016] all use their modified version of BLEU as a reward function.

2.3 Knowledge Distillation

Knowledge distillation [Hinton et al., 2015] describes a process where the knowledge of one neural network, the *teacher* model, is transferred to another neural network, the *student* model. The teacher model is used to produce a *transfer set*, which is then used to train the student model and thereby transfer the knowledge from the original model to a new one.

More precisely, rather than training a model on the *hard* targets from the original

training data, the student model is trained on *soft* targets produced by the teacher model. The soft targets are the class probabilities produced by the teacher model by applying a *softmax* output layer that converts the logit z_i of each class into a probability q_i

$$q_i = \frac{\exp(z_i/T)}{\sum_{j \in J} \exp(z_j/T)} \tag{2.13}$$

where T is a temperature to soften the probability distribution.

Hard targets from the training data are one-hot distributions of the target classes represented as vectors where every row corresponds to a class in the target space with 0s for all incorrect classes and a single 1 for the correct class. The softmax output of a neural classifier contains a probability for every class, and using a temperature $T > 1$ will produce a softer probability distribution over the classes.

$$\begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} \qquad \begin{bmatrix} 5.6e\text{-}09 \\ 0.99987 \\ 0.00012 \\ 1.5e\text{-}08 \end{bmatrix} \qquad \begin{bmatrix} 0.0610 \\ 0.6566 \\ 0.2132 \\ 0.0692 \end{bmatrix}$$

(a) One-hot vector encoding. (b) Softmax output with $T = 1$. (c) Softmax output with $T > 1$.

Figure 3: (a) shows a one-hot vector encoding of a training example, (b) shows a probability distribution over the target space produced by a softmax output layer and (c) shows the same outputs when the logits are normalized with a greater value for T.

These soft targets provide more information per training case than the hard targets from the original training data. This is because a model also assigns probabilities to all the incorrect classes. For instance, in a neural classifier that is trained to recognize handwritten digits, the soft targets of the teacher model will not only tell which "1"s look like "1"s, but also which "1"s look very similar to "7"s. Similarly, the probability assigned to "7" may be substantially smaller than the probability assigned to "1", yet it will still be many times more than the probability for an "8".

The student model trained on the predictions of the teacher model will make predictions that are similar to those of the teacher by learning to approximate the behavior and generalization ability of the teacher. However, the teacher model can be of any arbitrary size and complexity. The teacher model can be significantly larger than the student model or even be a set of models, i.e., an ensemble.

[Buciluǎ et al., 2006] introduced this approach as a form of *model compression*. In their approach, [Buciluǎ et al., 2006] trained an ensemble of models that is highly accurate but cumbersome to use in practice. Then, by training a single smaller model

on data produced by the ensemble, [Buciluǎ et al., 2006] were able to compress the knowledge of the large ensemble in the smaller model without sacrificing accuracy.

[Hinton et al., 2015] extended the approach of model compression [Buciluǎ et al., 2006] in a more general setting. In cases in which the teacher model produces the correct answer with very high confidence, the probabilities in the soft targets for incorrect classes are close to zero. This makes it difficult to transfer the knowledge from the teacher model because these values only have little influence on the cross-entropy cost function. [Hinton et al., 2015] show that this issue can be circumvented by using higher temperatures T to further soften the probability distribution.

Neural machine translation systems are typically very large. This makes them ideal candidates for knowledge distillation. [Kim and Rush, 2016] demonstrate that knowledge distillation can be applied to NMT models and introduce a novel version of knowledge distillation, *sequence-level* knowledge distillation.

NMT systems are commonly trained on multi-class predictions at the word-level, meaning that the training objective is to minimize the negative log-likelihood (NLL) at each iteration in the decoder. In this context, standard knowledge distillation can be directly applied, such that the student model mimics the predictions of the teacher model at each position. [Kim and Rush, 2016] refer to this approach as *word-level* knowledge distillation

During inference, however, we put more importance on the sequence distribution since the task of the NMT system is to predict a complete sentence. Thus, ideally, the student model learns to mimic the teacher model at the sequence-level. However, the sequence distribution over the sample space of all possible sequences is exponentially large, and thus computing and approximating this sequence is intractable.

[Kim and Rush, 2016] show that using the output from running beam search with the teacher model is a reasonable approximation of the teacher's sequence distribution.

Most surprisingly, [Kim and Rush, 2016] report that, after applying sequence-level knowledge distillation, the student model demonstrates similar performance to the teacher model, even when the student model decodes greedily. That is, [Kim and Rush, 2016] trained a neural translation model as in [Luong et al., 2015] with four layers and 1000 hidden units (4×1000) on the WMT 2014 `EN` $\leftrightarrow$ `DE` data set. Then, [Kim and Rush, 2016] used the outputs from running beam search with the teacher model and used these outputs to train two smaller student models with sizes 2×500 and 2×300. The teacher model achieved a `BLEU` score of 17.7 without beam search and a `BLEU` score of 19.5 with a beam size of 5. With greedy decoding, the student models achieved `BLEU` scores of 18.9 and 18.1, respectively. However,

on their hardware, the teacher model translates at only 425.5 words per second, whereas the smaller student models achieve 1051.3 and 1267.8 words per second, respectively. The larger student model produces translations with similar quality but at more than twice the speed. This allows [Kim and Rush, 2016] to deploy a translation model similar in performance to the large teacher model on significantly less capable hardware.

3 Eager Machine Translation

In this chapter, we describe the eager translation model as introduced by [Press and Smith, 2018] and show how eager translation is achieved. Particularly, we outline a detailed description of eager feasible pre-processing and describe and discuss the hyperparameters that eager translation offers for inference.

3.1 Introduction

Eager machine translation is a simultaneous model recently proposed by [Press and Smith, 2018]. The model is a modified version of a traditional encoder-decoder model [Bahdanau et al., 2014; Zaremba et al., 2014], that does not use attention and that combines the encoder and decoder into a single recurrent neural network. With this, the model is not divided into two consecutive modules and does not require processing the entire input sequence before emitting the first token of the translation. Consequently, the model is capable of translating *eagerly*. That is, the eager translation model returns exactly one translation token for every input token that it consumes, immediately after consuming it. Other than these changes, the eager model of [Press and Smith, 2018] strongly resembles the more traditional recurrent language model of [Zaremba et al., 2014].

3.2 Eager Feasible Pre-processing

To achieve the waiting dynamics of a simultaneous machine translation system, [Press and Smith, 2018] introduce a special ε token that is added to the training data in an additional step during pre-processing. Particularly, the ε token is used to resolve *non-monotonous* alignments between words of the source sentence and words of the target sentence.

Let $x_i \in x$ be the ith token of the source sentence x and let y_j be the jth token of the target sentence y such that x_i aligns with y_j, according to an alignment model.

We say that the alignment between x_i and y_j is *monotonous* if $i \leq j$. That is, an alignment is monotonous if the source word appears *before* or at the same index as the target word.

[Press and Smith, 2018] infer a correspondence between words in the source and target sentence using an alignment model [Dyer et al., 2013] and then add ε tokens to the target sentence such that no token in the target sentence appears before its corresponding token from the source sentence.

The number of ε tokens that are added to the target sentence is *minimal*.

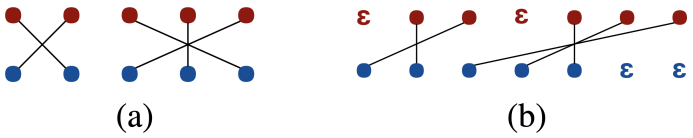


Figure 4: Word alignments before (a) and after (b) adding ε tokens. Source words are blue, target words are red and corresponding words are connected. Figure taken from [Press and Smith, 2018]

Figure 4 illustrates this with an example. Words from the source sentence are in blue, words from the target sentence are in red, and corresponding words are connected.

In (a) in Figure 4, some words from the source sentence (blue) only appear *after* the corresponding words from the target sentence (red). For instance, the first word from the target sentence is aligned to the second word of the source sentence. It appears in the translation *before* it appears in the source, making it impossible for the eager model to consider this source token for translation.

In (b) in Figure 4, we see the same sentence pair after pre-processing and after ε tokens were added. The first word of the target sentence is now an ε token. This moves the token that was previously at the first index to the right and thereby resolves the non-monotonous alignment between it and the second word from the source sentence.

After this pre-processing step, there is no word in the target sentence (red) that appears before its corresponding word from the source sentence (blue). That is, let (x_i, y_j) be a pair of aligned words from the source sentence x and the target sentence y. Before pre-processing, it is possible that $i > j$, i.e., the target token t_j appears *before* the source token s_i. After pre-processing, the target sentence is padded with ε tokens, such that $i \leq j$ for all word alignment pairs (x_i, y_j). If this inequality holds for all alignments of a pair of sentences, the source and target sentences are said to be *eager feasible*.

Eager feasible pre-processing is supposed to instill in the model the concept of waiting

for more information. In cases in which the translation of a given token of the input sentence follows subsequently in the output sentence, the model is trained to predict the ε token and thus *delay* further translation for another time step. Effectively, **the ε token can be considered a `WAIT` token** and, intuitively, by learning when to predict the `WAIT` token, the model learns when to wait. Since the set of ε tokens that are added during eager feasible pre-processing is minimal, the model also learns to avoid waiting unnecessarily.

Furthermore, since the eager model returns exactly one token for every input token, the model requires the source and target sentence pairs in the training data to have the same number of tokens. If this is not the case, the shorter sentence is padded with additional ε tokens. These padding ε tokens are added to the end of the sentence but before the `EOS`-token.

Next to the ε tokens, [Press and Smith, 2018] introduce so-called *start padding tokens*. These are padding tokens that are added to the start of the target sentence of a training pair in order to shift the original tokens in the target sentence to the right. This allows the model to consume more tokens from the source sentence before producing the first translations and, ideally, helps the model to produce better translations. The number of start padding tokens is the same for all target sentences in the training data and is a variable hyperparameter.

We illustrate eager feasible pre-processing with an example. Consider the following source and target sentence pair:

Source:	`I did not go to school today . EOS`
Target:	`Ich ging heute nicht zur Schule . EOS`

and their equivalents after eager feasible pre-processing:

Source:	`I did not go to school today .` $\varepsilon\ \varepsilon\ \varepsilon$ `EOS`
Target:	$\varepsilon\ \varepsilon$ `Ich ging` $\varepsilon\ \varepsilon$ `heute nicht zur Schule . EOS`

The target sentence after pre-processing contains two ε tokens at the beginning of the sentence. These are the start padding tokens. For this example, we chose two start padding tokens. Then, the pre-processed target sentence contains another two ε tokens. These ε tokens are added to resolve the alignment between "today" and "heute", because "heute" appears much earlier in the translation. After adding these tokens, the target sentence contains three more tokens than the source sentence. Hence, the source sentence is padded with three ε tokens.

Notice that the non-monotonous alignment between "go" and "ging" in the original sentence pair did not require any additional ε tokens because the mismatch was

already resolved with the start padding tokens. Without the start padding tokens, the target sentence after pre-processing would look as follows:

Source:	`I did not go to school today .` $\varepsilon\ \varepsilon\ \varepsilon$ `EOS`
Target:	`Ich` $\varepsilon\ \varepsilon$ `ging` $\varepsilon\ \varepsilon$ `heute nicht zur Schule . EOS`

Similarly, with four start padding tokens, the target sentence would not contain any additional ε tokens.

The alignments are computed on the original training data, and the number of ε tokens that are added to the target sentences is minimal in the sense that we only add the least number of ε tokens to satisfy the eager feasibility property. If start padding tokens are added to a target sentence, tokens in the target sentence are moved by the number of start padding tokens that are added. This moves the words in the target sentence and thereby also the alignments, which reduces the number of ε tokens required.

3.3 Model Architecture and Training

For training, all source and target sentences are concatenated into a single source and target string. This allows the authors to train the model on a single sequence, akin to how language models are trained, and initialize the hidden state of the model in a given batch with the last hidden state of the model from the previous batch. Eager feasible pre-processing ensures that a given source and target sentence pair have the same length. Thus, the source and target string also have the same length and sentence boundaries occur at the same indexes. No alignments or indexes relative to the corpus are changed.

[Press and Smith, 2018] define a backpropagation through time (`BPTT`) and batch size parameter such that each element in every batch contains `BPTT` source tokens and their respective target tokens.

At each time step, the eager translation model receives the current source input words and the previously selected output words. These inputs are embedded into a dense vector of dimension E each, with E being a configurable parameter. The two embedding vectors are then concatenated into a single vector of dimension $2E$ and fed into a multi-layer LSTM [Hochreiter and Schmidhuber, 1997; Gers et al., 1999] with $2E$ hidden units and an output size of $2E$.

