This Candidacy

entitled

IMPROVING NEURAL MACHINE TRANSLATION FOR LOW-RESOURCE LANGUAGES

typeset with NDdiss2$_\varepsilon$ v3.2017.2 (2017/05/09) on December 7, 2019 for

Toan Q. Nguyen

This LaTeX 2$_\varepsilon$ classfile conforms to the University of Notre Dame style guidelines as of Fall 2012. However it is still possible to generate a non-conformant document if the instructions in the class file documentation are not followed!

> Be sure to refer to the published Graduate School guidelines at `http://graduateschool.nd.edu` as well. Those guidelines override everything mentioned about formatting in the documentation for this NDdiss2$_\varepsilon$ class file.

This page can be disabled by specifying the "`noinfo`*" option to the class invocation.* (i.e.,`\documentclass[...,noinfo]{nddiss2e}`)

This page is *NOT* part of the dissertation/thesis. It should be disabled before making final, formal submission, but should be included in the version submitted for format check.

NDdiss2$_\varepsilon$ documentation can be found at these locations:

`http://graduateschool.nd.edu`
`https://ctan.org/pkg/nddiss`

IMPROVING NEURAL MACHINE TRANSLATION FOR LOW-RESOURCE LANGUAGES

A Candidacy

Submitted to the Graduate School
of the University of Notre Dame
in Partial Fulfillment of the Requirements
for the Degree of

Doctor of Philosophy

by

Toan Q. Nguyen

_____

David Chiang , Director

Graduate Program in
Notre Dame, Indiana
December 2019

IMPROVING NEURAL MACHINE TRANSLATION FOR LOW-RESOURCE LANGUAGES

Abstract

by

Toan Q. Nguyen

Neural Machine Translation (NMT), which exploits the power of continuous representation and neural networks, has recently become the *de facto* choice in both academic research and industry. While it started more as a proof of concept, with the invention of the attention mechanism, NMT has been able to achieve state-of-the-art performance. However, like other neural networks, these NMT systems are often data-hungry and require millions of training examples to achieve competitive results.

Unfortunately, only a few language pairs have this privilege. Most fall into the low-resource domain and often have only around 100k training sentence pairs or less. Some examples could be the LORELEI [1] or IWSLT datasets. This lack of data is particularly critical for the early recurrent neural network-based NMT systems which are often outperformed by the traditional Phrase-based Machine Translation (PBMT) ones. It also means more advanced techniques are required to effectively train a good NMT system for low-resource languages.

In this document, I will show that with better use of training data, improved model architecture or better normalization, we can in fact build a competitive NMT system with only limited data at hand. Firstly, I demonstrate how to exploit the relatedness between

[1] https://www.darpa.mil/program/low-resource-languages-for-emergent-incidents

languages for better transfer learning from one language pair to the other. Secondly, I propose a simple lexical module which greatly alleviates the mistranslation problem for rare words. Finally, I show how simple ℓ_2-based normalization at word embedding and hidden state levels can significantly improve translation for low-resource languages. Additionally, I propose two solutions to improve NMT performance by planning ahead the translation length and better architecture for exploiting multilingual data. Once completed, this thesis will be a combination of techniques which hopefully can be useful for NMT research in general, and for low-resource domain in particular.

CONTENTS

FIGURES

TABLES

CHAPTER 1

BACKGROUND

Neural Machine Translation is part of the *sequence to sequence learning* framework using neural networks. First introduced by Sutskever et al. [78], the model consists of an *encoder* which encodes the source sentence into a fixed-length vector, from which the *decoder* generates the target sentence. While achieving impressive result, this model still performs well below the traditional phrase-based systems. As pointed out by Cho et al. [9], the main reason lies in the limited capacity of the fixed-length vector representation to encode long sentences. To address this problem, Bahdanau et al. [5] proposes the *attention mechanism* which allows the decoder to, at every time step, dynamically select a subset of encoder outputs to focus on instead of relying on a fixed-length representation. This *encoder-decoder-attention* model significantly improves NMT translation on long sentences and is the first NMT system to achieve comparable performance with phrase-based ones.

Because NMT is a sequence to sequence learning problem, the apparent choice for the encoder or decoder architecture is the recurrent neural networks (RNNs). Oftentimes, RNN variants such as LSTM [28] or GRU [10] are used. The Transformer [81] takes a radical departure from this. This model relies entirely on the attention mechanism for the encoder to compute source sentence representation, or for the decoder to remember which part of the target sentence has been decoded so far. Aside from its state-of-the-art performance, the Transformer is also appealing for its train-time parallelism which better exploits the power of the graphics processing units (GPUs). Transformer was not the first to replace RNNs; in fact, an earlier work by Gehring et al. [15] uses convolutional neural networks

instead of LSTM and achieves state-of-the-art performance on many NMT tasks. However, since Transformer has been shown to perform better and becomes the *de facto* choice for NMT, I will only discuss it in this document and suggest readers to [15] for more details.

In this chapter, I will provide a rough overview of the RNN-based and attention-based (Transformer) NMT models. For a more complete picture of the former, I suggest readers to previous works [78, 10, 5, 9, 50]. For the latter, aside from the original work [81], Rush [69] provides a complete dissection on the model and is a good reference for implementation.

1.1 RNN-based NMT

A RNN-based NMT often consists of a source embedding $W_s \in \mathcal{R}^{d \times V_x}$, a target input embedding $W_y \in \mathcal{R}^{d \times V_y}$, a target output embedding $W_o \in \mathcal{R}^{V_y \times d}$, an encoder f and decoder g. V_x, V_y, d are source vocabulary size, target vocabulary size and the embedding size respectively. f and g are some variants of RNN; typically LSTM or GRU are used.

Each source sentence is represented as a sequence of one-hot vector $(x_1, \cdots, x_{L_x}), x_i \in \mathcal{R}^{V_x}$, with L_x be the source sentence length, then the input to the encoder is:

$$\mathbf{x} = (E_s x_1, \cdots, E_s x_{L_s}) \tag{1.1}$$

The encoder reads in this sequence step by step and generates, at time step s, an output vector $h_s = f(x_s, h_{s-1})$. The decoder works similarly and generates, at time step t, an output vector $h_t = g(y_t, h_{t-1})$. This h_t is used by the attention mechanism to calculate the weight α_s for each h_s. The context vector c_t is then simply a weighted sum over all encoder outputs:

$$c_t = \sum_{s=1}^{L_x} \alpha_s h_s \tag{1.2}$$

While I suggest readers to [5] or [50] for other ways to calculate α_s, a simple approach

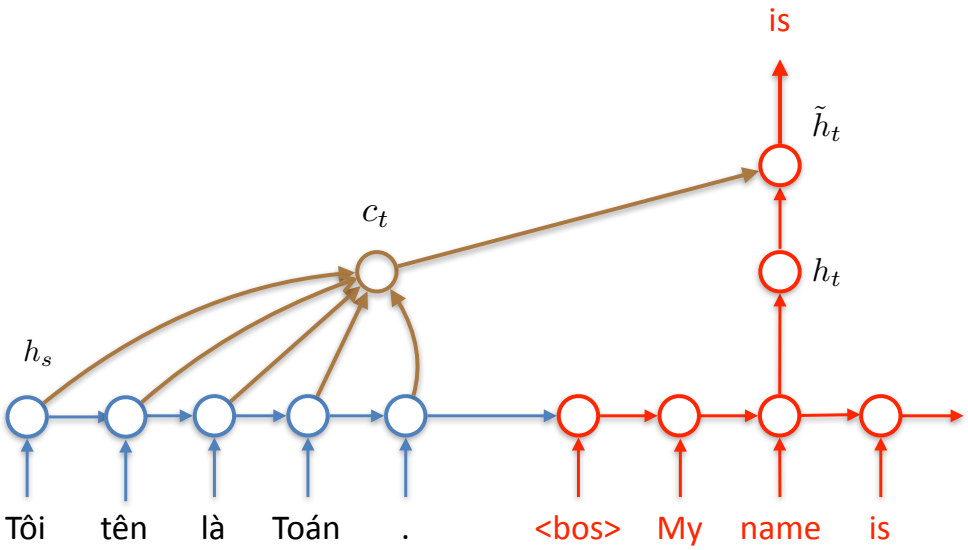


Figure 1.1: An RNN-based NMT with attention model. The encoder is represented in blue, the decoder is in red and the attention mechanism is in brown. Best viewed in color.

called *dot product* by Luong et al. [50] is:

$$\mathrm{e}_s = h_t^T h_s \tag{1.3}$$

$$\alpha_s = \frac{\exp(\mathrm{e}_s)}{\sum_{s'=1}^{L_x} \exp(\mathrm{e}_{s'})} \tag{1.4}$$

The decoder output h_t and context vector c_t are combined, either by a nonlinear feed-forward neural network or simply summed together, to produce the final output vector $\tilde{h}_t$. Finally, the predicted probability of the t-th target word is:

$$p(y_t \mid y_{<t}, \mathbf{x}) = \mathrm{softmax}(W_o \tilde{h}_t) \tag{1.5}$$

The model is trained to maximize the conditional probability:

$$p(\mathbf{y} \mid \mathbf{x}) = \prod_{t=1}^{L_y} p(y_t \mid y_{<t}, \mathbf{x}) \tag{1.6}$$

with L_y be the target sentence length.

Figure 1.1 shows the encoder in blue, decoder in red and the attention mechanism in brown.

1.2 Transformer

The main difference between the Transformer model and an RNN-based NMT model is Transformer replaces the RNN with the self-attention mechanism as shown in figure 1.2. While I suggest readers to Vaswani et al. [81] for a more complete formulation, in this section I will briefly describe four main components of Transformer.

Multi-head Attention Given an input source sequence $\mathbf{x} = (x_1, \cdots, x_{L_s}), x \in \mathcal{R}^d$, we can pack this sequence into a matrix $X = \text{concat}(\mathbf{x}), X \in \mathcal{R}^{L_s \times d}$. The i-th attention head then computes a output matrix H_i as:

$$Q = XW_i^Q, K = XW_i^K, V = XW_i^V \tag{1.7}$$

$$H_i = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V \tag{1.8}$$

where $Q_i, \in \mathcal{R}^{d \times d_k}$, $K_i, \in \mathcal{R}^{d \times d_k}$, $V_i, \in \mathcal{R}^{d \times d_k}$ and $d_k < d$.

These matrices are then combined as:

$$H = \text{concat}(H_1, \cdots, H_n)W^O \tag{1.9}$$

Oftentimes, we set $d = nd_k$ so that there is no increase in both number of parameters or computations compared to using a single attention model. On the decoder, it works similarly but we also need to make sure it does not see the future. This can be done by simply setting the corresponding values within the softmax in equation 1.8 to $-\infty$. Since these two attention mechanisms allow encoder or decoder to attend to themselves, they are also called *self-attention*.

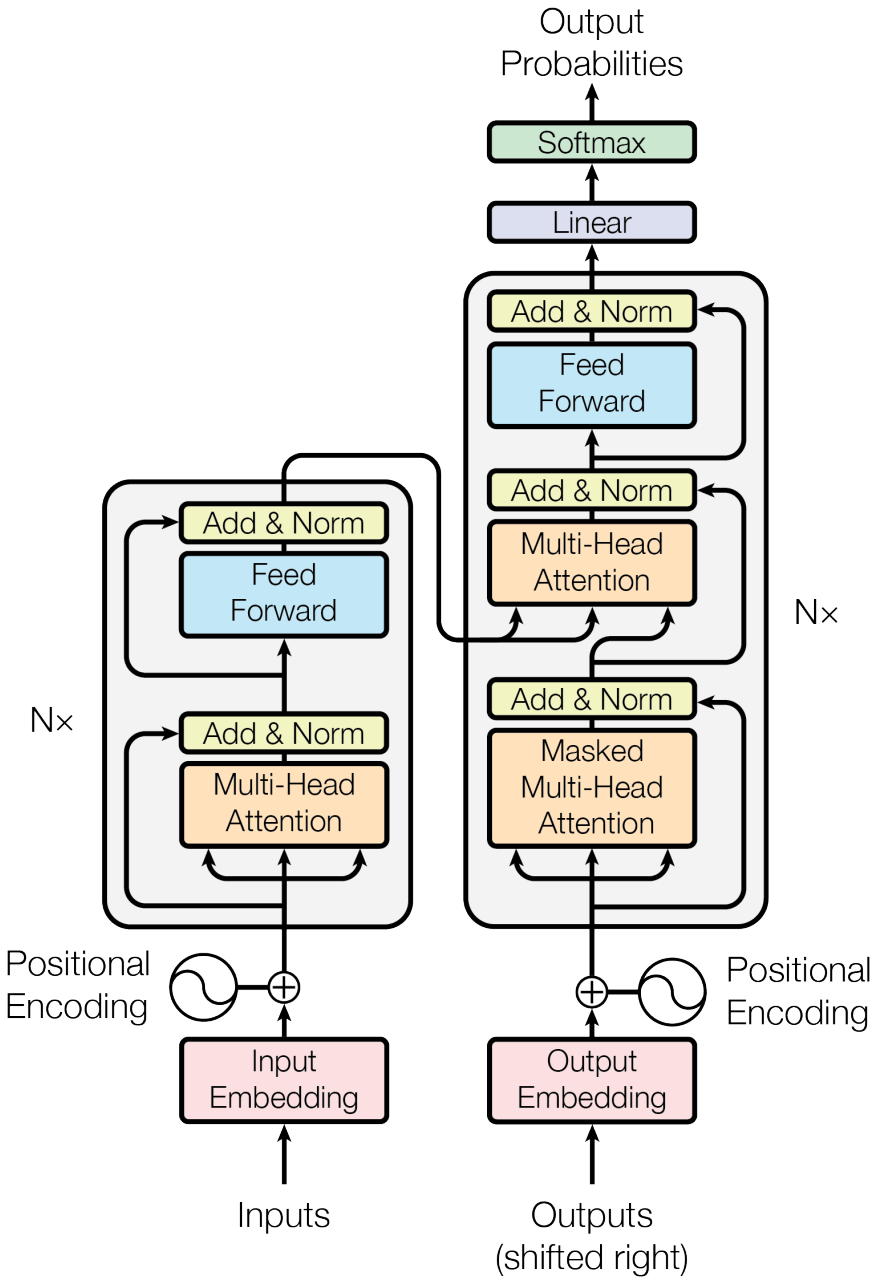


Figure 1.2: The Transformer model architecture [81]

At each decoder layer, the decoder also employs a similar attention mechanism to attend to encoder's output. One important change is:

$$Q = YW_i^Q \tag{1.10}$$

$$K = XW_i^K \tag{1.11}$$

$$V = XW_i^V \tag{1.12}$$

where X is the encoder's output, Y is the current decoder's output.

Feed-forwad networks The second important component of Transformer is the feed-forward networks. This is essentially two linear layers with RELU [17]:

$$FFN(X) = max(0, XW_1 + b_1)W_2 + b_2 \tag{1.13}$$

where $W_1 \in \mathcal{R}^{d \times d_{ff}}$, $W_w \in \mathcal{R}^{d_{ff} \times d}$, $b_1, b_2 \in \mathcal{R}^d_{ff}$, $d_{ff} > d$.

Layer Normalization and Residual Connection Residual connection [23] is employed at every sublayer of Transformer. This means the input to each multi-head attention or feed-forward layer is element-wise summed to their output. The output from each residual addition is then fed through a layer normalization layer [4]. The placement of the layer normalization is actually very important to Transformer's stability which we will discuss in section 3.3.

Positional Encoding To let Transformer aware of the order of the sequence, we use a cosine positional embedding which is defined as:

$$PE_{(pos,2i)} = \sin(\frac{pos}{10000^{2i/d}}) \tag{1.14}$$

$$PE_{(pos,2i+1)} = \cos(\frac{pos}{10000^{2i/d}}) \tag{1.15}$$

This positional embedding is simply element-wise summed with the word embedding before being fed into the encoder and decoder.

CHAPTER 2

OVERVIEW

Even though Bahdanau et al. [5] successfully train a NMT model that outperforms the PBMT system, they only experiment on a large dataset (348M tokens was used for training). Most languages in the world do not often fall in this high-resource category. For example, the LORELEI datasets often have only a few thousands to around 100k sentence pairs per language pair. Like other neural networks, NMT is known for being data-hungry which makes it difficult to train a good NMT system for such low-resource languages. In fact, Zoph et al. [93] show that out-of-the-box NMT systems perform significantly poorer for NMT in this domain. Most notably, Koehn and Knowles [39] quantitatively show that NMT needs up to around 100M tokens of training data to match PBMT's performance (figure 2.1). Digging deeper, Arthur et al. [3] points out that NMT often makes more mistakes for rare words than PBMT, which I will demonstrate in section 3.2 is a critical problem for low-resource languages.

