NAIST-IS-DD1561210

Doctoral Dissertation

Learning Lexical Representations for Neural Machine Translation

Philip Arthur

July 30, 2018

Graduate School of Information Science
Nara Institute of Science and Technology

A Doctoral Dissertation
submitted to Graduate School of Information Science,
Nara Institute of Science and Technology
in partial fulfillment of the requirements for the degree of
DOCTOR of ENGINEERING

Philip Arthur

Learning Lexical Representations for Neural Machine Translation[*]

Philip Arthur

Abstract

Words are the most natural unit of meaning with which most people think about language. Humans understand language by combining together spoken sounds, forming words. Different senses of words can totally change the meaning of the whole sentence thus it is important to represent words robustly in a computer. In order for computers to process words, it is important to study some various ways of representation so that we know which representation is better for NMT. In particular, this thesis focuses on representation of individual tokens in the computer, called the lexical representation.

Up to this point, there have been several different computational representations of words proposed. For example, the earliest natural language processing (NLP) work uses the word string itself to represent words. While simple and effective, this approach has several drawbacks. String is not a robust representation of the meaning which is contained inside a word, because words that are close in meaning but with slightly different surface forms (e.g. eats vs ate) to be treated as entirely different words. Thus it is hard to model the similarities or differences between words because all words in the corpus are treated as distinct entities and assigned to different parameters. A remedy to this problem is using continuous representations, which treat words as a vector in a continuous space. Continuous representations are better because meaning can now be represented as some degrees of magnitude and can be composed with many factors (represented as dimension).

However, in these continuous representations, while meaning is modeled more appropriately, there still exist some problems and dilemmas that are hard to deal with. This thesis attempts to study the better computational lexical representations of words,

[*]Doctoral Dissertation, Department of Information Systems, Graduate School of Information Science, Nara Institute of Science and Technology, NAIST-IS-DD1561210, July 30, 2018.

specifically focused on the continuous representations in the task of end-to-end neural machine translation (NMT). Sub-optimal representations of word units can make learning slow or even worse, make it fail to learn the patterns of the language in a generalizable way. This generalization failure can cause the translation systems to fail to learn and produce certain words. These failed words are often rare entities which are important to translate (e.g. political entities, races, religions, etc.).

Our first study proposes a better way of representing continuous word representations in the target side of NMT. The vectors that represent words (embeddings) tend to be similar for words that are infrequent in the training corpus, making it difficult for the translation system to precisely generate the words and entities. We use part of the previous count-based MT systems, the lexical translation probabilities, to provide a strong prior for the NMT system. Our experiments showed a significant improvement in rare-word translation over the NMT baseline.

The second study investigates better ways of representing the source words of NMT. NMT works on the set of lexical units that are defined using either full words or subwords. While using the full words form, there exist words that occur more often than the others. This implies that variants of machine learning algorithms that use a unique parameter for different words will update some word parameters more often than the others. This is the "rare word" problem, where it is hard to model and produce words that appear in the training corpus only a few times. Using subword units is a natural remedy to this problem. However, subword units also delete the natural segmentation of full word units, exchanging context for flexibilities. There also exist multiple ways in representing each token as a continuous vector. These are the dilemmas in representing each lexical unit with a vector in a continuous space. Despite a fair amount of previous work on this subject, because no systematic comparison between the various methods has been performed, there is no clear answer which combination works better. In this study, we tried to (1) compare each of the combinations of lexical unit and composition function, (2) combine the composition functions together. Experiments show that a composition function using character bag-of-ngrams information achieved the best accuracy when using the natural word segmentation. Moreover we find that by simply adding character embeddings information to the subword based systems also increased the system accuracy.

The third study proposed an automatic way of determining the most optimal set of lexical units for NMT. These lexical units are in the form of subwords that are optimized jointly with NMT. The subwords are latent variables that need to be discovered on-the-fly while also maximizing the potential of NMT system in producing better translation quality. Preliminary experiments show that the system is able to generate a set of meaningful lexical units, but also that stabilizing training is a major challenge. Qualitatively, the produced segmentation is somewhat intuitive and there are some consistent linguistic patterns being discovered.

Together, these three main components of the thesis, each studies better ways of representing lexical units for better NMT.[1]

Keywords:

Neural Machine Translation, Lexicon, Representation Learning, Reinforcement Learning, Hybrid Method, Embedding, Compositional Model, Computational Linguistic

[1]There is an updated version of this thesis available at `http://arthur91.com/work/dthesis/dthesis.pdf`

Acknowledgements

There are many parties involved in making this thesis. It has been a very enjoyable part of my life in learning and doing many things in Nara Institute of Science and Technology, Japan. First and foremost I want to thank my almighty God, Jesus Christ from whom by His grace, I might live, grow, and breath until today.

Second, I would like to thank Professor Nakamura for accepting me in his lab. He granted me an invaluable internship and later recommended me for MEXT scholarship which totally altered the course of my life. He is always funny and smiling, but very serious in doing his job. Nakamura-sensei always teaches me with his abundance experiences about making the best decision in my research career. His guidance and leadership are a very big inspiration to become a future researcher.

Third, I would like to thank Assistant Professor Graham Neubig from whom I have learned uncountable of invaluable experiences such as coding, research, English writing, time management, presentation, and manner. I can never say enough thanks for his big involvement in my Doctoral study. He is a man with unending patience whom I learned by examples from his very excellent works, and never gets angry to me even though I have totally messed up. He never hesitates in helping me hands-on with his excellent coding and debugging skills that I always admire. He also gave me a valuable 1 year experience to go to Carnegie Mellon University for getting more research experiences. Dr. Graham is always very humble in expressing his opinion and encourage me even in his most busiest time. If one day I am to be a teacher, he will always be one model that I always remember.

I also want to thank my only Indonesian supervisor, Associate Professor Sakriani Sakti for many valuable discussions about living in Japan. She also does not hesitate to bring us to see some nice places in Japan. She was also the first connection I had so I could come to Japan.

I would also like to thank all the thesis committee, Professor Yuji Matsumoto, Associate Professor Katsuhito Sudoh, and Dr. Toshiaki Nakazawa for giving me insightful comments and carefully checking my thesis.

I would like to thank to all the staffs of Nakamura laboratory. Thank you to Yoshino-sensei, who gives me many inputs about all my works and share his best experience in working with reinforcement learning. Thank you to Sudoh-sensei as the staff of Nakamura lab for he tried to give me many valuable inputs to my research even he was not following my works from the beginning. Special thanks to Matsuda-san who is always being involved to all the problems I encountered in Japan. She is very helpful and patience in helping me who has a very limited Japanese language abilities.

Thank you for all the members of Nakamura lab, especially the MT/NL group in which I learn, discuss, and grow up to be a researcher. It was very fun to work with others. Special thanks to Yusuke Oda who taught me some fundamental knowledge when I just started learning neural networks.

Thank you to all members of NeuLab for giving me the warmest place to discuss and hang out during my stay in Pittsburgh! Even though a year is a very short time, but I have made many invaluable connections while staying there.

Thank you to Dr. Hideto Kazawa and Keith Stevens for hosting me in Google Japan. It's been a very interesting and invaluable experience to work side by side with the most talented engineers that are working for the best translation service provider in the world.

Many thanks to all the people in Pittsburgh who makes my stay more wonderful! Thank you to Adhiguna Surya Kuncoro who is helping me in the first time in Pittsburgh. Thank you to Shirley, Aldrian, Karin, and Victoria for the most exciting trip I have in the US. Thank you to the Asian Church communities in Pittsburgh.

I would like to say thanks to GI3 Osaka for the church service and becoming the family I have in Japan. Thank you also to MEXT scholarhip and Microsoft that supported me during my research in Japan. Yamazaki-sensei, Nakanishi-sensei, Nakajima-sensei, Higa-sensei who taught me Japanese language, starting from none.

I would also like to say thank you to my friends and family who provided me endless emotional supports during my hardest time in the US. I always believe that everything is for everybody's goodness. These are friends who always support me with my decision. Thank you to Sendy, Ferty, Lasguido, Prildy, Shieron, Nathan, Venezia, Sally, Christian, Calvin, Fenny, Dennis, and Natan. Thank you for my steam friends who laugh and play together with me almost every night, making me feeling less lonely when studying abroad alone.

There are still many things and people I want to thank to, but it is impossible to mention all of them. Here are special mention for all the researchers who I think play some parts (and have not being mentioned yet) in the process of getting my degree during my whole 5 years study in Japan. Thank you to Professor Junichi Tsujii, Associate Professor Yuki Arase, Assistant Professor Kevin Duh, and Dr. Taro Watanabe.

And finally, for my big family: Papa, Mama, Cici, Dea, Pepet, Uncles, Aunts, Cousins, Kungkung and Pho2. They provide me with unending emotional supports and laughs. They always believe in and push me to pursue my dream, even it means leaving them to study abroad. Finally, for Ora Herfita who laughs, prays, smiles, and putting her trust in me every day, reminding me for what are the best and the most important things in the present and in the future.

Contents

Abstract **i**

Acknowledgements **iv**

1 Introduction **1**

1.1. Lexical Unit Acquisition for NMT 1

1.2. Improving Continuous Lexical Representation in NMT 4

2 Background **8**

2.1. Discrete and Continuous Lexical Representation 8

2.1.1 Word as Discrete Representation 8

2.1.2 Word as Continuous Representation 9

2.2. Constructing Statistical Machine Translation Systems with Neural Networks 13

2.2.1 Recurrent Neural Networks 13

2.2.2 Neural Machine Translation 14

2.2.3 Learning NMT with Maximum Likelihood Estimation 17

2.3. Learning Discrete Decisions with Reinforcement Learning 17

3 Generating Better Lexical Distribution using Lexicon **21**

3.1. Introduction 21

3.2. Integrating Lexicons into NMT 23

3.2.1 Converting Lexicon Probabilities into Conditioned Predictive Probabilities 23

3.2.2 Combining Predictive Probabilities 24

3.2.3 Constructing Lexicon Probabilities 25

3.3. Experiment & Result 27
3.3.1 Settings 27
3.3.2 Effect of Integrating Lexicons 31
3.3.3 Comparison of Integration Methods 33
3.3.4 Result Analysis 35
3.4. Related Work 35
3.5. Conclusion & Future Work 37

4 An Investigation of Lexical Representations for Neural Machine Translation 39
4.1. Background 40
4.2. Embedding in NMT 42
4.3. Choosing Lexical Units 43
4.4. Embedding Granularity 44
4.5. Composing Word Embedding From Character Information 44
4.5.1 Basic Composition Function 45
4.5.2 Learnable Composition Function 45
4.6. Experiments 47
4.6.1 Settings 47
4.6.2 Effect of Combining Character Models into Word Lookup Models 49
4.6.3 Effect of Choosing Different Lexical Representations 50
4.6.4 Parameter Size vs. Translation Accuracy 51
4.6.5 Translating Rare and Long Input Words 53
4.7. Conclusion 54

5 Discovering Optimizing Lexical Units For Neural Machine Translation 58
5.1. Introduction 58
5.2. The Segmenting Encoder 60
5.3. Joint Learning Segmentation and Translation with Reinforcement Learning 62
5.3.1 Learning the Segmentation Decision 62
5.3.2 Priors and Tricks for Helping Reinforcement Learning 64
5.4. Experiments Setup 67

5.5. Preliminary Results 70
5.5.1 Analysis on the Translation Results 71
5.5.2 Analysis on the Segmentation Results 71
5.5.3 Difficulties in Learning Discrete Decision 72
5.6. Conclusion 73

6 Conclusion and Future Work **75**
6.1. Conclusion 75
6.2. Future Work 76
6.2.1 Investigation of Lexicon for Subword Models 76
6.2.2 Real World Experiments on Endangered Languages 77
6.2.3 Better Reinforcement Learning Technique for Learning Segmentation 77

List of Publications **80**

References **82**

List of Figures

1.1 Syntax-based translation rules for the word “eat”. In the “Hiero” syntax-based translation model [12], these are the most elementary rules to model this phenomenon in a natural way. 3

1.2 The example of word embeddings learned by the end-to-end neural network. The multi-dimensional words are plotted using principal component analysis to reduce high dimensional vectors into lower dimensional vectors. The words that are close in proximity should be close to each other. The image was taken directly from the word2vec data and visualized by using `word2vec-web-visualization`. . 4

2.1 The bidirectional LSTM topology for transducing sequences. The input sentence is encoded twice from the forward LSTM and the backward LSTM, and their information is combined using an MLP layer. y_i is the encoded representation at timestep i. 15

2.2 The REINFORCE algorithm implementation of policy gradient method. Note that this algorithm does not sample the actions several states ahead such as is done in Monte Carlo tree search. 19

3.1 An example of a mistake made by NMT on low-frequency content words. 22

3.2 Training curves for the baseline `attn` and the proposed `bias` method. 31

3.3 Attention matrices for baseline `attn` and proposed `bias` methods. Lighter colors indicate stronger attention between the words, and boxes surrounding words indicate the correct alignments. 33

3.4 The F-measure comparison of the proposed method vs the traditional attention method on BTEC dataset. At the low count unigram (rare words), the proposed system is better at generating than that of the traditional attentional method. The x-axis indicates the count of a word in the source corpus, indicating how rare the word is. 36

4.1 Composing word representation from its character distributed representation. Here the Japanese word 'meal' (食) is combined with both Japanese words for 'morning' (朝) and 'evening' (夕), yielding a correct Japanese words for 'breakfast' and 'dinner.' 40

5.1 The segmenting encoder. The bottom 3 components are the basic NMT system. The segmenter will produce a single segmentation decision at every timestep. Then, the composer will compose the sequence of character embedding to a word embedding. The learning is done by reinforcement learning. 59

5.2 The example of learned segmentation compared to the KyTea gold baseline segmentation. The generated segmentation (GEN) was able to learn some meaningful words such as years and noun phrases. . . . 70

List of Tables

2.1 Example of representing words with feature structures. Here words are represented as a multidimensional vector where each dimension represents a particular feature value. Value 1 represents the value of "Yes" and -1 represents the value of "No." 9
2.2 One hot word distribution for discrete representation. Each element is a 1.0 probability distribution in its index. 10
2.3 The comparison of words that are represented as vectors in Table 2.1. Here the Hamilton and Euclidean distance show a contrast different between words that are related or not. 11
2.4 Examples of closely related words of "video" in the learned embedding space. 12

3.1 Corpus details. 28
3.2 Accuracies for the baseline attentional NMT (attn) and the proposed bias-based method using the automatic (auto-bias) or hybrid (hyb-bias) dictionaries. Bold indicates a gain over the attn baseline, † indicates a significant increase at $p < 0.05$, and $*$ indicates $p < 0.10$. Traditional phrase-based (pbmt) and hierarchical phrase based (hiero) systems are shown for reference. 29
3.3 Examples where the proposed `auto-bias` improved over the baseline system `attn`. Underlines indicate words were mistaken in the baseline output but correct in the proposed model's output. 30
3.4 A comparison of the `bias` and `linear` lexicon integration methods on the automatic, manual, and hybrid lexicons. The first line without lexicon is the traditional attentional NMT. 34

4.1 The number of sentences in the corpus. All experiments are translating into English (en). There are a total of 4 different test sets for IWSLT, so we are showinga the total number of sentences for IWSLT. 47
4.2 The main results of the experiments. Numbers are the percentage of the BLEU score. Here bold means the best performance on particular language pair. Underline means the best BLEU scores on that column. † indicates a statistical significance [42] with $p < 0.01$ compared to the W-SW lookup baseline. ‡ indicates the same statistical significance but against the SW-SW baseline. 49
4.3 The relationship between the number of parameters, model ratio size (compared to lookup baseline) and the gain/loss of BLEU scores. These are the numbers for the SW-SW en-cs IWSLT experiment. 52
4.4 The word F-Measure accuracies from W-SW, SW-SW lookup baseline and the W-SW lookup+charngram systems in the IWSLT cs-en experiment across different word lengths and rarities. It can be seen that the lookup+charngram is robust in handling very long rare words translations as the gain is much higher as the words become rarer and longer. 53
4.5 The comparison of translating "difficult" words across several NMT systems. The first input is a mismatch input where word "California" was spelled wrongly in the test set. The second example is an example of translating 20 characters word that does not appear in the source corpus. The composite changram system with word inputs successfully retrieved the correct target word "surreal," showing the success of the bag-of-ngrams composition function to correctly guess the meaning of the difficult input word. 56

5.1 The results of the segmentation experiment on the KFTT corpus. We use the Gold segmentation from KyTea and Mecab as the standard gold segmentation on Japanese texts. Here we tried several different settings but unfortunately the evaluation results are not good yet. Here we explain the abbreviation of the settings: (1) lp is the length prior, (2) cp is the confidence penalty, (3) z_norm is the z-normalization, and (4) compose_char is to use the character embedding instead of states from bidirectional LSTM. 67
5.2 The segmentation F1 accuracy compared to the segmentation that is based on heuristic. The abbreviations of the settings follow from Table 5.1 69

Chapter 1

Introduction

Understanding natural language is a problem of understanding sentence that are composed by words. Words are the smallest elementary units possible to carry semantics. The process of performing some natural language processing task, such as automatically translating a sentence, begins with capturing a robust representation of an input sentence in a computer-processable data structure. These words themselves are not necessarily stored as-is, but it can be decomposed as smaller lexical units that are more efficient for learning and storing the whole data. This learned lexical representation is one of the essential step toward adaptive NMT systems that are robust in learning and dealing with difficult and unseen words on the fly.

1.1. Lexical Unit Acquisition for NMT

The process of translating a sentence in natural language starts with understanding the smallest parts from which it is composed (lexical units) [70]. For English, spaces are natural delimiters of words in sentences that can be used as semantic boundaries. This natural segmentation on space boundaries can serve as a definition of "words". However, for some languages such as Japanese and Chinese, where words can be defined in multiple ways [67], the process of choosing the lexical units is not trivial.

There have been several attempts to define what lexical units are the best to be processed for NMT systems. Some use plain whole words that are segmented using a segmenter or tokenizer according to intuitive human-defined boundaries. Some [48, 80] instead use characters. In most cases, the word-based systems using human-defined boundaries tend to beat the character-based systems given enough training data to ensure that the parameters of the words are well trained, but in some cases we don't have such data. On the other hand, character based models are very robust in handling unknown or unseen words, especially for languages with a big word vocabulary units [40]. However, it also means that the other parameters of the system, need to model the sequence of characters better, and usually makes the sequence longer and harder to learn. As a remedy to this problem, [75, 46] propose intermediate representations using subwords that can compromise between two approaches while taking the good aspects of both.

This thesis studies about the good representation of lexical units. Before describing the problem of the current state of the art and answering the question of what is a good way of representing them computationally, first we describe two main data representation formats for these units.

Discrete Representation of Lexical Units

In the framework of traditional statistical machine translation [45], words are treated as the atomic units, and sentence are broken down into rules for the translation engine [12]. However, because of this discrete representation, there exists a problem of distinguishing between words that should be close in proximity (e.g. "eat" and "ate") and words that are totally different ("eat" and "tree"). The system should be intelligent enough to distinguish them, which is a non-trivial task.

Given that the system is robust enough in distinguishing between words that are similar and dissimilar, we also have a problem in storing them in a computationally tractable way. For example let us examine the rule in Figure 1.1. There we have three different rules for translating the elementary word "eat", but because the tenses are different, we need to enumerate every verb in English with verbose string rules. Moreover it is difficult to store the translation model on limited-resource devices such as mobile phones for offline processing that can be used in remote areas. We will discuss this further in Chapter 2.