The output of the LSTM is transformed into a vector of dimension E using a fully

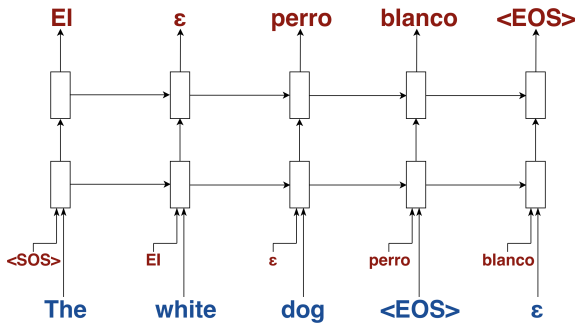


Figure 5: Eager translation model with two LSTM layers translating the English sentence "The white dog" into the Spanish sentence "El perro blanco". Figure taken from [Press and Smith, 2018]. Notice the additional input in the first layer. Next to the source words and the previous hidden state, [Press and Smith, 2018] feed the previous selection into the LSTM.

connected layer. This output is then transformed into a distribution over the target vocabulary using an output embedding matrix and the softmax function.

[Press and Smith, 2018] tie the input embedding matrix of the source language, the input embedding matrix of the target language, and the output embedding matrix [Press and Wolf, 2016; Inan et al., 2016]. They furthermore use teacher forcing [Williams and Zipser, 1989] and the cross-entropy loss function.

The ε token and the start padding token are tokens in the target vocabulary. They are not specially treated in any form or fashion. [Press and Smith, 2018] did not change the training objective or the loss function for the prediction of these tokens.

3.4 Inference

The eager translation model returns precisely one translation token for every input word it receives. Yet, [Press and Smith, 2018] use a modified beam search algorithm to improve the quality of the translations and return the translated sentence only once the entire input sentence has been processed. The modified beam search algorithm can be configured with four different parameters:

- **Beam Size:** The number of hypotheses to hold on to during the translation of a source sentence.
- **Start Padding Tokens:** The number of start padding tokens that will be returned before the first translation. This is supposed to make the translation task simpler for the eager model as it allows the model to consume a greater

number of tokens and to consider a greater amount of evidence before returning the first translation. This parameter can be understood as the initial *lag* at which the model translates the input sentence.

The start padding tokens are also added to the training data during preprocessing. Although this is a parameter that can be configured for inference, it is not meaningful to use a number of start padding tokens during inference that differs from the number of start padding tokens during training.

- **Padding Limit:** The maximum number of ε tokens that the eager model is allowed to return in a single sentence. This is an upper limit. The model may return any number of ε tokens between 0 and *padding limit*. Here and in the following, we also refer to this padding limit as *epsilon limit*.
- **Source Padding Injection (SPI):** The number of ε tokens that are added to the source sentence before the `EOS` token. Without this parameter, the length of a translated sentence is bound by the length of the respective input sentence because the model returns exactly one word for every word it receives. Adding additional tokens to source sentences artificially increases their lengths and thereby allows the eager model to output sentences that are longer than their respective inputs. [Press and Smith, 2018] report that during their experiments, translation quality improved when they inserted ε tokens into the source sentence. This is because the model assigns a high probability to the `EOS` token once it reads the `EOS` token from the input sentence, which would then terminate the translation process.

The authors [Press and Smith, 2018] report that the beam search process will consider any number of padding tokens between 0 and c, with c being the value that the SPI parameter is set to, *before* the source `EOS` token. In our analysis, we have noticed that this is not the case. During inference, once the end of the source sentence is reached and the source `EOS` token is read, the `EOS` token is replaced with an ε token **and** an `EOS` token, i.e., an array containing $[\varepsilon, \texttt{EOS}]$, resulting in two inputs being processed successively.

We illustrate the mechanism with an example. Suppose we are translating the following source sentence with the eager translation model:

`The white dog . EOS`

Suppose that for the translation we use an SPI value of 3. This is equivalent to translating the following source sentence:

`The white dog .` ε `EOS` ε `EOS` ε `EOS`

This is surprising and non-standard. A translation model is not supposed to receive multiple source `EOS` tokens for a single translation. We will see in Section 4 that this use of `EOS` tokens creates unexpected and complicated behavior during inference as well as how we can modify the beam search algorithm introduced by [Press and Smith, 2018] to remove this behavior.

Before the final translation is returned, all start padding tokens and ε tokens that were returned by the model are removed from the translation sentence.

4 Analysis of the Eager Translation Model

We have reproduced the eager translation model [Press and Smith, 2018] using the source code that the authors published and have trained our own eager translation model according to the specifications and instructions that [Press and Smith, 2018] outline in their work.

In this chapter, we compare our results to the results of the authors and establish an elaborate overview of the dynamics of eager translation. We quantitatively analyze the hyperparameters epsilon limit and SPI in great detail and show how they influence the translations that the model produces. We further show how appropriate values for these hyperparameters can be selected and show that excessive tuning of these hyperparameters does not guarantee performance improvements over an entire translation corpus.

Finally, in this chapter, we propose changes to the beam search algorithm that [Press and Smith, 2018] introduce and elaborate on the differences.

4.1 Setup

We have trained our own eager translation model using the source code that [Press and Smith, 2018] published[1]. We followed their specifications closely and have trained our model with the same configuration on the WMT 2014[2] `EN` $\leftrightarrow$ `DE` data set. We used *sockeye-autopilot*[3] to download the data set. The data set is segmented into a byte pair encoding (BPE) [Sennrich et al., 2015]. We held out $3,000$ sentence pairs from the training set as the validation set for early stopping when training the model. For testing, we use newstest2013.

[1] `https://github.com/ofirpress/YouMayNotNeedAttention`

[2] `http://www.statmt.org/wmt14/translation-task.html`

[3] `https://github.com/awslabs/sockeye/tree/master/sockeye_contrib/autopilot`

Beam Size	Start Pads	Epsilon Limit	SPI	BLEU (↑)
25	0	5	7	11.50
25	1	5	7	15.81
25	2	5	9	16.53
25	3	4	9	17.36
25	4	2	9	17.52
5	5	1	13	17.97
Ref. Model				18.89

Table 1: `BLEU` scores for different configurations from [Press and Smith, 2018]

We use *fast_align*[4] [Dyer et al., 2013] to compute word alignments on the training set and use the source code provided by [Press and Smith, 2018] to make the training data eager feasible. During this step, we add four start padding tokens to the target sentences.

[Press and Smith, 2018] trained the OpenNMT [Klein et al., 2017] implementation of [Bahdanau et al., 2014] and use it as their reference model. For reference and completeness, we have trained the same OpenNMT model and will compare our eager translation model to our OpenNMT model.

We furthermore follow the approach of [Press and Smith, 2018] for computing `BLEU` scores and use SacreBLEU [Post, 2018] to compute `BLEU` scores on the detokenized outputs of the models.

4.2 Baseline

We consider the configuration that [Press and Smith, 2018] outline in their instructions[5] as the default configuration and start experimenting from there. The default configuration uses beam size 5, epsilon limit 3, SPI 4, and 4 start padding tokens[6]. We report a `BLEU` score of 18.46 with the eager translation model in this configuration on WMT 2014 `EN` ↔ `DE`. This score and all other `BLEU` scores from this chapter are also shown in Table 2.

[4] `https://github.com/clab/fast_align`

[5] `https://github.com/ofirpress/YouMayNotNeedAttention`

[6] In fact, in the instructions, the authors add 5 start padding tokens during inference. They use 4 start padding tokens in the pre-processing step. We also used 4 start padding tokens in the pre-processing step. It is not meaningful to use a different number of start padding tokens for training and testing. Hence, we use 4 start padding tokens for both.

Model	Beam Size	Start Pads	Epsilon Limit	SPI	BLEU (↑)
Eager	5	4	3	4	18.46
Eager	25	4	2	9	19.18
Eager	5	4	3	9	19.28
Eager, Modified Beam Search	5	4	3	4	19.16
Eager, Modified Beam Search	5	4	3	5	19.24
OpenNMT Ref. Model					19.06
Oracle	5	4	-	-	18.40
Oracle, Modified Beam Search	5	4	-	-	18.07

Table 2: `BLEU` scores of our models.

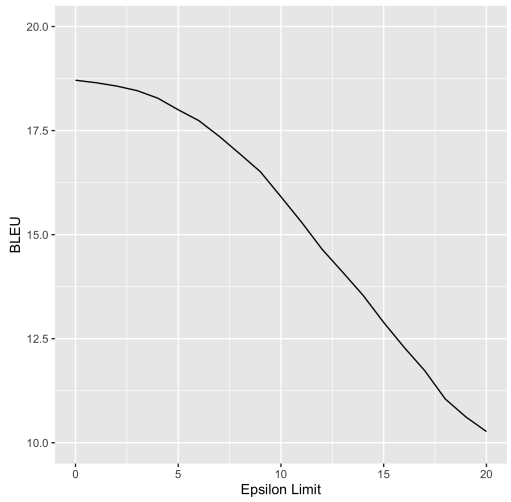
The authors report different `BLEU` scores for different language pairs and different configurations. Table 1 shows the `BLEU` scores that [Press and Smith, 2018] report for the language pair `EN` ↔ `DE`. The authors report a `BLEU` score of 17.97 as their highest score on `EN` ↔ `DE` with the eager model and a `BLEU` score of 18.89 with the OpenNMT reference model. We report a `BLEU` score of 19.06 with the OpenNMT reference model. Similar to the results of [Press and Smith, 2018], in the default configuration, our reference OpenNMT model outperforms our eager model by a small margin.

We did not reproduce the results from Table 1 because using a different number of start padding tokens requires to train a new model with the same number of start padding tokens in the training data. We did, however, reproduce the result for the configuration with 4 start padding tokens in Table 1. In this configuration, our eager translation model outperforms our reference model by a small margin and we report a `BLEU` score of 19.18.

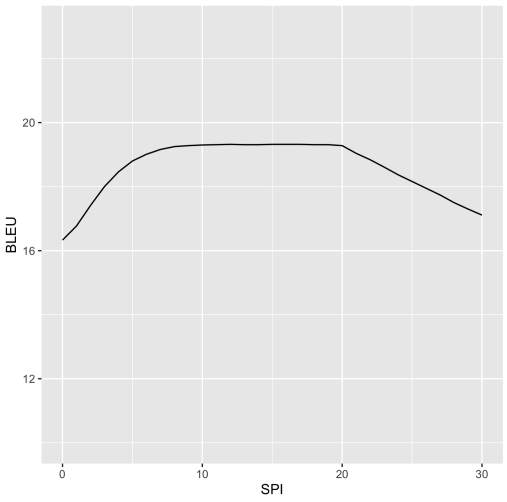
4.3 Influence of Epsilon Limit on BLEU

Naturally, we are interested in how the performance of the eager translation model changes when we deviate from the default configuration. Also, it is not entirely clear why an epsilon limit is needed at all. Apparently, the eager translation model must be limited in the total number of ε tokens that it produces. However, if the model is capable of learning when to produce ε tokens, such a limit should not be necessary.

We do not change the number of start padding tokens but will see how the model performs when we change one of the other parameters, one by one, all other things held constant.



(a) BLEU scores for the eager translation model with different epsilon limit values. Other parameters are beam size 5, 4 start padding tokens and SPI 4. Translation quality decreases rapidly with an increase in the epsilon limit. The highest BLEU score is achieved with an epsilon limit value of 0.



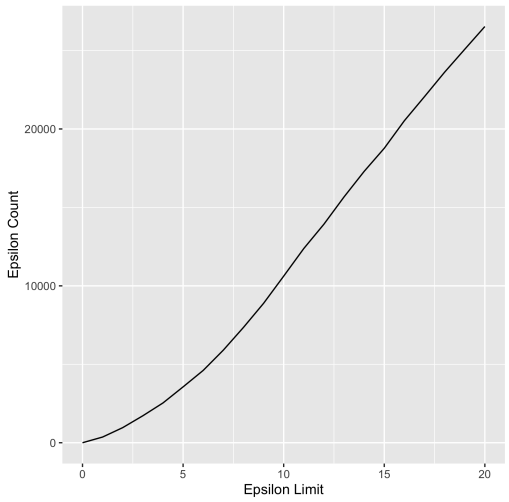
(b) BLEU scores for the eager translation model with different SPI values. Other parameters are beam size 5, epsilon limit 3 and 4 start padding tokens. Translation quality improves with diminishing rates and decreases rapidly beyond an SPI value of 20. Translation quality remains almost constant between SPI values 9 and 20.

Figure 6: BLEU scores for eager translation model with different epsilon limit and SPI values.

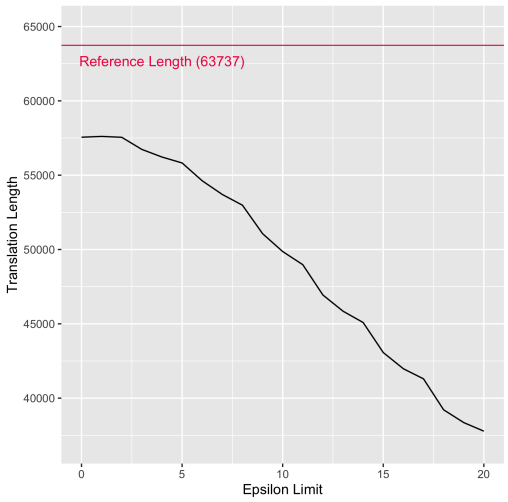
Figure 6a shows the BLEU scores of our eager translation model for various epsilon limit values. The translation quality decreases significantly with an increase in the number of ε tokens that the model is allowed to produce. The best scores are achieved when the model is prohibited from returning ε tokens entirely.