In this document, I demonstrate that we can improve NMT performance for low-resource languages by **better data exploitation**, **better modeling** or **better normalization**. Specifically, I list my existing contributions:

better data exploitation: I show that we can improve NMT performance on a language pair using another related language pair by exploiting their similarities at subword level (section 3.1).

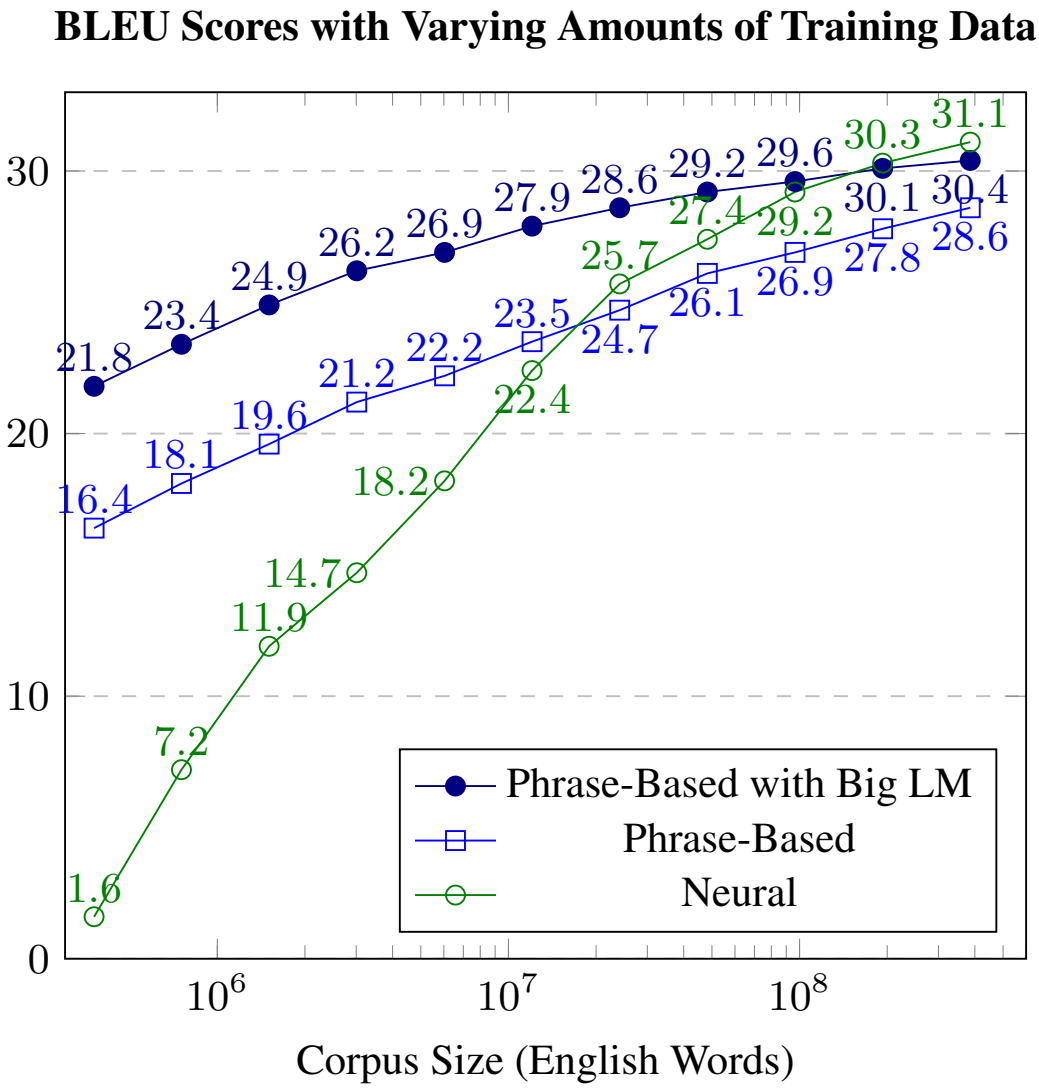


Figure 2.1: Performance of PBMT and NMT with different amount of data (Koehn and Knowles [38])

better modeling: I show that we can improve NMT by using a better neural network architecture. In particular, for RNN-based NMT, I propose a lexical module which can significantly alliviates the rare word mistranslation problem (section 3.2). For Transformer, I suggest a better placement of Layer Normalization and Residual Connection which helps to improve stability of training (section 3.3).

better normalization: I show that using simple ℓ_2 normalization, we can improve both the rare word mistranslation and training (section 3.2 and 3.3).

Additionally, I propose the next contributions:

- Improve NMT's modeling error which causes empty hypothesis is the best hypothesis [76] by informing the decoder the number of target words to translate in advance with the help of a length model (section 4.1).
- Improve multilingual NMT with shared-private encoder decoder which consists of shared components for all languages and private components which handle each language separately (section 4.2).

I do note that Transformer is a much stronger model than RNN-based NMT and was found to outperform PBMT out-of-the-box [44]. However, the first contribution regarding **better data exploitation** (section 3.1) is model-agnostic and I expect it to help with Transformer too. We will additionally see later on in section 3.3 how the **normalization** technique introduced in section 3.2 can be generalized and help improve Transformer as well. Finally, Transformer is not without its problems, and as mentioned above we will find out how some simple **modeling** changes can significantly improve Transformer's stability.

Thesis statement: *While NMT inherently requires large amount of data for training, with better data exploitation, better modeling and normalization we can successfully train strong NMT systems for low-resource languages.*

CHAPTER 3

EXISTING CONTRIBUTIONS

3.1 [Nguyen and Chiang, 2017] Transfer Learning across Low-Resource, Related Languages for Neural Machine Translation

Abstract *We present a simple method to improve neural translation of a low-resource language pair using parallel data from a related, also low-resource, language pair. The method is based on the transfer method of Zoph et al., but whereas their method ignores any source vocabulary overlap, ours exploits it. First, we split words using Byte Pair Encoding (BPE) to increase vocabulary overlap. Then, we train a model on the first language pair and transfer its parameters, including its source word embeddings, to another model and continue training on the second language pair. Our experiments show that transfer learning helps word-based translation only slightly, but when used on top of a much stronger BPE baseline, it yields larger improvements of up to 4.3 BLEU.*

3.1.1 Introduction

A common strategy to improve learning of low-resource languages is to use resources from related languages [55]. However, adapting these resources is not trivial. NMT offers some simple ways of doing this. For example, Zoph et al. [93] train a *parent* model on a (possibly unrelated) high-resource language pair, then use this model to initialize a *child* model which is further trained on a low-resource language pair. In particular, they showed that a French-English model could be used to improve translation on a wide range of low-resource language pairs such as Hausa-, Turkish-, and Uzbek-English.

In this section, we explore the opposite scenario, where the parent language pair is also low-resource, but related to the child language pair. We show that, at least in the case of three Turkic languages (Turkish, Uzbek, and Uyghur), the original method of Zoph et al. [93] does not always work, but it is still possible to use the parent model to considerably improve the child model.

The basic idea is to exploit the relationship between the parent and child language

lexicons. Zoph et al.'s original method makes no assumption about the relatedness of the parent and child languages, so it effectively makes a random assignment of the parent-language word embeddings to child-language words. But if we assume that the parent and child lexicons are related, it should be beneficial to transfer source word embeddings from parent-language words to their child-language equivalents.

Thus, the problem amounts to finding a representation of the data that ensures a sufficient overlap between the vocabularies of the languages. To do this, we map the source languages to a common alphabet and use Byte Pair Encoding (BPE) [74] on the union of the vocabularies to increase the number of common subwords.

In our experiments, we show that transfer learning helps word-based translation, but not always significantly. But when used on top of a much stronger BPE baseline, it yields larger and statistically significant improvements. Using Uzbek as a parent language and Turkish and Uyghur as child languages, we obtain improvements over BPE of 0.8 and 4.3 BLEU, respectively.

3.1.2 Method

The basic idea of our method is to extend the transfer method of Zoph et al. [93] to share the parent and child's source vocabularies, so that when source word embeddings are transferred, a word that appears in both vocabularies keeps its embedding. In order for this to work, it must be the case that the parent and child languages have considerable vocabulary overlap, and that when a word occurs in both languages, it often has a similar meaning in both languages. Thus, we need to process the data to make these two assumptions hold as much as possible.

3.1.2.1 Transliteration

If the parent and child language have different orthographies, it should help to map them into a common orthography. Even if the two use the same script, some transfor-

mation could be applied; for example, we might change French *-eur* endings to Spanish *-or*. Here, we take a minimalist approach. Turkish and Uzbek are both written using Latin script, and we did not apply any transformations to them. Our Uyghur data is written in Arabic script, so we transliterated it to Latin script using an off-the-shelf transliterator.[1] The transliteration is a string homomorphism, replacing Arabic letters with English letters or consonant clusters independent of context.

3.1.2.2 Segmentation

To increase the overlap between the parent and child vocabularies, we use BPE to break words into subwords. For the BPE merge rules to not only find the common subwords between two source languages but also ensure consistency between source and target segmentation among each language pair, we learn the rules from the union of source and target data of both the parent and child models. The rules are then used to segment the corpora. It is important to note that this results in a single vocabulary, used for both the source and target languages in both models.

	word-based		BPE 5k		BPE 60k	
model	toks	sents	toks	sents	toks	sents
Uzb par	1.5	102	2.4	92	1.9	103
Tur chd	0.9	56	1.5	50	1.2	57
Uzb par	1.5	102	2.4	90	2.0	103
Uyg chd	1.7	82	2.1	77	2.0	88

Table 3.1: Number of tokens ($\times 10^6$) and sentences ($\times 10^3$) in our training data.

3.1.3 Experiments

We used Turkish-, Uzbek-, and Uyghur-English parallel texts from the LORELEI program. We tokenized all data using the Moses toolkit [40]; for Turkish-English experiments, we also truecased the data. For Uyghur-English, the word-based models were trained in the original Uyghur data written in Arabic script; for BPE-based systems, we transliterated it to Latin script as described above.

For the word-based systems, we fixed the vocabulary size and replaced out-of-vocabulary words with `_UNK`. We tried different sizes for each language pair; however, each word-based system's target vocabulary size is limited by that of the child, so we could only use up to 45,000 word types for Turkish-English and 20,000 for Uyghur-English.

The BPE-based systems could make use of bigger vocabulary size thanks to the combination of both parent and child source and target vocabularies. We varied the number of BPE merge operations from 5,000 to 60,000. Instead of using a fixed vocabulary cutoff, we used the full vocabulary; to ensure the model still learns how to deal with unknown words, we trained on two copies of the training data: one unchanged, and one in which all rare words (whose frequency is less than 5) were replaced with `_UNK`. Accordingly, the number of epochs was halved.

Following common practice, we fixed an upper limit on training sentence length (discarding longer sentences). Because the BPE-based systems have shorter tokens and therefore longer sentences, we set this upper limit differently for the word-based and BPE-based systems to approximately equalize the total size of the training data. This led to a limit of 50 tokens for word-based models and 60 tokens for BPE-based models. See Table 3.1 for statistics of the resulting datasets.

We trained using Adadelta [90], with a minibatch size of 32 and dropout with a dropout rate of 0.2. We rescaled the gradient when its norm exceeded 5. For the Uzbek-English

[1] `https://cis.temple.edu/~anwar/code/latin2uyghur.html`

to Turkish-English experiment, the parent and child models were trained for 100 and 50 epochs, respectively. For the Uzbek-English to Uyghur-English experiment, the parent and child models were trained for 50 and 200, respectively. As mentioned above, the BPE models were trained for half as many epochs because their data is duplicated.

We used beam search for translation on the dev and test sets. Since NMT tends to favor short translations [9], we use the length normalization approach of Wu et al. [88] which uses a different score $s(e \mid f)$ instead of log-probability:

$$s(e \mid f) = \frac{\log p(e \mid f)}{lp(e)}$$
$$lp(e) = \frac{(5 + |e|)^\alpha}{(5 + 1)^\alpha}.$$

We set $\alpha = 0.8$ for all of our experiments.

We stopped training when the tokenized BLEU score was maximized on the development set. We also optimized the vocabulary size and the number of BPE operations for the word-based and BPE-based systems, respectively, to maximize the tokenized BLEU on the development set.

After translation at test time, we rejoined BPE segments, recased, and detokenized. Finally, we evaluated using case-sensitive BLEU.

As a baseline, we trained a child model using BPE but without transfer (that is, with weights randomly initialized). We also compared against a word-based baseline (without transfer) and two word-based systems using transfer without vocabulary-sharing, corresponding with the method of Zoph et al. [93]: one where the target word embeddings are fine-tuned, and one where they are frozen.

3.1.4 Results and Analysis

Our results are shown in Table 3.2. In this low-resource setting, we find that transferring word-based models does not consistently help. On Turkish-English, both transfer methods

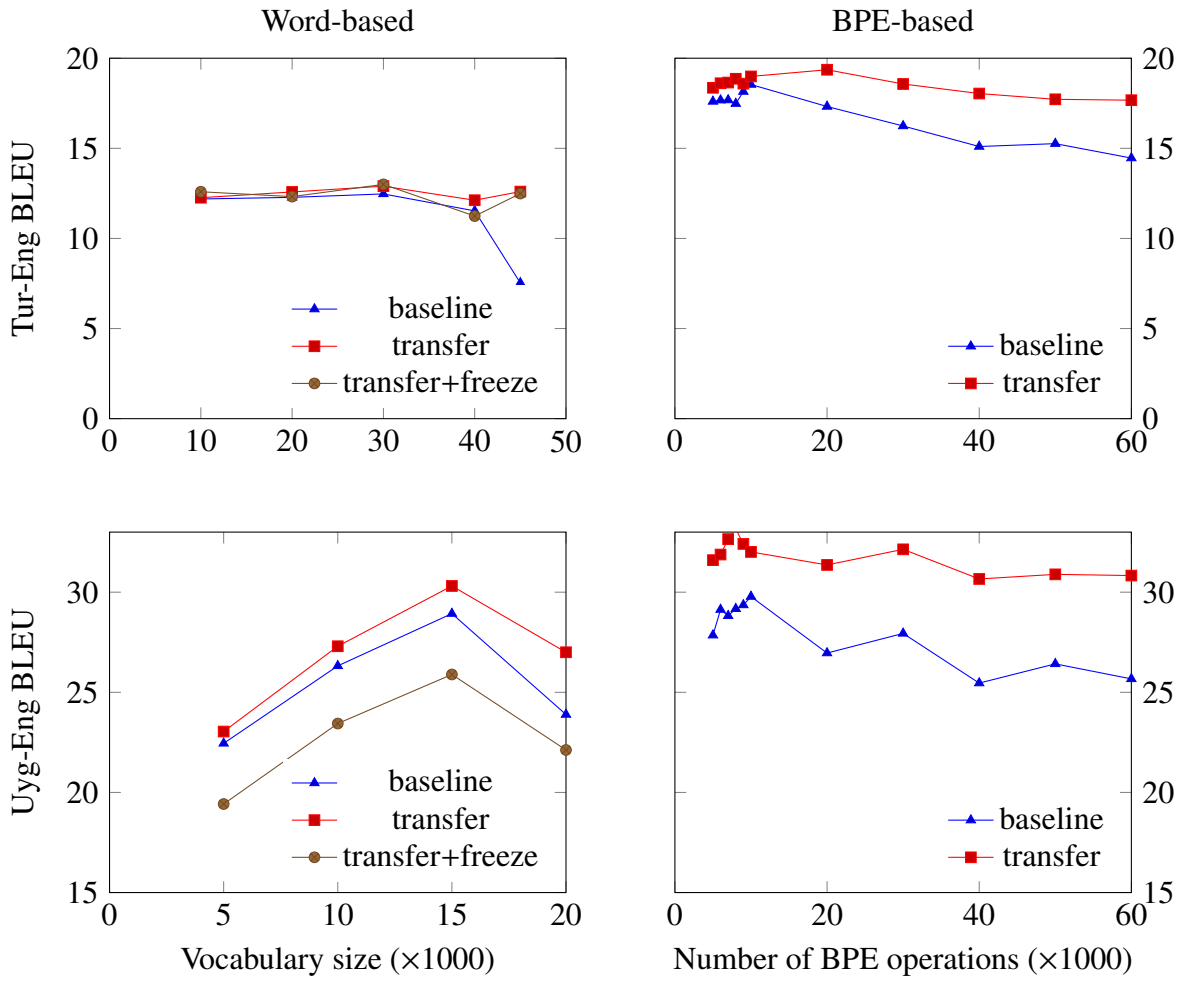


Figure 3.1: Tokenized **dev** BLEU scores for various settings as a function of the number of word/subword types. Key: baseline = train child model only; transfer = train parent, then child model; +freeze = freeze target word embeddings in child model.

		baseline		transfer		transfer+freeze	
		BLEU	size	BLEU	size	BLEU	size
Tur-Eng	word-based	8.1	30k	8.5*	30k	8.6*	30k
	BPE	12.4	10k	13.2†	20k	—	—
Uyg-Eng	word-based	8.5	15k	10.6†	15k	8.8*	15k
	BPE	11.1	10k	15.4‡	8k	—	—

Table 3.2: Whereas transfer learning at word-level does not always help, our method consistently yields a significant improvement over the stronger BPE systems. Scores are case-sensitive **test** BLEU. Key: `size` = vocabulary size (word-based) or number of BPE operations (BPE). The symbols † and ‡ indicate statistically significant improvements with $p < 0.05$ and $p < 0.01$, respectively, while $*$ indicates a statistically insignificant improvement ($p > 0.05$).

task	settings	train	dev
Tur-Eng	word-based 30k	3.9%	3.6%
	BPE 20k	58.8%	25.0%
Uyg-Eng	word-based 15k	0.5%	1.7%
	BPE 8k	57.2%	48.5%

Table 3.3: Amount of child's source types that appear in parent.

give only a statistically insignificant improvement ($p > 0.05$); on Uyghur-English, transfer without freezing target embeddings helps somewhat, but transfer with freezing helps only insignificantly.