There are several drawbacks in representing word as the most basic units. First, we are treating words that should convey closer meaning (e.g. boy and man) equally different to the words that are vastly different in meaning (e.g. boy and tree). This is suboptimal because the context indicating similarities of words boy and man can not

English Grammar	Japanese Grammar
X_0 eats X_1	X_0 wa X_1 wo taberu
X_0 eating X_1	X_0 wa X_1 wo tabeteiru
X_0 ate X_1	X_0 wa X_1 wo tabeta

Figure 1.1: Syntax-based translation rules for the word "eat". In the "Hiero" syntax-based translation model [12], these are the most elementary rules to model this phenomenon in a natural way.

be modeled. Second, for words that have the same lemmas, but are lexically different because of the conjugation, are also treated as equally different (e.g. eat vs ate and eat vs country). There were many works on character-based models for SMT and morphology in SMT, (e.g. the factored translation model [43]), but solutions to the problems remain elusive. In phrase-based and syntax-based systems, the actual rule table should be constructed with a combination of several words together, forming rule tables with sizes that exponentially increase as more words are added to the system. This leads to the third problem of the discrete word based systems, the computationally intractable problem.

Continuous Representation of Lexical Units

Continuous lexical unit representations are becoming more popular with the breakthroughs in the computational linguistic research area involving neural networks (NN) and long short term memory networks (LSTM) [7, 78, 5, 60, 59]. Most of the traditional discrete lexical modeling systems are continuously being outperformed by the new end-to-end systems that are purely built using NNs.

In the new NN end-to-end architecture, words are modeled using a real-valued multi-dimensional vector, which we call the continuous representation. This is the generalization of the discrete modeling, as we can also achieve the discrete modeling by a one-hot distribution that has the size of the vocabulary (further in Chapter 2). As the concept of the meaning itself is abstract, we argue that it is better to model that with a continuous representation. In this system, we use words as the input of the NNs and the representation of words is learned by a gradient flow from back-propagation. This learned representation of words in forms of continuous vector is called the "word embedding".

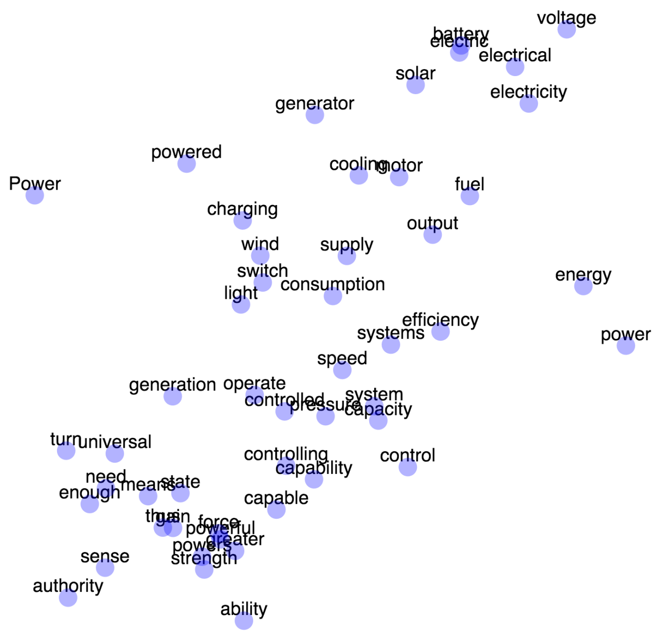


Figure 1.2: The example of word embeddings learned by the end-to-end neural network. The multi-dimensional words are plotted using principal component analysis to reduce high dimensional vectors into lower dimensional vectors. The words that are close in proximity should be close to each other. The image was taken directly from the word2vec data and visualized by using `word2vec-web-visualization`.

1.2. Improving Continuous Lexical Representation in NMT

In this section we discuss storing and representing sentences in NMT. Let us consider the sentence "今日私は最終試験をする" which means that "I have a final examination today." Based on this particular example, we list some problems that we attempt to solve in this thesis.

1. The first problem is that continuous representations make rare word (the words that have low counts in the training corpus) translation harder. Because of the low counts, the embedding parameters that are trained by using the general

lookup function will not be updated as much as the other non-rare words. This makes some words that are close to each other to be mistakenly generated. From the example we know that words such as “試験” which means an “examination” tends to be rarer than “今日” which means “today”. In general the vanilla NMT systems tend to fail in translating these kinds of words. More will be discussed in Chapter 3.

2. The second problem is that of how we properly model a particular word or lexical unit in a computer? The most basic modeling function is the lookup function that assigns a different vector to each lexical unit. This results in a massive vocabulary that usually needs to be pruned by discarding all rare words and replacing them with unknown words. This happens because words are being chosen to be the most basic units in the system. As the matter of fact, some of the complex words in a sentence can be represented or deduced by the smaller parts of lexical units from which it is composed. For example let us examine the word “今日”. Examining the character “今” which means “now” and “日” which means “day”, we can train a compositional embedding that deduce “now” and “day” are “today”.

   In addition, neural systems need to adapt to generate a special token that represents unknown words. This is done by replacing rare words with a special unknown token to simulate unknown words.l However, this reduces the amount of training data and systems are generating tokens that need to be processed further for the final output. In many cases, the neural machine translation system can not generate a continuous representation for an unknown word and just generalizes everything into the embedding of the special unknown token. Such embeddings can be properly deduced by looking at the constituent characters of the lexical unit. We will discuss this further in Chapter 4.

3. What are the best atomic units for representing a lexical unit in a language? Is it the word? Or the character? Or the subword? Subwords are designed to overcome the problems of the other choices. This is an interesting problem that we seek to solve in this thesis. While we have explained some pros and cons of using each variety of basic atomic unit, all of these were assumptions. First, let us define the variable that splits a stream of contiguous inputs into parts as seg-

mentation. All of the basic operations in the current neural machine translation systems operate on the segmented text (whether it is segmented into characters or some subwords). We are interested in discovering a segmentation that can further maximized the accuracy of the neural systems. The motivation of this study is that, we believe that there exist some segmentation that can be learned directly from parallel sentences. This will be further discussed in Chapter 5.

This thesis focuses on learning a good lexical representation for neural machine translation. We will start with the background theory of previous studies in the next chapter, and attempt to answer all the research questions on the previous sections in the chapters after that. Finally Chapter 6 will conclude the study and discuss what have we learned and what problems still remain. As was said in the abstract, there are three main components of the thesis, in each of which we study the better way of representing lexical units. Towards better NLP technology where words are not bounded by predefined vocabulary, we attempt to discover the lexical units in the third experiment. However, this is one first step toward open NLP system that can recognize and learn unknown words on the fly, with better estimation of meaning and boundaries, just like human adults when trying to guess the meaning of a sentence full of unknown vocabulary.

Chapter 2

Background

In this chapter, we describe all background technologies that support the main methods presented in this thesis. Some technologies are dated more than a decade ago, but are still used as the main foundation of Statistical Machine Translation (SMT) systems.

2.1. Discrete and Continuous Lexical Representation

As mentioned in Chapter 1, there are two ways to represent lexical units in computer systems. The first representation is a discrete character string (Section 2.1.1). The alternate representation is a dense continuous representation in the form of multi dimensional vectors (Section 2.1.2).

2.1.1 Word as Discrete Representation

Words are natural units of meaning. In the early days of SMT [9], SMT systems performed word]-level translation. SMT systems translate the source sentence (input) **f** into target sentence (output) **e**. This translation model is one of the most earliest data-driven translation models, and is based on the co-occurrence between source and target words in a sentence pair. The basis of this translation model is modeled by the following equation which is also the basis of all IBM translation models [9]:

$$p(\mathbf{e}|\mathbf{f}) = \sum_a p(\mathbf{e}, a|\mathbf{f}). \tag{2.1}$$

Word	Features		
	Is a Noun	Is a past tense?	Is a Verb?
eat	-1	-1	1
ate	-1	1	1
tree	1	-1	-1

Table 2.1: Example of representing words with feature structures. Here words are represented as a multidimensional vector where each dimension represents a particular feature value. Value 1 represents the value of "Yes" and -1 represents the value of "No."

Here a is the alignment of the words between **e** and **f**. The IBM models go all the way up to IBM model 5, which takes into account not only word alignments, but also other properties of word occurrence such as word fertility (IBM Model 3) and relative alignment (IBM model 4). All of these models operate on the word level. Based on the IBM models, the phrase-based [45] and syntax-based [27, 26, 14] translation systems were invented and became the state-of-the-art machine translation systems in industry. These systems treat words as the atomic units, and probabilities are counted based on their co-occurrences in the parallel corpus.

As discussed in Chapter 1, using a word string as the basic unit of NMT systems leads to the rare word problem. Each lexical unit should be trained enough so the NN parameters of NMT are robust in representing and generating it. Unfortunately, the frequency of occurrence of words in natural language follows Zipf's law. Indeed, infrequent words are learned only few times compared to that of the higher rank.

Ideally, SMT systems need to be robust and general enough in modeling the parallel corpus, and we can start with a better representation of words. We will discuss this further in the next section.

2.1.2 Word as Continuous Representation

Continuous representations are the better choice of representing meanings of words in a computer. This concept of meaning can be also composed by a few other concepts. For example, let us examine the word "ate", which is the past tense of the word eat. The meaning of this word can be composed by several defined attributes such as: (1)

word				
a	1	0	0	0
b	0	1	0	0
c	0	0	1	0
d	0	0	0	1

Table 2.2: One hot word distribution for discrete representation. Each element is a 1.0 probability distribution in its index.

is past tense (yes=1), (2) related to human need (yes=1), (3) positive word (70%). This is the example of representing meaning with a multidimensional vector. Each value of the attribute should thus be mapped to a real value, so it is easier to use this continuous meaning representation in neural models. The more dimensions we use to represent words, the more information can be stored inside the representation.

Before further describing the continuous representation of words, first let us revisit the discrete representation of a word. We can actually see the discrete representations as a one-hot vector with dimensions equal to the vocabulary size (Table 2.2). In that table, we use a total of 4 words as our vocabulary: {"a", "b", "c", "d" }. Each of the element in the Table 2.2 is represented as a vector of dimension H. We usually call H the feature dimension or embedding dimension of a word. We can define H as a number of handmade features we used to represent a word, or H as simply embedding dimension that can be learned from the end-to-end neural network systems. There are few ways to build a features and we will describe it in the following subsections.

Defined Features

In Table 2.1, there are 3 words, the example used in the Section 2.1.1, but here we preserve the semantic of the word by factorizing the word as feature vectors. We use 3 features "is the word a noun?", "is the word a past tense?", and "is the word a verb?" and assign the value accordingly. Based on these features we can model the close proximity of words "eat" and "ate", as opposed to the words "eat" and "tree".

Measuring the relationship of words can be done by measuring the distance of the vectors that represent it. Table 2.3 shows the comparison of the words using several vector distance comparison metrics. The equation for calculating the distance can be

Comparison	Cosine	Hamilton	Euclidean
eat & ate	0.33	2.00	2.00
eat & tree	-0.33	4.00	2.82
ate & tree	-1.00	6.00	3.46

Table 2.3: The comparison of words that are represented as vectors in Table 2.1. Here the Hamilton and Euclidean distance show a contrast different between words that are related or not.

seen in Equation 2.3, 2.4, and 2.4. It can be seen that "eat & ate" are closely related compared to "eat & tree," and "ate & tree." This is an example of using defined feature to represent words in a continuous representation. It is always possible to represent words with more features, so the representation of the word can be more robust. We can also use some non-binary features other than that of the examples in Table 2.3 such as probabilities of word appearing in a sentence to better represent the concept of meaning.

$$\text{cosine}(x, y) = \frac{x \cdot y}{|x||y|} \tag{2.2}$$

$$\text{hamilton}(x, y) = \sum_{i=1}^{N} |x_i - y_i| \tag{2.3}$$

$$\text{euclidean}(x, y) = \sqrt{\sum_{i=1}^{N} (x_i - y_i)^2} \tag{2.4}$$

Word Embedding

The defined features are easy to be understood by humans, however there are several problems in using the defined features. First it is expensive to annotate all the words in particular language with a set of defined features. These need to be annotated by a linguist with knowledge of the annotation standard, and it is of course time consuming. Next, we do not know whether a feature is needed in a particular problem or not. In machine learning, it is more beneficial if the machine can learn with the minimal amount of human supervision possible.

Word	Cosine Distance	Euclidean Distance
audio	0.704	1.187
videos	0.759	1.232
dvd	0.826	1.285
footage	0.869	1.318
console	0.875	1.323
consoles	0.890	1.334
picture	0.920	1.357
cd	0.922	1.358
computer	0.924	1.360
display	0.931	1.365
tv	0.934	1.367
playstation	0.936	1.368
broadcast	0.937	1.369
graphics	0.941	1.372

Table 2.4: Examples of closely related words of "video" in the learned embedding space.

Even though words in languages are abstract, there are still patterns the machine can extract given there are enough data and a good learning strategy. Previous studies [37, 29, 25, 13, 83] have shown that automatically learned continuous representations can be quite successful in extracting and representing patterns in some languages. Learning a word representation from a data produces automatically learned word embeddings. There are many method of learning word embeddings. With enough data and correct learning method, one can learn word embedding and visualize it[1]. We show the example of a closely related words of "video" from `word2vec` [59] embedding in Table 2.4.

In NMT systems, word embedding parameters are usually learned in an end-to-end fashion. Each discrete word in the training data is mapped into a vector of dimension H with a function embed(.):

[1]https://projector.tensorflow.org/

$$\text{embed}(x) = \text{lookup}[x]. \tag{2.5}$$

Here the lookup is a usually a hash map of word into a vector of dimension H. The number of parameters of embedding is usually $H \times |V|$, where $|V|$ is the total number of words in the vocabulary. This simple embedding function is usually used to power the architecture of the NMT and we will describe in on the next section.

2.2. Constructing Statistical Machine Translation Systems with Neural Networks

In this section we describe the new state of the art SMT systems that are constructed with an end-to-end neural network (NMT) [78]. Unlike the traditional SMT systems that consists of several separated components, the NMT systems are trained in single connected components of a neural network.

2.2.1 Recurrent Neural Networks

In particular, NMT often utilizes recurrent neural networks (RNNs) which also takes the last output of the network as an input (or state) to calculate the output at the next time-step. This property makes it convenient to model sequences, as we can model the probability of generating a particular token conditioned on the previously generated tokens

$$p(\mathbf{s}) = \prod_{i=1}^{|\mathbf{s}|} p(s_i | s_1 ... s_{i-1}) \tag{2.6}$$

$$\text{softmax}(\mathbf{x})_j = \frac{e^{x_j}}{\sum_i e^{x_i}}. \tag{2.7}$$

Here we can model the probability of a word s_i using a softmax function (Equation 2.7 over a vocabulary sized vector that is calculated using RNNs. Let h_i be an intermediate value that is calculated from the previous inputs. We define the probability of generating particular sequence with RNNs as below:

$$h_i \leftarrow \text{RNN}(h_{i-1}, s_i) \tag{2.8}$$

$$p(s_i|s_1...s_{i-1}) = \text{softmax}(h_i). \tag{2.9}$$

The choice of RNN is a problem of substituting the RNN function with an explicit instantiation in Equation 2.8. The most basic function will be using a concatenated input of a multi-layered perceptron (MLP) [24]. However, previous studies [33] has shown that this type of RNN suffered from modeling long distance sequences because of vanishing and exploding gradient problems.

In particular, some varieties of NMTs use Long Short Term Memory (LSTM) cells [33] to handle the long distance dependency problem. LSTM cells consists of 4 gates (input, output, forget, and cell) that control which information should flow through. This cell will also remember the previous information of a sequence in the LSTM state by preserving the cell state and the output state of the current time-step. Both of this states are used as the context of the next time-step. To put it formally, we define the calculation of both LSTM states (c_t & h_t) with input x_t by this LSTM cell formula:

$$\begin{aligned}
f_t =& \sigma(W_f x_t + U_f h_{t-1} + b_f) \\
i_t =& \sigma(W_i x_t + U_i h_{t-1} + b_i) \\
o_t =& \sigma(W_o x_t + U_o h_{t-1} + b_o) \\
c_t =& f_t \circ c_{t-1} + i_t \circ \sigma(W_c x_t + U_c h_{t-1} + b_c) \\
h_t =& o_t \circ \sigma(c_t).
\end{aligned} \tag{2.10}$$

This LSTM is particularly resilient toward the gradient vanishing or explosion problem. The problems of the previous RNN was that the gradients will become exponentially smaller or bigger overtime, depending on whether the value being multiplied is > 1 or < 1. The LSTM solved this problem by having a gated multiplication with the previous state. Particularly, it uses the sigmoid function which will squash the value of the multiplication to be in the range of 0 and 1.

2.2.2 Neural Machine Translation

The goal of machine translation is to translate a sequence of source words $F = f_1^{|F|}$ into a sequence of target words $E = e_1^{|E|}$. These words belong to the source vocabulary V_f, and the target vocabulary V_e respectively. NMT performs this translation by calculating the conditional probability $p_m(e_i|F, e_1^{i-1})$ of the ith target word e_i based on the source F and the preceding target words e_1^{i-1}. This is done by encoding the context $\langle F, e_1^{i-1}\rangle$ a fixed-width vector $\boldsymbol{\eta}_i$, and calculating the probability as follows (bold means a list of elements of the relevant type):

$$p_m(e_i|F, e_1^{i-1}) = \text{softmax}(W_s\boldsymbol{\eta}_i + \boldsymbol{b}_s), \tag{2.11}$$

where W_s and $\boldsymbol{b}_s$ are respectively weight matrix and bias vector parameters.

The exact variety of the NMT model depends on how we calculate $\boldsymbol{\eta}_i$ used as input. While there are many methods to perform this modeling, we opt to use attentional models [5], which focus on particular words in the source sentence when calculating the probability of e_i. These models represent the current state of the art in NMT, and are also convenient for use in our proposed method. Specifically, we use the method of [53], which we describe briefly here and refer readers to the original paper for details.

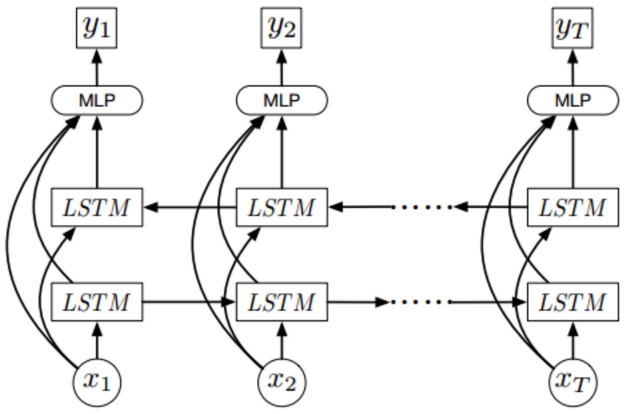


Figure 2.1: The bidirectional LSTM topology for transducing sequences. The input sentence is encoded twice from the forward LSTM and the backward LSTM, and their information is combined using an MLP layer. y_i is the encoded representation at timestep i.

First, an *encoder* converts the source sentence F into a matrix R where each column represents a single word in the input sentence as a continuous vector. This representation is generated using a bidirectional encoder (Figure 2.1[2])

$$\begin{aligned}\overrightarrow{\boldsymbol{r}}_j &= \text{enc}(\text{embed}(f_j), \overrightarrow{\boldsymbol{r}}_{j-1})\\ \overleftarrow{\boldsymbol{r}}_j &= \text{enc}(\text{embed}(f_j), \overleftarrow{\boldsymbol{r}}_{j+1})\\ \boldsymbol{r}_j &= [\overleftarrow{\boldsymbol{r}}_j; \overrightarrow{\boldsymbol{r}}_j].\end{aligned}$$

Here the embed$(\cdot)$ (Equation 2.5) function maps the words into a representation [6], and enc$(\cdot)$ is a stacking long short term memory (LSTM) neural network [33, 28, 78]. Finally we concatenate the two vectors $\overrightarrow{\boldsymbol{r}}_j$ and $\overleftarrow{\boldsymbol{r}}_j$ into a bidirectional representation $\boldsymbol{r}_j$. These vectors are concatenated into the matrix R where the jth column corresponds to $\boldsymbol{r}_j$.