We assume that this is a consequence of the model producing a greater number of ε tokens with a greater epsilon limit. This seems unexpected. An increase in the epsilon limit does not *force* the model to produce more ε tokens but only *enables* it to do so. Ideally, the model does not need to be limited in the number of ε tokens that it may return. Figure 7a shows that this is not the case. The eager translation model consistently produces more ε tokens when the epsilon limit is increased. The relationship between the epsilon limit and the number of ε tokens that the model returns are almost linear.

Producing more ε tokens also means not producing other tokens. The number of tokens that the eager translation model returns is limited by the number of source tokens it consumes. It cannot produce more tokens than it consumes, and for every



(a) Number of ε tokens returned by the eager translation model for different values of epsilon limit. The number of ε tokens that the model returns increases with the epsilon limit.



(b) Length of the final translations of the eager translation model for different values of epsilon limit. The length decreases almost linearly with an increase in the epsilon limit.

Figure 7: Number of epsilon tokens returned and total length of the translations of the eager translation model for various values of epsilon limits.

ε token it returns, it returns one fewer non-ε token. Since ε tokens are removed from the output once the translation is completed, the number of tokens that end up in the final translation is reduced by one for every ε token that the model produces. Consequently, the final translations become shorter if the model returns more ε tokens and more sentences are cut off and incomplete. This drastically reduces the quality of the translations, and hence, the `BLEU` scores decrease with an increase in the epsilon limit.

4.4 Influence of SPI on BLEU

Figure 6b shows the `BLEU` scores of our eager model for different SPI values. The translation quality increases gradually with an increase in SPI. The rate at which the translation quality improves diminishes quickly, and there is almost no improvement between the values 10 and 20. There appears to be an optimal SPI value around 20 and the translation quality decreases rapidly beyond that optimal value.

With SPI, [Press and Smith, 2018] artificially increase the number of tokens that the model returns by increasing the length of the sentence that is translated with

padding tokens. Intuitively, it seems comprehensible that the translation quality of the eager translation model improves when we allow the model to produce more tokens than there are in the source sentence. This is because we add padding tokens to the beginning of the translation sentence and allow the model to return ε tokens in order to wait for additional information. Both of these decrease the number of translation tokens that are returned by the model, and thus, padding the source sentence appears to be a reasonable approach to counteract this deficiency.

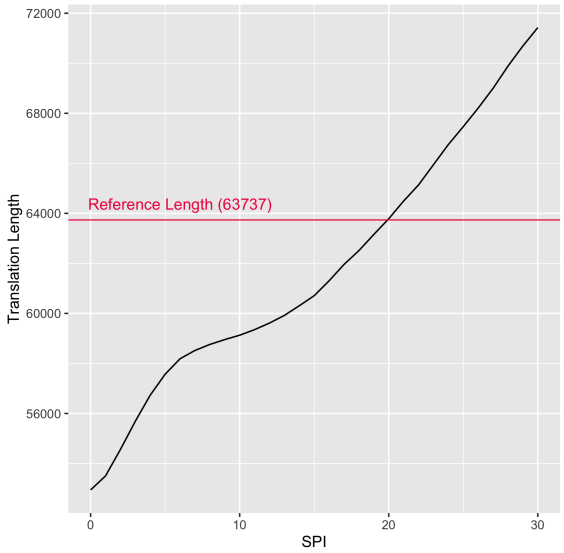


Figure 8: Lengths of the translations returned by the eager translation model relative to the SPI value set during inference. SPI increases the number of tokens that are consumed by the model and hence increases the number of tokens that the model returns.

We see a very similar picture regarding the influence of SPI on the lengths of the translations that are returned, as we have seen for the epsilon limit. Figure 8 shows the relationship between SPI and the lengths of the translations that are returned by the eager model. The lengths of the translations increase with SPI, the rate at which the translation lengths increase decreases between the values 6 and 13 and then grows again.

We assume that the non-linearity in the relationship between SPI and the translation lengths is a result of the way the ε tokens are added to the source sentences during inference. Rather than adding all ε tokens before the source `EOS` token, [Press and Smith, 2018] add an ε token to the source `EOS` token and process both tokens successively. This occurs c times in a configuration where SPI is set to c.

As a consequence of this, the eager translation model produces a much larger sample of translation candidates with a much greater variety in sentence lengths. The eager translation model assigns a high probability to the `EOS` token once it reads the source

EOS token. During beam search, hypotheses that already include an EOS token are not considered for further expansion. Repeatedly committing an ε token *and* the source EOS token allows the decoder to expand the hypotheses that terminated in previous iterations with an EOS token while retaining a high probability to terminate the translation by producing an EOS token *after* the expansion. Once all source tokens were processed, including the last ε and EOS token pair, the beam search algorithm considers every hypothesis that was produced, which also includes an EOS token. As a result, the eager model may produce a new candidate translation sentence during each iteration of this step. These candidate translations grow in length with every iteration such that each sentence will have k additional tokens where $0 \leq k \leq SPI$ and the eager model will have a larger set of candidate sentences with varying lengths to select from.

The model benefits from the increase in the length of the translations until the translations, on average, become too long, such that the number of cases in which the translations extend over sentence boundaries is larger than the number of cases in which the increase in SPI helps to prevent sentences from being cut off.

4.5 Finding a Good Value for Epsilon Limit

In Sections 4.3 and 4.4, we have seen that the epsilon limit and SPI parameters can be considered proxy variables that allow us to control the lengths of the translations that are produced by the eager model. Selecting non-optimal values for the two parameters may result in translations being too short or too long, i.e., translations being cut off or extending over sentence boundaries. The model is particularly sensitive to changes in the epsilon limit and the translation quality may suffer greatly if the model is allowed to return too many ε tokens.

A major difficulty in selecting appropriate values for the two parameters is that these parameters are static values that are used for every sentence that the model translates, whereas in reality, the ideal values, very probably, differ between sentences. The requirement to select specific values for the epsilon limit and SPI parameters constitutes an inherent weakness of the eager translation model.

We argue that the eager model is likely to perform well when given a configuration of parameters that reflects the configuration and data that the model experienced during training. That is, we assume that a reasonable selection for the values of the epsilon limit and SPI are close to statistics that prevail across the entire training set.

For the epsilon limit, we expect that a suitable value will be close to the number of ε tokens that are present in the target sentences in the training data. That is, the epsilon limit is the number of epsilon tokens that the eager model should return, on average, during inference and the number of ε tokens that are present in the target sentences in the training data, on average, is the number of ε tokens that the model is taught to return.

Min.	1st Qu.	Median	Mean	Mode	Variance	Std. Dev.	3rd Qu.	Max
0	0	1	5	0	334	18	6	13113

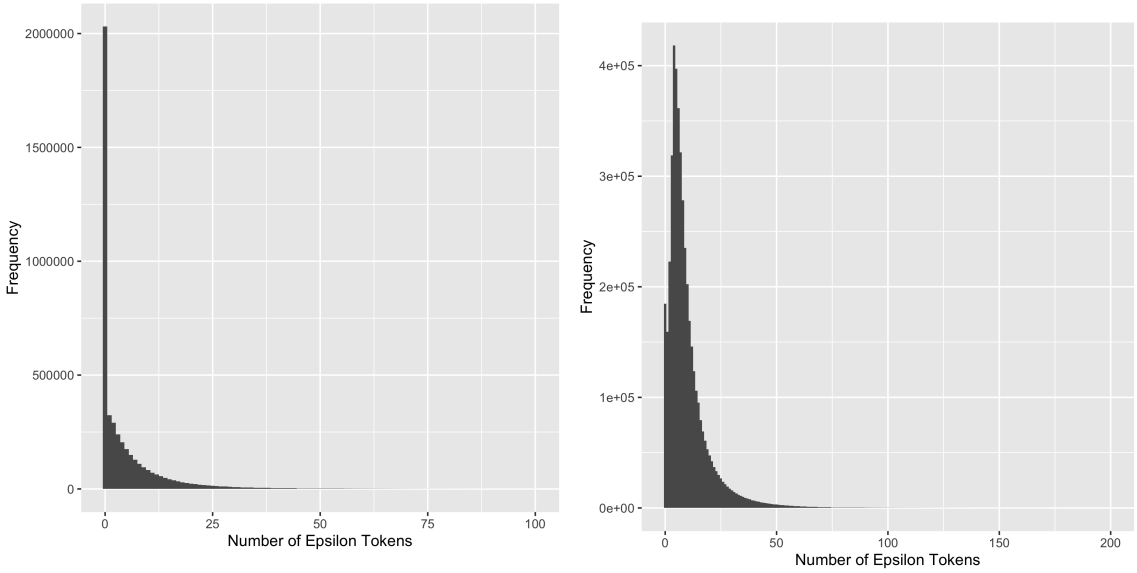
Table 3: Descriptive statistics over ε tokens in the target sentences in the training data.

Table 3 shows the measures of central tendency of the distribution of ε tokens in the target sentences in the training data. Surprisingly, the majority of all target sentences in the training data do not contain any ε token. The median number of ε tokens equals 1 and on average, a target sentence contains 5 ε tokens. The variance and standard deviation are comparably large. This indicates that the data set likely contains a few sentences with extremely large quantities of ε tokens that highly skew the distribution. Figure 9a shows the distribution of the number of ε tokens in the target sentences in the training data for all sentences with fewer than 100 ε tokens. This constitutes 99.9% of all sentences in the training data.

A closer look at the quantiles of the distribution shows that with 45%, an overwhelming majority of the target sentences do not contain any ε tokens.

Given that roughly half of all target sentences do not contain any ε tokens, it seems comprehensible that the model performs comparably well if we prohibit it from returning ε tokens entirely. That is, if 45% of all target sentences do not contain any ε tokens, the model is correct in 45% of all cases if it does not return any ε tokens and the likelihood of the model estimating the correct number of ε tokens between 0 and some arbitrary epsilon limit k appears to be significantly lower, especially if the model consistently exhausts its ε token contingent.

Nevertheless, an epsilon limit value of 0 arguably defeats the purpose of the model, and therefore, we intend to use an epsilon limit greater than that. Also, a substantial portion of the target sentences in the training data contains a low number of ε tokens between 1 and 5. Although the model has a strong tendency to produce more ε tokens when given the possibility, we expect that a reasonable performance can be achieved with a low but non-zero epsilon limit value.



(a) ε token distribution in target sentences in training data. Distribution follows a power-law. Almost half of all target sentences do not contain any ε tokens

(b) ε token distribution in source sentences in training data. Distribution follows a power-law. Most source sentences contain four ε tokens.

Figure 9: Distributions of ε tokens in the training data.

4.6 Finding a Good Value for SPI

Similarly, we are interested in finding a good value for the SPI parameter. We expect that a suitable value might be close to the number of ε tokens that are present in the source sentences in the training data. After all, some sentences need to be padded to ensure eager feasibility, and source padding injection is very similar in that it adds padding tokens to the source sentences to match the length of the target sentence better.

Min.	1st Qu.	Median	Mean	Mode	Variance	Std. Dev.	3rd Qu.	Max
0	4	7	10	4	224	15	12	8834

Table 4: Descriptive statistics over ε tokens in the source sentences in the training data.

Table 4 shows the measures of central tendency of the distribution of ε tokens in the source sentences in the training data. Most sentences contain 4 ε tokens, and on average, a source sentence contains 10 ε tokens. Given that target sentences are padded with 4 start padding tokens, it seems comprehensible that most sentences will contain somewhere around 4 fewer tokens and hence need to be padded with 4

ε tokens.

The variance and standard deviation are comparably large. This suggests that there might be a few sentences with very large quantities of ε tokens that skew the mean and median values.

Figure 9b shows the distribution of ε tokens in the source sentences in the training data. The distribution follows a power-law. Most sentences only have a few ε tokens, and only very few sentences contain a large number of ε tokens.

The distribution of ε tokens in the source sentences in the training data is less distinct than the distribution of ε tokens in the target sentences. Yet, we see a similar picture as we have seen for the epsilon limit. Most of the source sentences only contain very few ε tokens. More than half of all sentences contain 7 or fewer ε tokens.

Intuitively, we would expect an optimal SPI value to be close to the number of tokens that are removed from translation sentences due to start padding tokens and ε tokens that are produced. Given a start padding token parameter value of 4 and an epsilon limit of 3, an SPI value of 7 appears reasonable. We have also noticed that, in general, target sentences are, on average, slightly longer than source sentences. Also, since the beam search dynamics allow the model to produce translation sentences shorter than the length of the source sentence plus SPI, the model does not respond sensitively to a small increase in SPI. It, therefore, appears that a value slightly greater than the sum of the epsilon limit and the number of start padding tokens should be an appropriate selection for the value of the SPI parameter.