In both language pairs, the models that use BPE perform much better than their word-based counterparts. When we apply transfer learning to this much stronger baseline, we find that the relative improvements actually increase; that is, the combined effect of BPE and transfer learning is more than additive. On Turkish-English, the improvement is +0.8 BLEU over the BPE baseline; on Uyghur-English, a healthy +4.3 over the BPE baseline.

A similar pattern emerges when we examine the best BLEU scores on the development set (Figure 3.1). Whereas word-based transfer methods help very little for Turkish-English,

and help or hurt slightly for Uyghur-English, our BPE-based transfer approach consistently improves over both the baseline and transfer word-based models. We surmise that the improvement is primarily due to the vocabulary overlap created by BPE (see Table 3.3).

3.2 [Nguyen and Chiang, 2018] Improving Lexical Choice in Neural Machine Translation

Abstract *We explore two solutions to the problem of mistranslating rare words in neural machine translation. First, we argue that the standard output layer, which computes the inner product of a vector representing the context with all possible output word embeddings, rewards frequent words disproportionately, and we propose to fix the norms of both vectors to a constant value. Second, we integrate a simple lexical module which is jointly trained with the rest of the model. We evaluate our approaches on eight language pairs with data sizes ranging from 100k to 8M words, and achieve improvements of up to +4.3 BLEU, surpassing phrase-based translation in nearly all settings.*

3.2.1 Introduction

Despite their competitive performance, there are still many open problems in NMT [39]. One particular issue is mistranslation of rare words. For example, consider the Uzbek sentence:

Source: Ammo muammolar hali ko'p, deydi amerikalik olim Entoni Fauchi.

Reference: But still there are many problems, says American scientist Anthony Fauci.

Baseline NMT: But there is still a lot of problems, says James Chan.

At the position where the output should be *Fauci*, the NMT model's top three candidates are *Chan*, *Fauci*, and *Jenner*. All three surnames occur in the training data with reference to immunologists: Fauci is the director of the National Institute of Allergy and Infectious Diseases, Margaret (not James) Chan is the former director of the World Health Organization, and Edward Jenner invented smallpox vaccine. But *Chan* is more frequent in the training data than *Fauci*, and *James* is more frequent than either *Anthony* or *Margaret*.

Because NMT learns word representations in continuous space, it tends to translate words that "seem natural in the context, but do not reflect the content of the source sen-

tence" [3]. This coincides with other observations that NMT's translations are often fluent but lack accuracy [86, 88].

Why does this happen? At each time step, the model's distribution over output words e is

$$p(e) \propto \exp\left(W_e \cdot \tilde{h} + b_e\right)$$

where W_e and b_e are a vector and a scalar depending only on e, and $\tilde{h}$ is a vector depending only on the source sentence and previous output words. We propose two modifications to this layer. First, we argue that the term $W_e \cdot \tilde{h}$, which measures how well e fits into the context $\tilde{h}$, favors common words disproportionately, and show that it helps to fix the norm of both vectors to a constant. Second, we add a new term representing a more direct connection from the source sentence, which allows the model to better memorize translations of rare words.

Below, we describe our models in more detail. Then we evaluate our approaches on eight language pairs, with training data sizes ranging from 100k words to 8M words, and show improvements of up to +4.3 BLEU, surpassing phrase-based translation in nearly all settings. Finally, we provide some analysis to better understand why our modifications work well[2].

3.2.2 Neural Machine Translation

Given a source sequence $f = f_1 f_2 \cdots f_m$, the goal of NMT is to find the target sequence $e = e_1 e_2 \cdots e_n$ that maximizes the objective function:

$$\log p(e \mid f) = \sum_{t=1}^{n} \log p(e_t \mid e_{<t}, f).$$

[2]Our code for this work can be found at `https://github.com/tnq177/improving_lexical_choice_in_nmt`

	ha-en	tu-en	hu-en
untied embeddings	17.2	11.5	26.5
tied embeddings	17.4	13.8	26.5
don't normalize $\tilde{h}_t$	18.6	14.2	27.1
normalize $\tilde{h}_t$	20.5	16.1	28.8

TABLE 3.4

PRELIMINARY EXPERIMENTS SHOW THAT TYING TARGET EMBEDDINGS WITH OUTPUT LAYER WEIGHTS PERFORMS AS WELL AS OR BETTER THAN THE BASELINE, AND THAT NORMALIZING $\tilde{h}$ IS BETTER THAN NOT NORMALIZING $\tilde{h}$. ALL NUMBERS ARE BLEU SCORES ON DEVELOPMENT SETS, SCORED AGAINST TOKENIZED REFERENCES.

The predicted probability distribution of the t'th target word is:

$$p(e_t \mid e_{<t}, f) = \text{softmax}(W^o \tilde{h}_t + b^o). \tag{3.1}$$

The rows of the output layer's weight matrix W^o can be thought of as embeddings of the output vocabulary, and sometimes are in fact tied to the embeddings in the input layer, reducing model size while often achieving similar performance [29, 67]. We verified this claim on some language pairs and found out that this approach usually performs better than without tying, as seen in Table 3.4. For this reason, we always tie the target embeddings and W^o in all of our models.

3.2.3 Normalization

The output word distribution (3.1) can be written as:

$$p(e) \propto \exp\left(\|W_e\| \, \|\tilde{h}\| \cos\theta_{W_e,\tilde{h}} + b_e\right),$$

where W_e is the embedding of e, b_e is the e'th component of the bias b^o, and $\theta_{W_e,\tilde{h}}$ is the angle between W_e and $\tilde{h}$. We can intuitively interpret the terms as follows. The term $\|\tilde{h}\|$ has the effect of sharpening or flattening the distribution, reflecting whether the model is more or less certain in a particular context. The cosine similarity $\cos\theta_{W_e,\tilde{h}}$ measures how well e fits into the context. The bias b_e controls how much the word e is generated; it is analogous to the language model in a log-linear translation model [59].

Finally, $\|W_e\|$ also controls how much e is generated. Figure 3.2 shows that it generally correlates with frequency. But because it is multiplied by $\cos\theta_{W_e,\tilde{h}}$, it has a stronger effect on words whose embeddings have direction similar to $\tilde{h}$, and less effect or even a negative effect on words in other directions. We hypothesize that the result is that the model learns $\|W_e\|$ that are disproportionately large.

For example, returning to the example from Section 3.2.1, these terms are:

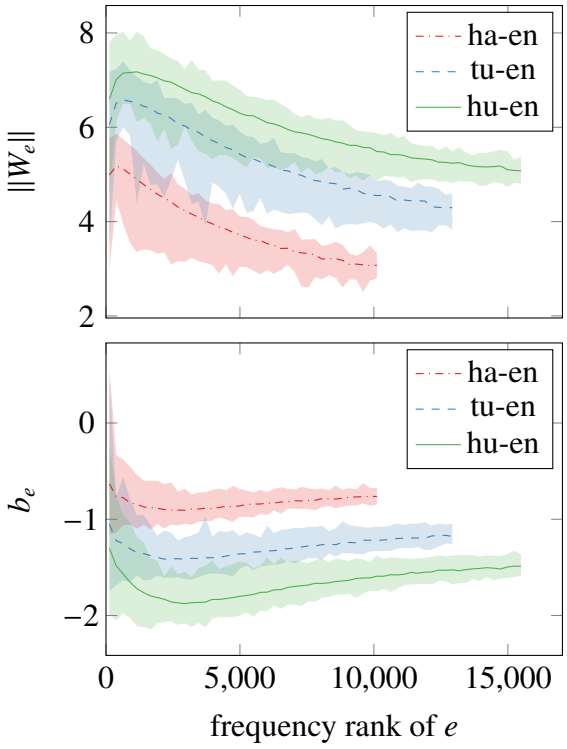


Figure 3.2. The word embedding norm $\|W_e\|$ generally correlates with the frequency of e, except for the most frequent words. The bias b_e has the opposite behavior. The plots show the median and range of bins of size 256.

e	$\|W_e\|$	$\|\tilde{h}\|$	$\cos\theta_{W_e,\tilde{h}}$	b_e	logit
Chan	5.25	19.5	0.144	−1.53	13.2
Fauci	4.69	19.5	0.154	−1.35	12.8
Jenner	5.23	19.5	0.120	−1.59	10.7

Observe that $\cos\theta_{W_e,\tilde{h}}$ and even b_e both favor the correct output word *Fauci*, whereas $\|W_e\|$ favors the more frequent, but incorrect, word *Chan*. The most frequently-mentioned immunologist trumps other immunologists.

To solve this issue, we propose to fix the norm of all target word embeddings to some value r. Followingthe weight normalization approach of Salimans and Kingma [71], we reparameterize W_e as $r\frac{v_e}{\|v_e\|}$, but keep r fixed.

A similar argument could be made for $\|\tilde{h}_t\|$: because a large $\|\tilde{h}_t\|$ sharpens the distribution, causing frequent words to more strongly dominate rare words, we might want to limit it as well. We compared both approaches on a development set and found that replacing $\tilde{h}_t$ in equation (3.1) with $r\frac{\tilde{h}_t}{\|\tilde{h}_t\|}$ indeed performs better, as shown in Table 3.4.

3.2.4 Lexical Translation

The attentional hidden state $\tilde{h}$ contains information not only about the source word(s) corresponding to the current target word, but also the contexts of those source words and the preceding context of the target word. This could make the model prone to generate a target word that fits the context but doesn't necessarily correspond to the source word(s). Count-based statistical models, by contrast, don't have this problem, because they simply don't model any of this context. Arthur et al. [3] try to alleviate this issue by integrating a count-based lexicon into an NMT system. However, this lexicon must be trained separately using GIZA++ [58], and its parameters form a large, sparse array, which can be difficult to store in GPU memory.

We propose instead to use a simple feedforward neural network (FFNN) that is trained jointly with the rest of the NMT model to generate a target word based directly on the source word(s). Let f_s ($s = 1, \ldots, m$) be the embeddings of the source words. We use the attention weights to form a weighted average of the embeddings (not the hidden states, as in the main model) to give an average source-word embedding at each decoding time step t:

$$f_t^\ell = \tanh \sum_s a_t(s) f_s.$$

Then we use a one-hidden-layer FFNN with skip connections [25]:

$$h_t^\ell = \tanh(W f_t^\ell) + f_t^\ell$$

and combine its output with the decoder output to get the predictive distribution over output

words at time step t:

$$p(y_t \mid y_{<t}, x) = \text{softmax}(W^o \tilde{h}_t + b^o + W^\ell h_t^\ell + b^\ell).$$

For the same reasons that were given in Section 3.2.3 for normalizing $\tilde{h}_t$ and the rows of W_t^o, we normalize h_t^ℓ and the rows of W^ℓ as well. Note, however, that we do not tie the rows of W^ℓ with the word embeddings; in preliminary experiments, we found this to yield worse results.

3.2.5 Experiments

We conducted experiments testing our normalization approach and our lexical model on eight language pairs using training data sets of various sizes. This section describes the systems tested and our results.

3.2.5.1 Data

We evaluated our approaches on various language pairs and datasets:

- Tamil (ta), Urdu (ur), Hausa (ha), Turkish (tu), and Hungarian (hu) to English (en), using data from the LORELEI program.
- English to Vietnamese (vi), using data from the IWSLT 2015 shared task.[3]
- To compare our approach with that of Arthur et al. [3], we also ran on their English to Japanese (ja) KFTT and BTEC datasets.[4]

We tokenized the LORELEI datasets using the default Moses tokenizer, except for Urdu-English, where the Urdu side happened to be tokenized using Morfessor FlatCat ($w = 0.5$). We used the preprocessed English-Vietnamese and English-Japanese datasets

[3] `https://nlp.stanford.edu/projects/nmt/`

[4] `http://isw3.naist.jp/~philip-a/emnlp2016/`

	tokens $\times 10^6$	vocab $\times 10^3$	layers num/size
ta-en	0.2/0.1	4.0/3.4	1/512
ur-en	0.2/0.2	4.2/4.2	1/512
ha-en	0.8/0.8	10.6/10.4	2/512
tu-en	0.8/1.1	21.1/13.3	2/512
uz-en	1.5/1.9	29.8/17.4	2/512
hu-en	2.0/2.3	27.3/15.7	2/512
en-vi	2.1/2.6	17.0/7.7	2/512
en-ja (BTEC)	3.6/5.0	17.8/21.8	4/768
en-ja (KFTT)	7.8/8.0	48.2/49.1	4/768

TABLE 3.5

STATISTICS OF DATA AND MODELS: EFFECTIVE NUMBER OF TRAINING SOURCE/TARGET TOKENS, SOURCE/TARGET VOCABULARY SIZES, NUMBER OF HIDDEN LAYERS AND NUMBER OF UNITS PER LAYER.

as distributed by Luong et al., and Arthur et al., respectively. Statistics about our data sets are shown in Table 3.5.

3.2.5.2 Systems

We compared our approaches against two baseline NMT systems:

untied, which does not tie the rows of W_o to the target word embeddings, and

tied, which does.

In addition, we compared against two other baseline systems:

Moses: The Moses phrase-based translation system [40], trained on the same data as the NMT systems, with the same maximum sentence length of 50. No additional data was used for training the language model. Unlike the NMT systems, Moses used the full vocabulary from the training data; unknown words were copied to the target sentence.

Arthur: Our reimplementation of the discrete lexicon approach of Arthur et al. [3]. We only tried their `auto` lexicon, using GIZA++ [58], integrated using their `bias` approach. Note that we also tied embedding as we found it also helped in this case.

Against these baselines, we compared our new systems:

fixnorm: The normalization approach described in Section 3.2.3.

fixnorm+lex: The same, with the addition of the lexical translation module from Section 3.2.4.

3.2.5.3 Details

Model We use the *global attentional* model with *general scoring function* and *input feeding* by Luong et al. [50]. Following their practice, we fed the source sentences to the encoder in reverse order during both training and testing. Information about the number and

size of hidden layers is shown in Table 3.5. The word embedding size is always equal to the hidden layer size.

Following common practice, we only trained on sentences of 50 tokens or less. We limited the vocabulary to word types that appear no less than 5 times in the training data and map the rest to `UNK`. For the English-Japanese and English-Vietnamese datasets, we used the vocabulary sizes reported in their respective papers [3, 48].

For **fixnorm**, we tried $r \in \{3, 5, 7\}$ and selected the best value based on the development set performance, which was $r = 5$ except for English-Japanese (BTEC), where $r = 7$. For **fixnorm+lex**, because $W_s\tilde{h}_t + W^\ell h_t^\ell$ takes on values in $[-2r^2, 2r^2]$, we reduced our candidate r values by roughly a factor of $\sqrt{2}$, to $r \in \{2, 3.5, 5\}$. A radius $r = 3.5$ seemed to work the best for all language pairs.