Next, we generate the output one word at a time while referencing this encoded input sentence and tracking progress with a *decoder* LSTM. The decoder's hidden state $\boldsymbol{h}_i$ is a fixed-length continuous vector representing the previous target words e_1^{i-1}, initialized as $\boldsymbol{h}_0 = \boldsymbol{0}$. Based on this $\boldsymbol{h}_i$, we calculate a similarity vector $\boldsymbol{\alpha}_i$, with each element equal to

$$\alpha_{i,j} = \text{sim}(\boldsymbol{h}_i, \boldsymbol{r}_j). \tag{2.12}$$

sim$(\cdot)$ can be an arbitrary similarity function. Following [53, 5] we list some possible attention functions

- dot$(h_i, r_j) = h_i^T r_j$
- general$(h_i, r_j) = h_i^T W_a r_j$
- MLP$(h_i, r_j) = \tanh(h_i^T W_a + b_a) r_j$

Once the attention is calculated, we then normalize this into an *attention* vector α, which weights the amount of focus that we put on each word in the source sentence

$$\boldsymbol{a}_i = \text{softmax}(\boldsymbol{\alpha}_i). \tag{2.13}$$

This attention vector is then used to weight the encoded representation R to create a context vector $\boldsymbol{c_i}$ for the current timestep

$$\boldsymbol{c} = R\boldsymbol{a}.$$

[2]https://stackoverflow.com/questions/42151994/use-stateful-lstm-with-mini-batching-and-input-with-variable-time-steps-in-kera

Finally, we create $\boldsymbol{\eta}_i$ by concatenating the previous hidden state $\boldsymbol{h}_{i-1}$ with the context vector, and performing an affine transform

$$\boldsymbol{\eta}_i = W_\eta[\boldsymbol{h}_{i-1}; \boldsymbol{c}_i] + b_\eta,$$

Once we have this representation of the current state, we can calculate $p_m(e_i|F, e_1^{i-1})$ according to Equation (2.11). The next word e_i is chosen according to this probability, and we update the hidden state by inputting the chosen word into the decoder LSTM (slightly modify Equation 2.8)

$$\boldsymbol{h}_i = \text{enc}(\text{embed}(e_i), \boldsymbol{h}_{i-1}). \tag{2.14}$$

2.2.3 Learning NMT with Maximum Likelihood Estimation

To train the NMT system we usually use maximum likelihood estimation. At each time-step of decoding, a word that has the highest probability is chosen from Equation 2.11. During training, we do not usually produce the real outputs which require us to calculate the softmax of a big vector.

Equation 2.11 produces a probability distribution at decoding time and this probability can be compared to the gold-standard probability (i.e. the one hot word vector distribution of the correct word). Finally, if we define the NMT system's parameters θ with a parallel corpus D, we can then train the model by minimizing the negative log-likelihood of the training data:

$$\hat{\theta} = \underset{\theta}{\text{argmin}} \sum_{\langle F,\ E\rangle \in D} \sum_{i} - \log(p_m(e_i|F, e_1^{i-1}; \theta)). \tag{2.15}$$

2.3. Learning Discrete Decisions with Reinforcement Learning

In the previous sections we have an assumption that our network is fully connected and every part is differentiable, thus the learning of the network can be done by backpropagating the gradient through the network with some learning equations such as MLE (Section 2.2.3). However, there is a case where discrete decisions need to be made in the middle, as in cases such as predicting the segmentation of words and making a network in accordance to the discrete decision (will be discussed in Chapter 5). The part where such a discrete decision is made is usually using a non backpropagateable function such as “argmax”. The learning of this part of the network should be done directly from the environment and thus we use reinforcement learning.

Reinforcement learning has been proven [22] successful to train neural network agents where such agents need to make some discrete decision (action a) from its policy network π. The agent is trying to maximize the reward that reflects how well the agents perform at that time, and the reward is directly received from the environment.

One procedure of learning a policy network is the policy gradient method [**?**]. In the policy gradient method, we estimate the gradient of the network by trying to maximize the expectation of the reward function $\mathcal{R}(x)$ from the trainable policy network $\pi(x)$:

$$\begin{aligned}\nabla_\theta E_x[\mathcal{R}(x)] &= \nabla_\theta \sum_x \pi(x)\mathcal{R}(x) \\ &= \sum_x \nabla_\theta \pi(x)\mathcal{R}(x) \\ &= \sum_x \pi(x)\frac{\nabla_\theta \pi(x)}{\pi(x)}\mathcal{R}(x) \\ &= \sum \pi(x)\nabla_\theta \log \pi(x)\mathcal{R}(x) \\ &= E_x[\mathcal{R}(x)\nabla_\theta \log \pi(x)],, \qquad (2.16)\end{aligned}$$

For each data point, first we sample a sequence of actions (the sample) from $\pi(x)$. From each sample, we calculate the reward function $\mathcal{R}$ which is the reward of generating some sample (discrete actions) that should be a scalar value. This equation tells us how we should update the network parameter θ, scaled by the reward value. The second term of the equation (the ∇_θ part is the direction onto which we should update the parameter θ. After the update of the policy network π, the probability of generating samples that yield better reward will be slightly increased.

In the policy gradient method (Figure 2.2), it is important to encourage the network to do enough exploration. This is also the reason why we sample some actions from the policy network, instead of using greedy search. In some situation, we also need to prevent the early convergence of the network. The network that is too sharp (approaching the 1-hot distribution) will make reinforcement learning fail because no different

```
Initialize $\theta$ randomly
**repeat**
    Sample $a \sim \pi(x)$
    Calculate $\mathcal{R} \leftarrow \mathcal{R}(a, x)$ from the environment
    Calculate $\nabla_\theta E_x[\mathcal{R}(x)]$ according to Equation 2.16
    Update $\theta \leftarrow \theta + \alpha \nabla_\theta E_x[\mathcal{R}(x)]$
**until** converge
```

Figure 2.2: The REINFORCE algorithm implementation of policy gradient method. Note that this algorithm does not sample the actions several states ahead such as is done in Monte Carlo tree search.

action can be created by the policy network. Methods such as label smoothing and adding entropy to the distribution were highly successful in the previous studies [72].

In Figure 2.2, not only a single action is sampled from the policy network. Previous studies have shown that sampling multiple actions from the same policy network can encourage exploration and can further stabilize the training. There are some techniques to stabilize the training such as using the actor-critic methods [81], z-normalization [47], and baselines. We are not going into the details one-by-one, but in this thesis we use a baseline, which is basically a function that tries to estimate the expected future reward directly from the input using a single layered perceptron.

$$\beta(x) = W_\beta x + b_\beta \tag{2.17}$$

Chapter 3

Generating Better Lexical Distribution using Lexicon

Neural machine translation (NMT) often makes mistakes in translating low-frequency content words that are essential to understanding the meaning of the sentence. This chapter describes a method to alleviate this problem by augmenting NMT systems with discrete translation lexicons that efficiently encode translations of these low-frequency words. We describe a method to calculate the lexicon probability of the next word in the translation candidate by using the attention vector of the NMT model to select which source word lexical probabilities the model should focus on. We test two methods to combine this probability with the standard NMT probability: (1) using it as a bias, and (2) linear interpolation. Experiments on two corpora show an improvement of 2.0-2.3 BLEU and 0.13-0.44 NIST score, and faster convergence time [3].[1]

3.1. Introduction

NMT has recently gained popularity due to its ability to model the translation process end-to-end using a single probabilistic model, and for its state-of-the-art performance on several language pairs [53, 74].

One feature of NMT systems is that they treat each word in the vocabulary as a vector of continuous-valued numbers (Section 2.1.2). This is in contrast to more tra-

[1]Tools to replicate our experiments can be found at http://isw3.naist.jp/~philip-a/emnlp2016/index.html

Input:	I come from Tunisia.
Reference:	チュニジア の 出身です。
	Chunisia no shusshindesu.
	(I'm from Tunisia.)
System:	ノルウェー の 出身です。
	Noruue- no shusshindesu.
	(I'm from Norway.)

Figure 3.1: An example of a mistake made by NMT on low-frequency content words.

ditional SMT methods such as phrase-based machine translation (PBMT; [45]), which represent translations as discrete pairs of word strings in the source and target languages. The use of continuous representations is a major advantage, allowing NMT to share statistical power between similar words (e.g. "dog" and "cat") or contexts (e.g. "this is" and "that is"). However, this property also has a drawback in that NMT systems often mistranslate into words that seem natural in the context, but do not reflect the content of the source sentence. For example, Figure 3.1 is a sentence from our data where the NMT system mistakenly translated "Tunisia" into the word for "Norway." This variety of error is particularly serious because the content words that are often mistranslated by NMT are also the words that play a key role in determining the whole meaning of the sentence.

In contrast, PBMT and other traditional SMT methods tend to rarely make this kind of mistake. This is because they base their translations on discrete phrase mappings, which ensure that source words will be translated into a target word that has been observed as a translation at least once in the training data. In addition, because the discrete mappings are memorized explicitly, they can be learned efficiently from as little as a single instance (barring errors in word alignments). Thus we hypothesize that if we can incorporate a similar variety of information into NMT, this has the potential to alleviate problems with the previously mentioned fatal errors on low-frequency words.

In this chapter, we propose a simple, yet effective method to incorporate discrete, probabilistic lexicons as an additional information source in NMT (Section 3.2). First we demonstrate how to transform lexical translation probabilities (Section 3.2.1) into a predictive probability for the next word by utilizing attention vectors from attentional NMT models [5]. We then describe methods to incorporate this probability into NMT, either through linear interpolation with the NMT probabilities (Section 3.2.2) or as the bias to the NMT predictive distribution (Section 3.2.2). We construct these lexicon probabilities by using traditional word alignment methods on the training data (Section 3.2.3), other external parallel data resources such as a handmade dictionary (Section 3.2.3), or using a hybrid between the two (Section 3.2.3).

We perform experiments (Section 3.3) on two English-Japanese translation corpora to evaluate the method's utility in improving translation accuracy and reducing the time required for training.

3.2. Integrating Lexicons into NMT

In Section 2.2.2 we described how traditional NMT models calculate the probability of the next target word $p_m(e_i|e_1^{i-1}, F)$. Our goal in this chapter is to improve the accuracy of this probability estimate by incorporating information from discrete probabilistic lexicons. We assume that we have a lexicon that, given a source word f, assigns a probability $p_l(e|f)$ to target word e. For a source word f, this probability will generally be non-zero for a small number of translation candidates, and zero for the majority of words in the target vocabulary unit V_e. In this section, we first describe how we incorporate these probabilities into NMT, and explain how we actually obtain the $p_l(e|f)$ probabilities in Section 3.2.3.

3.2.1 Converting Lexicon Probabilities into Conditioned Predictive Probabilities

First, we need to convert lexical probabilities $p_l(e|f)$ for the individual words in the source sentence F to a form that can be used together with $p_m(e_i|e_1^{i-1}, F)$. Given input sentence F, we can construct a matrix in which each column corresponds to a word in the input sentence, each row corresponds to a word in the V_e, and the entry corresponds to the appropriate lexical probability:

$$L_F = \begin{bmatrix} p_l(e = 1|f_1) & \cdots & p_l(e = 1|f_{|F|}) \\ \vdots & \ddots & \vdots \\ p_l(e = |V_e||f_1) & \cdots & p_l(e = |V_e||f_{|F|}) \end{bmatrix}. \tag{3.1}$$

This matrix can be precomputed during the encoding stage because it only requires information about the source sentence F.

Next we convert this matrix into a predictive probability over the next word: $p_l(e_i|F, e_1^{i-1})$. To do so we use the alignment probability $\boldsymbol{a}$ from Equation (2.13) to weight each column of the L_F matrix:

$$p_l(e_i|F, e_1^{i-1}) = L_F \boldsymbol{a}_i = \begin{bmatrix} p_l(e=1|f_1) & \cdots & p_{lex}(e=1|f_{|F|}) \\ \vdots & \ddots & \vdots \\ p_l(e=V_e|f_1) & \cdots & p_{lex}(e=V_e|f_{|F|}) \end{bmatrix} \begin{bmatrix} a_{i,1} \\ \vdots \\ a_{i,|F|} \end{bmatrix}. \quad (3.2)$$

This calculation is similar to the way how attentional models calculate the context vector $\boldsymbol{c}_i$, but over a vector representing the probabilities of the target vocabulary, instead of the distributed representations of the source words. The process of involving $\boldsymbol{a}_i$ is important because at every time step i, the lexical probability $p_l(e_i|e_1^{i-1}, F)$ will be influenced by different source words.

3.2.2 Combining Predictive Probabilities

After calculating the lexicon predictive probability $p_l(e_i|e_1^{i-1}, F)$, next we need to integrate this probability with the NMT model probability $p_m(e_i|e_1^{i-1}, F)$. To do so, we examine two methods: (1) adding it as a bias, and (2) linear interpolation.

Model Bias

In our first `bias` method, we use $p_l(\cdot)$ to bias the probability distribution calculated by the vanilla NMT model. Specifically, we add a small constant ϵ to $p_l(\cdot)$, take the logarithm, and add this adjusted log probability to the input of the softmax as follows:

$$p_b(e_i|F, e_1^{i-1}) = \text{softmax}(W_s \boldsymbol{\eta}_i + b_s + \log(p_l(e_i|F, e_1^{i-1}) + \epsilon)). \quad (3.3)$$

We take the logarithm of $p_l(\cdot)$ so that the values will still be in the probability domain after the softmax is calculated, and add the hyper-parameter ϵ to prevent zero probabilities from becoming $-\infty$ after taking the log. When ϵ is small, the model will be more heavily biased towards using the lexicon, and when ϵ is larger the lexicon probabilities will be given less weight. We use $\epsilon = 0.001$ for this experiment.

Linear Interpolation

We also attempt to incorporate the two probabilities through linear interpolation between the standard NMT probability model probability $p_m(\cdot)$ and the lexicon probability $p_l(\cdot)$. We will call this the `linear` method, and define it as follows:

$$p_o(e_i|F, e_1^{i-1}) = \begin{bmatrix} p_l(e_i = 1|F, e_1^{i-1}) & p_m(e = 1|F, e_1^{i-1}) \\ \vdots & \vdots \\ p_l(e_i = |V_e||F, e_1^{i-1}) & p_m(e = |V_e||F, e_1^{i-1}) \end{bmatrix} \begin{bmatrix} \lambda \\ 1 - \lambda \end{bmatrix}, \quad (3.4)$$

where λ is an interpolation coefficient that is the result of the sigmoid function $\lambda = \text{sig}(x) = \frac{1}{1+e^{-x}}$. x is a learnable parameter, and the sigmoid function ensures that the final interpolation level falls between 0 and 1. We choose $x = 0$ ($\lambda = 0.5$) at the beginning of training.

This notation is partly inspired by [2] and [31] who use linear interpolation to merge a standard attentional model with a "copy" operator that copies a source word as-is into the target sentence. The main difference is that they use this to copy words into the output while our method uses it to influence the probabilities of all target words.[2]

3.2.3 Constructing Lexicon Probabilities

In the previous section, we have defined some ways to use predictive probabilities $p_l(e_i|F, e_1^{i-1})$ based on word-to-word lexical probabilities $p_l(e|f)$. Next, we define three ways to construct these lexical probabilities using automatically learned lexicons, handmade lexicons, or a combination of both.

Automatically Learned Lexicons

In traditional SMT systems, lexical translation probabilities are generally learned directly from parallel data in an unsupervised fashion using a model such as the IBM models [9, 68]. These models can be used to estimate the alignments and lexical

[2]Note that while we are only using two distributions here, in general, if we have a normalized interpolation parameters $\lambda_1, \cdots, \lambda_n$ where $\sum_i \lambda_i = 1$, we can merge n probability distributions.

translation probabilities $p_l(e|f)$ between the tokens of the two languages using the expectation maximization (EM) algorithm.

First in the expectation step, the algorithm estimates the expected count $c(e, f)$. In the maximization step, lexical probabilities are calculated by dividing the expected count by all possible counts:

$$p_{l,a}(e|f) = \frac{c(f, e)}{\sum_{\tilde{e}} c(f, \tilde{e})},$$

The IBM models vary in level of refinement, with Model 1 relying solely on these lexical probabilities, and latter IBM models (Models 2, 3, 4, 5) introducing more sophisticated models of fertility and relative alignment. Even though IBM models also occasionally have problems when dealing with the rare words (e.g. "garbage collecting" effects [50]), traditional SMT systems generally achieve better translation accuracies of low-frequency words than NMT systems [78], indicating that these problems are less prominent than they are in NMT.

Note that in many cases, NMT limits the target vocabulary [35] for training speed or memory constraints, resulting in rare words not being covered by the NMT vocabulary V_e. Accordingly, we allocate the remaining probability assigned by the lexicon to the unknown word symbol $\langle\text{unk}\rangle$:

$$p_{l,a}(e = \langle\text{unk}\rangle|f) = 1 - \sum_{i \in V_e} p_{l,a}(e = i|f). \tag{3.5}$$

Manual Lexicons

In addition, for many language pairs, broad-coverage handmade dictionaries exist, and it is desirable that we be able to use the information included in them as well. Unlike automatically learned lexicons, however, handmade dictionaries generally do not contain translation probabilities. To construct the probability $p_l(e|f)$, we define the set of translations K_f existing in the dictionary for particular source word f, and assume a uniform distribution over these words:

$$p_{l,m}(e|f) = \begin{cases} \frac{1}{|K_f|} & \text{if } e \in K_f \\ 0 & \text{otherwise} \end{cases}.$$

Following Equation (3.5), unknown source words will assign their probability mass to the $\langle\text{unk}\rangle$ tag.

Hybrid Lexicons

Handmade lexicons have broad coverage of words but their probabilities might not be as accurate as the learned ones, particularly if the automatic lexicon is constructed on in-domain data. Thus, we also test a `hybrid` method where we use the handmade lexicons to complement the automatically learned lexicon.[3] [4] Specifically, inspired by phrase table fill-up used in PBMT systems [8], we use the probability of the automatically learned lexicons $p_{l,a}$ by default, and fall back to the handmade lexicons $p_{l,m}$ only for uncovered words:

$$p_{l,h}(e|f) = \begin{cases} p_{l,a}(e|f) & \text{if } f \text{ is covered} \\ p_{l,m}(e|f) & \text{otherwise} \end{cases} \tag{3.6}$$

3.3. Experiment & Result

In this section we first describe the setting of the NMT systems and the results of the experiment.

3.3.1 Settings

- **Dataset:** We perform experiments on two widely-used tasks for the English-to-Japanese language pair: KFTT [62] and BTEC [38]. KFTT is a collection of Wikipedia article about city of Kyoto and BTEC is a travel conversation corpus. BTEC is an easier translation task than KFTT, because KFTT covers a broader domain, has a larger vocabulary of rare words, and has relatively long sentences. The details of each corpus are depicted in Table 3.1.

  We tokenize English according to the Penn Treebank standard [56] and lowercase, and tokenize Japanese using KyTea [65]. We limit training sentence length up to 50 in both experiments and keep the test data at the original length. We replace words of frequency less than or equal to a threshold u in both languages

[3]Alternatively, we could imagine a method where we combined the training data and dictionary before training the word alignments to create the lexicon. We attempted this, and results were comparable to or worse than the fill-up method, so we use the fill-up method for the remainder of the chapter.

[4]While most words in the V_f will be covered by the learned lexicon, many words (13% in experiments) are still left uncovered due to alignment failures or other factors.

Data	Corpus	Sentence	Tokens	
			En	Ja
Train	BTEC	464K	3.60M	4.97M
	KFTT	377K	7.77M	8.04M
Dev	BTEC	510	3.8K	5.3K
	KFTT	1160	24.3K	26.8K
Test	BTEC	508	3.8K	5.5K
	KFTT	1169	26.0K	28.4K

Table 3.1: Corpus details.

with the $\langle$unk$\rangle$ symbol and exclude them from our vocabulary. We choose $u = 1$ for BTEC and $u = 3$ for KFTT, resulting in $|V_f| = 17.8$k, $|V_e| = 21.8$k for BTEC and $|V_f| = 48.2$k, $|V_e| = 49.1$k for KFTT.

- **NMT Systems:** We build the described models using the Chainer[5] toolkit. The depth of the stacking LSTM is $d = 4$ and hidden node size $h = 800$. We concatenate the forward and backward encodings (resulting in a 1600 dimension vector) and then perform a linear transformation to 800 dimensions.