We conclude that optimal epsilon limit and SPI parameter values do not definitively reveal themselves even over a large set of translations. However, we achieve reasonable improvements in the quality of our translations, compared to the default configuration, when we select parameter values near to measures of central tendency of the distributions of ε tokens in the training data. We believe that a reasonable configuration for further analysis includes 4 start padding tokens, an SPI value of 9, an epsilon limit of 3, and a beam size of 5. We report a `BLEU` score of 19.28 for the eager translation model with this configuration.

4.7 Oracle Experiment

Translations produced by the eager translation model are prone to be too short or too long. To some extent, we can control the lengths of the translations and

generate shorter or longer sentences with a greater epsilon limit or greater SPI value, respectively.

In Sections 4.3 and 4.4, we investigated how these parameters change the translations that the eager model produces and have found a reasonable set of parameters in Sections 4.5 and 4.6. Although this set of parameters might perform well, or even best, on a given corpus, the selection of parameter values might still not be optimal for a large subset of the sentences in the corpus. When we change these parameters and measure the differences in the translation quality over all translations using `BLEU`, we only select the values for these parameters such that they fit best *on average*. That is, when we increase SPI by one and the translation quality improves, it might just be that the subset of all sentences in the translations that benefit from the increase is larger than the subset of all sentences that suffer from the increase such that we experience a net benefit over the entire set of translations.

Finding the best combination of these hyperparameters involves producing a great number of translations with a great number of combinations of different values for these hyperparameters. Once a good configuration was found for one data set, it might not be the best configuration for another data set, or even all other data sets. Therefore, we investigate the performance of the eager translation model under optimal circumstances. That is, we investigate the quality of the translations produced by the model when given the exact number of ε tokens and source padding tokens for every sentence it consumes and produces.

Using an oracle configuration, where the model is supplied with optimal hyperparameter values for every sentence in the data set, will tell us whether using non-optimal hyperparameters has a significant impact on the quality of the translations of the eager model.

We have modified the source code such that during inference, before starting the translation, we retrieve the corresponding target sentence from the translation corpus and compute the ideal values for epsilon limit and SPI for the given sentence pair. Particularly, we count the number of tokens in both sentences and compute the difference. This difference in sentence lengths is the value we assign to SPI. We then count the number of ε tokens in the target sentence and use this value as the epsilon limit, i.e., we allow the model to produce as many ε tokens as there are in the reference translation. These two values are computed for every sentence pair. With this, the model is configured with the optimal values for every sentence pair and, presumably, well-equipped for producing high-quality translations.

We report a `BLEU` score of 18.4 for the eager model with perfect parameters and the

original beam search algorithm and report a BLEU score of 18.07 with the modifications that we added to the beam search algorithm.

This seems unexpected. When we set the hyperparameters epsilon limit and SPI, we select static values that are used throughout all translations. We select these values such that they fit best on average, by measuring the translation quality over all translations. These values, however, are likely wrong for a substantial subset of the translations, and we would expect that a more careful and dynamic selection of these hyperparameters would improve the translations that the eager model produces.

Yet, this is not the case. We are unable to improve the translation quality using dynamically set values for epsilon limit and SPI. On the contrary, translation quality drops significantly in the oracle experiment. We conclude that the use of static global values is not a key reason why the eager translation model might underperform the OpenNMT reference model. Also, we do not expect large improvements in translation quality with a better configuration and with further tuning of these hyperparameters.

4.8 Modifying the Modified Beam Search

[Press and Smith, 2018] use a modified beam search algorithm to improve the quality of the translations of the eager model. During inference, the source sentence that is translated is padded with ε tokens *and* source EOS tokens to increase the number of tokens that the eager model can produce. We argue that the way the source sentences are padded, particularly that the decoder receives multiple EOS tokens, is not common practice and leads to strange behavior during beam search.

We have modified the beam search algorithm that [Press and Smith, 2018] applied such that source sentences are still padded but will only ever receive a single source EOS token. With our modifications, padding a source sentence with c ε tokens is equivalent to inserting c ε tokens just before the EOS token.

Suppose we are translating the following source sentence with the eager translation model:

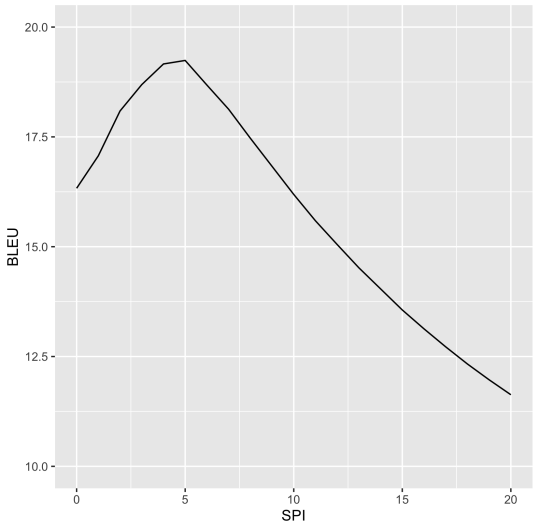
```
The white dog .  EOS
```

Suppose that for the translation, we use an SPI value of 3. With our modifications, this is equivalent to translating the following source sentence:

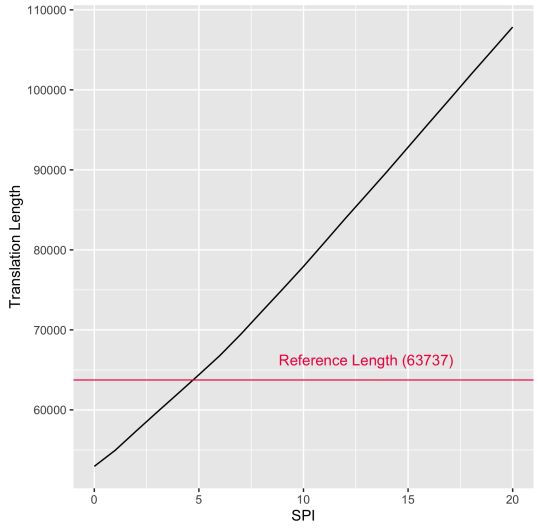
```
The white dog .  ε ε ε EOS
```

Figure 10a shows the BLEU scores of the eager translation model for various SPI values after our modification. As we have seen before, the translation quality of the model improves, if the source sentence is padded with ε tokens during inference. However, the curve is now much steeper, and the translation quality deteriorates rapidly beyond a much lower optimal value for SPI. The optimal value, which is now around 4 or 5, is also closer to the number of start padding tokens and thus seems more comprehensible.

Figure 10b shows the translation lengths for various values of SPI. Compared to the original beam search algorithm, the translation lengths grow more consistently and more quickly than before. We also noticed that the total translation length grows by approximately the number of sentences in the data set when we increase SPI by one. That is, an increase in the value of SPI results in approximately the same increase, in absolute terms, in the number of tokens per sentence in the translations. There is now a one-to-one relationship between the number of source padding tokens and the number of tokens that the model returns.



(a) BLEU scores for the eager translation model with different SPI values with our modifications to the beam search algorithm.



(b) Translation lengths for the eager translation model with different SPI values with our modifications to the beam search algorithm.

Figure 10: The eager translation model responds much more sensitively to changes in SPI with our modifications to the beam search algorithm.

With this configuration and our modifications, the eager translation model performs best with 5 and slightly worse with 4 source padding tokens. We report BLEU scores of 19.16 and 19.24 for SPI values of 4 and 5 respectively and an epsilon limit of 3, beam size 5, and 4 start padding tokens.

Our modifications appear reasonable, and this might be how [Press and Smith, 2018]

may have initially envisioned and planned to modify their beam search algorithm. That is to say, it does fit their description much more closely.

5 Qualitative Analysis of Translations

The eager translation model is trained on a corpus where there are only monotonous alignments. All non-monotonous alignments are removed during eager feasible preprocessing and resolved by adding ε tokens to the target sentences. The ε tokens are placed where the model requires additional information to translate eagerly. In essence, the ε token is a `WAIT` token that allows the model to wait for essential information from the source sentence. Ideally, by training on such a corpus, the model learns when to wait for more information from the source sentence and is capable of translating eagerly.

In Chapter 4, we have analyzed the translations that the eager translation model produces and have shown that allowing the model to wait, i.e., using a positive epsilon limit, influences the quality of the translations that the model produces. However, these automated analyses only give us a good understanding of the translations *in aggregate* and do not reveal, on a sentence-level, if the eager model produces *reasonable* translations. The most crucial question, whether or not the eager translation model is capable of waiting for relevant information from the source sentence, has not been answered so far.

No metric to assess the quality of a translation system beats a manual analysis performed by a human. In this chapter, we take a closer look at the translations that the eager translation model produces and manually analyze the waiting behavior of the eager model. We show an example where the eager translation model produces unreasonable ε tokens and also show examples where the eager translation model exhibits good waiting behavior and uses ε tokens reasonably.

5.1 Setup

We translated newstest2013 `EN` $\leftrightarrow$ `DE`, that we downloaded using sockeye-autopilot[1], with the eager translation model using 4 start padding tokens, an epsilon limit of

[1] `https://github.com/awslabs/sockeye/tree/master/sockeye_contrib/autopilot`

3, an SPI of 9 and a beam size of 5. This is the configuration that we selected in Chapter 4.

The data set contains 3′000 sentences. Of all translations produced by the eager model, 724 sentences contain at least one ε token. We randomly selected 60 sentences and assessed the following criteria:

- **Are the ε tokens in the translations placed reasonably?** We consider the use of ε tokens reasonable if it is evident from the translation that the model was waiting for additional information. Since we translate from English to German and English and German have different word orders, i.e., verbs that typically appear late in English sentences commonly appear early in their German translations, we assume that these will be cases in which the eager translation model might make good use of ε tokens.
- **Did the model use all ε tokens?** If the model consistently exhausts the epsilon limit, particularly if the ε tokens are not produced reasonably, this reduces the number of tokens in the final translation. If translations are consistently too short, this might be one of the reasons why.
- **Could the model have used fewer ε tokens?** It might be possible that the eager translation model simply produces ε tokens early during the translation to read more tokens as a form of exploit to collect more information, without resolving any alignments between the source sentence and the translation it produces.
- **Could the model have required more ε tokens?** Similarly, the model might require more ε tokens than it is allowed to produce and might have started producing ε tokens at a reasonable point during the translation but was then restricted by the epsilon limit.
- **Is the translation too long?** If the translation exceeds the sentence boundary and produces too many tokens.
- **Is the translation too short?** If the translation is cut off.
- **Is there only one block of epsilon tokens?** We do not know beforehand how ε tokens are produced most reasonably. We anticipate that reasonably placed ε tokens are spread throughout the entire translation and that an excess of blocks of ε tokens might indicate the eager model trying to exploit the use of ε tokens to obtain more information, even when that information is not crucial at the current translation step.

- **Did the model simplify / shorten translation tokens?** The model may shorten the translation compared to the source sentence by translating tokens into shorter equivalents. For instance, the translation model might consistently translate "United States of America" into "Amerika", which shortens the translation. If translations are consistently too short, this might be one of the reasons why.
- **Did the model add translation tokens?** Similarly, the model might lengthen a translation compared to the source sentence by translating into longer equivalents. For instance, the same example might go the other way, and the model might consistently translate "America" into "Vereinigte Staaten von Amerika", which increases the number of tokens it needs to produce. If translations are consistently too long, this might be one of the reasons why.

To perform this manual evaluation, we built our own application. The application uses a NodeJS[2] server and a ReactJS[3] frontend that runs in a browser. We used the MeteorJS framework[4] for better prototyping and stored all translations in a MongoDB[5] database. Figure 11 shows a screenshot of our interface.

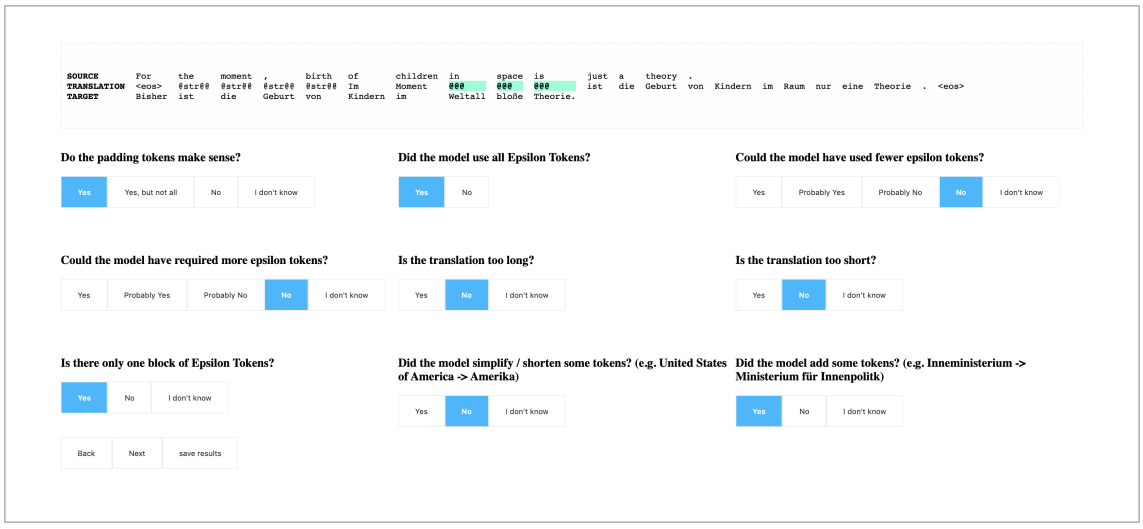


Figure 11: Interface of our evaluation application. The top box contains the source sentence, the translation, and the target sentence. Below the box there are evaluation questions with buttons that immediately store the selected answer in the database.