Training We trained all NMT systems with Adadelta [90]. All parameters were initialized uniformly from $[-0.01, 0.01]$. When a gradient's norm exceeded 5, we normalized it to 5. We also used dropout on non-recurrent connections only [89], with probability 0.2. We used minibatches of size 32. We trained for 50 epochs, validating on the development set after every epoch, except on English-Japanese, where we validated twice per epoch. We kept the best checkpoint according to its BLEU on the development set.

Inference We used beam search with a beam size of 12 for translating both the development and test sets. Since NMT often favors short translations [9], we followed Wu et al. [88] in using a modified score $s(e \mid f)$ in place of log-probability:

$$s(e \mid f) = \frac{\log p(e \mid f)}{lp(e)}$$
$$lp(e) = \frac{(5 + |e|)^\alpha}{(5 + 1)^\alpha}$$

We set $\alpha = 0.8$ for all of our experiments.

Finally, we applied a postprocessing step to replace each UNK in the target translation with the source word with the highest attention score [51].

Evaluation For translation into English, we report case-sensitive NIST BLEU against detokenized references. For English-Japanese and English-Vietnamese, we report tokenized, case-sensitive BLEU following Arthur et al. [3] and Luong and Manning [48]. We measure statistical significance using bootstrap resampling [37].

3.2.6 Results and Analysis

3.2.6.1 Overall

Our results are shown in Table 3.6. First, we observe, as has often been noted in the literature, that NMT tends to perform poorer than PBMT on low resource settings (note that the rows of this table are sorted by training data size).

Our **fixnorm** system alone shows large improvements (shown in parentheses) relative to **tied**. Integrating the lexical module (**fixnorm+lex**) adds in further gains. Our **fixnorm+lex** models surpass Moses on all tasks except Urdu- and Hausa-English, where it is 1.6 and 0.7 BLEU short respectively.

The method of Arthur et al. [3] does improve over the baseline NMT on most language pairs, but not by as much and as consistently as our models, and often not as well as Moses. Unfortunately, we could not replicate their approach for English-Japanese (KFTT) because the lexical table was too large to fit into the computational graph.

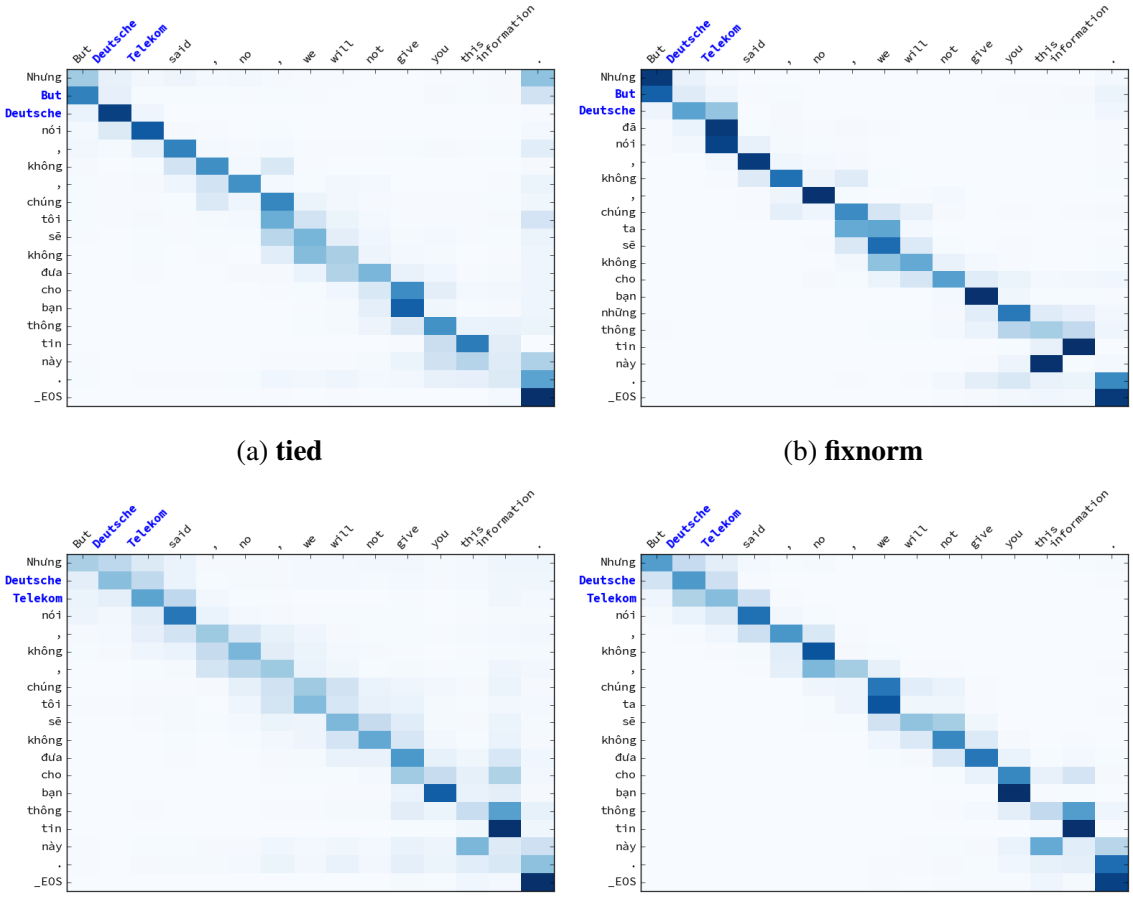
For English-Japanese (BTEC), we note that, due to the small size of the test set, all systems except for Moses are in fact not significantly different from **tied** ($p > 0.01$). On all other tasks, however, our systems significantly improve over **tied** ($p < 0.01$).

	untied	**tied**	**fixnorm**	**fixnorm+lex**	**Moses**	**Arthur**
ta-en	10.3	11.1	14 (+2.9)	15.3 (+4.2)	10.5 (−0.6)	14.1 (+3.0)
ur-en	7.9	10.7	12 (+1.3)	13 (+2.3)	14.6 (+3.9)	12.5 (+1.8)
ha-en	16.0	16.6	20 (+3.4)	21.5 (+4.9)	22.2 (+5.6)	18.7 (+2.1)
tu-en	12.2	12.6	16.4 (+3.8)	19.1 (+6.5)	18.1 (+5.5)	16.3 (+3.7)
uz-en	14.9	15.7	18.2 (+2.5)	19.3 (+3.6)	17.2 (+1.5)	17.1 (+1.4)
hu-en	21.6	23.0	24.0 (+1.0)	25.3 (+2.3)	21.3 (−1.7)	22.7 (-0.3)†
en-vi	25.1	25.3	26.8 (+1.5)	27 (+1.7)	26.7 (+1.4)	26.2 (+0.9)
en-ja (BTEC)	51.2	53.7	52.9 (-0.8)†	51.3 (−2.6)†	46.8 (−6.9)	52.4 (−1.3)†
en-ja (KFTT)	24.1	24.5	26.1 (+1.6)	26.2 (+1.7)	21.7 (−2.8)	—

Table 3.6: Test BLEU of all models. Differences shown in parentheses are relative to **tied**, with a dagger (†) indicating an *insignificant* difference in BLEU ($p > 0.01$). While the method of Arthur et al. [3] does not always help, **fixnorm** and **fixnorm+lex** consistently achieve significant improvements over **tied** ($p < 0.01$) except for English-Japanese (BTEC). Our models also outperform the method of Arthur et al. on all tasks and outperform Moses on all tasks but Urdu-English and Hausa-English.

input	Dushanba kuni Hindistonda kamida **34** kishi halok bo'lgani xabar qilindi .
reference	At least **34** more deaths were reported Monday in India .
untied	At least **UNK** people have died in India on Monday .
tied	It was reported that at least **700** people died in Monday .
fixnorm	At least **34** people died in India on Monday .
fixnorm+lex	At least **34** people have died in India on Monday .
input	Yarın **Kenya'da** bir yardım konferansı düzenlenecek .
reference	Tomorrow a conference for aid will be conducted in **Kenya** .
untied	Tomorrow there will be an **Afghan** relief conference .
tied	Tomorrow there will be a relief conference in **Myanmar** .
fixnorm	Tomorrow it will be a aid conference in **Kenya** .
fixnorm+lex	Tomorrow there will be a relief conference in **Kenya** .
input	Ammo muammolar hali ko'p , deydi amerikalik olim **Entoni Fauchi** .
reference	But still there are many problems , says American scientist **Anthony Fauci** .
untied	But there is still a lot of problems , says **James Chan** .
tied	However , there is still a lot of problems , says American scientists .
fixnorm	But there is still a lot of problems , says American scientist **UNK UNK** .
fixnorm+lex	But there are still problems , says American scientist **Anthony Fauci** .

Table 3.7: Example translations, in which **untied** and **tied** generate incorrect, but often semantically related, words, but **fixnorm** and/or **fixnorm+lex** generate the correct ones.



(a) **tied** (b) **fixnorm**

(c) **fixnorm+lex** (d) Arthur et al. [3]

Figure 3.3: While the **tied** and **fixnorm** systems shift attention to the left one word (on the source side), our **fixnorm+lex** model and that of Arthur et al. [3] put it back to the correct position, improving unknown-word replacement for the words *Deutsche Telekom*. Columns are source (English) words and rows are target (Vietnamese) words. Bolded words are unknown.

3.2.6.2 Impact on translation

In Table 3.7, we show examples of typical translation mistakes made by the baseline NMT systems. In the Uzbek example (top), **untied** and **tied** have confused *34* with `UNK` and *700*, while in the Turkish one (middle), they incorrectly output other proper names, *Afghan* and *Myanmar*, for the proper name *Kenya*. Our systems, on the other hand, translate these words correctly.

The bottom example is the one introduced in Section 3.2.1. We can see that our **fixnorm** approach does not completely solve the mistranslation issue, since it translates *Entoni Fauchi* to `UNK UNK` (which is arguably better than *James Chan*). On the other hand, **fixnorm+lex** gets this right. To better understand how the lexical module helps in this case, we look at the top five translations for the word *Fauci* in **fixnorm+lex**:

e	$\cos\theta_{W_e,\tilde{h}}$	$\cos\theta_{W^l_e,h_l}$	$b_e + b^l_e$	logit
Fauci	0.522	0.762	−8.71	7.0
UNK	0.566	−0.009	−1.25	5.6
Anthony	0.263	0.644	−8.70	2.4
Ahmedova	0.555	0.173	−8.66	0.3
Chan	0.546	0.150	−8.73	−0.2

As we can see, while $\cos\theta_{W_e,\tilde{h}}$ might still be confused between similar words, $\cos\theta_{W^l_e,h_l}$ significantly favors *Fauci*.

3.2.6.3 Alignment and unknown words

Both our baseline NMT and **fixnorm** models suffer from the problem of shifted alignments noted by Koehn and Knowles [39]. As seen in Figure 3.3a and 3.3b, the alignments for those two systems seem to shift by one word to the left (on the source side). For example, *nói* should be aligned to *said* instead of *Telekom*, and so on. Although this is not a problem *per se*, since the decoder can decide to attend to any position in the encoder states

as long as the state at that position holds the information the decoder needs, this becomes a real issue when we need to make use of the alignment information, as in unknown word replacement [51]. As we can see in Figure 3.3, because of the alignment shift, both **tied** and **fixnorm** incorrectly replace the two unknown words (in bold) with *But Deutsche* instead of *Deutsche Telekom*. In contrast, under **fixnorm+lex** and the model of Arthur et al. [3], the alignment is corrected, causing the `UNK`s to be replaced with the correct source words.

3.2.6.4 Impact of r

The single most important hyper-parameter in our models is r. Informally speaking, r controls how much surface area we have on the hypersphere to allocate to word embeddings. To better understand its impact, we look at the training perplexity and dev BLEUs during training with different values of r. Table 3.8 shows the train perplexity and best tokenized dev BLEU on Turkish-English for **fixnorm** and **fixnorm+lex** with different values of r. As we can see, a smaller r results in worse training perplexity, indicating underfitting, whereas if r is too large, the model achieves better training perplexity but decrased dev BLEU, indicating overfitting.

3.2.6.5 Lexicon

One byproduct of **lex** is the lexicon, which we can extract and examine simply by feeding each source word embedding to the FFNN module and calculating $p^\ell(y) = \text{softmax}(W^\ell h^\ell + b_\ell)$. In table 3.9, we show the top translations for some entries in the lexicons extracted from **fixnorm+lex** for Hungarian, Turkish, and Hausa-English. As expected, the lexical distribution is sparse, with a few top translations accounting for the most probability mass.

3.2.6.6 Byte Pair Encoding

Byte-Pair-Encoding (BPE) [74] is commonly used in NMT to break words into word-pieces, improving the translation of rare words. For this reason, we reran our experiments

system	r	train ppl	dev BLEU
fixnorm	3	3.9	13.6
	5	2.5	16.1
	7	2.3	14.4
fixnorm+lex	2	4.2	12.3
	3.5	2.0	17.5
	5	1.4	16.0

TABLE 3.8

WHEN r IS TOO SMALL, HIGH TRAIN PERPLEXITY AND LOW DEV BLEU INDICATE UNDERFITTING; WHEN r IS TOO LARGE, LOW TRAIN PERPLEXITY AND LOW DEV BLEU INDICATE OVERFITTING.

hu-en	244 befektetéseinek kutatás-fejlesztésre	244 (0.599) document (0.005) By (0.003) investments (0.151) investment (0.017) Investments (0.015) research (0.227) Research (0.040) Development (0.014)
tu-en	ifade cumhurbaşkanı Göstericiler	expression (0.109) expressed (0.061) express (0.056) President (0.573) president (0.030) Republic (0.027) protesters (0.115) demonstrators (0.050) Protesters (0.033)
ha-en	😒 Wayoyin manzonsa	😒 (0.469) cholera (0.003) `EOS` (0.001) phones (0.414) wires (0.097) mobile (0.088) Prophet (0.080) His (0.041) Messenger (0.015)

Table 3.9: Top five translations for some entries of the lexical tables extracted from **fixnorm+lex**. Probabilities are shown in parentheses.

	tied	fixnorm	fixnorm+lex	
ta-en	13	15 (+2.0)	15.9	(+2.9)
ur-en	10.5	12.3 (+1.8)	13.7	(+3.2)
ha-en	18	21.7 (+3.7)	22.3	(+4.3)
tu-en	19.3	21 (+1.7)	22.2	(+2.9)
uz-en	18.9	19.8 (+0.9)	21	(+2.1)
hu-en	25.8	27.2 (+1.4)	27.9	(+2.1)
en-vi	26.3	27.3 (+1.0)	27.5	(+1.2)
en-de (newstest2014)	19.7	22.2 (+2.5)	20.4	(+0.7)
en-de (newstest2015)	22.5	25 (+2.5)	23.2	(+0.7)

Table 3.10: Test BLEU for all BPE-based systems. Our models significantly improve over the baseline ($p < 0.01$) for both high and low resource when using BPE.

using BPE on the LORELEI and English-Vietnamese datasets. Additionally, to see if our proposed methods work in high-resource scenarios, we run on the WMT 2014 English-German (en-de) dataset,[5] using *newstest2013* as the development set and reporting tokenized, case-sensitive BLEU on *newstest2014* and *newstest2015*.

We validate across different numbers of BPE operations; specifically, we try {1k, 2k, 3k} merge operations for ta-en and ur-en due to their small sizes, {10k, 12k, 15k} for the other LORELEI datasets and en-vi, and 32k for en-de. Using BPE results in much smaller vocabulary sizes, so we do not apply a vocabulary cut-off. Instead, we train on an additional copy of the training data in which all types that appear once are replaced with `UNK`, and halve the number of epochs accordingly. Our models, training, and evaluation processes are largely the same, except that for en-de, we use a 4-layer decoder and 4-layer bidirectional encoder (2 layers for each direction).

Table 3.10 shows that our proposed methods also significantly improve the translation when used with BPE, for both high and low resource language pairs. With BPE, we are only behind Moses on Urdu-English.

[5] `https://nlp.stanford.edu/projects/nmt/`

3.2.7 Related Work

The closest work to our **lex** model is that of Arthur et al. [3], which we have discussed already in Section 3.2.4. Recent work by Liu et al. [45] has very similar motivation to that of our **fixnorm** model. They reformulate the output layer in terms of directions and magnitudes, as we do here. Whereas we have focused on the magnitudes, they focus on the directions, modifying the loss function to try to learn a classifier that separates the classes' directions with something like a margin. Wang et al. [84] also make the same observation that we do for the **fixnorm** model, but for the task of face verification.