  We train the system using the Adam [40] optimization method with the default settings: $\alpha = 1\text{e}{-3}, \beta_1 = 0.9, \beta_2 = 0.999, \epsilon = 1\text{e}{-8}$. Additionally, we add dropout [77] with drop rate $r = 0.2$ at the last layer of each stacking LSTM unit to prevent overfitting. We use a batch size of $B = 64$ and we run a total of $N = 14$ epochs for all data sets. We also use a "dot" attention as described in [53] because it is the simplest one. All of the experiments are conducted on a single GeForce GTX TITAN X GPU with a 12 GB memory cache.

  At test time, we use beam search with beam size $b = 5$. We follow [54] in replacing every unknown token at position i with the target token that maximizes the probability $p_{l,a}(e_i|f_j)$. We choose source word f_j according to the highest alignment score in Equation (2.13). This unknown word replacement is applied to both baseline and proposed systems. Finally, because NMT models tend to give higher probabilities to shorter sentences [15], we discount the probability

[5]`http://chainer.org/index.html`

System	BTEC			KFTT		
	BLEU	NIST	RECALL	BLEU	NIST	RECALL
`pbmt`	48.18	6.05	27.03	22.62	5.79	13.88
`hiero`	52.27	6.34	24.32	22.54	5.82	12.83
`attn`	48.31	5.98	17.39	20.86	5.15	17.68
`auto-bias`	**49.74***	**6.11***	**50.00**	**23.20**†	**5.59**†	**19.32**
`hyb-bias`	**50.34**†	**6.10***	**41.67**	**22.80**†	**5.55**†	16.67

Table 3.2: Accuracies for the baseline attentional NMT (attn) and the proposed bias-based method using the automatic (auto-bias) or hybrid (hyb-bias) dictionaries. Bold indicates a gain over the attn baseline, † indicates a significant increase at $p < 0.05$, and $*$ indicates $p < 0.10$. Traditional phrase-based (pbmt) and hierarchical phrase based (hiero) systems are shown for reference.

of ⟨EOS⟩ token by 10% to correct for this bias.

- **Traditional SMT Systems:** We also prepare two traditional SMT systems for comparison: a PBMT system [45] using Moses[6] [44], and a hierarchical phrase-based MT system [14] using Travatar[7] [63], Systems are built using the default settings, with models trained on the training data, and weights tuned on the development data.

- **Lexicons:** We use a total of 3 lexicons for the proposed method, and apply `bias` and `linear` method for all of them, totaling 6 experiments. The first lexicon (`auto`) is built on the training data using the automatically learned lexicon method of Section 3.2.3 separately for both the BTEC and KFTT experiments. Automatic alignment is performed using GIZA++ [68]. The second lexicon (`man`) is built using the popular English-Japanese dictionary Eijiro[8] with the manual lexicon method of Section 3.2.3. Eijiro contains 104K distinct word-to-word translation entries. The third lexicon (`hyb`) is built by combining the first and second lexicon with the hybrid method of Section 3.2.3.

[6]`http://www.statmt.org/moses/`
[7]`http://www.phontron.com/travatar/`
[8]`http://eijiro.jp`

- **Evaluation:** We use standard single reference BLEU-4 [69] to evaluate the translation performance. Additionally, we also use NIST [21], which is a measure that puts a particular focus on low-frequency word strings, and thus is sensitive to the low-frequency words we are focusing on in this chapter. We measure the statistical significant differences between systems using paired bootstrap resampling [42] with 10,000 iterations and measure statistical significance at the $p < 0.05$ and $p < 0.10$ levels.

  Additionally, we also calculate the recall of rare words from the references. We define "rare words" as words that appear less than eight times in the target training corpus or references, and measure the percentage of time they are recovered by each translation system.

Input	Do you have an opinion regarding extramarital affairs?
Reference	不倫 に 関して 意見 が あります か。 Furin ni kanshite iken ga arimasu ka.
`attn`	サッカー に 関する 意見 は あります か。 Sakkā ni kansuru iken wa arimasu ka. *(Do you have an opinion about soccer?)*
`auto-bias`	不倫 に 関して 意見 が あります か。 Furin ni kanshite iken ga arimasu ka. *(Do you have an opinion about extramarital affairs?)*
Input	Could you put these fragile things in a safe place?
Reference	この 壊れ物 を 安全な 場所 に 置いて もらえません か 。 Kono kowaremono o anzen'na basho ni oite moraemasen ka.
`attn`	貴重品 を 安全 に 出したい の ですが 。 Kichō-hin o anzen ni dashitai nodesuga. *(I'd like to safely put out these valuables.)*
`auto-bias`	この 壊れ物 を 安全な 場所 に 置いて もらえません か 。 Kono kowaremono o anzen'na basho ni oite moraemasen ka. *(Could you put these fragile things in a safe place?)*

Table 3.3: Examples where the proposed `auto-bias` improved over the baseline system `attn`. Underlines indicate words were mistaken in the baseline output but correct in the proposed model's output.

In this section, we first a detailed examination of the utility of the proposed `bias` method when used with the `auto` or `hyb` lexicons, which empirically gave the best results, and perform a comparison among the other lexicon integration methods in the following section. Table 3.2 shows the results of these methods, along with the corresponding baselines.

3.3.2 Effect of Integrating Lexicons

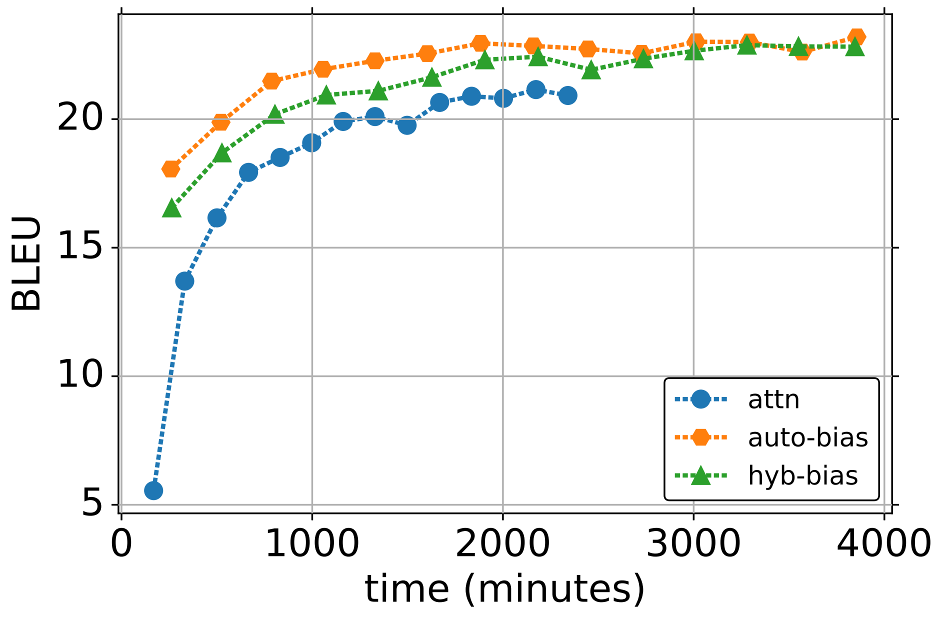


Figure 3.2: Training curves for the baseline `attn` and the proposed `bias` method.

First, compared to the baseline `attn`, our `bias` method achieved consistently higher scores on both test sets. In particular, the gains on the more difficult KFTT set are large, up to 2.3 BLEU, 0.44 NIST, and 30% Recall, demonstrating the utility of the proposed method in the face of more diverse content and fewer high-frequency words.

Compared to the traditional `pbmt` systems `hiero`, particularly on KFTT we can see that the proposed method allows the NMT system to exceed the traditional SMT methods in BLEU. This is despite the fact that we are not performing ensembling, which has proven to be essential to exceed traditional systems in several previous works [78, 53, 74]. Interestingly, despite gains in BLEU, the NMT methods still fall behind in NIST score on the KFTT data set, demonstrating that traditional SMT systems still tend to have a small advantage in translating lower-frequency words, despite the gains made by the proposed method.

In Table 3.3, we show some illustrative examples where the proposed method (`auto-bias`) was able to obtain a correct translation while the normal attentional model was not. The first example is a mistake in translating "extramarital affairs" into the Japanese equivalent of "soccer," entirely changing the main topic of the sentence. This is typical of the errors that we have observed NMT systems make (the mistake from Figure 3.1 is also from `attn`, and was fixed by our proposed method). The second example demonstrates how these mistakes can then affect the process of choosing the remaining words, propagating the error through the whole sentence.

Next, we examine the effect of the proposed method on the training time for each neural MT method, drawing training curves for the KFTT data in Figure 3.2. Here we can see that the proposed `bias` training methods achieve reasonable BLEU scores in the upper 10s even after the first iteration. In contrast, the baseline `attn` method has a BLEU score of around 5 after the first iteration, and takes significantly longer to approach values close to its maximal accuracy. This shows that by incorporating lexical probabilities, we can effectively bootstrap the learning of the NMT system, allowing it to approach an appropriate answer in a more timely fashion. [9]

It is also interesting to examine the alignment vectors produced by the baseline and proposed methods, a visualization of which we show in Figure 3.3. For this sentence, the outputs of both methods were both identical and correct, but we can see that the proposed method (right) placed sharper attention on the actual source word corresponding to content words in the target sentence. This trend of peakier attention distributions in the proposed method held throughout the corpus, with the per-word entropy of the attention vectors being 3.23 bits for `auto-bias`, compared with 3.81 bits for `attn`, indicating that the `auto-bias` method places more certainty in its attention decisions.

We realize that our baseline systems are slightly lower than PBMT and Hiero, as in [78] reported that their NMT systems are better than PBMT baseline. The reason behind this is that we have a lower settings of systems ($h = 800$ vs $h = 1000$, $b = 5$ vs $b = 12$, and no assembling vs 5 systems assembling). While we think it is important

[9]Note that these gains are despite the fact that one iteration of the proposed method takes a longer (167 minutes for `attn` vs. 275 minutes for `auto-bias`) due to the necessity to calculate and use the lexical probability matrix for each sentence. It also takes an additional 297 minutes to train the lexicon with GIZA++, but this can be greatly reduced with more efficient training methods [23].

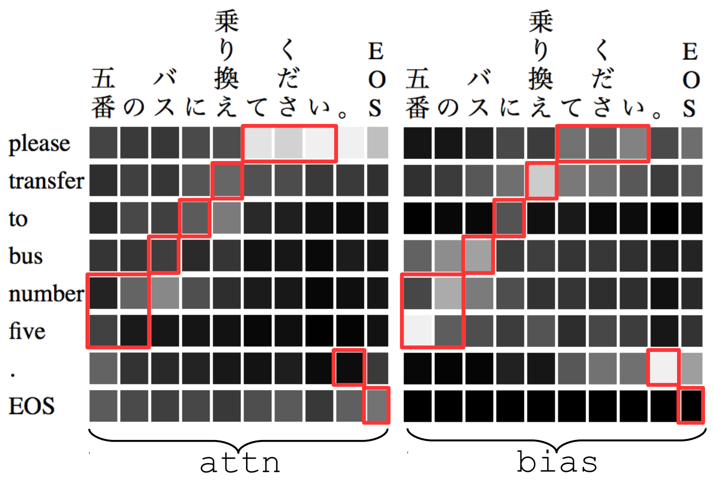


Figure 3.3: Attention matrices for baseline `attn` and proposed `bias` methods. Lighter colors indicate stronger attention between the words, and boxes surrounding words indicate the correct alignments.

that our baseline accuracies should be at least the same as PBMT baseline (which we achieved only in BTEC corpora), but we feel that it needs a great amount of engineering effort to do so. In general, however, we are certain that our baseline is comparable to the other systems, such systems described in [5] and [53].[10]

3.3.3 Comparison of Integration Methods

Finally, we perform a full comparison between the various methods for integrating lexicons into the translation process, with results shown in Table 3.4. In general the `bias` method improves accuracy for the `auto` and `hyb` lexicon, but is less effective for the `man` lexicon. This is likely due to the fact that the manual lexicon, despite having broad coverage, did not sufficiently cover target-domain words (coverage of unique words in the source vocabulary was 35.3% and 9.7% for BTEC and KFTT respectively).

Interestingly, the trend is reversed for the `linear` method, with it improving `man` systems, but causing decreases when using the `auto` and `hyb` lexicons. This indicates

[10]Note that the implementation of this work is using old technology and there might be a possible bugs. As of 2018, we find that the newest implementation of our NMT baseline is far better than the Hiero and PBMT system. The results of this method still show the same trend as shown in [64].

(a) BTEC

Lexicon	BLEU		NIST	
	`bias`	`linear`	`bias`	`linear`
-	48.31		5.98	
`auto`	**49.74***	47.97	**6.11**	5.90
`man`	**49.08**	**51.04**†	**6.03***	**6.14**†
`hyb`	**50.34**†	**49.27**	**6.10***	5.94

(b) KFTT

Lexicon	BLEU		NIST	
	`bias`	`linear`	`bias`	`linear`
-	20.86		5.15	
`auto`	**23.20**†	18.19	**5.59**†	4.61
`man`	20.78	**20.88**	5.12	5.11
`hyb`	**22.80**†	20.33	**5.55**†	5.03

Table 3.4: A comparison of the `bias` and `linear` lexicon integration methods on the automatic, manual, and hybrid lexicons. The first line without lexicon is the traditional attentional NMT.

that the `linear` method is more suited for cases where the lexicon does not closely match the target domain, and plays a more complementary role. Compared to the log-linear modeling of `bias`, which strictly enforces constraints imposed by the lexicon distribution [41], linear interpolation is intuitively more appropriate for integrating this type of complimentary information.

On the other hand, the performance of linear interpolation was generally lower than that of the bias method. One potential reason for this is the fact that we use a constant interpolation coefficient that was set fixed in every context. [31] have recently developed methods to use the context information from the decoder to calculate the different interpolation coefficients for every decoding step, and it is possible that introducing these methods would improve our results.

We additionally train the baseline `attn` and `auto-bias` systems with a parallel data that is created by combining the manual dictionary Eijiro and training data together as complimentary comparison. This scenario will train an NMT system with bigger vocabulary and with more training examples. The results of this experiment (`attn`/`auto-bias`) are 48.11/49.71 BLEU for BTEC and 18.75/20.54 BLEU for KFTT, which are comparable but sligthly lower to that of the original training scenario, showing that our method is better than the standard way of using dictionary in training SMT systems.

To test whether the proposed method is useful on larger data sets, we also performed follow-up experiments on the larger Japanese-English ASPEC dataset [61] that consist of 2 million training examples, 63 million tokens, and 81,000 vocabulary size. We gained an improvement in BLEU score from 20.82 using the `attn` baseline to 22.66 using the `auto-bias` proposed method. This experiment shows that our method scales to larger datasets.

3.3.4 Result Analysis

We add an additional analysis using [1] toolkit to further analyze the results of generated 4-gram as additional to our recall metrics over the rare words translation. The results is shown in Figure 3.4. The results in the table further confirm our claim that the lexicon can increase not only the recall over the rare word translation, but also the overall ngram count F-measure, and thus benefiting the BLEU score.

3.4. Related Work

From the beginning of work on NMT, unknown words that do not exist in the system vocabulary have been focused on as a weakness of these systems. Early methods to handle these unknown words replaced them with appropriate words in the target vocabulary [35, 54] according to a lexicon similar to the one used in this work. In contrast to our work, these only handle unknown words and do not incorporate information from the lexicon in the learning procedure.

There have also been other approaches that incorporate models that learn when to copy words as-is into the target language [2, 31, 32]. These models are similar to the `linear` approach of Section 3.2.2, but are only applicable to words that can be copied as-is into the target language. In fact, these models can be thought of as a subclass of the proposed approach that use a lexicon that assigns a all its probability to target words

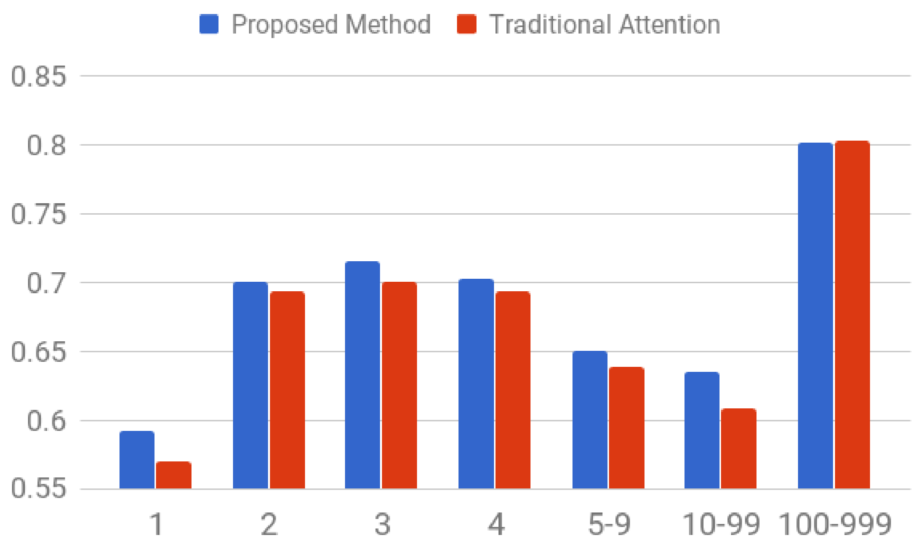


Figure 3.4: The F-measure comparison of the proposed method vs the traditional attention method on BTEC dataset. At the low count unigram (rare words), the proposed system is better at generating than that of the traditional attentional method. The x-axis indicates the count of a word in the source corpus, indicating how rare the word is.

that are the same as the source. On the other hand, while we are simply using a static interpolation coefficient λ, these works generally have a more sophisticated method for choosing the interpolation between the standard and "copy" models. Incorporating these into our `linear` method is a promising avenue for future work.

In addition [58] have also recently proposed a similar approach by limiting the number of vocabulary being predicted by each batch or sentence. This vocabulary is made by considering the original HMM alignments gathered from the training corpus. Basically, this method is a specific version of our bias method that gives some of the vocabulary a bias of negative infinity and all other vocabulary a uniform distribution. Our method improves over this by considering actual translation probabilities, and also considering the attention vector when deciding how to combine these probabilities.

Finally, there have been a number of recent works that improve accuracy of low-frequency words using character-based translation models [51, 18, 16]. However, [52] have found that even when using character-based models, incorporating information about words allows for gains in translation accuracy, and it is likely that our lexicon-based method could result in improvements in these hybrid systems as well.

3.5. Conclusion & Future Work

In this chapter, we have proposed a method to incorporate discrete probabilistic lexicons into NMT systems to solve the difficulties that NMT systems have demonstrated with low-frequency words. As a result, we achieved substantial increases in BLEU (2.0-2.3) and NIST (0.13-0.44) scores, and observed qualitative improvements in the translations of content words. This further strengthen the claim of the thesis that we have improved the ability of NMT system to represent/producing words, especially unknown words. As we can see that the representation of the target word can be further be more accurate by adding a prior predictive probabilities from the traditional SMT systems.

For future work, we are interested in conducting the experiments on larger-scale translation tasks. We also plan to do subjective evaluation, as we expect that improvements in content word translation are critical to subjective impressions of translation results. Finally, we are also interested in improvements to the `linear` method where λ is calculated based on the context, instead of using a fixed value.

Chapter 4

An Investigation of Lexical Representations for Neural Machine Translation

There are now a plethora of ways to represent source words in neural machine translation (NMT) systems, leading to a wide variety of design decisions and combinations thereof for NMT system builders. These choices include: (1) tokenization granularity – whether to represent words themselves or split into subword units, (2) embedding granularity – whether to simply look up token representations or use a character based composition function to capture regularities in spelling, (3) embedding method – if using character-based composition functions, what method should be used. However, while many options exist for each of these questions, they are largely evaluated in isolation, and thus it is unclear which formula is most effective for this task. In this paper we perform a comparison of various ways to represent source language words in neural machine translation, and find that over three language pairs, methods that sum word or subword embeddings with character-derived embeddings consistently lead to significant gains in accuracy over other options.