[2] https://nodejs.org/

[3] https://reactjs.org/

[4] https://www.meteor.com/

[5] https://www.mongodb.com/

Were ε tokens placed reasonably?			
Yes	No	Don't know	Yes, but not for all ε tokens
45	9	2	4

Table 5: Summary of the manual evaluation. The eager translation model makes good use of ε tokens in most translations.

Translation Length		
Too Short	Too Long	Total
24	0	60

Table 6: Number of translations from the manual evaluation that were either too short or too long. Almost half of all translations are too short.

5.2 Results

We summarize our findings in Table 5. In most cases that we evaluated manually, the model used ε tokens to resolve non-monotonous alignments between the source sentence and the translation. ε tokens were consistently placed reasonably in the sense that the model returned a translation of the source word that followed after predicting the last ε token.

Whether the use of wait tokens is reasonable is, of course, entangled with translation quality in general. In some cases, the model starts producing a poor translation, mixes up word orders early in the sentence, and does not produce translations for some source tokens. Due to this, it then lacks "progress", falls behind, and requires unnecessary ε tokens to catch up again. Figure 12a illustrates this with an example. The first sentence is the source sentence and the second sentence is the translation produced by the eager translation model. The first 4 tokens returned in the translation are start padding tokens.

The sentences where this occurs are typically longer and more complex. There are also cases where the model produces ε tokens immediately at the beginning of the translation with no apparent reason. In other cases, the model produces ε tokens at the end of the translation, even when consuming source padding tokens, i.e., even when there is no more relevant information to obtain from the source sentence. This does not occur often in the translations that we manually evaluated.

The eager translation model predominantly uses ε tokens reasonably. We particularly notice that the model consistently uses ε tokens to resolve inherent differences in word orders between the source and target language. We translate from English

to German. Verbs that appear late in an English sentence commonly appear early in its German translation. We noticed that the model consistently uses ε tokens to resolve this type of misalignment.

SOURCE	Which	is	what	makes	a	player	like	me	want	to	face	up	and	give	my	best	.
TRANSLATION	@str@@	@str@@	@str@@	@str@@	Was	macht	ein	Spieler	wie	ich	möchte	@@@	@@@	@@@	meinen	besten	Eindruck	<eos>
TARGET	Dies	führt	dazu,	dass	ein	Spieler	wie	ich,	die	Stirn	bieten	muss	und	sein	Bestes	geben	will.	

(a) The model starts producing an incorrect translation from the very beginning and then uses ε tokens to catch up again.

SOURCE	On	the	other	hand	,	the	administration	does	not	often	ease	their	life	.					
TRANSLATION	@str@@	@str@@	@str@@	@str@@	Auf	der	anderen	Seite	@@@	@@@	erleichtert	die	Administration	ihr	Leben	oft	nicht	.	<eos>
TARGET	Andererseits	macht	die	Unternehmensführung	es	ihnen	nicht	einfach.											

(b) The model produced ε tokens to wait for "ease" from the source sentence. This resolves the non-monotonous alignment between the verbs from the source and target sentence.

SOURCE	In	the	meantime	,	most	of	the	m@@	igrant	workers	continue	to	live	in	Russia	illegally	.	
TRANSLATION	@str@@	@str@@	@str@@	@str@@	In	der	Zwischenzeit	@@@	@@@	@@@	gehen	die	meisten	M@@	igran@@	t@@	innen	in	Russland	<eos>
TARGET	Währenddessen	hält	sich	der	Großteil	der	Gastarbeiter	weiterhin	illegal	in	Russland	auf.								

(c) The model could have produced more ε tokens but was limited by the epsilon limit and could not wait as long as it needed to.

SOURCE	In	the	meantime	,	most	of	the	m@@	igrant	workers	continue	to	live	in	Russia	illegally	.		
TRANSLATION	@str@@	@str@@	@str@@	@str@@	In	der	Zwischenzeit	@@@	@@@	@@@	@@@	@@@	leben	die	meisten	M@@	igran@@	t@@	innen	<eos>
TARGET	Währenddessen	hält	sich	der	Großteil	der	Gastarbeiter	weiterhin	illegal	in	Russland	auf.								

(d) Shows the same translation as (c) but with a higher epsilon limit. The waits reasonably and minimally when given the chance.

Figure 12: Translations produced by the eager translation model. The first sentence is the source sentence, the second sentence is the translation produced by the model and the third sentence is the target sentence. The first four tokens in every translation ("@@@") are the start padding tokens. (a) Shows a bad use of ε tokens, (b) shows the ideal use, (c) shows a good waiting behavior but a lack of ε tokens due to a too low epsilon limit and (d) shows the same sentence as (c) but with a higher epsilon limit.

Figure 12b shows an example of a reasonable use of ε tokens. Notice that the verb "ease", which is the 11th token in the source sentence, appears much earlier in the translation. Notice also that the eager translation model waits no longer than it needs. That is, the eager translation model produces the *minimal* number of ε tokens in order to consume all relevant source words.

Table 6 shows the number of translations that are too short or too long. Almost half of all translations are too short. The eager translation model has a strong tendency to produce too few tokens and does so very consistently. We did not find a clear connection between the model producing too few tokens and the model lengthening translations. Neither could we find a clear connection between translation lengths

and whether the model exhausted the epsilon limit.

We further notice cases where the model starts producing ε tokens reasonably but is then restricted by the epsilon limit, cannot wait as long as it should and fails in resolving the non-monotonous alignment due to this restriction. Figure 12c shows an example where we believe that the eager translation model has started producing ε tokens reasonably, but was then restricted in producing the necessary number of ε tokens by the epsilon limit.

The model correctly starts waiting at the point where it needs the verb from the source sentence. The last word it is able to consume before producing something other than an ε token is the source word "continue", which it then translates to "gehen". At this point, a reasonable translation. However, the relevant information is still hidden in the following two source words.

We confirm our hypothesis by translating the same sentence with a higher epsilon limit and observe that the model indeed produces an additional two ε tokens to wait until it reads the source word "live". Figure 12d shows this translation. Notice that, again, the model produces the minimal number of ε tokens to resolve the alignment between the source token and the translation token.

Using a greater epsilon limit evidently improved the waiting behavior of the eager translation model. However, it also decreased the quality of the translation that it produced because by producing more ε tokens, the model inevitably also produces fewer non-ε tokens since it is limited in the number of tokens that it can produce. Because of this, the translation that the model produced is too short and cut off, which reduces the quality of the translation.

We confidently conclude that the model is able to learn and apply *some* of the waiting mechanics created by the use of ε tokens. The eager translation model predominantly uses ε tokens to *wait* for crucial information from the source sentence. Frequently, the number of ε tokens that the model produces is *minimal*, and the model does not wait any longer than it needs. However, waiting for information from the source sentence also always reduces the number of translation tokens that the model can return and that constitute the final translation.

Consequently, waiting for more information reduces the quality of the final translation. There is a direct trade-off between waiting for relevant information and translating the entire source sentence.

6 Knowledge Distillation

Sequence-level knowledge distillation was shown to eliminate the need for beam search [Kim and Rush, 2016] and has helped to improve the performance of non-autoregressive neural machine translation models [Gu et al., 2017]. In this chapter, we apply sequence-level knowledge distillation to the eager translation model and show if sequence-level knowledge distillation can be used to eliminate the need for beam search in eager translation [Press and Smith, 2018]. We further show how the distilled data sets differ from the original training data in terms of parallelism and monotonicity. Also, we analyze the confidence with which our student models produce translations and compare the determinism in the outputs of the student models with the determinism in the predictions of the original eager translation model.

6.1 Introduction

Translating simultaneously comes at the cost of being unable to revert past decisions. The eager translation model introduced by [Press and Smith, 2018] produces exactly one translation token for every source token that it consumes and returns the translated token immediately after consuming the source token. This makes it impossible for the eager translation model to consider subsequent tokens for the translation of the current source token. The eager translation model is trusted to anticipate future words [Ma et al., 2018].

However, anticipating future events may not always be possible. Hence, [Press and Smith, 2018] use beam search to counteract this shortcoming of the eager translation model. With beam search, the eager model holds on to multiple hypotheses during the translation process and subsequently selects the most likely translation among them.

Figure 13 shows the `BLEU` scores of our eager translation model for various beam sizes. We used an SPI value of 9, an epsilon limit of 3 and 4 start padding tokens for all translations shown in Figure 13. This is the configuration that we selected in

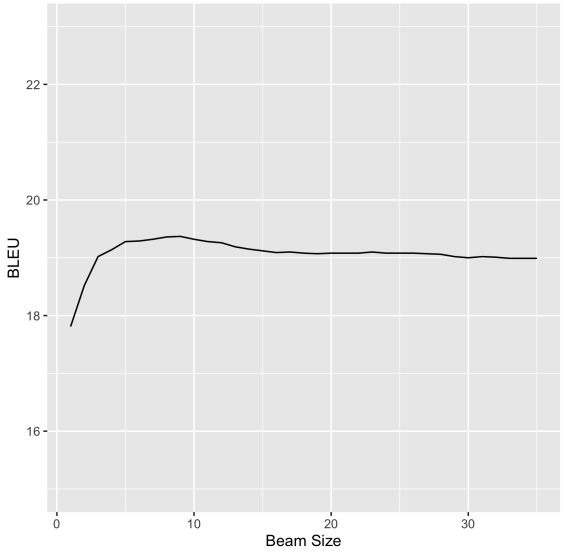


Figure 13: BLEU scores of the eager translation model for various beam sizes. Other parameters are SPI 9, epsilon limit 3 and 4 start padding tokens.

Beam Size	Start Pads	Epsilon Limit	SPI	BLEU (↑)
5	4	3	9	19.28
9	4	3	9	19.37
1	4	3	9	17.81

Table 7: BLEU scores of our models.

Section 4. We achieve the highest BLEU score with beam size 9. With this beam size, the eager translation model achieves a BLEU score of 19.37. With greedy decoding, i.e., a beam size of 1, the model performs substantially worse and achieves a BLEU score of only 17.81. We also show these scores in Table 7.

Evidently, the translation quality of the eager translation model increases significantly with beam search. In their work, [Press and Smith, 2018] report their best scores with beam sizes as large as 35. However, with beam search, the translation cannot be returned until the entire input sequence was processed. The model still returns a translation token immediately after consuming a word from the source sentences, but the final translation is only returned after the model has consumed and processed the last word from the source sentence. This effectively makes eager translation a non-simultaneous procedure and defeats the purpose of translating eagerly.

We hypothesize that we might be able to eliminate the need for beam search in the eager translation model by applying sequence-level knowledge distillation as proposed by [Kim and Rush, 2016]. More precisely, we investigate if we can elim-

inate the need for beam search in the eager translation model without sacrificing translation quality.

Knowledge distillation was initially proposed as a method to compress large or cumbersome models, or even ensembles of models, into smaller models that are easier to use [Buciluǎ et al., 2006]. [Kim and Rush, 2016] have shown that knowledge distillation can be applied to neural machine translation systems. Most surprisingly, [Kim and Rush, 2016] demonstrate that, after applying sequence-level knowledge distillation, their student models were able to decode greedily with only a negligible loss in translation quality.

6.2 Applying Sequence-Level Knowledge Distillation

We have translated the source sentences from our training data with the eager translation model that we have trained using an epsilon limit of 3, an SPI value of 9, a beam size of 5, and 4 start padding tokens. This is the configuration that we selected in Chapter 4. Next, we used these translations as the target sentences during the training of another eager translation model and trained the model with the same specifications and configuration as the first model.