Handling rare words is an important problem for NMT that has been approached in various ways. Some have focused on reducing the number of `UNK`s by enabling NMT to learn from a larger vocabulary [31, 52]; others have focused on replacing `UNK`s by copying source words [19, 18, 51]. However, these methods only help with unknown words, not rare words. An approach that addresses both unknown and rare words is to use subword-level information [74, 11, 49]. Our approach is different in that we try to identify and address the root of the rare word problem. We expect that our models would benefit from more advanced `UNK`-replacement or subword-level techniques as well.

Recently, Liu and Kirchhoff [44] have shown that their baseline NMT system with BPE already outperforms Moses for low-resource translation. However, in their work, they use the Transformer network [81], which is quite different from our baseline model. It would be interesting to see if our methods benefit the Transformer network and other models as well.

3.2.8 Conclusion

In this section, we have presented two simple yet effective changes to the output layer of a NMT model. Both of these changes improve translation quality substantially on low-resource language pairs. In many of the language pairs we tested, the baseline NMT system performs poorly relative to phrase-based translation, but our system surpasses it (when

both are trained on the same data). We conclude that NMT, equipped with the methods demonstrated here, is a more viable choice for low-resource translation than before, and are optimistic that NMT's repertoire will continue to grow.

3.3 [Nguyen and Salazar, 2018] Transformers without Tears: Improving the Normalization of Self-Attention

Abstract *We evaluate three simple, normalization-centric changes to improve Transformer training. First, we show that pre-norm residual connections (*PreNorm*) and smaller initializations enable warmup-free, validation-based training with large learning rates. Second, we propose ℓ_2 normalization with a single scale parameter (*ScaleNorm*) for faster training and better performance. Finally, we reaffirm the effectiveness of normalizing word embeddings to a fixed length (*FixNorm*). On five low-resource translation pairs from TED Talks-based corpora, these changes always converge, giving an average +1.1 BLEU over state-of-the-art bilingual baselines and a new 32.8 BLEU on IWSLT'15 English-Vietnamese. We observe sharper performance curves, more consistent gradient norms, and a linear relationship between activation scaling and decoder depth. Surprisingly, in the high-resource setting (WMT'14 English-German),* ScaleNorm *and* FixNorm *remain competitive but* PreNorm *degrades performance.*

3.3.1 Introduction

The Transformer [80] has become the dominant architecture for neural machine translation (NMT) due to its train-time parallelism and strong downstream performance. Various modifications have been proposed to improve the efficiency of its multi-head attention and feedforward sublayers [20, 77]. Our work focuses on *layer normalization* (LayerNorm) [4], which we show has an outsized role in the convergence and performance of the Transformer in two ways:

Placement of normalization. The original Transformer uses *post-norm residual units* (PostNorm), where layer normalization occurs after the sublayer and residual addition. However, Chen et al. [8] found that *pre-norm residual units* (PreNorm), where layer normalization occurs immediately before the sublayer, were instrumental to their model's per-

formance. Wang et al. [85] compare the two, showing that PreNorm makes backpropagation more efficient over depth and training Transformers with deep, 30-layer encoders.

Our work demonstrates additional consequences in the base (≤6-layer encoder) Transformer regime. We show that PreNorm enables warmup-free, validation-based training with large learning rates even for small batches, in contrast to past work on scaling NMT [60]. We also partly reclaim PostNorm's stability via smaller initializations, although PreNorm is less sensitive to this magnitude and can improve performance. However, despite PreNorm's recent adoption in many NMT frameworks, we find it degrades base Transformer performance on WMT'14 English-German.

Choice of normalization. Santurkar et al. [72] show that batch normalization's effectiveness is not from reducing internal covariate shift, but from smoothing the loss landscape. They achieve similar or better performance with non-variance-based normalizations in image classification. Hence, we propose replacing LayerNorm with the simpler *scaled* ℓ_2 *normalization* (ScaleNorm), which normalizes activation vectors to a *single* learned length g. This is both inspired by and synergistic with jointly fixing the word embedding lengths (FixNorm) [57]. These changes improve the training speed and low-resource performance of the Transformer without affecting high-resource performance.

On five low-resource pairs from the TED Talks [68] and IWSLT'15 [7] corpora, we first train state-of-the-art Transformer models (+4.0 BLEU on average over the best published NMT bitext-only numbers). We then apply PreNorm, FixNorm, and ScaleNorm for an average total improvement of +1.1 BLEU, where each addition contributes at least +0.3 BLEU (Section 3.3.3), and attain a new 32.8 BLEU on IWSLT'15 English-Vietnamese. We validate our intuitions in Section 3.3.4 by showing sharper performance curves (i.e., improvements occur at earlier epochs) and more consistent gradient norms. We also examine the per-sublayer g's learned by ScaleNorm, which suggest future study[6].

[6]Reimplementation available at `https://github.com/tnq177/transformers_without_tears`

3.3.2 Background

3.3.2.1 Identity mappings for transformers

Residual connections [23] were first introduced to facilitate the training of deep convolutional networks, where the output of the ℓ-th layer F_ℓ is summed with its input:

$$\boldsymbol{x}_{\ell+1} = \boldsymbol{x}_\ell + F_\ell(\boldsymbol{x}_\ell). \tag{3.2}$$

The identity term $\boldsymbol{x}_\ell$ is crucial to greatly extending the depth of such networks [24]. If one were to scale $\boldsymbol{x}_\ell$ by a scalar λ_ℓ, then the contribution of $\boldsymbol{x}_\ell$ to the final layer F_L is $(\prod_{i=\ell}^{L-1} \lambda_i)\boldsymbol{x}_\ell$. For deep networks with dozens or even hundreds of layers L, the term $\prod_{i=\ell}^{L-1} \lambda_i$ becomes very large if $\lambda_i > 1$ or very small if $\lambda_i < 1$, for enough i. When backpropagating from the last layer L back to ℓ, these multiplicative terms can cause exploding or vanishing gradients, respectively. Therefore they fix $\lambda_i = 1$, keeping the total residual path an identity map.

The original Transformer applies LayerNorm after the sublayer and residual addition (PostNorm):

$$\boldsymbol{x}_{\ell+1} = \text{LayerNorm}(\boldsymbol{x}_\ell + F_\ell(\boldsymbol{x}_\ell)). \tag{3.3}$$

We conjecture this has caused past convergence failures [65, 75], with LayerNorms in the residual path acting similarly to $\lambda_i \neq 1$; furthermore, warmup was needed to let LayerNorm safely adjust scale during early parts of training. Inspired by He et al. [24], we apply LayerNorm immediately before each sublayer (PreNorm):

$$\boldsymbol{x}_{\ell+1} = \boldsymbol{x}_\ell + F_\ell(\text{LayerNorm}(\boldsymbol{x}_\ell)). \tag{3.4}$$

This is cited as a stabilizer for Transformer training [8, 85] and is already implemented in popular toolkits [82, 61, 27], though not necessarily used by their default recipes. Wang et al. [85] make a similar argument to motivate the success of PreNorm in training very

deep Transformers. Note that one must append an additional normalization after both encoder and decoder so their outputs are appropriately scaled. We compare PostNorm and PreNorm throughout Section 3.3.3.

3.3.2.2 Weight initialization

Xavier normal initialization [16] initializes a layer's weights $\boldsymbol{W}_\ell \in \mathbb{R}^{d_{\ell+1} \times d_\ell}$ (d_ℓ is the hidden dimension) with samples from a centered normal distribution with layer-dependent variance:

$$(\boldsymbol{W}_\ell)_{i,j} \sim \mathcal{N}\left(0, \sqrt{\frac{2}{d_\ell + d_{\ell+1}}}\right). \tag{3.5}$$

Our experiments with this default initializer find that PostNorm sometimes fails to converge, especially in our low-resource setting, even with a large number of warmup steps. One explanation is that Xavier normal yields initial weights that are too large. In implementations of the Transformer, one scales the word embeddings by a large value (e.g., $\sqrt{d} \approx 22.6$ for $d = 512$), giving vectors with an expected square norm of d. LayerNorm's unit scale at initialization preserves this same effect. Since feedforward layers already have their weights initialized to a smaller standard deviation, i.e., $\sqrt{\frac{2}{d+4d}}$, we propose reducing the attention layers' initializations from $\sqrt{\frac{2}{d+d}}$ to $\sqrt{\frac{2}{d+4d}}$ as well (SmallInit), as a corresponding mitigation. We evaluate the effect of this on PostNorm vs. PreNorm in Section 3.3.3.2.

3.3.2.3 Scaled ℓ_2 normalization and FixNorm

LayerNorm is inspired by batch normalization [30], both of which aim to reduce internal covariate shift by fixing the mean and variance of activation distributions. Both have been applied to self-attention [80, 41]. However, Santurkar et al. [72] show that batch normalization's success has little to do with covariate shift, but comes instead from smoothing the loss landscape. For example, they divide by the pre-centered ℓ_p norm instead of the

variance and achieve similar or better results in image classification.

Hence, we propose replacing LayerNorm with *scaled ℓ_2 normalization*:

$$\textsc{ScaleNorm}(\boldsymbol{x}; g) = g\frac{\boldsymbol{x}}{||\boldsymbol{x}||}. \tag{3.6}$$

This can be viewed as projecting d-dimensional vectors onto a $(d-1)$-dimensional hypersphere with learned radius g. This expresses the inductive bias that each sublayer's activations has an ideal "global scale," a notion we empirically validate in Section 3.3.4.2. ScaleNorm replaces the $2d$ scale and shift parameters of LayerNorm with a single learned scalar, improving computational and parameter efficiency while potentially regularizing the loss landscape.

This bias has an explicit interpretation at the final layer: large inner products sharpen the output distribution, causing frequent words to disproportionately dominate rare words. This led Nguyen and Chiang [57] to introduce $\textsc{FixNorm}(\boldsymbol{w}) = g\frac{\boldsymbol{w}}{||\boldsymbol{w}||}$ with fixed g at the last linear layer, to maximize the angular difference of output representations and aid rare word translation. By making g learnable, we can apply ScaleNorm and FixNorm jointly, which means applying the following at the final linear layer:

$$\begin{aligned}&(\textsc{ScaleNorm+FixNorm})(\boldsymbol{x}, \boldsymbol{w}; g)\\ &\qquad = g\frac{\boldsymbol{w}\cdot\boldsymbol{x}}{||\boldsymbol{w}||||\boldsymbol{x}||}.\end{aligned} \tag{3.7}$$

Note that this combination at the last layer is equivalent to cosine normalization [47] with a learned scale.

3.3.2.4 Learning rates

Despite using an adaptive optimizer, Adam [35], Transformer training uses a learning rate (LR) schedule with a linear *warmup* and an inverse square root *decay* (InvSqrtDecay):

	# egs.	# src + tgt toks.	# iters/epoch	max epoch	# enc/dec layers	# heads/layer	dropout	# BPE
gl→en	10k	0.37M	100	1000	4	4	0.4	3k
sk→en	61k	2.32M	600	200	6	8	0.3	8k
en→vi	133k	5.99M	1500	200	6	8	0.3	8k
en→he	212k	7.88M	2000	200	6	8	0.3	8k
ar→en	214k	8.09M	2000	200	6	8	0.3	8k

Table 3.11: Data and model properties for low-resource NMT. *en→vi* is from IWSLT 2015; the rest are from the TED Talks corpus.

$$\mathrm{LR}(n) = \frac{\lambda}{\sqrt{d}} \min\left(\frac{1}{\sqrt{n}}, \frac{n}{n_{\text{warmup}}^{1.5}}\right), \tag{3.8}$$

where d is the hidden dimension of the self-attention layers, and λ, n_{warmup} are hyperparameters that determine the highest learning rate achieved and the number of steps to reach it, respectively. These two hyperparameters have been the subject of much empirical study [65, 60]. In light of our modifications however, we revisit various aspects of this schedule:

Warmup-free training. We conjectured that warmup is primarily needed when using PostNorm to gradually learn LayerNorm parameters without gradient explosion/vanishing (Section 3.3.2.1). Hence, we evaluate both PreNorm and PostNorm without warmup in Section 3.3.3.4.

Large learning rates. To speed up training, one often explores using larger learning rates. In the context of Transformer, Ott et al. [60] and Aharoni et al. [1] take $\lambda \in \{2, 3\}$ instead of the conventional $\lambda = 1$. Ott et al. [60] showed that one can scale up Adam's learning rate to 10^{-3} with an extremely large batch (400k tokens). However, the improved convergence provided by our modifications could enable higher learning rates with much small batch sizes (4k tokens), as examined in Section 3.3.3.4.

Validation-based decay. For similar reasons, one might wish to adopt a classic validation-based decay, i.e., training at a high learning rate for as long as tenable, decaying rapidly

when development scores flatline. This has inspired usage of fixed decay schemes upon convergence with InvSqrtDecay [13, 70]. We revisit ValDecay under our modifications, where we still perform a linear warmup but then multiply by a scale $\alpha_{\text{decay}} < 1$ when performance on a development set does not improve over *patience* evaluations.

3.3.3 Experiments and results

We train Transformer models for a diverse set of five low-resource translation pairs from the TED Talks [68] and the IWSLT'15 [7] corpora. Details are summarized in Table 3.11.

3.3.3.1 Training details

Data and preprocessing. The pairs are English (en) to Hebrew (he), Vietnamese (vi), and Galician (gl), Slovak (sk), Arabic (ar) to English (en). Because the data is already preprocessed, we only apply BPE [74] with `fastBPE`[7]. Depending on the data size, we use different numbers of BPE operations.

We wanted to compare with the latest low-resource works of [56, 1] on the TED Talks corpus [68]. In particular, Aharoni et al. [1] identified 4 very low-resource pairs (<70k); we took the two (gl→en, sk→en) that were not extremely low (≤6k). They then identified 4 low-resource pairs with 100k-300k examples; we took the top two (ar→en, en→he). To introduce a second English-source pair and to showcase on a well-understood task, we used the *en→vi* pair from IWSLT'15 with an in-between number of examples (133k). In this way, we have examples of different resource levels, language families, writing directions, and English-source versus -target.

Model configuration. We set the hidden dimension of the feedforward sublayer to 2048 and the rest to 512, matching Vaswani et al. [80]. We use the same dropout rate for output

[7] `https://github.com/glample/fastBPE`

of sublayers, ReLU, and attention weights. Additionally, we also do word dropout [73] with probability 0.1. However, instead of zeroing the word embeddings, we randomly replace tokens with `UNK`. For all experiments, we use label smoothing of 0.1 [79, 64]. The source and target's input and output embeddings are shared [66], but we mask out words that are not in the target's vocabulary at the final output layer before softmax, by setting their logits to $-\infty$.

Training. We use a batch size of 4096 and optimize using Adam [35] with the default parameters $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\epsilon = 10^{-8}$. Gradients are clipped when global norm exceeds 1.0 [63]. An epoch is a predefined number of iterations for each pair. We stop training when a maximum number of epochs has been met or the learning rate becomes too small (10^{-6}). We also do early stopping when the development BLEU has not improved for 20 evaluations. For gl→en, this number is 50. When doing validation-based decay, we use $\alpha_{decay} = 0.8$ and $patience = 3$. For complete data and model statistics, please refer to Table 3.11. The best checkpoint is selected based on the development BLEU score during training.

Evaluation. We report tokenized BLEU [62] with `multi-bleu.perl` to be comparable with previous works. We also measure statistical significance using bootstrap resampling [36]. For WMT '14 English-German, note that one needs to put compounds in ATAT format[8] before calculating BLEU score to be comparable with previous works.

[8] `https://github.com/tensorflow/tensor2tensor/blob/master/tensor2tensor/utils/get_ende_bleu.sh`

3.3.3.2 Large vs. small initialization

To see the impact of weight initialization, we run training on the *en→vi* dataset using warmup steps of 4k, 8k, 16k (Table 3.12). With default initialization, PostNorm fails to converge on this dataset even with a long warmup of 16k steps, only reaching 5.76 BLEU.

Xavier normal		# warmup steps		
		4k	8k	16k
Baseline	PostNorm	fail	fail	5.76
	PreNorm	28.52	28.73	28.32
SmallInit	PostNorm	28.17	28.20	28.62
	PreNorm	28.26	28.44	28.33

TABLE 3.12

DEVELOPMENT BLEU ON *EN→VI* USING XAVIER NORMAL INITIALIZATION (BASELINE VERSUS SMALLINIT).