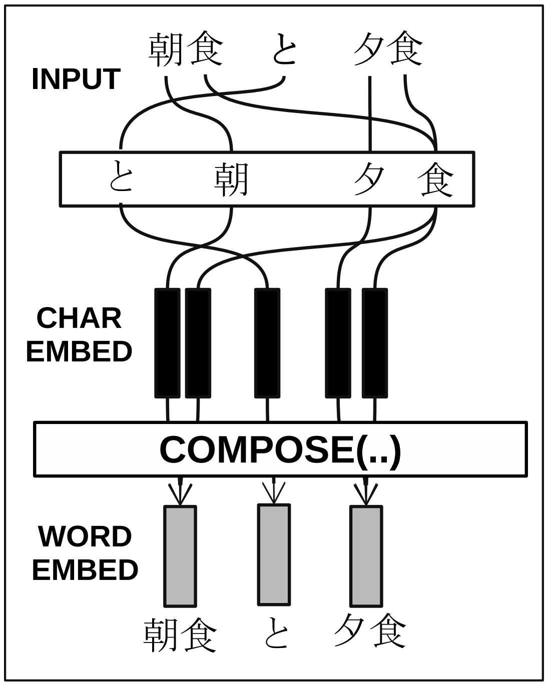


Figure 4.1: Composing word representation from its character distributed representation. Here the Japanese word ‘meal’ (食) is combined with both Japanese words for ‘morning’ (朝) and ‘evening’ (夕), yielding a correct Japanese words for ‘breakfast’ and ‘dinner.’

4.1. Background

Neural machine translation (NMT; Section 4.2, [36, 78, 5]) is now the de-facto state-of-the-art in machine translation (MT) technology, generally achieving superior accuracy compared to that of its predecessors, phrase-based and syntax-based SMT. There are a large number of ways to construct NMT systems, but all have one thing in common: treating the input tokens as multi-dimensional vectors. This is conducive to further processing within the neural net, and also allows for natural modeling of similarity of words along multiple dimensions, aiding generality.

Originally, the NMT systems were based on word-level input where we choose the input units were full words in the predefined vocabulary [78]. However, these kind of NMT systems were not able to handle unknown/rare words properly due to problems of sparsity and computational infeasibility of prediction over large output spaces. As a remedy to this, it is common to alter the granularity of segmenting the MT input or output into tokens, using sub-word units derived through unsupervised methods [**?**], or in the extreme case characters [49].

However, this does not change the fact that words are natural units of meaning and space markers often delimit these semantically meaningful word boundaries[1]. From a practical perspective, previous work has demonstrated that operating on the word level is preferable, given that some intelligent method of handling the sparsity caused thereby is used [57].

Once we decide the unit with which we model words in NLP applications such as NMT, there is also the question of how we turn each word into a distributed representation. The standard way of doing so is to simply have a single embedding vector for each token. However, the characters inside words are often highly indicative of the meaning of the words themselves [17, 48, 4], and several works have noted that we can gain by creating word representations from the constituent characters, as shown in Figure 4.1 [51]. Alternatively, [52] propose a method that uses word embeddings for high-frequency words, and character-derived embeddings for low-frequency words, falling back to characters only when sparsity entails that we should do so.

Finally, given that we use a character-derived representation in some part of our model, it becomes necessary to choose exactly which method we use for this purpose. There are a number of methods to do so, which we describe further in Section 4.5.

In summary, there are three major design decisions that go into how we represent words in neural MT: (1) tokenization granularity – whether to represent words themselves or split into subword units, (2) embedding granularity – whether to simply look up token representations or use a character based composition function to capture regularities in spelling, (3) embedding method – if using character-based composition functions, what method should be used. The major objective of this work is to perform a comprehensive investigation of the effect of these design choices, and how they interact, with the goal of providing recommendations for how words should be represented in NMT going forward.[2]

[1]This is not true for every language in the world, but for languages where word boundaries are not clear (e.g. Chinese or Japanese), supervised word segmenters are often available.

[2]Specifically, we focus on representing words on the *source* side of NMT systems, which precludes the computational considerations of predicting target-side words based on composed representations,

Experiments (Section 4.6) over three language pairs demonstrate that the NMT systems that make use of both word embeddings and character-derived information achieve the best performance in most of the language pairs. We also found that incorporating character information to baseline systems almost never hurts overall system performance, and often leads to significant gains. In addition we also find that on average the best composition function is one that makes use of the character bag of n-grams information [82], a method that is well known for being useful for training word embeddings, but seldom used in NMT.

4.2. Embedding in NMT

The goal of machine translation is to translate a sequence of source words $F = f_1^{|F|}$ into a sequence of target words $E = e_1^{|E|}$. These words belong to the source vocabulary V_f, and the target vocabulary V_e respectively.

First we need to convert each lexical unit f_j in the input sentence F into their continuous representations. Each lexical unit f_j in V_f is assigned to a continuous vector of fixed size $\overline{f}_j$, which is the word embedding. Let "embed" be a function that performs word embedding. There are many different implementations of the function "embed" and we will discuss each of them in the next section.

$$\overline{f}_j = \text{embed}(f_j) \tag{4.1}$$

After embedding the words we use the bidirectional encoder to produce the contextualized word representations in both directions.

$$\begin{aligned} \overrightarrow{\boldsymbol{r}}_j &= \text{enc}(\overline{f}_j, \overrightarrow{\boldsymbol{r}}_{j-1}) \\ \overleftarrow{\boldsymbol{r}}_j &= \text{enc}(\overline{f}_j, \overleftarrow{\boldsymbol{r}}_{j+1}) \\ \boldsymbol{r}_j &= [\overleftarrow{\boldsymbol{r}}_j; \overrightarrow{\boldsymbol{r}}_j]. \end{aligned} \tag{4.2}$$

enc$(\cdot)$ is a usually a stacked long short term memory (LSTM) neural network [33, 28, 78]. Finally we concatenate the two vectors $\overrightarrow{\boldsymbol{r}}_j$ and $\overleftarrow{\boldsymbol{r}}_j$ into a bidirectional representation $\boldsymbol{r}_j$. These vectors $\boldsymbol{r}_j$ are the contextual informations that contain the information of the source sentence and are used to initialize and generate words using the NMT decoder.

which is computationally more difficult than simply encoding words on the source side [79].

4.3. Choosing Lexical Units

The choice of lexical units of vocabulary V_f is one of the challenge that can define whether an NMT model is robust in handling unknown word or not. As mentioned in Section 1, the main purpose of modifying the lexical units of a sentence is to overcome the low counts data problem that is created because there are so many unique words in one language. For example, let us examine the word "study" and "studies". In NMT, they are represented by two different set of parameters. Even their meaning could be captured during training and there is a high possibility that they will be clustered in the same category, but this still creates a problem that we also need to learn the same representation for other correlated words such as "studied" and "studying."

To overcome this problem, a set of fixed size lexical units V_f are chosen using some unsupervised segmentation algorithm. BPE/The BPE algorithm [**?**] is based on merging two lexical units that has the highest occurrences. On the other hand, [46]'s segmentation algorithm is finding a set of lexical units that maximizes the unigram likelihood of the training corpus. Let $\boldsymbol{x}$ be a subword that is composed by several character $x_1, x_2, ..., x_n$. The probability of subword $\boldsymbol{x}$ is a unigram probability of each lexical units.

$$P(\boldsymbol{x}) = \prod_{i=1}^{N} p(x_i) \tag{4.3}$$

$$\forall_i x_i \in V_f, \sum_{x \in V_f} p(x) = 1$$

The segmentation algorithm starts from the character tokens (subword of size one). At each iteration, a new subword that maximizes Equation 4.3 if formed by merging two subwords together. We stop after we reach some number of merge operations or we have reached the desired vocabulary size. In this paper, we use the segmentation algorithm of [46] as our choice of subword based system to define the lexical units of the vocabulary V_f although other choices such as byte-pair encoding would be equally appropriate.

4.4. Embedding Granularity

The word embedding function is a function that maps a single string (word) into a continuous representation (a vector). The most common embedding function in NMT is the lookup embedding which assign a different set of continuous parameters to a different lexical unit.

$$\text{embed\_lookup}(f_j) = \text{lookup}[f_j] \tag{4.4}$$

The lookup embedding function only looks at the whole word only. On the other hand there are other composition functions that compose characters into word. Composing word from characters is inspired by the fact that the meaning of a certain word (lexical unit) can be derived by looking at the elements that are composing it [70]. In languages with many different character units such as Japanese and Chinese that have several thousand characters, the information of the character units can become a strong context to predict the meaning of the words.

We go back to the description of Figure 4.1. There we compose “朝食” which means “breakfast” from the word “朝” which means morning and “食” which means meal. In the English language, we can associate that “morning meal” is a breakfast. Same goes for the “dinner” example.

4.5. Composing Word Embedding From Character Information

In this section we will describe several ways to implement the word embedding function at Equation 4.1 using character information. A number of previous studies have proposed several composition functions that model the word representation $\overline{f}_j$ given the character embeddings of token $\overline{f}_{j,k}$ and its surface form f_j. For example k is the counter for the character in word/subword. we list below several methods that we use in our empirical comparison below.[3]

[3]There are other ways of turning characters into words e.g. methods that use explicit morphology [55]. However, we do not cover this on our paper because using morphological information is out of this paper's scope.

4.5.1 Basic Composition Function

The basic composition functions are only a mathematical operator to reduce a list of vectors into a single vector. These composition functions have no parameters and are relatively cheap to compute. By using this, we are relying more on the encoder to learn the understand the representation of the composed lexical units.

Sum: This embedding function is simply summing all the character embedding inputs.

$$\text{embed\_sum}(f_j) = \sum_{k=1}^{|f_j|} \overline{f}_{j,k} \tag{4.5}$$

Average: This composition function has been used in a previous study [34]. It simply calculates the normalized sum of character embeddings.

$$\text{embed\_avg}(f_j) = \frac{\text{embed\_sum}(f_j)}{|f_j|} \tag{4.6}$$

Max: This embedding function constructs a word embedding by taking the maximum of each dimension over all the character embeddings. Let n be a counter that runs over the vector dimension. We construct the element of the produced embedding one dimension at one time.

$$\text{embed\_max}(f_j)[n] = \max_k \overline{f}_{j,k}[n] \tag{4.7}$$

4.5.2 Learnable Composition Function

This type of composition functions have some learning capabilities and become a special part of the NMT system that model the word embedding given the characters embedding as an input. The learnable composition function is basically more expensive to compute, but more robust in modeling the lexical units.

BiLSTM: Bidirectional LSTMs have been widely used to model a sequence. Following Equation 4.2, first we encode the character embeddings $\overline{f}_{j,1..|f_j|}$ using a recurrent LSTM network. Then, we take the last step of forward and backward encodings, concatenate them together, and project them using affine transformation [51]

$$\text{embed\_bilstm}(f_j) = \tanh(W * [\overrightarrow{\boldsymbol{r}}_{|f_j|}, \overleftarrow{\boldsymbol{r}}_1] + b). \tag{4.8}$$

<u>CNN</u>: Convolutional neural networks, firstly designed for images, now have been widely used to model a sentence [39, 84]. CNN is useful to capture the relationship between adjacent inputs. First we treat every hidden dimension of the input embedding as the input channels of the convolution neural network. We then construct a filter of $1 \times K$ with K the window size that slides from the beginning of the sentence until the end of the sentence (and 1 is used as clarity purpose only because sentence/word has only 1 dimension and). Each element $\overline{x}_j$ at the input channel n is extracted from the filter starting from the position j, and taking up to K elements (K-1 zero padding are appended at the right of the input sentence).

$$\overline{x}_j[n] = \text{tanh}(\left[\sum_{l=j}^{j+K} W_l \overline{f}_{j,l}[n]\right] + b[n])$$

Then we apply a single pooling layer to choose the best filtered value $\overline{x}_i$ over all strides, for every input channel:

$$\text{embed\_cnn}(f_j)[n] = \max_j \overline{x}_j[n] \tag{4.9}$$

<u>CharNGram</u>: This composition function takes a bag of characters n-grams into consideration by first collecting the counts, and takes in a linear projection onto the desired embedding dimension with σ non-linearity transformation [82]. Previously it was employed to capture the embedding of a sentence, but like the other composition functions, we can also use this to capture the embedding of a word from its characters.

First we need to define the vocabulary V_{f_c} of the character n-grams by decomposing every word in the source corpus. Then, we map every word into a count vector with a fixed dimension of $|V_{f_c}|$. Let c_f be a count of character n-grams in a sparse vector of dimension $|V_{f_c}|$ and N be the n-gram size.

$$c_f(f_j) = \text{bag\_of\_ngrams}(f_j, N, |V_{f_c}|)$$

Then we can use a tanh non-linearity to produce the composed word embedding.

$$\text{embed\_charngram}(f_j) = \text{tanh}(W * c_f(f_j) + b) \tag{4.10}$$

Dataset	Src	Train	Dev	Test
KFTT	ja	∼440K	1166	∼1.1K
IWSLT	cs	∼122K	480	∼5.2K
	ar	∼239K	887	∼5.6K

Table 4.1: The number of sentences in the corpus. All experiments are translating into English (en). There are a total of 4 different test sets for IWSLT, so we are showinga the total number of sentences for IWSLT.

4.6. Experiments

Following from our discussion in Section 1, there are three effects that we want to closely take a look on. Before, going into the details, first we describe all the settings we use in this experiments.

4.6.1 Settings

Dataset: In this experiment, we use 3 language pairs that are all translating into English from 2 different corpora. The first and second source languages are Czech (cs) and Arabic (ar) from the IWSLT corpus [11], a collections of news articles. The third source language is Japanese (ja) from KFTT corpus [62], a corpus crawled from Wikipedia in mainly article about Kyoto city. The details of the corpora can be seen in Table 4.1.

Preprocessing: We prepare the data as needed. To summarize, we are doing the tokenization, lowercasing, and the filtering process. First we run 3 different types of tokenizers for 4 different languages. To preprocess the English and Czech data we run the Moses tokenizer. For Arabic, we use the Stanford NLP tokenizer for Arabic tokens [30]. For Japanese, we use the KyTea tokenizer [65]. We remove all sentences that have 50 words or more from the training data.

We prepare the subword data that are trained jointly from both source and target training texts using [46]'s unigram-based algorithm. Note that we have a choice of source lexical units which is either full word (W) or subword (SW), but we always translate into subwords, which mitigates problems of generating unknown words and also reduces computational burden for calculating softmaxes over large vocabularies.[4] We set the joint vocabulary size to be 32,000. We choose 32,000 after finding that it yields the best overall translation qualities after doing preliminary experiments with 8K, 16K, and 32K settings. After subwords/full-words are generated on both sides, we replace every lexical units whose occurrence is less than 2 in the whole corpus with a special unknown token. We want to emphasize that we also do the same procedure on the source side. The reason is that when we get an unknown test word, we want to be able to calculate its representation from the characters. We append a special end-of-sentence token at the end of each sentence and *do not* append a special begin-of-sentence, or any token that marks the start/end of a word.

Baseline NMT Systems: We build LSTM-based attentional NMT systems [5, 53] with the XNMT [66] toolkit, using the default settings of the toolkit unless otherwise specified. The model has a single layer bidirectional LSTM encoder with a single layer LSTM decoder. We use 512 hidden dimension for all layers. We use the multi-layered perceptron attention as specified in [5]. Dropout of 50% is employed in the output LSTM layer to make the network more robust.

To train the system, we use the batch size of 2048 words, counting the total number of words on both source and target sides. We use Adam [40] trainer with an initial learning rate of 0.001, and halve the learning rate every epoch the model loss (Equation 2.15) goes up on the development set. We observe the number of decays the system made, and stop the system immediately at the 4th decay, rolling back to the model with the lowest development loss before performing testing.

Composition Functions: It is our interest to see which character composition functions produce the best translation results. As a representative of the de-facto standard in current NMT systems, we compare with SW-SW NMT systems using the `lookup` composition function (Equation 4.4). We additionally try all 6 composition functions in Section 4.5: `sum` (Equation 4.5), `avg` (Equation 4.6), `max` (Equation 4.7), `bilstm` (Equation 4.8), `conv` (Equation 4.9), and finally `charngram` (Equation 4.10). We use a window size and n-gram size of 4 for both convolution and charngram composition functions.

[4]Noteably, [**?**] find that joint training of subword units is highly advantageous as it leads to consistently sized units over the input and output languages. This advantage will be lost in methods that use word-sized units on the input side, so any gains achieved by the methods that use word-based encoding on the source side will be in spite of losing this inherent advantage.

Composition Function	cs-en		ar-en		ja-en	
	W-SW	SW-SW	W-SW	SW-SW	W-SW	SW-SW
lookup	21.12	23.24	29.52	30.27	22.78	22.54
avg	18.05	18.91	24.03	25.07	21.35	21.28
max	18.56	19.59	24.37	24.14	21.85	21.54
sum	18.77	19.29	24.79	25.06	21.59	21.62
bilstm	22.27†	21.26	28.11	28.00	21.57	21.42
conv	21.11	21.39	28.24	28.22	22.20	20.50
charngram	23.50†	22.89†	29.93†	29.35	22.35	22.51
lookup+avg	21.84†	23.91†‡	29.96†	**30.55**†	22.67	22.78
lookup+max	21.89†	23.68†‡	30.06†	30.36†	22.28	**23.18**‡
lookup+sum	22.53†	23.60†‡	29.98†	30.13†	22.76	22.33
lookup+bilstm	22.72†	23.62†‡	30.05†	30.37†	22.97	22.47
lookup+conv	21.32	23.47†	30.05†	30.28†	22.27	22.80
lookup+charngram	**23.99**†‡	23.78†‡	30.46†	30.37†	21.89	22.77

Table 4.2: The main results of the experiments. Numbers are the percentage of the BLEU score. Here bold means the best performance on particular language pair. Underline means the best BLEU scores on that column. † indicates a statistical significance [42] with $p < 0.01$ compared to the W-SW lookup baseline. ‡ indicates the same statistical significance but against the SW-SW baseline.

Additionally, for all 6 character composition functions, we combine them with the `lookup` word composition function by simply adding the embeddings together.
Evaluation: We use standard single reference BLEU-4 [69] to measure the translation accuracy.

4.6.2 Effect of Combining Character Models into Word Lookup Models

The full result of the experiments can be seen in Table 4.2, where we can observe the effects of different segmentation granularities, representation granularities, and composition functions.

First, comparing the `lookup` function with the character only composition functions, we can see that the results are mixed, with word-based composition function achieving the best results in ja-en, and the `charngram` and `bilstm` achieving better results in cs-en and `charngram` achieving better results in the ar-en experiments.

When we combine the character composition functions with the lookup composition function by simply adding the embeddings together, we can see that almost all experiments in the cs-en and ar-en settings show positive results, with only one experiment (lookup+sum for ar-en) achieving worse results than the lookup. This results show that adding character embedding information to the existing baseline does not hurt the accuracy, but can contribute significant gains in many cases.

The gains when using the W-SW lookup function are larger, with increases of up to 2.87 for cs-en, and 0.94 for ar-en. Interestingly, even when using subwords on the source side in the SW-SW model, we can still see gains of up to 0.75 and 0.19 for cs-en and ar-en respectively. This indicates that the use of subwords do not entirely alleviate the problems of sparsity, and there is still significant potential for improvement through the incorporation of character-level information when using subword representations.

However we do not see similar gains in the Japanese language experiments. One possible reason that the character informations does not yield large gains in the Japanese language experiments is the low average number of characters in a word in Japanese language. To be precise, there are in average 4.30 characters per word in the Czech, 3.43 characters per word in the Arabic, and only 1.41 characters per word in the Japanese.[5] This indicates that in our Japanese language experiment, the composed method is comparable to using character based translation. Thus, it is likely the case that incorporation of character-based representations will yield larger gains in languages with longer words, and fewer gains in languages with shorter words.