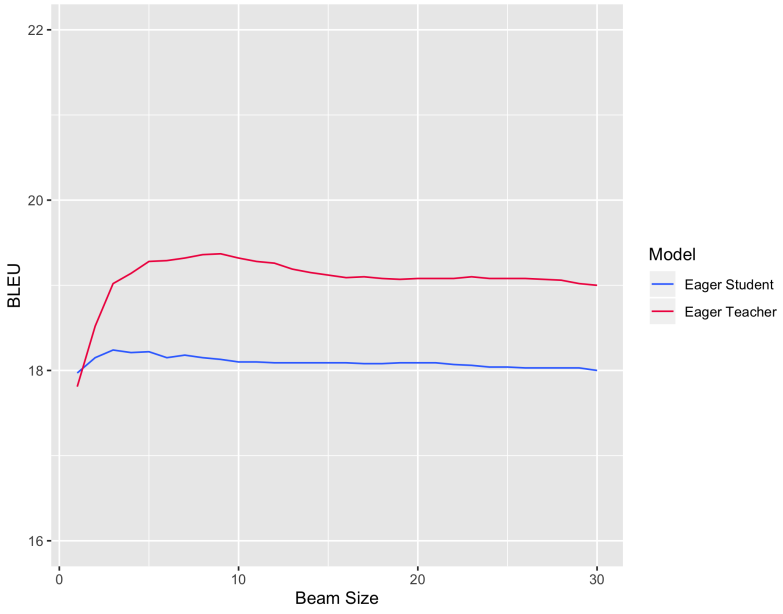


Figure 14: `BLEU` scores for the eager teacher and eager student models for various values of beam size. Other parameters are SPI 9, epsilon limit 3 and 4 start padding tokens. The eager student performs significantly worse for all beam sizes other than beam size 1.

Figure 14 shows the `BLEU` scores for the teacher model and the student model for

Model	Beam Size	Start Pads	Epsilon Limit	SPI	BLEU (↑)
Eager Teacher	5	4	3	9	19.28
Eager Teacher	9	4	3	9	19.37
Eager Teacher	1	4	3	9	17.81
Eager Student	1	4	3	9	17.97
Eager Student	5	4	3	9	18.22
Transformer Student	**1**	**4**	**3**	**9**	**21.44**
Transformer Student	**3**	**4**	**3**	**9**	**21.75**
OpenNMT Transformer Student					21.29

Table 8: `BLEU` scores of all models. Eager student performs significantly worse than the teacher model in all configurations except during greedy decoding. The Transformer student outperforms all other eager translation models by a substantial margin. Notice that we show the translation quality for several different beam sizes. Beam size 5 is our default beam size for the eager teacher model. The other beam sizes show the scores for greedy decoding and the highest scores that we achieve given the same values for the other hyperparameters.

increasing beam sizes. The benefit from beam search in the student model is significantly smaller than in the teacher model. That is, the translation quality of the student model improves only slightly with beam search, and the translation quality of the student model decreases for beam sizes greater than 2. This is as expected. If knowledge distillation reduces beam sizes or even eliminates the need for beam search, the student model will not benefit from beam search.

The student model performs on par with the teacher model with greedy decoding. For all beam sizes other than beam size 1, the student model performs significantly worse than the teacher model. Clearly, applying sequence-level knowledge distillation between two eager translation models does not reduce the necessary beam size at all. The eager student model we trained is not a reasonable replacement for the teacher model.

However, knowledge distillation is not limited to two instances of the same model. On the contrary, a key benefit of knowledge distillation is that the teacher model can be any arbitrary model or even an ensemble of models [Buciluǎ et al., 2006; Hinton et al., 2015; Freitag et al., 2017].

A model that has recently proven to perform well in machine translation tasks and was shown to improve the translation quality of student models when used for knowledge distillation [Gu et al., 2017], is the Transformer introduced by [Vaswani et al., 2017].

We have trained a Transformer model and used it to translate the source sentences from the training data. We used the translations of the Transformer model and trained another eager translation model with this data set with the same configuration we used for all eager models that we trained previously.

This, of course, means that we invested additional time and resources to train another teacher model. However, a resource-intensive training phase is better than a resource-intensive usage, once a model is deployed. A model is typically used many more times than it is trained, and the higher cost incurred by longer training will be amortized by using a model many times if it is faster at inference time.

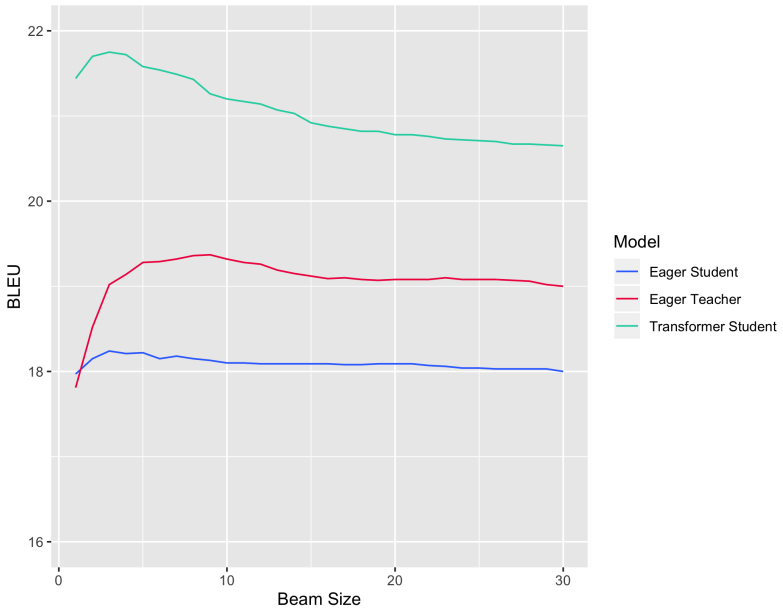


Figure 15: BLEU scores of all models for various values of beam size. Other parameters are SPI 9, epsilon limit 3 and 4 start padding tokens. The Transformer student model outperforms both eager models in every dimension by a substantial margin.

Figure 15 shows the BLEU scores of the Transformer student model for various beam sizes compared to the scores of the eager student and the original eager translation model. The figure shows that the Transformer student model performs significantly better than all previous models, regardless of the beam size. It also shows that the Transformer student model only benefits from beam search with beam sizes 1, 2, and 3. For any beam size greater than 3, the translation quality of the Transformer student model decreases. Also, using beam search with the Transformer student model improves the translation quality only by a small margin.

We achieve the highest BLEU score of 21.75 with a beam size of 3 and report a slightly lower BLEU score of 21.44 for the Transformer student model with greedy decoding. These and the scores of all other student models can also be found in Table 8.

Model	Beam Size (↓)	BLEU (↑)	Sentences per Second (↑)
Eager Teacher	1	17.81	4.58
Eager Teacher	5	19.28	2.40
Eager Teacher	25	19.08	0.97
Transformer Student	1	21.44	4.48
Transformer Student	5	21.58	2.46
Transformer Student	25	20.71	1.02

Table 9: `BLEU` scores and sentences translated per second for eager teacher and Transformer student. There is no significant difference in the translation speed between the models given the same beam size. With beam size 1, the Transformer student produces significantly more accurate translations than the eager teacher model at almost twice the speed.

Our results demonstrate that it is possible to eliminate the need for beam search in the eager translation model without sacrificing translation quality. Indeed, using a more powerful teacher model, we were able to train a student model that outperforms the original eager translation model with greedy decoding by a substantial margin. Similarly, we expect that the translation quality would be even better if we used an ensemble of Transformers for producing the transfer set, rather than only a single model.

6.3 Performance Improvement with Greedy Decoding

In the previous section, we have shown that we can eliminate the need for beam search in the eager translation model by applying sequence-level knowledge distillation using the predictions of a Transformer model as the transfer set. Because beam search requires the model to process all source tokens before producing the final translation, there is no immediate advantage from producing translation tokens eagerly. Eliminating the need for beam search eliminates the need to process all source tokens before committing to a final translation and thereby restores the simultaneity in the eager translation model.

In addition, translating greedily is significantly faster. Figure 16 shows the translation quality of the eager teacher model and the Transformer student model relative to their translation speeds for beam sizes 25, 5, and 1. The translations are most accurate with beam size 5 and are produced fastest with beam size 1, for both models. We show the same results in Table 9.

The translation speed does not differ between the models given the same beam size. The speed improvement is caused entirely by the elimination of beam search. Compared to our baseline beam size of 5, the Transformer student produces much better translations at almost twice the speed. Compared to the beam size of 25, which [Press and Smith, 2018] use in some of their best configurations, the Transformer student is capable of translating at more than four times the speed of the original eager translation model.

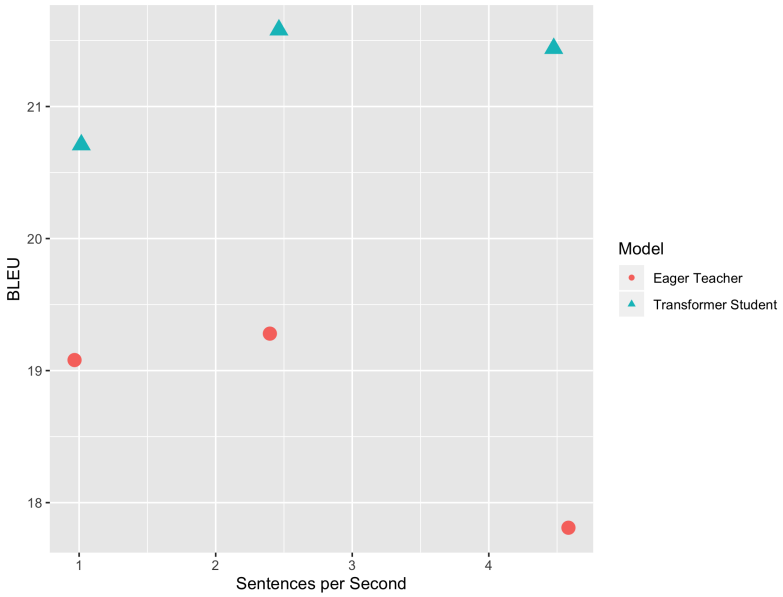


Figure 16: Translation quality relative to translation speed for the eager teacher and Transformer student model. The translation speed does not differ between the models given the same beam size.

6.4 Qualitative Analysis of Student Models

The Transformer student model evidently produces much better translations than the eager teacher model. The quality of these translations is better even when the Transformer student decodes greedily. But, does the Transformer student model also wait more reasonably?

In this chapter, we repeat the manual analysis that we have performed in Chapter 5 and compare the results between the eager teacher model and the student models that we have trained.

6.4.1 Setup

We translated newstest2013 EN $\leftrightarrow$ DE, that we downloaded using sockeye-autopilot[1], with the eager teacher model, the eager student, and the Transformer student. We used the same configuration of 4 start padding tokens, an epsilon limit of 3, an SPI of 9, and a beam size of 5 for all models. This is the configuration that we selected in Chapter 4 and that we also used in Chapter 5, when we manually evaluated the eager teacher model.

The data set contains 3′000 sentences. Of all translations produced by the eager model, 724 sentences contain at least one ε token. The eager student and the Transformer student produce 463 and 311 translations with at least one ε token, respectively. A total of 303 sentences contain ε tokens in the translations of all models, and 9 of these translations are identical across all three models.

We randomly selected 60 of these 303 sentences from all models and manually evaluated the quality of the translations. During the evaluation, the identity of the model was hidden from us, and we did not know which model produced the translation. Furthermore, we evaluated all sentences in random order such that the translations of the same source sentence from different models did not follow immediately. We assessed the same criteria as we did in Chapter 5 with a particular focus on whether the models produced ε tokens reasonably and if the sentences are too long or too short.

6.4.2 Results

Both student models exhibit a similar waiting pattern as the eager teacher model. All models predominantly use ε tokens to wait reasonably.

The eager student closely resembles the eager teacher in terms of waiting behavior but has a very strong tendency to produce translations that are too short. The vast majority of the translations that the eager student produces contain too few tokens. This appears comprehensible. The eager teacher model already produces translations that are too short. Thus, a large number of translations in the transfer set that we produced using the eager teacher model contain too few tokens such that knowledge distillation might amplify this weakness in the eager student model.

The Transformer student outperforms the eager teacher also in terms of waiting behavior. The Transformer student exhibits a reasonable waiting behavior in al-

[1] https://github.com/awslabs/sockeye/tree/master/sockeye_contrib/autopilot

Did the use of ε tokens make sense?				
Model	Yes	No	Don't know	Yes, but not for all ε tokens
Eager	45	9	2	4
Eager Student	48	5	3	4
Transformer Student	56	2	2	0

Table 10: Summary of the manual evaluation. The eager translation model makes good use of ε tokens in most translations. The Transformer student performs best overall.

Translation Length		
Model	Too Short	Too Long
Eager Teacher	24	0
Eager Student	43	1
Transformer Student	13	20

Table 11: Number of translations from the manual evaluation that were either too short or too long. All models produce translations that are too short. The eager student has the strongest tendency to produce translations that are too short. The Transformer student is the only model that consistently produces translations that are too long.

most all translations that we evaluated. We assume that this is a consequence of the Transformer student generally producing better translations and being able to model the structure of the source sentence more accurately. The Transformer student also produces significantly fewer translations that are too short. However, the Transformer student has a strong tendency to produce translations that are too long.

6.5 Understanding Knowledge Distillation

[Zhou et al., 2019] report improvements in the translation quality of non-autoregressive translation models after applying sequence-level knowledge distillation on the same models that they trained on the original data set. We report the same results for our OpenNMT reference model and achieve a `BLEU` score of 21.29 with the OpenNMT student model compared to a `BLEU` score of 19.06 that we achieve with the original OpenNMT reference model.