The second row shows that taking a smaller standard deviation on the attention weights (SmallInit) restores convergence to PostNorm. Though the $\sqrt{2/5} \approx 0.63$ adjustment used here seems marginal, operations like residual connections and the products between queries and keys can compound differences in scale. Though both models now achieve similar performance, we note that PreNorm works in all setups, suggesting greater stability during training. For all remaining experiments, we use PostNorm and PreNorm with SmallInit. We find this choice does not affect the performance of PreNorm.

	gl→en	sk→en	*en→vi*	en→he	ar→en	average Δ
PostNorm + LayerNorm (published)	16.2	24.0	29.09	23.66	27.84	-4.05
PostNorm + LayerNorm (1)	18.47	29.37	31.94	27.85	33.39	+0.00
PreNorm + LayerNorm (2)	19.09	29.45	31.92	28.13	33.79	+0.27
PreNorm + **fixnorm** + LayerNorm (3)	19.38	29.50	32.45	28.39	$34.35^{\dagger}$	+0.61
PreNorm + **fixnorm** + ScaleNorm (4)	$20.91^{\ddagger *}$	$30.25^{\ddagger *}$	32.79^{*}	28.44^{*}	34.15^{*}	+1.10

Table 3.13: Test BLEU using PostNorm or PreNorm and different normalization techniques. Published values are from Wang et al. [87], Neubig and Hu [56], Aharoni et al. [1]. †, ‡ and $*$ indicate significant improvement of (3) over (2), (4) over (3), and (4) over (1), respectively; $p < 0.01$ via bootstrap resampling [36].

	gl→en	sk→en	*en→vi*	en→he	ar→en
NoWarmup	18.00	28.92	28.91	30.33	35.40
InvSqrtDecay	22.18	29.08	28.84	30.30	35.33
ValDecay	21.45	29.46	28.67	30.69	35.46
InvSqrtDecay + 2×LR	21.92	29.03	28.76	30.50	35.33
ValDecay + 2×LR	21.63	29.49	28.46	30.13	34.95

Table 3.14: Development BLEU for PreNorm + **fixnorm** + ScaleNorm, trained with different learning rate schedulers.

3.3.3.3 Scaled ℓ_2 normalization and FixNorm

To compare ScaleNorm and LayerNorm, we take 8k warmup steps for all further experiments. Since we tie the target input word embedding and the last linear layer's weight (Section 3.3.3.1), FixNorm is implemented by applying ℓ_2 normalization to the word embedding, with each component initialized uniformly in $[-0.01, 0.01]$. For non-FixNorm models, word embeddings are initialized with mean 0 and standard deviation $\sqrt{1/d}$ so they sum to unit variance. All g's in ScaleNorm are initialized to $\sqrt{d}$.

Table 3.13 shows our results along with some published baselines. First, note that our Transformer baselines with PostNorm + LayerNorm (1) are very strong non-multilingual NMT models on these pairs. They outperform the best published numbers, which are all Transformer models in the past year, by an average margin of +4.0 BLEU. Then, we see

that PreNorm (2) achieves comparable or slightly better results than PostNorm on all tasks. FixNorm (3) gives an additional gain, especially on ar→en ($p < 0.01$).

Finally, we replace LayerNorm with ScaleNorm (4). ScaleNorm significantly improves on LayerNorm for two very low-resource pairs, gl→en and sk→en. On the other tasks, it performs comparably to LayerNorm. Upon aggregating all changes, our final model with ScaleNorm and FixNorm improves over our strong baseline with PostNorm on all tasks by an average of +1.1 BLEU ($p < 0.01$), with each change contributing an average of at least +0.3 BLEU. In Section 3.3.4.2, we further examine where the performance gains of ScaleNorm come from.

Moreover, ScaleNorm is also faster than LayerNorm. Recall that for each vector of size d, LayerNorm needs to compute mean, standard deviation, scaling, and shifting, which costs $O(7d)$ operations. For ScaleNorm, we only need $O(3d)$ operations to perform normalization and global scaling. This does not account for further gains due to reduction in parameters. In our implementation, training with ScaleNorm is around 5% faster than with LayerNorm, similar to the speedups on NMT observed by Zhang and Sennrich [91]'s RMSNorm (which can be viewed as ScaleNorm with per-unit scales; see Section 3.3.4.2).

3.3.3.4 Learning rates

We compare the original learning rate schedule in equation 3.8 (InvSqrtDecay) with validation-based decay (ValDecay), possibly with no warmup (NoWarmup). We use $\lambda = 1$, $n_{warmup} = 8$k for InvSqrtDecay and ValDecay. For NoWarmup, we instead use a learning rate of $3 \cdot 10^{-4}$ for all datasets. For both ValDecay and NoWarmup, we take $\alpha_{decay} = 0.8$ and $patience = 3$. For experiments with high learning rate, we use either ValDecay or InvSqrtDecay with $\lambda = 2$ (giving a peak learning rate of $\approx 10^{-3}$). All experiments use PreNorm + FixNorm + ScaleNorm.

In Table 3.14, we see that NoWarmup performs comparably to InvSqrtDecay and ValDecay except on gl→en. We believe that in general, one can do without warmup,

though it remains useful in the lowest resource settings. In our 2×LR experiments, we can still attain a maximum learning rate of 10^{-3} without disproportionately overfitting to small datasets like gl→en.

One might hypothesize that ValDecay converges more quickly to better minima than InvSqrtDecay by staying at high learning rates for longer. However, both schedulers achieve similar results with or without doubling the learning rate. This may be due to the tail-end behavior of ValDecay methods, which can involve multiplicative decays in rapid succession. Finally, our 2×LR experiments, while not yielding better performance, show that PreNorm allows us to train the Transformer with a very high learning rate despite small batches (4k tokens).

Since PreNorm can train without warmup, we wonder if PostNorm can do the same. We run experiments on *en→vi* with NoWarmup, varying the number of encoder/decoder layers. As seen in Table 3.15, PostNorm often fails without warmup even with 5 or 6 layers. Even at 4 layers, one achieves a subpar result compared to PreNorm. This reaffirms Section 3.3.3.2 in showing that PreNorm is more stable than PostNorm under different settings.

	4 layers	5 layers	6 layers
PostNorm	18.31	fails	fails
PreNorm	28.33	28.13	28.32

TABLE 3.15

DEVELOPMENT BLEU ON *EN→VI* USING NOWARMUP, AS NUMBER OF ENCODER/DECODER LAYERS INCREASES.

3.3.3.5 High-resource setting

Since all preceding experiments were in low-resource settings, we examine if our claims hold in a high-resource setting. We train the Transformer base model on WMT'14 English-German using FAIRSEQ and report tokenized BLEU scores on *newstest2014*.

In Table 3.16, SCALENORM and FIXNORM achieve equal or better results than LAYERNORM. Since SCALENORM is also faster, we recommend using both as drop-in replacements for LAYERNORM in all settings. Surprisingly, in this task POSTNORM works notably better than PRENORM; one observes similar behavior in Wang et al. [85]. We speculate this is related to identity residual networks acting like shallow ensembles [83] and thus undermining the learning of the longest path; further study is required.

	newstest2014
POSTNORM + LAYERNORM (published)	27.3
PRENORM + LAYERNORM	26.83
PRENORM + **fixnorm** + SCALENORM	27.07
POSTNORM + LAYERNORM	27.58
POSTNORM + **fixnorm** + SCALENORM	27.57

TABLE 3.16

BLEU SCORES FROM WMT'14 ENGLISH-TO-GERMAN. PUBLISHED VALUE IS FROM Vaswani et al. [80].

3.3.4 Analysis

3.3.4.1 Performance curves

Figure 3.4 shows that PreNorm not only learns faster than PostNorm, but also outperforms it throughout training. Adding FixNorm also gives faster learning at first, but only achieves close performance to that with PreNorm and no FixNorm. However, once paired with ScaleNorm, we attain a better BLEU score at the end. Because of the slow warmup period, ScaleNorm with warmup learns slower than ScaleNorm without warmup initially; however, they all converge at about the same rate.

To visualize how PreNorm helps backpropagation, we plot the global gradient norms from our runs in Figure 3.5. PostNorm produces noisy gradients with many sharp spikes, even towards the end of training. On the other hand, PreNorm has fewer noisy gradients with smaller sizes, even without warmup. LayerNorm has lower global norms than ScaleNorm + FixNorm but it has more gradient components corresponding to normalization.

	gl→en	**sk→en**	***en→vi***	**en→he**	**ar→en**
RMSNorm + **fixnorm**	20.92	30.36	32.54	28.29	33.67
ScaleNorm + **fixnorm**	20.91	30.25	32.79	28.44	34.15
ScaleNorm (g= $\sqrt{d}$) + **fixnorm** (learned)	21.18	30.36	32.66	28.19	34.11
ScaleNorm (g= $\sqrt{d}$) + **fixnorm** (learned) + ValDecay	20.36	30.45	32.83	27.97	33.98
ScaleNorm (g= $\sqrt{d}$) + **fixnorm** (learned) + ValDecay + 2×LR	21.15	30.57	31.81	25.00	28.92

Table 3.17: Test BLEU of ℓ_2-based normalization techniques with different numbers of learned g: $O(Ld)$ vs. $O(L)$ vs. $O(1)$.

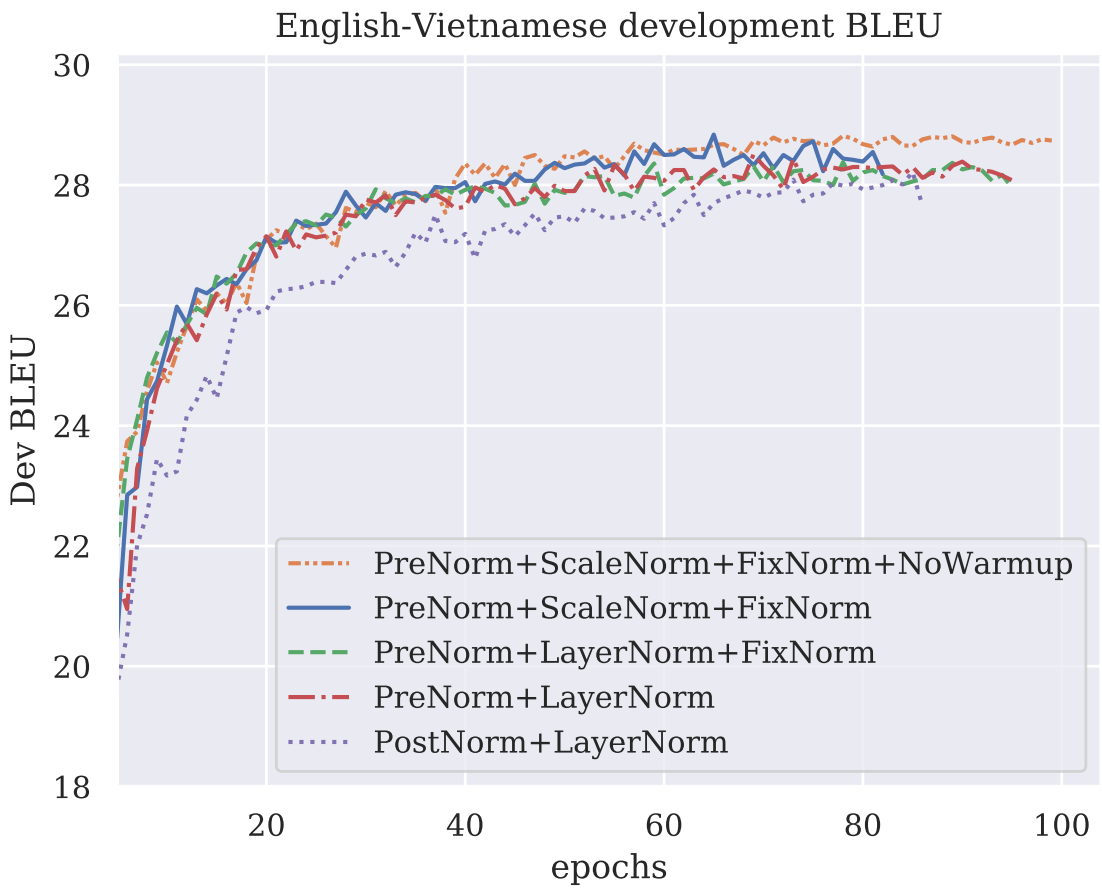


Figure 3.4. Development BLEU on *en→vi* with PostNorm or PreNorm, and with LayerNorm or ScaleNorm.

3.3.4.2 Activation scaling and the role of g

One motivation for ScaleNorm was that it expressed a good inductive bias for the global scaling of activations, independent of distributional stability (Section 3.3.2.3). In contrast, a contemporaneous work [91] proposes *root mean square layer normalization* (RMSNorm), which still follows layer normalization's motivation but reduces overhead by forgoing additive adjustments, using only a scaling g_i per activation a_i. Despite their differing motives, tying the g_i of RMSNorm and dividing by $\sqrt{d}$ retrieves ScaleNorm.

Hence we can frame our comparisons in terms of number of learnable parameters. We rerun our PreNorm experiments with RMSNorm. We also consider fixing $g = \sqrt{d}$ for

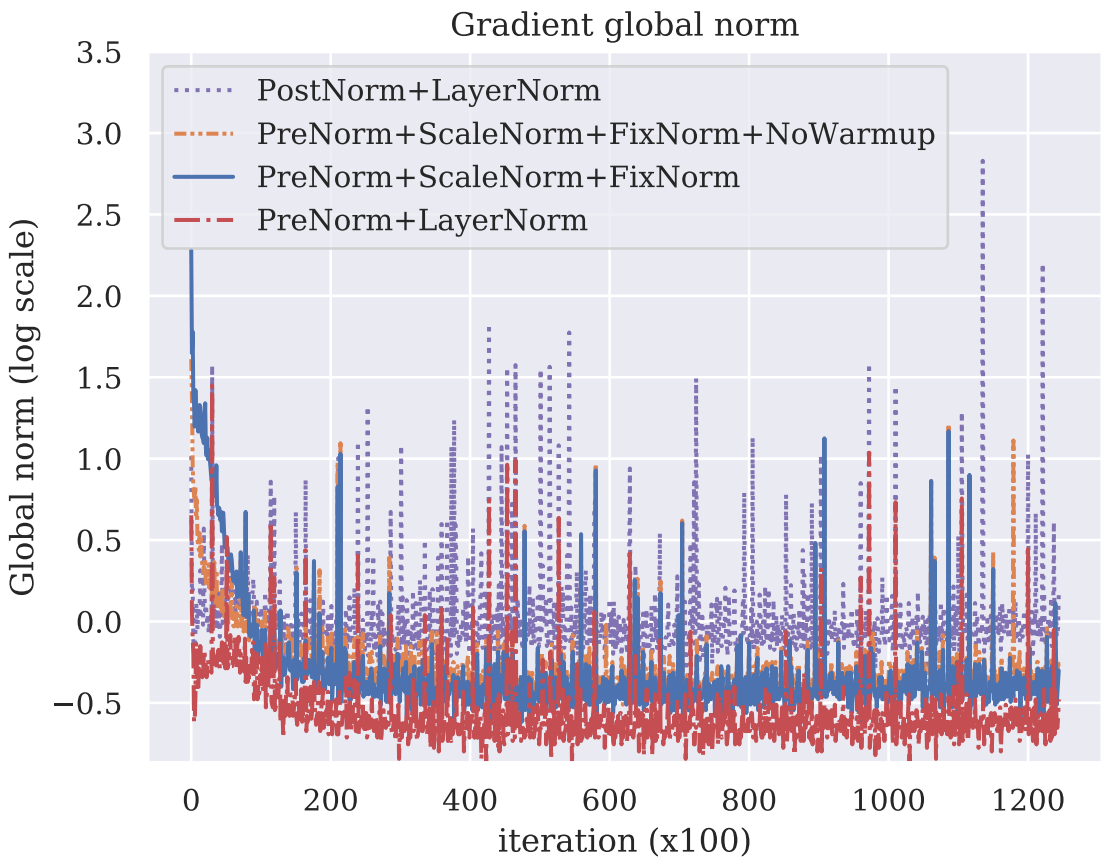


Figure 3.5. The global norm of gradients when using PostNorm or PreNorm, and with LayerNorm, ScaleNorm and FixNorm. Best viewed in color.