4.6.3 Effect of Choosing Different Lexical Representations

From the same Table 4.2 we can see that going from the word lexical representation into subword lexical representation already gave us big gain as reported in the previous studies of [**?**, 46]. This is actually expected because subword units are robust in handling unknown words in general, and can generally mitigates when unknown words

[5]It should be noted that the KyTea segmenter that we used in our experiments produces a relatively fine-grained segmentation, and other segmenters may yield different results. However the scope of comparing different segmenters is out of the scope for this paper.

appear. Moreover the subword units are relatively more stable due to overall number of characters per word to be not very short or very long, compared to that of the word lexical representation.

The same effect can be seen in the Japanese language. Overall, by using the subword units, gains are relatively more stable if we compared the composite systems with the lookup baseline with the same inputs. The best accuracies are gained by the lookup+max baseline with 0.64 BLEU score improvement and 90% statistical significant against the best lookup baseline.

4.6.4 Parameter Size vs. Translation Accuracy

As noted previously, for each composition function, the level of parameterization[6] differs. Here we are referring to the total size of the actual NMT model parameters. In Table 4.3, we provide statistics about the number of parameters in each model with respect to the translation accuracy.

First we look at the first part of the table where systems are using purely character-based embeddings to estimate the word embeddings. From this table we can see that in the SW-SW experiments, The models that use character embeddings are practically much smaller than the lookup baselines, but there can be a significant BLEU penalty incurred by decomposing the word model into a character-based model. The BiLSTM and the Convolution composition functions used in previous studies give a better estimate of word embeddings but still are inferior to that of the lookup method. The charngram model is the only exception, as it needs to store a different parameter vector for a different character ngram (part of words), so the charngram model is significantly bigger than that of the other character composition models.

On the other hand, by purely using the simple `avg`, `max`, and `sum`, composition functions with the lookup embedding, we consistently gain accuracy if we are using subwords as the lexical unit of our models. These composition functions add few additional parameters to the model and relatively cheap to calculate, so systems this represents a cheap yet effective way to improve accuracy of systems.

However, the best accuracy is achieved by the lookup & charngram system with

[6]Note that the lookup baseline parameter should actually have a tiny lesser parameter, our actual implementation includes some unused character embedding parameters and made the actual parameters size to be a bit bigger than it actually is.

System	#params	Ratio	ΔBLEU
lookup	32.3M	1.00	-
avg	19.2M	0.59	-4.33
max	19.2M	0.59	-3.65
sum	19.2M	0.59	-3.95
bilstm	20.8M	0.64	-2.36
conv	20.3M	0.63	-1.85
charngram	39.3M	1.22	-0.36
lookup+avg	32.3M	1.00	0.66
lookup+max	32.3M	1.00	0.44
lookup+sum	32.3M	1.00	0.36
lookup+bilstm	33.9M	1.05	0.38
lookup+conv	33.4M	1.03	0.23
lookup+charngram	52.4M	1.62	0.54

Table 4.3: The relationship between the number of parameters, model ratio size (compared to lookup baseline) and the gain/loss of BLEU scores. These are the numbers for the SW-SW en-cs IWSLT experiment.

word input. This is the most highly parameterized model, which also gives it room to more capture regularities in the spelling of words, and thus the fact that it achieves the highest accuracy is understandable.

From the parameter count to accuracy tradeoff point of view, there is no clear winner in the choosing the composition functions. The most effective is the lookup & charngram combination, but for models that require extremely small model size, purely

	Min Length			
Max Count	1	5	10	15
all	56.67/58.67/**59.51**	47.09/49.48/**50.24**	28.87/33.82/**35.49**	24.65/31.65/**34.36**
100	48.86/51.13/**51.97**	45.21/47.67/**48.45**	28.87/33.82/**35.49**	24.65/31.65/**34.36**
5	33.25/37.24/**38.32**	32.12/36.23/**37.22**	22.51/28.70/**30.98**	21.96/29.10/**32.70**
1	19.87/25.29/**26.62**	19.27/24.75/**26.13**	13.90/20.58/**22.94**	13.61/21.11/**24.75**

Table 4.4: The word F-Measure accuracies from W-SW, SW-SW lookup baseline and the W-SW lookup+charngram systems in the IWSLT cs-en experiment across different word lengths and rarities. It can be seen that the lookup+charngram is robust in handling very long rare words translations as the gain is much higher as the words become rarer and longer.

character-based methods may also be an option. [7]

4.6.5 Translating Rare and Long Input Words

As one of the potential merits of using character-based representations is the ability to model low-frequency words where data sparsity is a problem, we also measure the F-measure of rare and long words. To measure this we use the simple word aligner of [23] to retrieve the word alignment between the reference and the source of the test sentence. Then for every word in the reference sentence, we retrieve the aligned source words from the input. We do the same procedure for the hypotheses from all the systems. Then we calculate the F-measure of each bucket of source words, treating source words as matched if the hypothesis matches the reference translation.

Table 4.4 shows an experiment of 3 different systems (from left-to-right) the W-SW lookup baseline, the SW-SW lookup baseline and the W-SW charngram composite system. Going down the row indicates the rarer the words being measured, while going right through the column indicates the longer the words being measured. Experiments shows that the charngram systems are by far the most robust system in handling the rare and longer words. The changram composite system is 1% better translating words with frequency one (the rarest words to appear in the training corpus), 2.7% better at

[7]The same trends also happen to the W-SW baseline with Δ BLEU scores are a bit toward to a more positive numbers, but providing almost the same insights as the SW-SW experiments.

the long words (length 15) and 3.6% better at the combination of these compared to the de-facto standard SW-SW lookup baseline.

The real example is shown in the Table 4.5. Here we show two cases where input tokens are very difficult to translate. We include Google Translate results for reference as well[8]. The first input is the misspelled word "Californii" which should be California. The vanilla lookup NMT failed in translating this, making a copy of the previously generated output, a mistake that is common in poorly performing NMT systems. The SW-SW lookup baseline produce unrecognizable entities. However the composite charngram system predicts the output of the word correctly. The Google Translate system was able to generate a sensible and smart output by disregarding the mistaken word and produce the abbreviation of "California" instead.

The next example is translating a very long Czech input word *nejsurrealističtější*. Here the correct translation for this 20 character word is "surreal." The vanilla baseline, once again failed badly by producing a nonsense "cado-ro-ro-like" like. The SW-SW baseline and Google Translate (they presumably use a similar lexical units) were able to predict a close words but not the precise translation. Finally, the composite charngram system, once again showed its ability to translate difficult words by correctly predicting the "surreal" output.

4.7. Conclusion

In conclusion we have investigated a number of choices in representing source-side lexical units for NMT. We compared 13 ways to create token representation from characters combined with two types of inputs (word and subword). We believe our findings to be useful in suggesting best practices for modeling words in NMT systems.

First, when aiming for the best accuracy, we suggest to build composite NMT system that is based on the charngram composition method (Equation 4.10), combined with word-based input. For languages that have no natural segmentation, we suggest to first split the sentence into words by using a supervised segmenter.

Second, if we aim to quickly increase the current recipe for subword based NMT system, we can quickly train the system jointly with the character embeddings that are composed using the average composition function. Incorporating other composition

[8]Taken on July 2018 from https://translate.google.com/

functions also will provide comparable results.

Third, using subword units as the input of NMT systems overall yields much robust NMT system. We suggest to follow the common practice of using subword as inputs. Fourth, we find that using the basic lookup composition function already provide NMT with a strong results. However, it does take up significant memory. If our concern is the memory, we would suggest to train the bidirectional LSTM or convolution based composition functions. They are weaker than the lookup composition function (with subword input) but the number of parameters is greatly reduced.

For future work, we might want to investigate other ways to incorporate the character and word embeddings together (currently it is just summing the embedding together). It would also be interesting to perform a similar examination on target-side word embeddings, such as those proposed by [79, 71]. We also think that a further investigation of incorporating the morphology of a language and type of segmentation of the source sentence can also further affect the overall ability of the system in modeling the source language.

Input	Měl jsem jen 85 dolaru a skončil jsem v San Franciscu, **<u>Californii</u>** - tam jsem potkal přítele – a v 80. letech jsem začal pracovat pro organizace zabývající se AIDS.
Reference	I had a thumb, I had 85 dollars, and I ended up in San Francisco, **<u>California</u>** – met a lover – and back in the '80s, found it necessary to begin work on AIDS organizations.
W-SW lookup	I had only 85 dollars, and I ended up in San Francisco – **<u>I met a friend –</u>** I met a friend – and in the 1980s, I started working for AIDS organizations.
SW-SW lookup	I had just about 85 dollars, and I ended up in San Francisco, **<u>Calinidi</u>** – I met a friend there – and in the 1980s, I started working for AIDS organizations.
W-SW charngram composite	I had only about 85 bucks, and I ended up in San Francisco, **<u>California</u>** – I met a friend – and in the 1980s, I started working for AIDS organizations.
Google Translate	I had only $ 85, and I ended up in San Francisco, **<u>Calif.</u>** - I met a friend there - and in the 1980s I started working for organizations dealing with AIDS.
Input	Byl to **<u>nejsurrealističtější</u>** druh deja vu, jaké jsem kdy měl, protože jsem věděl předem, co je za rohem, ještě než jsem tam zatočil a světla to odhalila, jelikož jsem strávil měsíce mezi kulisami, když jsme film natáčeli.
Reference	And it was the most **<u>surreal</u>** kind of deja vu experience I've ever had, because I would know before I turned a corner what was going to be there before the lights of the vehicle actually revealed it, because I had walked the set for months when we were making the movie.
W-SW lookup	This was a kind of **<u>cado-ro-ro-like</u>** kind of speaker I've ever known, because I knew what was around the corner, even before I'd returned that, because I spent months amongst the moon when we filmed the film.
SW-SW lookup	It was an **<u>unreal</u>** species where I had ever been, because I knew what was around the corner, even before i turned there and light that revealed, because i spent months between *alisa*, when we filmed films.
W-SW charngram composite	It was the most **<u>surreal</u>** species upon what I've ever had, because I knew what was in the corner, until I knew what was happening there, and the lights revealed, because I spent the thousands of months when we filmed the film.
Google Translate	It was the most **<u>realistic</u>** type of deja vu I ever had because I knew in advance what was around the corner before I turned and the lights revealed it because I spent the months between the scenes when we were filming the film.

Table 4.5: The comparison of translating "difficult" words across several NMT systems. The first input is a mismatch input where word "California" was spelled wrongly in the test set. The second example is an example of translating 20 characters word that does not appear in the source corpus. The composite changram system with word inputs successfully retrieved the correct target word "surreal," showing the success of the bag-of-ngrams composition function to correctly guess the meaning of the difficult input word.

Chapter 5

Discovering Optimizing Lexical Units For Neural Machine Translation

We propose a novel model and method to discover a hidden segmentation from a parallel texts. Our approach is purely unsupervised and able to exploit hidden linguistic properties from a stream of unsegmented inputs to learn and improve the accuracy of the translation systems. We also able to purely learn some linguistically intelligent segmentation without any prior supervision which is very promising to salvage language properties from extinct and endangered languages. These segmentations are modeled as discrete decisions that are learned from an end-to-end neural network. We model a segmentation as a sequence of discrete decision made when reading an input from left to right and use reinforcement learning to directly learn it from the data. This was our first attempt to learn segmentation purely unsupervised from the NMT, there are still further room for improvement and further studies are needed.

5.1. Introduction

Word units are the basic input of the state-of-the-art neural machine translation (NMT) systems [5, 78, 53, 52]. Starting from the word inputs, the state-of-the-art NMT systems read the input from left-to-right with a gated recurrent neural network [78] that uses a memory cells such as long short term memory [33] to remember long dependencies. Previous studies show that NMT systems are able to surpass the accuracy of its predecessor phrase-based & syntax based statistical machine translation systems by

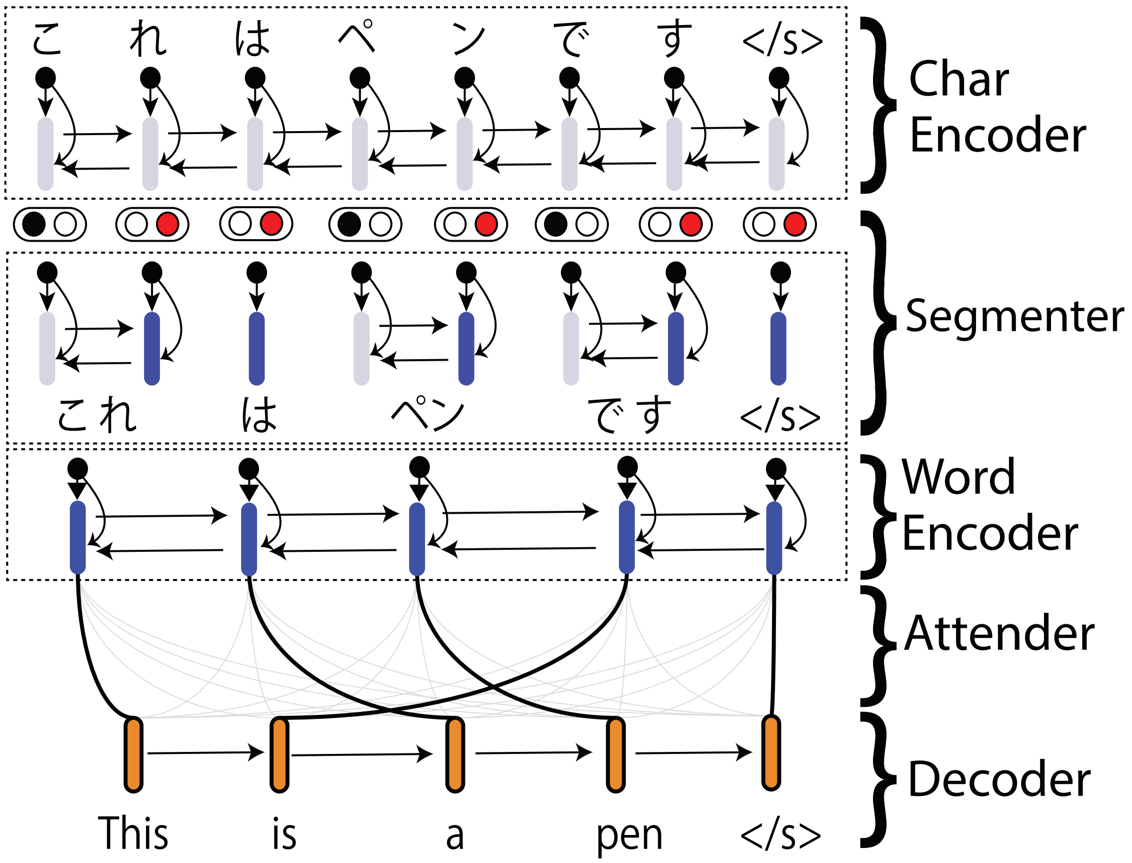


Figure 5.1: The segmenting encoder. The bottom 3 components are the basic NMT system. The segmenter will produce a single segmentation decision at every timestep. Then, the composer will compose the sequence of character embedding to a word embedding. The learning is done by reinforcement learning.

a quite far margins.

However, the most basic NMT system suffer from input sparsity. [74, 46] alleviates this problem by combining the most basic units (character) from a text into subwords. Here, we can control the sparsity of the input space for the system. While effective, this approach has a drawback because the produced subwords are uninterpretable by most humans. This makes that this approach can not be applied to the discovery of words unit in rare, endangered, and extinct languages.

Since babies, human are trained to identify words from sequence of speech. To do so infants must uncover at least some of the units that belong to their native language from a largely continuous stream of sounds in which words are seldom surrounded by pauses [73]. After these units are discovered, the acquisition of language starts with discovering a more complex linguistic properties such as entities, particles, and pronouns.

Based on that, this paper is motivated by mimicking the ability of language acquisition by infants. The most simplest task can be by discovering a hidden linguistic properties such as segmentation from parallel texts. Such texts/transcriptions are commonly made by recording speeches from native speakers that can still speak the language. Mostly, these languages are not well documented and it is somehow difficult to identify words from within the texts. In this paper, we propose a method to discover a hidden segmentation from a parallel texts.

On the other hand, segmentation has been one of the most prominent first step in the translation task. Previous studies [65, 12] showed the importance of segmentation toward the translation qualities. In this paper, we showed that the learned segmentation is more optimal than that of the unsupervised segmentation or even the resource heavy supervised segmentation.

Section 5.2 describes our proposed encoder that can perform and learn segmentation from parallel texts as depicted in Figure 5.1. Section 5.3 describes the detailed reinforcement learning process of the proposed model. Section 5.4 shows the generated segmentation results with addition translation and Section 5.6 finally concludes this paper.

5.2. The Segmenting Encoder

Our proposed segmentation model is a single decoder that both encodes and segments an unsegmented input embedding F into a segmented input $\mathcal{F}$ with a latent segmentation $\mathcal{S}$. Figure 5.1 illustrates the proposed segmenting encoder. First, on the char encoder module we project each token of the input F into its embedding space and run the bidirectional LSTM over the embedded input to get the positional encoding $\hat{F}_i$ which is a projection of the concatenation of both forward and backward encoding of the input token at time step i. The segmentation model (the black-red part of the segmenter) is a straightforward binary linear regression that takes the positional encoding

$\hat{F}_i$ as an input and outputs black decision (not segment) or red decision (segment):

$$P(S_i|F) = \text{softmax}(W_s * \hat{F}_i + b_s) \tag{5.1}$$

The derived segmentation decisions S are used to segment and encode the input accordingly with $\mathcal{F} = \text{segment}(F, \mathcal{S})$ Each element of the segmented $\mathcal{F}_j$ represents a word that is produced by the segmentation $\mathcal{S}$.

Conversely, we can calculate the probability of generating particular word from the segmenter $P(\mathcal{F}_j)$ as a probability of emitting the word F_j given the segmentation $\mathcal{S}$ that follows the Bernoulli distribution. This is because there should be several events of "non-segment" and followed by a single "segment" event to produce a single word.

Here, we pick all the character embeddings (i as a counter) starting from the last segmentation d_{j-1} decision until reaches the point of segmentation ($\mathcal{S}_i =$"segment") d_j:

$$P(\mathcal{F}_j|\mathcal{S}) = \prod_{i=d_{j-1}}^{d_j} P(\mathcal{S}_i|F) \tag{5.2}$$

The encoding of the segmented word $\hat{\mathcal{F}_j}$ is composed based on the composition of the character encodings $[\hat{F}_{d_{j-1}} ... \hat{F}_{d_j}]$. Figure 5.1 shows this process at the segmenter part. The Japanese characters (transliterated) [ko, re, wa, pe, n, de, su] are segmented into [kore, wa, pen, desu], which is the correct segmentation of this Japanese sentence. As there are many ways to compose the $\hat{\mathcal{F}_j}$, we currently simply run an additional stack LSTM over the character encodings $[\hat{F}_{d_{j-1}} ... \hat{F}_{d_j}]$ and get the last encoding states to represent the word representation $\hat{\mathcal{F}_j}$.

Finally, we run a separate stack LSTM over the word representation $[\hat{\mathcal{F}_1}, ... \hat{\mathcal{F}_{|\mathcal{F}|}}]$ to learn the relationship between each produced word vectors. This final results will be the final output of the segmenting encoder as shown in the "word encoder" part of Figure 5.1.