The reason for the improvement in translation quality is not entirely clear. We assume that two properties of the distilled training data may cause the superior translation quality at lower beam sizes. We assume that the distilled training data

is more deterministic, and thus less noisy, and more parallel and monotonous than the original training data. In this chapter, we propose two measures, a measure to compute the parallelism of a pair of languages and a measure to compute the determinism of the predictions of a translation model. We compute and compare these values between our data sets and models.

6.5.1 Parallelism

Simultaneous machine translation is particularly challenging when the information that appears early in the target sentence appears late in the source sentence [Grissom II et al., 2014]. The eager translation model [Press and Smith, 2018] even includes a strict requirement that no token from the target sentence may appear before its corresponding token from the source sentence. This is because the eager translation model produces exactly one translation token for every source token it consumes, immediately after consuming it. If the target token appears before the source token, the eager model has no chance of obtaining the information required to produce the correct translation. Thus, if this is not naturally the case in a given sentence pair, we add ε tokens to the target sentence in order to resolve the misalignment. This makes the sentence pair eager feasible.

After eager feasible pre-processing, no word from any target sentence appears before its corresponding word in the source sentence. In that case, the alignment between those words is said to be *monotonous*. That is, from the perspective of the source token, the index of the corresponding word from the target sentence is always equal to or greater than the index of the source token. For an example and a detailed description of this process, see Figure 5 in Chapter 3.

Although all sentence pairs that we train the eager translation model with are monotonous, according to the definition above, the translations are not parallel. That is, corresponding words from the source and target sentence appear at different indexes in these sentences.

We propose a measure of translation *parallelism* of a corpus measured as the sum of the normalized distances between corresponding words, drawn from word alignments.

Let $x_i \in x$ be the ith token in a source sentence x and let y_j be the jth token in the corresponding target sentence y such that x_i aligns with y_j, according to an alignment model. We call the difference $|i - j|$ between the indexes the *distance* between the tokens x_i and y_j.

If the word from the source sentence x_i appears before the corresponding word from the target sentence y_j, i.e., if $i < j$, we call the distance between the words *forward distance*. From the perspective of the source sentence, the translation of the ith token lies in the future. The information must flow *forward*.

Similarly, if the word from the source sentence x_i appears after the corresponding word from the target sentence y_j, i.e., if $i > j$, we call the distance between the words *backward distance*. From the perspective of the source sentence, the translation of the ith token lies in the past. The information must flow *backward*. Because information cannot flow backward in the eager translation model, backward distances must be resolved with ε tokens. Notice that the number of ε tokens required is typically lower than the total backward distance in a sentence pair. This is because adding one ε token at index j also moves all following tokens by one and thus reduces the number of ε tokens required to resolve these alignments.

We call the sum of all distances in a sentence pair the *total distance* in that sentence. We normalize the total distance of a sentence with the number of tokens in the target sentence, compute the sum of all normalized distances over all sentences and normalize this sum with the number of sentence pairs in the corpus

$$P(C) = \frac{1}{|C|} \sum_{(x,y) \in C} \sum_{i,j \in a(x,y)} \frac{abs(i-j)}{T} \tag{6.1}$$

where T is the length of the target sentence, C is the corpus, and a is an alignment model.

The normalized total distance is the average distance between tokens in a sentence pair, and the normalized sum is the average distance between tokens in all sentences of the corpus. This allows us to compute these values for the original training data as well as the two distilled data sets and compare if the distances between tokens have changed.

We further compute the minimal *number of ε tokens required* to resolve all backward distances in a sentence pair. We normalize this number with the number of tokens in the target sentence, compute the sum over all sentences, and normalize this number again with the number of sentence pairs.

We used *fast_align*[2] to compute the word alignments on the original training data, the distilled data that we used for the eager student model, and the distilled data that we used to train the Transformer student model and used these alignments to

[2] `https://github.com/clab/fast_align`

	Eager Teacher	Eager Student	Transformer Student
Forward Distance (↓)	2.16	0.68	1.24
Backward Distance (↓)	1.71	0.87	1.15
Total Distance (↓)	3.87	1.55	2.39
ε Tokens Required (↓)	0.30	0.14	0.18

Table 11: Normalized distances and ε tokens required in all data sets.

compute $P(C)$ for each data set.

Table 11 shows all normalized distances and the normalized sum of ε tokens required to resolve all non-monotonous alignments in the distilled data sets.

Corresponding words in the translations produced by the eager model are, over the entire corpus, 60% closer than in the original training data. The backward distances were reduced by almost half, and the forward distances were reduced by more than two thirds. Similarly, the training data used for the eager student requires 51% fewer ε tokens than the original data set.

The translations produced by the Transformer require 61% of the number of ε tokens of the original data set to make the training data eager feasible and corresponding tokens are 38% closer. The difference between the forward and backward distance is not as distinct as in the translations produced by the eager model.

The distances between corresponding words decreased significantly in both distilled data sets, and the number of ε tokens required to make each distilled data set eager feasible is also substantially lower than in the original training data. Both distillation methods increase parallelism by roughly cutting distances in half. However, the translations of the eager model have much lower distances and require substantially fewer ε tokens than the translations of the Transformer teacher, despite the eager student model producing translations with a much lower quality. Thus, increased parallelism and less total distance between aligned words from the source and target sentences cannot be the only reason for superior translation quality at lower beam sizes in student models.

6.5.2 Determinism

Translations are not deterministic. In many cases, there are multiple semantically equivalent translations for the same source sentence. For instance, the sentence "Thank you" could be translated to "Danke", "Dankeschön" or "Danke dir", with all

	Eager Teacher	Eager Student	Transformer Student
$D(C) \uparrow$	-12.48	-3.24	-4.66

Table 12: Determinism scores for all models of their predictions of the development set. The numbers are log probabilities. Higher values are better.

translations being equally correct (example taken from [Gu et al., 2017]). However, a neural machine translation model will consistently generate the same translation for the same input sentence and thus, when we use the predictions of a neural machine translation model, i.e., the teacher model, as the target sentences for the training of another neural machine translation model, i.e., the student model, the training data of the student model is expected to contain fewer variations of translations of the same source tokens. That is, the distilled training data is expected to be more deterministic and less noisy [Gu et al., 2016, 2017; Zhou et al., 2019].

Inspired by the observations and computations of [Zhou et al., 2019], we propose a measure of translation *confidence* of a translation model measured as determinism $D(C)$ of a model over a corpus C.

Let x be a source sentence from corpus C and let $\hat{y}$ be a possible translation of the sentence x from all possible translations y of a model. Then, the determinism D of a model for the corpus C is the sum of the log probabilities of the most likely target sentences of all sentences, normalized with the total number of sentences in the corpus.

$$D(C) = \frac{1}{|C|} \sum_{x \in C} \underset{\hat{y}}{\operatorname{argmax}} \log p(y|x) \tag{6.2}$$

The determinism D of a model for a corpus C is the average *confidence* with which the model produces its predictions. If the determinism scores between two models differ for the same corpus, we say that the model with the greater value D produces its translations with greater confidence and the predictions are more deterministic.

We assume that the student models produce predictions with higher confidence than the teacher model because the data sets that we used to train the student models contain fewer variations given the same input sequences.

We computed $D(C)$ for our eager translation model, the eager student model, as well as the Transformer student model for the development set of the WMT 2014 `EN` $\leftrightarrow$ `DE` data set.

Table 12 shows the normalized sum of the log probabilities of the predictions produced by our models. The student models return their translations with much greater certainty than the original eager model. This is as we expected. However, the log probabilities of the eager student model are greater than the log probabilities of the Transformer student model. This means that the eager student produces its translations with more confidence than the Transformer student. Yet again, the translations produced by the Transformer student are much better than those of the eager student, and therefore, increased determinism does not explain the superior translation quality of the student models.

7 Limitations

We summarize the findings from our experiments. We analyzed the eager translation model and the translations that the model produces in great detail. In Chapter 4, we have shown that a major challenge for the eager translation model is to produce the correct number of tokens in the translation. We demonstrate that the number of tokens that the model produces can be controlled, to some extent, with the inference hyperparameters epsilon limit and SPI. In Chapter 5, we show that the eager translation model is able to learn to wait and is capable of producing ε tokens when it requires more information from the source sentence. However, for a great number of sentences, the eager translation model produces too many ε tokens. This becomes evident in Chapter 4.3. We show that, in aggregate, the number of tokens in the final translations decreases almost linearly with an increase in the epsilon limit.

The first limitation that we draw is that we based our analysis in Chapter 4 only on newstest2013. We did not use this data set during training, so the model has not seen these sentences before. Nevertheless, using another data set could have helped in confirming our results.

The eager translation model exhibits conflicting properties. On the one hand, it needs to be able to produce ε tokens in order to be able to wait. On the other hand, it must be limited in the number of ε tokens that it produces, or otherwise, it will produce an excessive number of ε tokens. We were unable to reveal the mechanics that cause this behavior and our analysis does not reveal clearly what causes the waiting behavior.

We demonstrate that the Transformer student exhibits a better waiting behavior than the eager teacher model, but it remains unclear why that is.

In our manual analysis, we focused on sentences where all our models produced ε tokens. It might be that this selection caused a strong sampling bias towards good translations and that other translations, on average, exhibit different properties.

Similarly, we could have extended our analysis with more translations from only

one of the models. That is, rather than analyze how the translations compare, we could have investigated only the translations of a single model and obtain a more detailed view on how the eager translation model behaves over a greater number of translations.

On the same note, in this work, we only cover one language pair. It is not evident from our analysis how the eager translation model behaves when trained on a different language pair.

In Chapter 6, we have shown that the eager translation model benefits greatly from knowledge distillation. That is, we demonstrated significant improvements in terms of translation quality and translation speed. However, the Transformer student still translates with a significantly lower quality than the Transformer teacher model.

8 Future Work

In this chapter, we share our thoughts on future work regarding eager translation.

We have demonstrated that the eager translation model is prone to producing translations that are too short or too long. Almost half of all translations that the eager teacher model produces are too short. Similarly, about a third of all translations that the Transformer student model produces are too long. To some extent, this can be controlled with the SPI hyperparameter. However, this hyperparameter is a static value that will be used for all translations in a translation corpus. We anticipate that a more dynamic approach, such as increasing SPI by one for every ε token that the model produces, might improve the ability of the eager translation model to produce the correct number of tokens.

We have shown that the eager translation model is capable of learning a reasonable waiting behavior to obtain crucial information from the source sentence. It is, however, not entirely clear when, how, and why the model decides to wait. This is particularly the case in translations in which the eager translation model produces an excessive number of ε tokens. We propose further research around the waiting behavior of the eager translation model.

The eager translation model closely resembles a traditional encoder-decoder translation model [Zaremba et al., 2014] but does not use attention. This is by design to keep the eager translation model lean. Nevertheless, attention was shown to improve the capabilities of traditional neural translation systems significantly. Similarly, computing attention over the source tokens seen so far might improve the capabilities of the eager translation model and we propose to extend eager translation with such an attention mechanism.

Similarly, we propose to apply eager feasible pre-processing and eager translation to other successful machine learning architectures. More specifically, we propose research on eager Transformer models [Vaswani et al., 2017]. In this setting, self-attention would only be computed between tokens from the source sentence that were already revealed. This would make encoding in the Transformer an iterative process.

Recent proposals for simultaneous neural machine translation [Grissom II et al., 2014; Satija and Pineau, 2016; Gu et al., 2016] show how waiting behavior can be trained using reinforcement learning. [Graves, 2016] show that a similar behavior can be trained in RNNs without reinforcement learning. More precisely, [Graves, 2016] modify an RNN and add a scalar *halting unit* to perform a variable number of state transitions and produce a variable number of outputs at each time step. With this modification, the neural network can dynamically vary the amount of computation it applies to a given input. The network can "ponder" each input. Similarly, we anticipate that the waiting behavior could be implemented as accumulating a scalar halting unit and trade this off with a penalty for pondering time. This would also avoid some known problems with reinforcement learning, most notably, unstable learning dynamics.

9 Conclusion

[Press and Smith, 2018] report that during their experiments, the eager translation model performed on par with their OpenNMT reference model. We confirm their finding and report that, in our setting, the eager translation model outperforms our OpenNMT reference model by a small margin in almost all configurations.

We quantitatively analyzed the hyperparameters used during inference of the eager translation model in Section 4 and have shown how the translations produced by the eager model change when we change any of these hyperparameters.

The eager translation model is prone to producing translations that contain either too few or too many tokens. That is, the translations are consistently either too short or too long. This is a direct consequence of the model producing exactly as many tokens as it consumes. We have shown in Section 4.3 and Section 4.4 that the hyperparameters epsilon limit and SPI can be used as proxy parameters to control the lengths of the translations. The model is particularly sensitive to changes in the epsilon limit and consistently produces shorter translations when the epsilon limit is increased. The model is less sensitive to changes in SPI, which we have shown is a direct consequence of the non-standard use of consecutive source `EOS` tokens in the beam search algorithm that [Press and Smith, 2018] use.