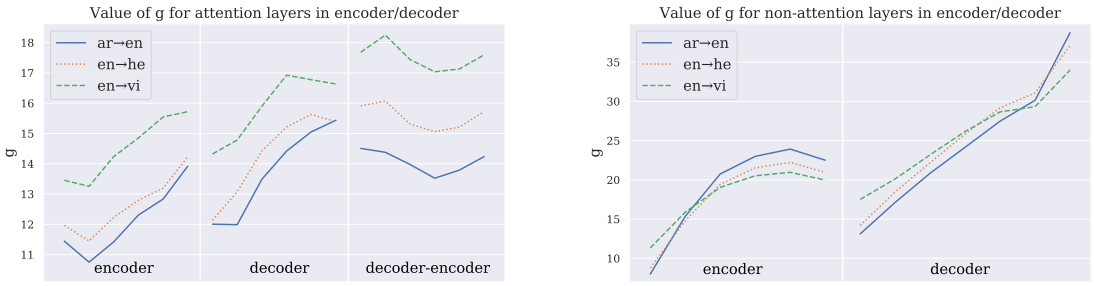


Figure 3.6: Learned g values for PreNorm + ScaleNorm + FixNorm models, versus depth. **Left:** Attention sublayers (*decoder-encoder* denotes decoder sublayers attending on the encoder). **Right:** Feedforward sublayers and the final linear layer.

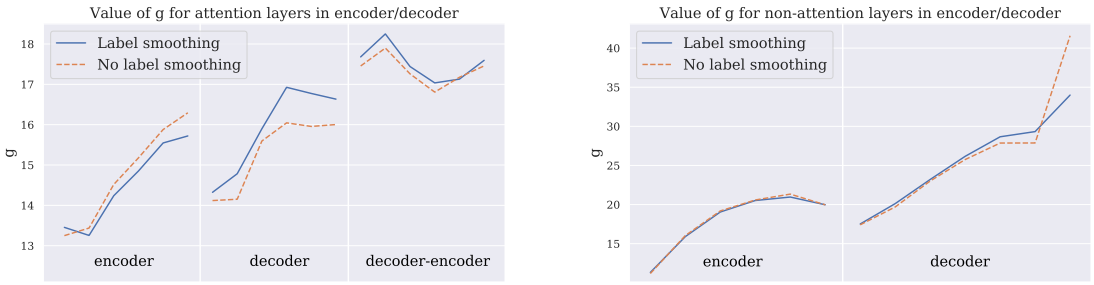


Figure 3.7: Learned g values for our PreNorm + ScaleNorm + FixNorm *en→vi* model (with and without label smoothing), versus depth. **Left** and **Right** are the same as in Figure 3.6.

ScaleNorm, where only FixNorm has learnable g. Table 3.17 shows that ScaleNorm always performs comparably or better than RMSNorm. Surprisingly, the fixed-g model performs comparably to the one with learnable g. However, at higher learning rates (ValDecay with and without 2×LR), fixed-g models perform much worse on ar→en, en→he and *en→vi*. We conjecture that learning g is required to accommodate layer gradients.

In Figure 3.6, we plot the learned g values for pairs with 100k+ examples. For all but the decoder-encoder sublayers, we observe a positive correlation between depth and g, giving credence to ScaleNorm's inductive bias of global scaling. This trend is clearest in the decoder, where g linearly scales up to the output layer, perhaps in tandem with the discriminativeness of the hidden representations [43]. We also note a negative correlation between the number of training examples and the magnitude of g for attention sublayers, which may reflect overfitting.

Finally, to affirm our intuition for interpreting g, we plot g values with and without label smoothing (Figure 3.7). We see a difference in later layers of the decoder; there, removing label smoothing results in lower g values except at the output layer, where g increases sharply. This corresponds to the known overconfidence of translation models' logits, on which label smoothing has a downscaling effect [53].

CHAPTER 4

PROPOSED CONTRIBUTIONS

Through out this document, I have shown three simple ways to improve NMT performance for low-resource languages: **better data exploitation**, **better modeling**, and **better normalization**. I have demonstrated how effective our simple ℓ_2 normalization technique is with both RNN-based and Transformer NMT. For the next step, we will focus on **better modeling** which is also related to **better data exploitation** as explained later.

I have previously shown how to improve the already-impressive performance of Transformer on low-resource with better normalization at its core components: the word embedding, self-attention and feed-forward layers (section 3.3). In section 4.1, we will look at its last important element: the positional embedding. I propose a new way to apply positional embedding for Transformer which allows it to plan in advance how many tokens to generate during decoding. I will show how this can potentially help to alleviate an important modeling error in the current model which makes empty translation to be usually the best hypothesis.

In section 3.1, we have discussed a simple way to improve performance on a low-resource language pair by transferring learning from another related language pair. While effective, it requires training two different models which is not ideal. In section 4.2, I will look into multilingual neural machine translation: a single NMT system that can handle multiple language pairs. Aside from its apparent convenience, multilingual NMT is also appealing for its implicit transfer learning ability across languages during training. We will discuss related work on multilingual NMT, where it shines and where it falls short. From this, I suggest to rethink the NMT architecture for better use of multilingual data.

Specifically, I propose to factorize each component in the model into a shared part which helps to transfer learning across languages and a private part which is language dependent to handle specific characteristics of a language.

All proposed work is roughly estimated to be submitted to some *ACL or WMT conferences in 2020 or early 2021.

4.1 Improving NMT modeling errors by planning ahead

4.1.1 Introduction

Because machine translation inference is intractable in nature, it is often approximated using beam search. However, unlike PBMT, using wider beam for NMT tends to produce shorter translations [39], thus poorer result. This bias towards short translations has recently demonstrated quantitatively by Stahlberg and Byrne [76]. In fact, the authors show that NMT assigns the best score to empty translations for more than 50% of test sentences.

Murray and Chiang [54] attributes this problem to the locally normalized nature of NMT. As seen in figure 4.1, NMT is trained to assign very low probability to the empty hypothesis (with only the "end of sentence" special token </s>). However, as we widen the beam, this hypothesis could appear early which means any other hypothesis with lower score than it will be discarded by the search.

4.1.2 Proposed Solution

The popular solutions to this problem are either length normalization [32, 6, 39] or word rewards [26, 54]. However, these solutions only try to partially alleviate the problem by assigning higher scores to long sentences. We believe the main problem is the use of </s> to signify the end of a hypothesis as described above. This puts a burden on the decoder to translate and predict the length dynamically at the same time. Therefore we propose to remove the length prediction by explicitly let the decoder know how many

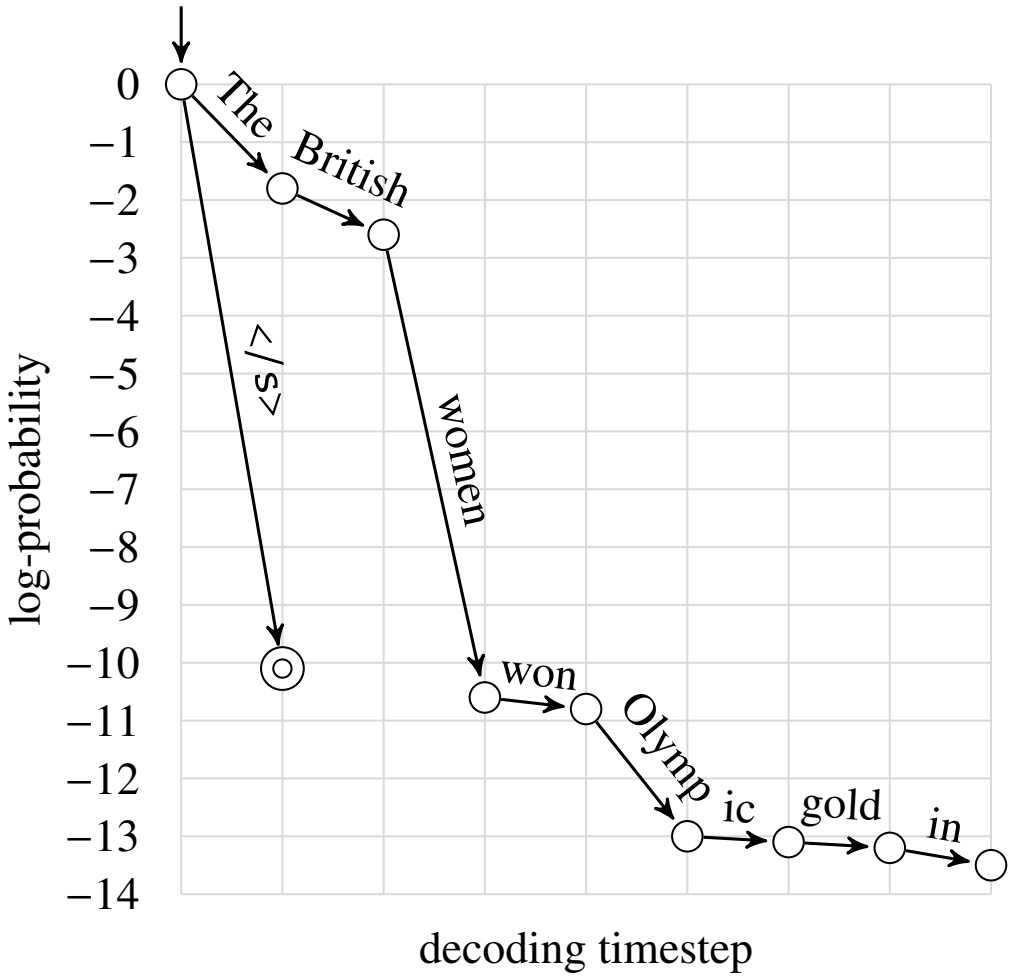


Figure 4.1: Widening beam search allows empty hypothesis to appear early on in decoding. This puts a lower bound on the score another hypothesis can have which causes longer translations to be excluded including the correct translation in this example [54].

tokens left to decode.

Recall from section 1.2 that in Transformer, we model the order of a sequence by using the cosine positional embedding which is element-wise summed with the word embedding. For example, we would provide each token in the sentence "My name is Toan ." with positional embeddings of indices 0, 1, 2, 3, 4. To make decoder aware of the number of tokens left to translate, we propose to "count down" by indexing with 4, 3, 2, 1, 0 instead. Using this approach, we can always stop decoding whenever the position at index 0 has been met. Additionally, we need a separate length model which helps NMT to plan ahead how many tokens it should decode. Specifically, instead of learning to maximize $p_{nmt}(\mathbf{y} \mid \mathbf{x})$, we maximize:

$$p_{length}(|\mathbf{y}| \mid \mathbf{x}) p_{nmt}(\mathbf{y} \mid \mathbf{x}) \tag{4.1}$$

where p_{length} is produced by some neural network.

At inference time, we will first select some top k lengths produced by the length model. Then NMT can start decoding accordingly each possible length. The best hypothesis is chosen according to the score in equation 4.1.

Note that this "count down" method has been previously investigated by Kikuchi et al. [34] for summarization task using RNN-based encoder-decoder. The main differences are we use cosine positional embedding instead of learned embedding and we also incorporate a length prediction model. By using cosine embedding, we hope the model can generalize to unseen sentence length as well.

4.2 Improving multilingual NMT with Shared-Private Encoder-Decoder

4.2.1 Introduction

Multilingual NMT [14, 12, 33, 21, 1] aims to build a single NMT system for many language pairs. These multilingual systems can share certain components such as the encoder, decoder or attention module across languages, or better yet use the same model

for all of them. In terms of translation direction, this task can be categorized into one-to-many, many-to-one, or ideally many-to-many. Multilingual NMT is appealing since it can handle many translation directions with a single model, thus saving space and easing deployment. Furthermore, it can also help improve translation quality by transferring learning from high-resource to low-resource languages. The most attractive feature, however, is the potential to do zero-shot translation between language pairs unseen in training [33, 22, 2, 42].

Early work on using a single multilingual NMT model shows the task is not that straightforward. The general consensus is training many-to-one often helps, one-to-many helps mostly for low-resource but hurts high-resource languages, and many-to-many tends to hurt more than help [33]. Recently, Aharoni et al. [1] conduct a massive experiment on more than 100 high-resource language pairs (each has more than 1M examples) using Transformer. Their finding can be summarized as:

- all one-to-many, many-to-one, and many-to-many outperform baseline
- one-to-many or many-to-one are always better than many-to-many
- many-to-one is better than one-to-many

This result is interesting as it demonstrates that we can indeed use a single encoder-decoder model to train for all translation directions. However, the fact that many-to-many is often less superior to the other settings despite being trained on more data demonstrates the limited capacity of the NMT model, specifically the decoder, to handle different languages. Ideally, we do not want to resort to separate encoder/decoder which can potentially limit the transfer learning ability. This begs for a solution somewhere in the middle.

4.2.2 Proposed Solution

We propose to decouple each component of NMT into two parts: a shared part which is used for all languages, and a private part which is language dependent. A simple example

using feed-forward layer could be rewriting the following:

$$y = \tanh(Wx) \tag{4.2}$$

with $W \in \mathcal{R}^{d \times d}$, into:

$$y = \tanh(\text{concat}([W_{shared}, W_{private}])x) \tag{4.3}$$

with $W_{shared}, W_{private} \in \mathcal{R}^{\frac{d}{2} \times d}$.

With this approach, we hope W_{shared} could help to transfer learning across all languages. And by using a separate $W_{private}$ per language, we hope to increase the capacity of NMT model and account for the specific characteristics of each language.

Factorizing neural networks into shared and private parts is not new. For example, Zhang et al. [92] have used this approach for text style adaptation. In the context of NMT, Liu et al. [46] propose to factorize the word embedding into different components. One feature of their method is if the two source and target words have the same form, they only share a part of the word embedding vector, and the rest is language dependent. As far as we know, generalizing this approach to the whole NMT has not been investigated before. We hope that doing so can either improve multilingual NMT or shed some light into how to better design NMT architecture for multilingual translation.

CHAPTER 5

CONCLUSION

In this proposal, I have covered some background on Neural Machine Translation with a particular focus on low-resource domain. Through existing contributions and related work, I have demonstrated the weakness of vanilla NMT systems when dealing with limited data. My contributions have addressed these problems one by one, allowing us to successfully train NMT models that significantly improves over the baseline.

It is hard to tell if a technique works in low-resource can apply for high-resource and vice versa. The hope is all contributions in this thesis can shed some light into future research. For example, we have seen in section 3.3 that the **fixnorm** proposed in section 3.2 can generalize to high-resource domain, or the transfer learning approach in section 3.1 gives a better understanding on how to share learning across languages. With that in mind, my next proposed contributions will focus on the core problem of NMT in general and multilingual NMT in particular. Once completed, I hope this work can bring together a collection of useful techniques to train good NMT systems.

BIBLIOGRAPHY

1. R. Aharoni, M. Johnson, and O. Firat. Massively Multilingual Neural Machine Translation. In *NAACL-HLT*, pages 3874–3884, 2019. doi: 10.18653/v1/N19-1388.

2. N. Arivazhagan, A. Bapna, O. Firat, R. Aharoni, M. Johnson, and W. Macherey. The missing ingredient in zero-shot neural machine translation. *arXiv preprint arXiv:1903.07091*, 2019.

3. P. Arthur, G. Neubig, and S. Nakamura. Incorporating discrete translation lexicons into neural machine translation. In *Proc. EMNLP*, 2016.

4. J. L. Ba, J. R. Kiros, and G. E. Hinton. Layer Normalization. *CoRR*, abs/1607.06450, 2015.

5. D. Bahdanau, K. Cho, and Y. Bengio. Neural machine translation by jointly learning to align and translate. In *Proc. ICLR*, 2015.

6. N. Boulanger-Lewandowski, Y. Bengio, and P. Vincent. Audio chord recognition with recurrent neural networks. In *ISMIR*, 2013.

7. M. Cettolo, J. Niehues, L. Bentivogli, R. Cattoni, and M. Federico. The IWSLT 2015 Evaluation Campaign. In *IWSLT*, pages 3–4, 2015. URL `http://workshop2015.iwslt.org/downloads/IWSLT_2015_EP_0.pdf`.

8. M. X. Chen, O. Firat, A. Bapna, M. Johnson, W. Macherey, G. Foster, L. Jones, N. Parmar, N. Shazeer, A. Vaswani, J. Uszkoreit, L. Kaiser, M. Schuster, Z. Chen, Y. Wu, and M. Hughes. The best of both worlds: Combining recent advances in neural machine translation. In *ACL*, pages 76–86, 2018. ISBN 9781948087322. doi: 10.18653/v1/P18-1008.

9. K. Cho, B. van Merriënboer, D. Bahdanau, and Y. Bengio. On the properties of neural machine translation: Encoder–decoder approaches. In *Proc. Workshop on Syntax, Semantics and Structure in Statistical Translation*, 2014.

10. 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. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1724–1734, Doha, Qatar, Oct. 2014. Association for Computational Linguistics. doi: 10.3115/v1/D14-1179. URL `https://www.aclweb.org/anthology/D14-1179`.

11. J. Chung, K. Cho, and Y. Bengio. A character-level decoder without explicit segmentation for neural machine translation. In *Proc. ACL*, 2016.