5.3. Joint Learning Segmentation and Translation with Reinforcement Learning

In order to learn a good and robust segmentation model, here we use reinforcement learning to train the binary classifier while also learning the translation model. We model the segmentation learning as a multi task learning whose objective is to find the best segmentation that yields the best translation. Equation 5.3 denotes the generative process of finding the best segmentation S given the the input sentence F and its reference E.

$$
\begin{aligned}
&\underset{\mathcal{S}}{\operatorname{argmax}}\{P(\mathcal{S}|F,E)\}\\
&\quad= \underset{\mathcal{S}}{\operatorname{argmax}}\left\{\frac{P(F,E|\mathcal{S})P(\mathcal{S})}{\cancel{P(F,E)}}\right\}\\
&\quad= \underset{\mathcal{S}}{\operatorname{argmax}}\{P(E|F,\mathcal{S})P(F|\mathcal{S})P(\mathcal{S})\}\\
&\quad= \underset{\mathcal{S}}{\operatorname{argmax}}\left\{P(E|F,\mathcal{S})\frac{P(\mathcal{S}|F)\cancel{P(F)}}{\cancel{P(\mathcal{S})}}\cancel{P(\mathcal{S})}\right\}\\
&\quad= \underset{\mathcal{S}}{\operatorname{argmax}}\{P(E|F,\mathcal{S})P(\mathcal{S}|F)\}
\end{aligned}
\tag{5.3}
$$

In details, the joint learning of the model is composed by two separates models: (1) **segmentation model** $P(\mathcal{S}|F)$ that produce the segmentations $\mathcal{S}$ given the input sentence F and (2) **translation model** $P(E|F,\mathcal{S})$ that use both the segmentation $\mathcal{S}$ onto the input sentence F to find the best translation E.

Before going into the detail to the technique of learning the segmentation $\mathcal{S}$, first we will describe and denote all the notations for the state of the art neural machine translation systems in Section 2.2.3 and finally the method of learning the segmentation in Section 5.3.1.

$$\hat{\theta}_{MLE} = \underset{\hat{\theta}}{\operatorname{argmin}} \sum_{\langle E,F\rangle} \mathcal{L}_{MLE}(F,E) \tag{5.4}$$

5.3.1 Learning the Segmentation Decision

Because the segmentation discrete decisions parameters cannot be learned with the end-to-end back propagation, we use the reinforcement learning technique that collect the rewards from the environment and learn the segmentation. As we are interested in a purely unsupervised training, we construct the reward purely based on the system. We treated the system as an environment that provide feedback to the reinforcement learning agent.

One way to construct reward will be using the evaluation measure of the translation outputs whenever systems produce translations based on some produced segmentation (e.g. BLEU) as done in some evaluation-metrics training for NMT [76]. However, this has a drawback on sacrificing parallelism as decoding of the NMT needs to be done per sentence basis and the calculation of the evaluation metric such as BLEU cannot be parallelized in GPU.

For the sake of maintaining efficiency and speed, we use the likelihood of the model as it is the most direct reward that is also being optimized by the system in Equation 5.4. We define the reward of generating target sentence E with a source sentence F as the normalized likelihood of the model[1]:

$$\mathcal{R}(F, E) = \frac{\log(P_{MLE}(E|F))}{|E|} \tag{5.5}$$

Next, we describe how we properly learn the best segmentation $\mathcal{S}$ from the system. First let us define $\mathcal{S}$ as a binary random variable with members $\{0, 1\}$ with 0 represents "no segmentation" and 1 represents "segmentation". Like most common practice in reinforcement learning, we sample the segmentation $\mathcal{S}$ directly from the policy network, given the source sentence F as an input:

$$[\mathcal{S}_1, ..., \mathcal{S}_{|F|}] \sim P(\mathcal{S}|F) \tag{5.6}$$

.

Following the REINFOCE algorithm, we collect all the probability of the sampled segmentation and calculate their gradients in respective to the cumulative rewards. In formal, let $\pi(\mathcal{S}_i) = P(\mathcal{S} = \mathcal{S}_i|F)$, the loss of the policy network π is calculated as follow:

$$\mathcal{L}_R(F, \mathcal{S}, E) = -\sum_{i=1}^{|F|} \pi(\mathcal{S}_i)[\mathcal{R}(F, E) - \beta(\hat{F}_i)] \tag{5.7}$$

[1]We normalize the likelihood because it is easier to control the number of the normalized version.

Here we subtract the reward with a baseline to reduce the variance of the reward. We calculate the baseline as a the expected cumulative future reward. To do so, we simply perform a linear regression from the encoded input (before segmentations take place):

$$\beta(F_i) = W_\beta * \hat{F}_i + b_\beta \tag{5.8}$$

The parameter of the baseline is trained by directly calculating the squared distance of the actual reward with the predicted reward:

$$\mathcal{L}_\beta(F, E) = \sum_{i=1}^{|F|} ||\mathcal{R}(F, E) - \beta(F_i)||_2 \tag{5.9}$$

It is important to make sure that we sever the connection of the main network and the baseline network when doing the join training. This is to prevent the network to overly trivialize the reward. The baseline value should be output as a constant that contains no gradient flow.

Finally, the final losses are composed by joining all the losses together:

$$\mathcal{L}_\theta = \sum_{\langle F,E \rangle} \mathcal{L}_{MLE}(F, E) + \mathcal{L}_R(F, \mathcal{S}, E) + \mathcal{L}_\beta(F, E), \tag{5.10}$$

and the parameter learning is simply search of parameters that minimizes the cumulative losses on Equation 5.10. In practice, we assign some scaling weight γ_R to the $\mathcal{L}_R$ to control its strength.

5.3.2 Priors and Tricks for Helping Reinforcement Learning

Reinforcement learning is notoriously difficult and sometimes needs proper sets of settings and tricks in order to make it work. In this paper, we introduce several priors and tricks for learning the segmentation. These priors are usually added as additional rewards and also used as additional losses (with the opposite sign) whenever possible.[2]

- **Length Prior** λ is the expected number of characters in a word. We can use this prior to control how many expected characters in the resulting segmentations.

[2]We assign a proper scaling factor γ to each of the prior so we can control its strength.

Normally we can calculate this from the known target corpus, given our input is a mystery language.

Given a stream of unsegmented input F and the length prior λ, we calculate the expected number of words in a sentence by simply taking a fraction of the total number of characters and λ.

$$\mu_\lambda = \frac{|F|}{\lambda} \tag{5.11}$$

This expected prior is then used to calculate the probability of some segmentation $\mathcal{S}$ happening, calculated based on the number of words produced by $\mathcal{S}$. Simply, we can use the Poisson distribution that contains the similar property:

$$P_\lambda(\mathcal{S}|F) \approx \text{POISSON}(|\mathcal{F}|, \text{k} = \mu_\lambda) \tag{5.12}$$

Integrating this probability is simply multiplying the reward with the prior with some scaling factor γ_λ:

$$\mathcal{R}(F, E) = \mathcal{R}(F, E) * P_\lambda(\mathcal{S}|F)^{\gamma_\lambda} \tag{5.13}$$

- **Global Fertility** is a measurement of proper number of produced target words given a particular source words. The proper global fertility is a single source word will contribute to also a single target word production. So we can calculate this by summing the attention value of all target words, and inspect the value of the fertility of each source words. This is to make sure that there is a proper amount segmentations given the total target words.

$$\mathcal{L}_F(F, E, \alpha) = -\sum_{i=1}^{|F|} \left[(1 - \sum_{j=1}^{|E|} \alpha_{i,j})^2 \right] \tag{5.14}$$

- **Confidence Penalty**: Following [72], we add additional penalty to the network with high confidence. This is an important penalty as the reinforcement learning usually tends to trivialize the discrete decision in the early iterations because the translation model that provide the reward is not trained yet. We basically sum the entropy of each probability distribution produced by the policy network, and turns the entropy into additional loss by adding the negative sign.[3]

$$\mathcal{L}_{CP}(F) = -\sum_{i=1}^{|F|} \sum_{S_i \in [0,1]} -\pi(S_i) \log \pi(S_i) \tag{5.15}$$

- **Epsilon Greedy**: A simple procedure to sample action spaces from a prior distribution. Here we use the same length prior λ to determine the average number of characters in a word. We sample a random number k from a poisson distribution and segment a k length character from the source input. This procedure is repeated until all characters are segmented.

$$a \sim \begin{cases} P(S|F) & \text{if uniform}(\epsilon) > 0.5 \\ Poisson(S|F, \lambda) & \text{otherwise} \end{cases}$$

The $Poisson$ distribution is built on the probability of some event k occurring on the specific rate of occurrence λ:

$$Poisson(\lambda, k) = \frac{\lambda^k e^{-\lambda}}{k!}$$

- **Z Normalization**: It is a common reinforcement learning tricks that normalize the rewards across all the items in the batch. We assume that the rewards in the batch has a 0 mean and variance of 1.

$$\mathcal{R} = \frac{\mathcal{R} - \mu(\mathcal{R})}{\sigma(\mathcal{R})} \tag{5.16}$$

- **Multiple Segmentations Decoding**: Because the segmentation model $P(\mathcal{S}|F)$ is a probability distribution, we can sample a segmentation $\mathcal{S}$ during the decoding. This allow us to perform decoding with multiple segmentation candidate and pick a hypothesis that yields the best translation from the translation model $P(E|F, \mathcal{S})$.

[3]Following from the Equation 5.10, we add 2 more weighted loss $\mathcal{L}_F, \mathcal{L}_C$ to the sum of the losses, resulting that the final loss is composed by 5 separate losses if all the priors are used.

Experiments	Configuration	BLEU	Loss	KyTea	MECAB
cbaseline	-	21.2378	2.669	0.81539	0.77870
wbaseline	-	22.0218	2.631	1.00000	0.93329
presegment	-	21.0659	2.638	1.00000	0.93329
segment	lp=3.3	19.9132	2.693	0.55087	0.56665
	cp_weight=0.1	7.4752	3.570	0.56241	0.57745
	scale_reinf	19.6457	2.723	0.23295	0.24126
	eps_greedy	11.6760	3.111	0.54252	0.56060
	cp_weight=1	9.3456	3.411	0.63707	0.62509
	lp=2.0	20.0097	2.687	0.23802	0.24515
	lp=1.25	19.7998	2.673	0.75217	0.75887
	z_norm	18.2747	2.731	0.81119	0.77634
	compose_char	18.9114	2.763	0.52800	0.53789
	lp_weight=0.05	18.9735	2.698	0.52798	0.53402

Table 5.1: The results of the segmentation experiment on the KFTT corpus. We use the Gold segmentation from KyTea and Mecab as the standard gold segmentation on Japanese texts. Here we tried several different settings but unfortunately the evaluation results are not good yet. Here we explain the abbreviation of the settings: (1) lp is the length prior, (2) cp is the confidence penalty, (3) z_norm is the z-normalization, and (4) compose_char is to use the character embedding instead of states from bidirectional LSTM.

5.4. Experiments Setup

Here we describe all the settings we use for this experiment. Unless otherwise specified, we use the most basic settings of hyperparameters for the external toolkit that we use.

- **Corpus:** We perform experiments on KFTT [62] dataset on Japanese to English translation tasks. KFTT is a collection of Wikipedia article about city of Kyoto. In total there is 440K training example with 1169 development sets and 1160 test sets.

- **Preprocessing**: We tokenize English according to the Penn Treebank standard. [56] and lowercase, and tokenize Japanese using KyTea [65] and MeCab[4]. We limit training sentence length up to 50 tokens and keep the test data at the original length.

- **Baseline NMT Systems**: Similar to chapter 4, we build the NMT systems based on the system specifications of [5] and [53] with the XNMT [66] toolkit. The model is a single layer bidirectional LSTM encoder with a single layer LSTM decoder. We use 512 hidden dimension for all the parameters. We use the multi-layered perceptron attention as specified in [5]. The dropout of 50% is employed in the output LSTM layer to make the network more robust.

  To train the system, we use the batch size of 2048 words. This is counting the total words in both source and target sides. We use the Adam [40] trainer with the initial learning rate of 0.001, and halving the learning rate every time the MLE loss improved. We observe the number of decay the system made, and stop the system immediately at the 4th decay. Then we choose the model that has the best development perplexity across the training epochs. Unless otherwise specified, we are using the default setting of the XNMT toolkit [66]. Please note that the baseline systems are using the lookup embedding functions.

  There are three baseline systems that we prepare. The first two baseline are using the lookup embedding function and the third baseline is built using bidirectional composition function with the proposed segmenting encoder framework (pre-segment). The first baseline is using character based input (cbaseline), and the second baseline is using word based inputs (wbaseline). All of the baseline has a knowledge about the gold segmentation with character baseline being the merely baseline that make no use of segmentation.

- **Evaluation:** We use standard single reference BLEU-4 [69] to evaluate the translation performance. Additionally, we also measure the F-measure of the segmentation being produced by the systems toward the heavily supervised Japanese gold segmentation that is built using KyTea [65].

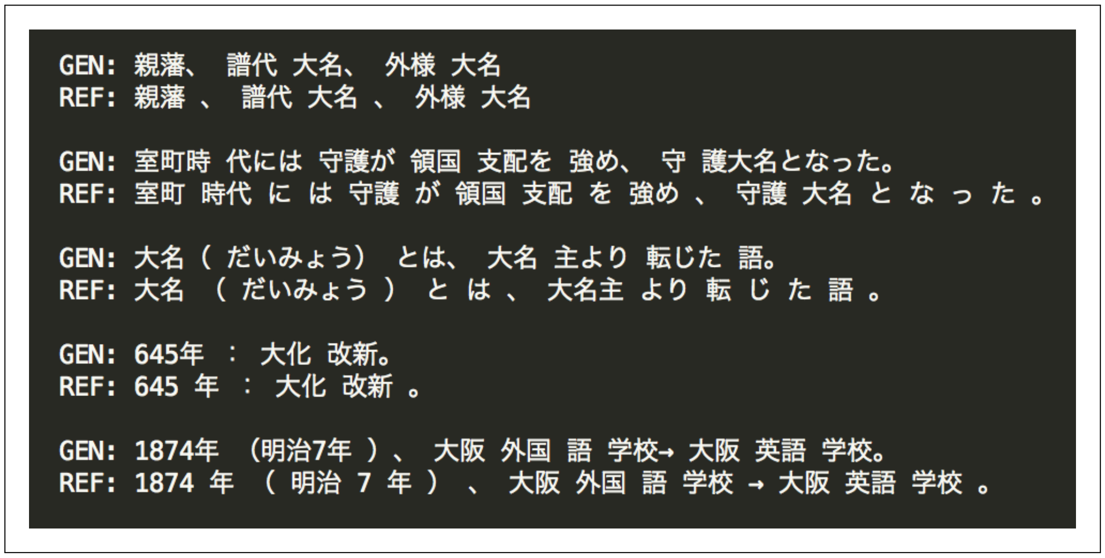
- **Heuristic Segmentation:** Additionally we also prepare 2 heuristic segmenta-

[4]https://github.com/taku910/mecab

Experiments	Configuration	Heuristic 1		Heuristic 2	
		Punc	NoPunc	Punc	NoPunc
cbaseline	-	0.84410	0.83316	0.68836	0.65560
wbaseline	-	0.68149	0.64637	0.65015	0.60434
presegment	-	0.67435	0.63684	0.66532	0.61888
segment	lp=3.3	0.56860	0.61389	0.60908	0.67165
	cp_weight=0.1	0.26674	0.28971	0.25164	0.28050
	scale_reinf	0.53376	0.57674	0.60327	0.66712
	eps_greedy	0.68306	0.69948	0.57352	0.58057
	cp_weight=1	0.25155	0.27875	0.23309	0.26738
	lp=2.0	0.66041	0.66412	0.64351	0.64572
	lp=1.25	0.83460	0.82538	0.68980	0.66127
	z_norm	0.43042	0.47606	0.40716	0.46224
	compose_char	0.46761	0.48814	0.55933	0.59913
	lp_weight=0.05	0.82706	0.81466	0.68761	0.65489

Table 5.2: The segmentation F1 accuracy compared to the segmentation that is based on heuristic. The abbreviations of the settings follow from Table 5.1

tion baseline. These heuristics are based on a simple segmentation of Japanese language. We prepare these heuristics to show that the proposed systems are able to learn something that require some prior knowledge of a language. The first heuristic is to join the function words that have POS tags of ‘助詞’, ‘語尾’, ‘助動詞’, and also joining the ‘する’ together (Heuristic 1). These are the most common function words in Japanese language. The second heuristic is to also learn the noun phrase (Heuristic 2). To do this we simply join the consecutive ‘名詞’ POS tag together.



```
GEN: 親藩、 譜代 大名、 外様 大名
REF: 親藩 、 譜代 大名 、 外様 大名

GEN: 室町時 代には 守護が 領国 支配を 強め、 守 護大名となった。
REF: 室町 時代 に は 守護 が 領国 支配 を 強め 、 守護 大名 と な っ た 。

GEN: 大名（ だいみょう） とは、 大名 主より 転じた 語。
REF: 大名 （ だいみょう ） と は 、 大名主 より 転 じ た 語 。

GEN: 645年 ： 大化 改新。
REF: 645 年 ： 大化 改新 。

GEN: 1874年 （明治7年 ）、 大阪 外国 語 学校→ 大阪 英語 学校。
REF: 1874 年 （ 明治 7 年 ） 、 大阪 外国 語 学校 → 大阪 英語 学校 。
```

Figure 5.2: The example of learned segmentation compared to the KyTea gold baseline segmentation. The generated segmentation (GEN) was able to learn some meaningful words such as years and noun phrases.

5.5. Preliminary Results

First we discuss the overall translation quality on Section 5.5.1 and we discuss the produced segmentation accuracy on Section 5.5.2.

5.5.1 Analysis on the Translation Results

First we examine the results on Table 5.1 clearly it is shown that our proposed method are able to learn some translations but still can not beat the character baseline. We tried several settings and did a grid search to some hyper parameters such as the weight, and tried to scale it over the epochs, however, so far, we were not able to beat the character and word baseline. Next we show that when we run our proposed segmentation framework with the bidirectional LSTM composition function, we are still behind the character and word baseline. This would further suggest that maybe one of the problem that the learning did not go well was because of the wrong choice of the composition function.

Of all the results on Table 5.1, the best accuracies of the proposed systems achieved by setting length prior to 2.0. It means that the system will be heavily biased toward producing words that have around 2 characters. Actually, the results almost tied with the length prior that is calculated on the English corpus (3.3). Still, the optimization problems are very difficult and it is harder to learn from inputs that changed from time to time.

One of the culprit of the failed learning is that the reward is calculated directly from the maximum likelihood. At the first iteration, this rewards are still untrained. Thus leading to a bad segmentation and making learning of the translation more difficult. We tried to include the scaling of weight of the reinforce loss overtime, but the results on "scale_reinf" settings showed that this has no effect in overall learning. We tried to learn the learn the segmentation after having several iterations, but this leads to several problems. First is to use the segmentation prior to the learning

5.5.2 Analysis on the Segmentation Results

The preliminary results of the segmentation can be seen in Table 5.1 and Table 5.2. There the character baseline already has a respectable F1 accuracies baseline of 81% in Japanese and 77% in English. This was due to that Japanese language has a low number of characters/words (around 1.7 characters per words), compared to our method that use a bias from the English language (3.3 characters per words). The resulting segmentation has fewer segmentation and number of words as can be seen in Table 5.2.

However, we argue that our proposed framework was also able to learn some meaningful words. These results can be seen in Figure 5.2. In this table the systems actually were able to discover:

1. Parentheses boundaries. It is always putting space after the open bracket, and the closing bracket.
2. Joining most of the numbers forming years together.
3. Was able to detect and extract some multilingual terms. Though it is still hard to correctly predict rare multilingual terms.
4. Joining most of the particle boundaries to the previous noun phrase. This is not favored by the gold segmentation, but we discover that the discovered segmentation are quite consistent in making these decision.

The biggest help comes from the length prior. We tried not including any priors and the learning failed to discover any meaningful segmentation. Even though we are using prior, but the resulting segmentation are not highly randomized following the given prior, for example, it is not always random to segment years, whereas if the system are following the poisson distribution with some priors, the segmentation should be more randomized.

5.5.3 Difficulties in Learning Discrete Decision

As it has been discussed in the previous sections. We are still facing the problem of unfinished/failed learning using the proposed learning scheme. Here we outline several analysis and remedies that can be done to make the learning works.

1. Currently the rewards are being given from the environment that is all being optimized (the MLE loss). We know that the neural machine translation is not good at adapting with unknown words. Sampling a new set of segmentation is the same as introducing new words. Even though the sampled segmentation is the correct one, but if the system has not seen the segmentation yet, the loss of MLE is still big and thus the reward will decrease. In most cases, the system will every time try to sample the same decision so the learning will become

more stable. This leads to the problem of not having enough exploration and no real guidance to guide the exploration. We might need to design the new reward function.