We have established a good foundation for estimating reasonable values for the epsilon limit and SPI hyperparameters using measures of central tendency that reflect the configuration that the model experienced during training.

We achieve the highest `BLEU` score of 19.37 with a low epsilon limit of 3 and an SPI value between 9 and 20, a beam size of 9 and 4 start padding tokens.

We modified the beam search algorithm that [Press and Smith, 2018] introduced and removed the part where the decoder repeatedly receives source `EOS` tokens. The translation quality is slightly lower, yet still higher than with the baseline configuration. However, with our modifications, the results and the translations seem much more comprehensible.

We investigated the performance of the eager model in the ideal setting where the

inference hyperparameters are set dynamically by an oracle. In this setting, the epsilon limit and SPI parameters are always set to match the actual numbers from the target sentence. Surprisingly, the model does not benefit from this, and we report worse BLEU scores in the oracle experiment. This shows that the selection of hyperparameters is not a key stumbling block why the eager translation model might underperform a traditional encoder-decoder model, such as the OpenNMT reference model. This also shows that excessive tuning of the inference hyperparameters does not lead to significant improvements in translation quality.

In Chapter 5, we show that the eager translation model consistently produces ε tokens to wait for crucial information from the source sentence. The eager translation model predominantly uses ε tokens to resolve non-monotonous alignments between the verb in the source sentence and the same verb in the translation. We observe and show that, in many cases, the eager model waits not only reasonably and correctly but also *minimally* in that it only produces the minimal number of ε tokens that it requires in order to receive all relevant information. Evidently, the eager translation model is able to capture, learn, and apply a reasonable waiting behavior.

In aggregate, however, the eager translation model consistently produces more ε tokens when given the chance. In Section 4.3, we have shown that increasing the epsilon limit hyperparameter consistently increases the total number of ε tokens that the eager model produces over an entire translation corpus. This drastically reduces the lengths of the final translations and, with that, the overall translation quality.

It is not entirely clear why and when the eager translation model produces ε tokens. The eager translation model exhibits two conflicting properties. Limiting the number of ε tokens that the eager model may produce improves the general translation quality. Yet, removing this limit appears to improve the general waiting behavior of the model.

We were unable to identify the source of this conflict and were unable to reveal how and when precisely the eager translation model produces *incorrect* ε tokens. It could be that our selection of translations for manual analysis has a strong bias towards good translations, as these are translations where all our models exhibit a similar behavior and waiting pattern.

We applied sequence-level knowledge distillation and have trained an eager translation model with translations produced by another eager translation model. The eager teacher model outperforms the eager student model in every dimension, and we were unable to remove beam search without sacrificing translation quality sub-

stantially. We have also trained a Transformer model and applied sequence-level knowledge distillation to the eager translation model using the predictions of the Transformer. The Transformer student model outperforms the original eager translation model in every configuration. With both models decoding greedily, the Transformer student model outperforms the original eager translation model by more than 3.6 BLEU. Compared to our baseline eager translation model with beam search, the Transformer student model, without beam search, outperforms the original model by more than 2 BLEU. In this comparison, the Transformer student model produces substantially better translations at almost twice the translation speed. Furthermore, the Transformer student exhibits a more consistent waiting behavior, uses ε tokens more reasonably, and produces significantly fewer translations that are too short. It does, however, have a much higher tendency for producing sentences that are too long.

We investigated the effects of sequence-level knowledge distillation on the quality of the training data by analyzing the properties of monotonicity, parallelism, and determinism in all data sets. We confirm the tendency that these three measures increase through knowledge distillation. However, we were unable to show a clear connection between the magnitude with which monotonicity, parallelism, and determinism increase through knowledge distillation and the improvement in the translation quality. Both distilled data sets are significantly more parallel and monotonous than the original data set. The effect is substantially more prominent in the translations produced by the eager model. Similarly, the eager student produces translations with a much higher certainty than the Transformer student. Yet, the eager student produces the worst translations of all our models.

References

D. Bahdanau, K. Cho, and Y. Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

S. Bangalore, V. K. Rangarajan Sridhar, P. Kolan, L. Golipour, and A. Jimenez. Real-time incremental speech-to-speech translation of dialogs. In *Proceedings of the 2012 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 437–445. Association for Computational Linguistics, 2012.

C. Buciluǎ, R. Caruana, and A. Niculescu-Mizil. Model compression. In *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 535–541, 2006.

K. Cho and M. Esipova. Can neural machine translation do simultaneous translation? *arXiv preprint arXiv:1606.02012*, 2016.

K. Cho, B. Van Merriënboer, D. Bahdanau, and Y. Bengio. On the properties of neural machine translation: Encoder-decoder approaches. *arXiv preprint arXiv:1409.1259*, 2014a.

K. Cho, B. Van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014b.

F. Dalvi, N. Durrani, H. Sajjad, and S. Vogel. Incremental decoding and training methods for simultaneous translation in neural machine translation. *arXiv preprint arXiv:1806.03661*, 2018.

C. Dyer, V. Chahuneau, and N. A. Smith. 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, 2013.

M. Freitag, Y. Al-Onaizan, and B. Sankaran. Ensemble distillation for neural machine translation. *arXiv preprint arXiv:1702.01802*, 2017.

T. Fujita, G. Neubig, S. Sakti, T. Toda, and S. Nakamura. Simple, lexicalized choice of translation timing for simultaneous speech translation. In *INTERSPEECH*, pages 3487–3491, 2013.

J. Gehring, M. Auli, D. Grangier, D. Yarats, and Y. N. Dauphin. Convolutional sequence to sequence learning. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 1243–1252. JMLR. org, 2017.

F. A. Gers, J. Schmidhuber, and F. Cummins. Learning to forget: Continual prediction with lstm. 1999.

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

A. Grissom II, H. He, J. Boyd-Graber, J. Morgan, and H. Daumé III. 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, 2014.

J. Gu, G. Neubig, K. Cho, and V. O. Li. Learning to translate in real-time with neural machine translation. *arXiv preprint arXiv:1610.00388*, 2016.

J. Gu, J. Bradbury, C. Xiong, V. O. Li, and R. Socher. Non-autoregressive neural machine translation. *arXiv preprint arXiv:1711.02281*, 2017.

G. Hinton, O. Vinyals, and J. Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015.

S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

H. Inan, K. Khosravi, and R. Socher. Tying word vectors and word classifiers: A loss framework for language modeling. *arXiv preprint arXiv:1611.01462*, 2016.

N. Kalchbrenner and P. Blunsom. Recurrent continuous translation models. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 1700–1709, 2013.

Y. Kim and A. M. Rush. Sequence-level knowledge distillation. *arXiv preprint arXiv:1606.07947*, 2016.

D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

G. Klein, Y. Kim, Y. Deng, J. Senellart, and A. M. Rush. Opennmt: Open-source toolkit for neural machine translation. *arXiv preprint arXiv:1701.02810*, 2017.

M.-T. Luong, H. Pham, and C. D. Manning. Effective approaches to attention-based neural machine translation. *arXiv preprint arXiv:1508.04025*, 2015.

M. Ma, L. Huang, H. Xiong, R. Zheng, K. Liu, B. Zheng, C. Zhang, Z. He, H. Liu, X. Li, et al. Stacl: Simultaneous translation with implicit anticipation and controllable latency using prefix-to-prefix framework. *arXiv preprint arXiv:1810.08398*, 2018.

S. Merity, N. S. Keskar, and R. Socher. Regularizing and optimizing lstm language models. *arXiv preprint arXiv:1708.02182*, 2017.

Y. Oda, G. Neubig, S. Sakti, T. Toda, and S. Nakamura. 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, 2014.

M. Ott, M. Auli, D. Grangier, and M. Ranzato. Analyzing uncertainty in neural machine translation. *arXiv preprint arXiv:1803.00047*, 2018.

M. Post. A call for clarity in reporting bleu scores. *arXiv preprint arXiv:1804.08771*, 2018.

O. Press and N. A. Smith. You may not need attention. *arXiv preprint arXiv:1810.13409*, 2018.

O. Press and L. Wolf. Using the output embedding to improve language models. *arXiv preprint arXiv:1608.05859*, 2016.

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

R. Sennrich, B. Haddow, and A. Birch. Neural machine translation of rare words with subword units. *arXiv preprint arXiv:1508.07909*, 2015.

V. K. R. Sridhar, J. Chen, S. Bangalore, A. Ljolje, and R. Chengalvarayan. Segmentation strategies for streaming speech translation. In *Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 230–238, 2013.

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

A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008, 2017.

Y. Vyas, X. Niu, and M. Carpuat. Identifying semantic divergences in parallel text without annotations. *arXiv preprint arXiv:1803.11112*, 2018.

R. J. Williams and D. Zipser. A learning algorithm for continually running fully recurrent neural networks. *Neural computation*, 1(2):270–280, 1989.

Y. Wu, M. Schuster, Z. Chen, Q. V. Le, M. Norouzi, W. Macherey, M. Krikun, Y. Cao, Q. Gao, K. Macherey, et al. Google's neural machine translation system: Bridging the gap between human and machine translation. *arXiv preprint arXiv:1609.08144*, 2016.

M. Yarmohammadi, V. K. R. Sridhar, S. Bangalore, and B. Sankaran. Incremental segmentation and decoding strategies for simultaneous translation. In *Proceedings of the Sixth International Joint Conference on Natural Language Processing*, pages 1032–1036, 2013.

W. Zaremba, I. Sutskever, and O. Vinyals. Recurrent neural network regularization. *arXiv preprint arXiv:1409.2329*, 2014.

C. Zhou, G. Neubig, and J. Gu. Understanding knowledge distillation in non-autoregressive machine translation. *arXiv preprint arXiv:1911.02727*, 2019.

A Tables

Beam Size	Start Pads	Epsilon Limit	SPI	BLEU
25	0	5	7	11.50
25	1	5	7	15.81
25	2	5	9	16.53
25	3	4	9	17.36
25	4	2	9	17.52
5	5	1	13	17.97
OpenNMT Ref. Model				18.89

Table 13: `BLEU` scores for different configurations from Press and Smith [2018].

Model	Beam Size	Start Pads	Epsilon Limit	SPI	BLEU
Eager	5	4	3	4	18.46
Eager	25	4	2	9	19.18
Eager	5	4	3	9	19.28
Eager	1	4	3	9	17.81
Eager	9	4	3	9	19.37
Eager, Modified Beam Search	5	4	3	4	19.16
Eager, Modified Beam Search	5	4	3	5	19.24
OpenNMT Ref. Model					19.06
Oracle	5	4	-	-	18.40
Oracle, Modified Beam Search	5	4	-	-	18.07
Eager Student	1	4	3	9	17.97
Eager Student	5	4	3	9	18.22
Transformer Student	**1**	**4**	**3**	**9**	**21.44**
Transformer Student	3	4	3	9	21.75
OpenNMT Transformer Student	-	-	-	-	21.29
Transformer Teacher	-	-	-	-	25.90

Table 14: `BLEU` scores for different configurations of our models.

B Training Details

This appendix shows the training configurations that we used for our models.

B.0.1 Eager Model Training Configuration

We used exactly the same configuration for training our eager translation model as [Press and Smith, 2018] outline in their work. For reference and completeness, we include the configuration in this appendix.

We trained the eager translation model with four LSTM layers with $1,000$ units each and an embedding size of 1000. The model was regularized during training using dropout on the LSTM as well as the word embeddings [Merity et al., 2017]. We trained the model for 25 epochs with a batch size of 200 tokens with backpropagation through time of 60 tokens. The optimization algorithm was Stochastic Gradient Descent with an initial learning rate of 20. The learning rate was halved whenever there was no improvement on the validation set after $6,500$ updates.

B.0.2 OpenNMT Model Training Configuration

We used the exact same configuration for training the OpenNMT reference model as [Press and Smith, 2018] have. The authors present their configuration in [Press and Smith, 2018]. For reference and completeness, we include the configuration in this appendix.

We trained the OpenNMT reference model with two LSTM layers in the encoder, two LSTM layers in the decoder and an embedding size of 500. The optimization algorithm was Stochastic Gradient Descent and we trained the model until there was no improvement on the validation set for $50,000$ updates.

B.0.3 Transformer Training Configuration

In the following we describe the training configuration that we used for the Transformer model from Chapter 6. We trained a Transformer model [Vaswani et al., 2017] using sockeye[1] with 6 layers and 8 attention heads each. We used an embedding size of 512 and 2048 hidden units in the feed forward networks. The model was regularized during training using dropout. We trained the model in batches of $4,096$ words until there was no improvement on the validation set for 32 checkpoints with a checkpoint frequency of 4000 steps and an initial learning rate of 0.0001. The optimization algorithm that we used was Adam [Kingma and Ba, 2014]. We further used weight tying [Press and Wolf, 2016] and label smoothing. The training script with all our configurations is publicly available[2].

[1] `https://github.com/awslabs/sockeye`

[2] `https://github.com/bricksdont/sockeye-toy-models/blob/wmt14-ende-teacher/scripts/train.sh`