12. D. Dong, H. Wu, W. He, D. Yu, and H. Wang. Multi-task learning for multiple language translation. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 1723–1732, Beijing, China, July 2015. Association for Computational Linguistics. doi: 10.3115/v1/P15-1166. URL https://www.aclweb.org/anthology/P15-1166.

13. L. Dong, S. Xu, and B. Xu. Speech-Transformer: A No-Recurrence Sequence-to-Sequence Model for Speech Recognition. In *ICASSP*, pages 5884–5888, 2018. ISBN 978-1-5386-4658-8. doi: 10.1109/ICASSP.2018.8462506.

14. O. Firat, K. Cho, and Y. Bengio. Multi-way, multilingual neural machine translation with a shared attention mechanism. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 866–875, San Diego, California, June 2016. Association for Computational Linguistics. doi: 10.18653/v1/N16-1101. URL https://www.aclweb.org/anthology/N16-1101.

15. J. Gehring, M. Auli, D. Grangier, D. Yarats, and Y. N. Dauphin. Convolutional sequence to sequence learning, 2017. arXiv:1705.03122.

16. X. Glorot and Y. Bengio. Understanding the difficulty of training deep feedforward neural networks. In *AISTATS*, pages 249–256, 2010. ISBN 9781937284275. doi: 10.1.1.207.2059.

17. X. Glorot, A. Bordes, and Y. Bengio. Deep sparse rectifier neural networks. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pages 315–323, 2011.

18. J. Gu, Z. Lu, H. Li, and V. O. Li. Incorporating copying mechanism in sequence-to-sequence learning. In *Proc. ACL*, 2016.

19. C. Gulcehre, S. Ahn, R. Nallapati, B. Zhou, and Y. Bengio. Pointing the unknown words. In *Proc. ACL*, 2016.

20. Q. Guo, X. Qiu, P. Liu, Y. Shao, X. Xue, and Z. Zhang. Star-Transformer. In *NAACL-HLT*, pages 1315–1325, 2019.

21. T.-L. Ha, J. Niehues, and A. Waibel. Toward multilingual neural machine translation with universal encoder and decoder. *arXiv preprint arXiv:1611.04798*, 2016.

22. T.-L. Ha, J. Niehues, and A. Waibel. Effective strategies in zero-shot neural machine translation. *arXiv preprint arXiv:1711.07893*, 2017.

23. K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *CVPR*, pages 770–778, 2016. ISBN 9781467388504. doi: 10.1109/CVPR.2016.90.

24. K. He, X. Zhang, S. Ren, and J. Sun. Identity Mappings in Deep Residual Networks. In *ECCV*, pages 630–645, 2016.

25. K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proc. CVPR*, 2016.

26. W. He, Z. He, H. Wu, and H. Wang. Improved neural machine translation with smt features, 2016. URL https://www.aaai.org/ocs/index.php/AAAI/AAAI16/paper/view/12189.

27. F. Hieber, T. Domhan, M. Denkowski, D. Vilar, A. Sokolov, A. Clifton, and M. Post. The Sockeye neural machine translation toolkit. In *AMTA*, pages 200–207, 2018. URL https://www.aclweb.org/anthology/W18-1820/.

28. S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural Computation*, 9 (8), 1997.

29. H. Inan, K. Khosravi, and R. Socher. Tying word vectors and word classifiers: A loss framework for language modeling. In *Proc. ICLR*, 2017.

30. S. Ioffe and C. Szegedy. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. In *ICML*, pages 448–456, 2015. ISBN 9780874216561. doi: 10.1007/s13398-014-0173-7.2.

31. S. Jean, K. Cho, R. Memisevic, and Y. Bengio. On using very large target vocabulary for neural machine translation. In *Proc. ACL-IJCNLP*, 2015.

32. S. Jean, O. Firat, K. Cho, R. Memisevic, and Y. Bengio. Montreal neural machine translation systems for WMT'15. In *Proceedings of the Tenth Workshop on Statistical Machine Translation*, pages 134–140, Lisbon, Portugal, Sept. 2015. Association for Computational Linguistics. doi: 10.18653/v1/W15-3014. URL https://www.aclweb.org/anthology/W15-3014.

33. M. Johnson, M. Schuster, Q. V. Le, M. Krikun, Y. Wu, Z. Chen, N. Thorat, F. Viégas, M. Wattenberg, G. Corrado, M. Hughes, and J. Dean. Google's multilingual neural machine translation system: Enabling zero-shot translation. *Transactions of the Association for Computational Linguistics*, 5:339–351, 2017. doi: 10.1162/tacl_a_00065. URL https://www.aclweb.org/anthology/Q17-1024.

34. Y. Kikuchi, G. Neubig, R. Sasano, H. Takamura, and M. Okumura. Controlling output length in neural encoder-decoders. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 1328–1338, Austin, Texas, Nov. 2016. Association for Computational Linguistics. doi: 10.18653/v1/D16-1140. URL https://www.aclweb.org/anthology/D16-1140.

35. D. P. Kingma and J. Ba. Adam: A Method for Stochastic Optimization. In *ICLR*, 2015.

36. P. Koehn. Statistical significance tests for machine translation evaluation. In *EMNLP*, pages 388–395, 2004. ISBN 978-1-4503-0717-8. doi: 10.1145/2063576.2063688.

37. P. Koehn. Statistical significance tests for machine translation evaluation. In *Proc. EMNLP*, 2004.

38. P. Koehn and R. Knowles. Six challenges for neural machine translation. In *Proc. First Workshop on Neural Machine Translation*. Association for Computational Linguistics, 2017.

39. P. Koehn and R. Knowles. Six challenges for neural machine translation. In *Proc. Workshop on Neural Machine Translation*, 2017.

40. P. Koehn, H. Hoang, A. Birch, C. Callison-Burch, M. Federico, N. Bertoldi, B. Cowan, W. Shen, C. Moran, R. Zens, et al. Moses: Open source toolkit for statistical machine translation. In *Proc. ACL*, 2007.

41. W. Kool, H. Van Hoof, and M. Welling. Attention, Learn to Solve Routing Problems! In *ICLR*, 2019.

42. S. M. Lakew, Q. F. Lotito, M. Negri, M. Turchi, and M. Federico. Improving zero-shot translation of low-resource languages. *arXiv preprint arXiv:1811.01389*, 2018.

43. D. Liang, Z. Huang, and Z. C. Lipton. Learning noise-invariant representations for robust speech recognition. In *SLT*, pages 56–63, 2018. doi: 10.1109/SLT.2018.8639575.

44. A. Liu and K. Kirchhoff. Context models for oov word translation in low-resource languages. In *Proceedings of AMTA 2018, vol. 1: MT Research Track*. AMTA, 2018.

45. W. Liu, Y. Wen, Z. Yu, and M. Yang. Large-margin softmax loss for convolutional neural networks. In *Proc. ICML*, 2016.

46. X. Liu, D. F. Wong, Y. Liu, L. S. Chao, T. Xiao, and J. Zhu. Shared-private bilingual word embeddings for neural machine translation. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 3613–3622, Florence, Italy, July 2019. Association for Computational Linguistics. doi: 10.18653/v1/P19-1352. URL `https://www.aclweb.org/anthology/P19-1352`.

47. C. Luo, J. Zhan, L. Wang, and Q. Yang. Cosine Normalization: Using Cosine Similarity Instead of Dot Product in Neural Networks. In *ICANN*, pages 382–391, 2018.

48. M.-T. Luong and C. D. Manning. Stanford neural machine translation systems for spoken language domain. In *Proc. IWSLT*, 2015.

49. M.-T. Luong and C. D. Manning. Achieving open vocabulary neural machine translation with hybrid word-character models. In *Proc. ACL*, 2016.

50. M.-T. Luong, H. Pham, and C. D. Manning. Effective approaches to attention-based neural machine translation. In *Proc. EMNLP*, 2015.

51. T. Luong, I. Sutskever, Q. Le, O. Vinyals, and W. Zaremba. Addressing the rare word problem in neural machine translation. In *Proc. ACL-IJCNLP*, 2015.

52. H. Mi, Z. Wang, and A. Ittycheriah. Vocabulary manipulation for neural machine translation. In *Proc. ACL*, 2016.

53. R. Müller, S. Kornblith, and G. Hinton. When Does Label Smoothing Help? In *NeurIPS*, 2019.

54. K. Murray and D. Chiang. Correcting length bias in neural machine translation. In *Proceedings of the Third Conference on Machine Translation: Research Papers*, pages 212–223, Brussels, Belgium, Oct. 2018. Association for Computational Linguistics. doi: 10.18653/v1/W18-6322. URL https://www.aclweb.org/anthology/W18-6322.

55. P. Nakov and H. T. Ng. Improved statistical machine translation for resource-poor languages using related resource-rich languages. In *Proc. EMNLP*, 2009. ISBN 978-1-932432-63-3.

56. G. Neubig and J. Hu. Rapid Adaptation of Neural Machine Translation to New Languages. In *EMNLP*, pages 875–880, 2018. doi: 10.18653/v1/d18-1103.

57. T. Nguyen and D. Chiang. Improving Lexical Choice in Neural Machine Translation. In *NAACL-HLT*, pages 334–343, 2018. doi: 10.18653/v1/n18-1031.

58. F. J. Och and H. Ney. A systematic comparison of various statistical alignment models. *Computational Linguistics*, 29(1), 2003.

59. F. J. Och and H. Ney. Discriminative training and maximum entropy models for statistical machine translation. In *Proc. ACL*, 2002.

60. M. Ott, S. Edunov, D. Grangier, and M. Auli. Scaling Neural Machine Translation. In *WMT*, pages 1–9, 2018. URL https://www.aclweb.org/anthology/W18-6301/.

61. M. Ott, S. Edunov, A. Baevski, A. Fan, S. Gross, N. Ng, D. Grangier, and M. Auli. fairseq: A Fast, Extensible Toolkit for Sequence Modeling. In *NAACL-HLT (Demonstrations)*, pages 48–53, 2019. doi: 10.18653/v1/n19-4009.

62. K. Papineni, S. Roukos, T. Ward, and W.-J. Zhu. Bleu: a method for automatic evaluation of machine translation. In *ACL*, pages 311–318, 2002. doi: 10.3115/1073083.1073135.

63. R. Pascanu, T. Mikolov, and Y. Bengio. On the difficulty of training recurrent neural networks. In *ICML*, pages 1310–1318, 2013. ISBN 08997667 (ISSN). doi: 10.1109/72.279181.

64. G. Pereyra, G. Tucker, J. Chorowski, L. Kaiser, and G. E. Hinton. Regularizing neural networks by penalizing confident output distributions. In *ICLR (Workshop)*, 2017. URL https://openreview.net/forum?id=HyhbYrGYe.

65. M. Popel and O. Bojar. Training Tips for the Transformer Model. *Prague Bull. Math. Linguistics*, 110(1):43–70, 2018. doi: 10.2478/pralin-2018-0002.

66. O. Press and L. Wolf. Using the Output Embedding to Improve Language Models. In *EACL*, pages 157–163, 2017. URL https://www.aclweb.org/anthology/E17-2025/.

67. O. Press and L. Wolf. Using the output embedding to improve language models. In *Proc. EACL*, 2017.

68. Y. Qi, D. Sachan, M. Felix, S. Padmanabhan, and G. Neubig. When and Why Are Pre-Trained Word Embeddings Useful for Neural Machine Translation? In *NAACL-HLT*, pages 529–535, 2018. doi: 10.18653/v1/n18-2084.

69. A. Rush. The annotated transformer. In *Proceedings of Workshop for NLP Open Source Software (NLP-OSS)*, pages 52–60, Melbourne, Australia, July 2018. Association for Computational Linguistics. doi: 10.18653/v1/W18-2509. URL https://www.aclweb.org/anthology/W18-2509.

70. J. Salazar, K. Kirchhoff, and Z. Huang. Self-attention Networks for Connectionist Temporal Classification in Speech Recognition. In *ICASSP*, pages 7115–7119, 2019. ISBN 978-1-4799-8131-1. doi: 10.1109/ICASSP.2019.8682539.

71. T. Salimans and D. P. Kingma. Weight Normalization: A Simple Reparameterization to Accelerate Training of Deep Neural Networks. *ArXiv e-prints*, Feb. 2016.

72. S. Santurkar, D. Tsipras, A. Ilyas, and A. Madry. How does batch normalization help optimization? In *NeurIPS*, pages 2483–2493, 2018. URL http://papers.nips.cc/paper/7515-how-does-batch-normalization-help-optimization.

73. R. Sennrich, B. Haddow, and A. Birch. Edinburgh Neural Machine Translation Systems for WMT 16. In *WMT*, pages 371–376, 2016.

74. R. Sennrich, B. Haddow, and A. Birch. Neural machine translation of rare words with subword units. In *Proc. ACL*, 2016.

75. N. Shazeer and M. Stern. Adafactor: Adaptive Learning Rates with Sublinear Memory Cost. In *ICML*, pages 4603–4611, 2018.

76. F. Stahlberg and B. Byrne. On NMT search errors and model errors: Cat got your tongue? In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 3354–3360, Hong Kong, China, Nov. 2019. Association for Computational Linguistics. doi: 10.18653/v1/D19-1331. URL https://www.aclweb.org/anthology/D19-1331.

77. S. Sukhbaatar, E. Grave, G. Lample, H. Jegou, and A. Joulin. Augmenting Self-attention with Persistent Memory. *CoRR*, abs/1907.01470, 2019.

78. I. Sutskever, O. Vinyals, and Q. V. Le. Sequence to sequence learning with neural networks. In *NIPS 27*, 2014.

79. C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna. Rethinking the Inception Architecture for Computer Vision. In *CVPR*, pages 2818–2826, 2016. ISBN 9781467388504. doi: 10.1109/CVPR.2016.308.

80. A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin. Attention is All you Need. In *NeurIPS*, pages 5998–6008. 2017. URL http://papers.nips.cc/paper/7181-attention-is-all-you-need.

81. 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 6000–6010, 2017.

82. A. Vaswani, S. Bengio, E. Brevdo, F. Chollet, A. N. Gomez, S. Gouws, L. Jones, Ł. Kaiser, N. Kalchbrenner, N. Parmar, R. Sepassi, N. Shazeer, and J. Uszkoreit. Tensor2Tensor for Neural Machine Translation. In *AMTA*, pages 193–199, 2018.

83. A. Veit, M. Wilber, and S. Belongie. Residual networks behave like ensembles of relatively shallow networks. *NeurIPS*, pages 550–558, 2016. ISSN 10495258. URL http://papers.nips.cc/paper/6556-residual-networks-behave-like-ensembles-of-relatively-shallow-networks.

84. F. Wang, X. Xiang, J. Cheng, and A. L. Yuille. Normface: L2 hypersphere embedding for face verification. In *Proceedings of the 25th ACM international conference on Multimedia*. ACM, 2017. doi: https://doi.org/10.1145/3123266.3123359.

85. Q. Wang, B. Li, T. Xiao, J. Zhu, C. Li, D. F. Wong, and L. S. Chao. Learning Deep Transformer Models for Machine Translation. In *ACL*, pages 1810–1822, 2019. URL https://www.aclweb.org/anthology/P19-1176/.

86. X. Wang, Z. Lu, Z. Tu, H. Li, D. Xiong, and M. Zhang. Neural machine translation advised by statistical machine translation. In *Proc. AAAI*, 2017.

87. X. Wang, H. Pham, Z. Dai, and G. Neubig. Switchout: an efficient data augmentation algorithm for neural machine translation. In *EMNLP*, pages 856–861, 2018. URL https://www.aclweb.org/anthology/D18-1100/.

88. 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, 2016. arXiv:1609.08144.

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

90. M. D. Zeiler. ADADELTA: An adaptive learning rate method, 2012. arXiv:1212.5701v1.

91. B. Zhang and R. Sennrich. Root Mean Square Layer Normalization. *NeurIPS*, 2019. URL https://openreview.net/pdf?id=SygkZ3MTJE.

92. Y. Zhang, N. Ding, and R. Soricut. SHAPED: Shared-private encoder-decoder for text style adaptation. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 1528–1538, New Orleans, Louisiana, June 2018. Association for Computational Linguistics. doi: 10.18653/v1/N18-1138. URL https://www.aclweb.org/anthology/N18-1138.

93. B. Zoph, D. Yuret, J. May, and K. Knight. Transfer learning for low-resource neural machine translation. In *Proc. EMNLP*, 2016.

This document was prepared & typeset with pdfLaTeX*, and formatted with* NDdiss2$_\varepsilon$ *classfile (v3.2017.2[2017/05/09])*