2. Currently we are using only one sample and not doing monte carlo tree search from the softmax and this is probably not the standard way of training the segmentation method. Currently the architecture to include multiple samples are still being developed in the translation toolkit that we use.

3. We might want to training the system in turn. First we train the decoder only using some samples of the segmentations that can be generated from the current system. Next we train the segmentation based on the fixed parameter decoder. This might help to stabilize the training because the previous problem was that it is hard to jointly train the segmentation and maximum likelihood together. This training scheme was actually close to that of generative adversarial network where the decoder is used as generator, but now, we also want to optimize jointly with the segmentation model.

4. On Chapter 4 we noted that using the character based composition model are not very good even when we know the gold segmentation. We might argue that the character based baseline here is already very strong because characters in Japanese language have rich contents in it might not be necessary to define what words are [65]. We might need to switch to other dataset to further demonstrate the effectiveness of our proposed method, for example the IWSLT, as used in Chapter 4.

5.6. Conclusion

In conclusion, we have presented a novel framework to unsupervisedly learn the segmentation from parallel texts. While the motivation is ambitious and justified, we are not still able to achieve the ideal translation results that we aim for. These was due to several reasons. However, we are able to produce some segmentation that are meaningful to some extents, and looks promising. Further studies and investigation are required in order to stabilize the training of the systems.

Chapter 6

Conclusion and Future Work

We summarize all the contents of Chapter 3, 4, and 5 to answer the problems we posed in Chapter 1.

6.1. Conclusion

Even though ambiguities are inevitable and there are many choices in choosing lexical representation for computer systems, this thesis has provided survey and several suggestions in attempting to solve these problems. The conclusion will be written as paragraphs of the chapter contents.

First, we discover that it is possible to better represent the output probability distribution with addition of a lexicon that is trained with traditional statistical machine translation systems. We have successfully combined the good features of the old count based conditional probability systems with the naturalness of the newer neural network models. This leads into a system that is more capable in generating rare words and better in correlating the output with the input by the attention matrix. This hybrid system has also proven to be effective in other recent works [64] .

Second, we discover that using the combination of embedding words and their constituent characters improves the overall robustness of NMT Systems. This effect can be directly seen by the improvement of the BLEU score of the systems and the ability of the system to generate translations for rare words. We might argue that the systems that benefit from character embeddings are also able to better generate unknown word embeddings. We also discover that by using the natural segmentation of words in some languages improve the performance of NMTs.

Third, we have attempted to build an unsupervised framework that is able to learn segmentation of source languages with the objective to maximize the accuracy of NMT. The system was also able to generate segmentations which are correlated to the natural segmentation to some extent. Even though our tricks and attempts to make the method work are still not enough and we are only seeing moderately good results. As a result, we have not yet able to answer the question which atomic input forms are the best, but we hypothesize that further study on this topic can reveal the answer that we have pursued.

This thesis has given us some motivation and hope, advancing us one step forward in understanding better representations and also choosing lexical units for natural language in computational models. This thesis can also be seen as one case study that attempts to uncover what inside the representation of the neural "black box" (in this case, the lexical representation) in the most appropriate sense possible. This is one of the first step toward the open NLP system that can recognize and learn unknown words on the fly, with better estimation of meaning and boundaries.

6.2. Future Work

Every chapter of the thesis has given us a clue of a possible future directions. Here we list some of questions and problems that emerge from this study and also are still unsolved.

6.2.1 Investigation of Lexicon for Subword Models

In Chapter 3, we proposed the use of lexicons to generate better rare word translations. Dated back when this method was developed, the subword models had not been invented yet. With the increasing of data sparsity of using neural network for building the NMT systems, the subword models now have become the basic component and a must-do element in state-of-the-art NMT systems.

We have not pursued the direction of using the lexicon for the subword models. While it is intuitive enough to make use of the lexicon to help generate rare words, it is still a mystery if such effect can be achieved with the subword models. Even though some studies have reported gain of the accuracies in using the lexicon on subword inputs [20], we have not yet uncover why such effects occur and how well it can impact the subword based systems.

6.2.2 Real World Experiments on Endangered Languages

Chapter 4 and 5 have given us motivation that lexical representation can be learned from an unsegmented input of tokens. Chapter 4 provided us some insights in choosing the best way in composing sequences of characters into a word. This knowledge is very important for Chapter 5 where we want to be able to discover the best lexical units in representing a language.

This framework should be applied to real world scenarios. There are thousands of endangered languages that are still unsupported by good tokenization tools such as Stanford Core NLP, KyTea, or NLTK. This makes it hard to do analysis on the languages when word boundaries are not clear. For humans, words make more sense than strings of characters. Fortunately there have been many corpora collections that collect sentences in spoken languages from people who speak languages without concrete documentation [10, 19]. These corpora can be processed further into parallel corpora and our discovery method can further extract meaningful language properties that can be used to further understanding of the language. Unfortunately due to our time constraint, we were not able to perform such experiments.

6.2.3 Better Reinforcement Learning Technique for Learning Segmentation

In Chapter 5, we just partially succeeded in learning the lexical units from unsegmented inputs. We believe that there is much room for improvement in stabilizing the methods, or there are other tricks to be applied to make the methods work. As we know that the reinforcement learning is difficult to optimize, and researchers have been using numerous tricks to make it work and it also needs to a correct set of hyper parameters.

We also think that the reward function that is mainly scored by using the likelihood of the models can be made even better. Though, constructing a reward function is always the main secret of successful reinforcement learning. At first we are using the maximum likelihood so that part of the reward computation can be calculated in a parallel manner and there is no decoding needed. However, using BLEU score of the immediate translation can be one potential dire

List of Publications

- Odette Scharenborg, Laurent Besacier, Alan Black, Mark Hasegawa-Johnson, Florian Metze, Graham Neubig, Sebastian Stuker, Pierre Godard, Markus Muller, Lucas Ondel, Shruti Palaskar, Philip Arthur, Francesco Ciannella, Mingxing Du, Elin Larsen, Danny Merkx, Rachid Riad, Liming Wang, Emmanuel Dupoux. *Linguistic Unit Discovery from Multi-modal Inputs in Unwritten Languages: Summary of the "Speaking Rosetta"*. JSALT 2017 Workshop 2018 IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP). Calgary, Canada. April 2018.

- Graham Neubig, Matthias Sperber, Xinyi Wang, Matthieu Felix, Austin Matthews, Sarguna Padmanabhan, Ye Qi, Devendra Singh Sachan, Philip Arthur, Pierre Godard, John Hewitt, Rachid Riad, Liming Wang . *XNMT: The eXtensible Neural Machine Translation Toolkit*. Conference of the Association for Machine Translation in the Americas (AMTA) Open Source Software Showcase. Boston. March 2018.

- Yusuke Oda, Philip Arthur, Graham Neubig, Koichiro Yoshino, Satoshi Nakamura. *Neural Machine Translation via Binary Code Prediction*. The 55th Annual Meeting of the Association for Computational Linguistics (ACL). Vancouver, Canada. July 2017.

- Philip Arthur, Graham Neubig, Satoshi Nakamura. *Incorporating Discrete Translation Lexicons into Neural Machine Translation*. Conference on Empirical Methods in Natural Language Processing (EMNLP). November 2016.

- Philip Arthur, Graham Neubig, Sakriani Sakti, Tomoki Toda, Satoshi Nakamura. *Semantic Parsing of Ambiguous Input using Multi Synchronous Grammars*. Association for Natural Language Processing. March 2016.

- Philip Arthur, Graham Neubig, Sakriani Sakti, Tomoki Toda, Satoshi Nakamura. *Semantic Parsing of Ambiguous Input through Paraphrasing and Verification*. Transactions of the Association of Computational Linguistics. Transactions of ACL. September 2015.

- Graham Neubig, Philip Arthur, Kevin Duh. *Multi-Target Machine Translation with Multi-Synchronous Context-free Grammars*. Meeting of the North American Chapter of the Association for Computational Linguistics (NAACL). Denver, USA. May 2015.

- Philip Arthur, Graham Neubig, Sakriani Sakti, Tomoki Toda, Satoshi Nakamura. *Inter-Sentence Features and Thresholded Minimum Error Rate Training: NAIST at CLEF 2013 QA4MRE*. Conference and Labs of the Evaluation Forum (CLEF). Valencia, Spain. September 2013.

References

[1] AKABE, K., NEUBIG, G., SAKTI, S., TODA, T., AND NAKAMURA, S. Discriminative language models as a tool for machine translation error analysis. In *The 25th International Conference on Computational Linguistics (COLING)* (Dublin, Ireland, August 2014), pp. 1124–1132.

[2] ALLAMANIS, M., PENG, H., AND SUTTON, C. A convolutional attention network for extreme summarization of source code. In *Proceedings of the 33th International Conference on Machine Learning (ICML)* (2016).

[3] ARTHUR, P., NEUBIG, G., AND NAKAMURA, S. Incorporating discrete translation lexicons into neural machine translation. In *Conference on Empirical Methods in Natural Language Processing (EMNLP)* (Austin, Texas, USA, November 2016), pp. 1557–1567.

[4] ATAMAN, D., AND FEDERICO, M. Compositional representation of morphologically-rich input for neural machine translation. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)* (Melbourne, Australia, July 2018), Association for Computational Linguistics, pp. 305–311.

[5] BAHDANAU, D., CHO, K., AND BENGIO, Y. Neural machine translation by jointly learning to align and translate. In *Proceedings of the 4th International Conference on Learning Representations (ICLR)* (2015).

[6] BENGIO, Y., DUCHARME, R., VINCENT, P., AND JANVIN, C. A neural probabilistic language model. *Journal of Machine Learning Research* (2003), 1137–1155.

[7] BENGIO, Y., RÉJEAN, D., VINCENT, P., AND JANVIN, C. A neural probabilistic language model. *The Journal of Machine Learning Research 3* (2003), 1137–1155.

[8] BISAZZA, A., RUIZ, N., AND FEDERICO, M. Fill-up versus interpolation methods for phrase-based SMT adaptation. In *Proceedings of the 2011 International Workshop on Spoken Language Translation (IWSLT)* (2011), pp. 136–143.

[9] BROWN, P. F., PIETRA, V. J. D., PIETRA, S. A. D., AND MERCER, R. L. The mathematics of statistical machine translation: Parameter estimation. *Computational Linguistics* (1993).

[10] CAVAR, M., CAVAR, D., AND CRUZ, H. Endangered language documentation: Bootstrapping a chatino speech corpus, forced aligner, asr. In *LREC* (2016).

[11] CETTOLO, M., NIEHUES, J., STÜKER, S., BENTIVOGLI, L., AND FEDERICO, M. Report on the 11th iwslt evaluation campaign, iwslt 2014. In *Proceedings of the International Workshop on Spoken Language Translation, Hanoi, Vietnam* (2014).

[12] CHANG, P.-C., GALLEY, M., AND MANNING, C. D. Optimizing chinese word segmentation for machine translation performance. In *Proceedings of the Third Workshop on Statistical Machine Translation* (2008), pp. 224–232.

[13] CHEN, X., XU, L., LIU, Z., SUN, M., AND LUAN, H.-B. Joint learning of character and word embeddings. In *IJCAI* (2015), pp. 1236–1242.

[14] CHIANG, D. Hierarchical phrase-based translation. *Computational Linguistics* (2007), 201–228.

[15] CHO, K., VAN MERRIENBOER, B., BAHDANAU, D., AND BENGIO, Y. On the properties of neural machine translation: Encoder–decoder approaches. In *Proceedings of the Workshop on Syntax and Structure in Statistical Translation (SSST)* (2014), Association for Computational Linguistics, pp. 103–111.

[16] CHUNG, J., CHO, K., AND BENGIO, Y. A character-level decoder without explicit segmentation for neural machine translation. In *Proceedings of the 54th*

Annual Meeting of the Association for Computational Linguistics (ACL) (2016), pp. 1693–1703.

[17] CHUNG, J., CHO, K., AND BENGIO, Y. A character-level decoder without explicit segmentation for neural machine translation. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)* (2016), Association for Computational Linguistics, pp. 1693–1703.

[18] COSTA-JUSSÀ, M. R., AND FONOLLOSA, J. A. R. Character-based neural machine translation. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (ACL)* (2016), pp. 357–361.

[19] COX, C. Corpus linguistics and language documentation: challenges for collaboration. *Language and Computers-Studies in Practical Linguistics 73*, 1 (2011), 239.

[20] DENKOWSKI, M., AND NEUBIG, G. Stronger baselines for trustable results in neural machine translation. *arXiv preprint arXiv:1706.09733* (2017).

[21] DODDINGTON, G. Automatic evaluation of machine translation quality using n-gram co-occurrence statistics. In *Proceedings of the Second International Conference on Human Language Technology Research* (2002), Morgan Kaufmann Publishers Inc., pp. 138–145.

[22] DULAC-ARNOLD, G., EVANS, R., VAN HASSELT, H., SUNEHAG, P., LILLICRAP, T., HUNT, J., MANN, T., WEBER, T., DEGRIS, T., AND COPPIN, B. Deep reinforcement learning in large discrete action spaces. *arXiv preprint arXiv:1512.07679* (2015).

[23] DYER, C., CHAHUNEAU, V., AND SMITH, N. A. A simple, fast, and effective reparameterization of IBM model 2. In *Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies* (2013), Association for Computational Linguistics, pp. 644–648.

[24] ELMAN, J. L. Finding structure in time. *Cognitive science 14*, 2 (1990), 179–211.

[25] FARUQUI, M., AND DYER, C. Improving vector space word representations using multilingual correlation. In *Proceedings of the 14th Conference of the European Chapter of the Association for Computational Linguistics* (2014), pp. 462–471.

[26] GALLEY, M., GRAEHL, J., KNIGHT, K., MARCU, D., DENEEFE, S., WANG, W., AND THAYER, I. Scalable inference and training of context-rich syntactic translation models. In *Proceedings of the 44th Annual Meeting of the Association for Computational Linguistics (ACL)* (2006), pp. 961–968.

[27] GALLEY, M., HOPKINS, M., KNIGHT, K., AND MARCU, D. What's in a translation rule? In *Proceedings of the 2004 Human Language Technology Conference of the North American Chapter of the Association for Computational Linguistics (HLT-NAACL)* (2004), pp. 273–280.

[28] GERS, F. A., SCHMIDHUBER, J. A., AND CUMMINS, F. A. Learning to forget: Continual prediction with LSTM. *Neural Computation* (2000), 2451–2471.

[29] GRAVE, E., BOJANOWSKI, P., GUPTA, P., JOULIN, A., AND MIKOLOV, T. Learning word vectors for 157 languages. *arXiv preprint arXiv:1802.06893* (2018).

[30] GREEN, S., WANG, S., CER, D., AND MANNING, C. D. Fast and adaptive online training of feature-rich translation models. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)* (2013), Association for Computational Linguistics, pp. 311–321.

[31] GU, J., LU, Z., LI, H., AND LI, V. O. K. Incorporating copying mechanism in sequence-to-sequence learning. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (ACL)* (2016), pp. 1631–1640.

[32] GÜLÇEHRE, Ç., AHN, S., NALLAPATI, R., ZHOU, B., AND BENGIO, Y. Pointing the unknown words. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (ACL)* (2016), pp. 140–149.

[33] HOCHREITER, S., AND SCHMIDHUBER, J. Long short-term memory. *Neural Computation* (1997), 1735–1780.

[34] Iyyer, M., Manjunatha, V., Boyd-Graber, J., and Daumé III, H. Deep unordered composition rivals syntactic methods for text classification. 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)* (2015), vol. 1, pp. 1681–1691.

[35] Jean, S., Cho, K., Memisevic, R., and Bengio, Y. On using very large target vocabulary for neural machine translation. In *Proceedings of the 53th Annual Meeting of the Association for Computational Linguistics (ACL) and the 7th Internationali Joint Conference on Natural Language Processing of the Asian Federation of Natural Language Processing, ACL 2015, July 26-31, 2015, Beijing, China, Volume 1: Long Papers* (2015), pp. 1–10.

[36] Kalchbrenner, N., and Blunsom, P. Recurrent continuous translation models. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing (EMNLP)* (2013), Association for Computational Linguistics, pp. 1700–1709.

[37] Kholghi, M., Vine, L. D., Sitbon, L., Zuccon, G., and Nguyen, A. The benefits of word embeddings features for active learning in clinical information extraction. In *Proceedings of the Australasian Language Technology Association Workshop 2016* (December 2016), pp. 25–34.

[38] Kikui, G., Sumita, E., Takezawa, T., and Yamamoto, S. Creating corpora for speech-to-speech translation. In *8th European Conference on Speech Communication and Technology, EUROSPEECH 2003 - INTERSPEECH 2003, Geneva, Switzerland, September 1-4, 2003* (2003), pp. 381–384.

[39] Kim, Y. Convolutional neural networks for sentence classification. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)* (2014), Association for Computational Linguistics, pp. 1746–1751.

[40] Kingma, D. P., and Ba, J. Adam: A method for stochastic optimization. *CoRR* (2014).

[41] KLAKOW, D. Log-linear interpolation of language models. In *Proceedings of the 5th International Conference on Speech and Language Processing (ICSLP)* (1998).

[42] KOEHN, P. Statistical significance tests for machine translation evaluation. In *Proceedings of the 2004 Conference on Empirical Methods in Natural Language Processing (EMNLP)* (2004).

[43] KOEHN, P., AND HOANG, H. Factored translation models. In *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL)* (2007).

[44] KOEHN, P., HOANG, H., BIRCH, A., CALLISON-BURCH, C., FEDERICO, M., BERTOLDI, N., COWAN, B., SHEN, W., MORAN, C., ZENS, R., DYER, C., BOJAR, O., CONSTANTIN, A., AND HERBST, E. Moses: Open source toolkit for statistical machine translation. In *Proceedings of the 45th Annual Meeting of the Association for Computational Linguistics (ACL)* (2007), Association for Computational Linguistics, pp. 177–180.

[45] KOEHN, P., OCH, F. J., AND MARCU, D. Statistical phrase-based translation. In *Proceedings of the 2003 Human Language Technology Conference of the North American Chapter of the Association for Computational Linguistics (HLT-NAACL)* (2003), pp. 48–54.

[46] KUDO, T. Subword regularization: Improving neural network translation models with multiple subword candidates. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)* (2018), Association for Computational Linguistics, pp. 66–75.

[47] LECUN, Y., BOTTOU, L., BENGIO, Y., AND HAFFNER, P. Gradient-based learning applied to document recognition. *Proceedings of the IEEE 86*, 11 (1998), 2278–2324.

[48] LEE, J., CHO, K., AND HOFMANN, T. Fully character-level neural machine translation without explicit segmentation. *Transactions of the Association for Computational Linguistics 5* (2017), 365–378.

[49] LEE, J., CHO, K., AND HOFMANN, T. Fully character-level neural machine translation without explicit segmentation. *Transactions of the Association for Computational Linguistics 5* (2017), 365–378.

[50] LIANG, P., TASKAR, B., AND KLEIN, D. Alignment by agreement. In *Proceedings of the 2006 Human Language Technology Conference of the North American Chapter of the Association for Computational Linguistics (HLT-NAACL)* (2006), Association for Computational Linguistics, pp. 104–111.

[51] LING, W., TRANCOSO, I., DYER, C., AND BLACK, A. W. Character-based neural machine translation. *CoRR* (2015).

[52] LUONG, M., AND MANNING, C. D. Achieving open vocabulary neural machine translation with hybrid word-character models. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (ACL)* (2016), pp. 1054–1063.

[53] LUONG, M.-T., PHAM, H., AND MANNING, C. D. Effective approaches to attention-based neural machine translation. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing (EMNLP)* (2015), Association for Computational Linguistics, pp. 1412–1421.

[54] LUONG, M.-T., SUTSKEVER, I., LE, Q. V., VINYALS, O., AND ZAREMBA, W. Addressing the rare word problem in neural machine translation. In *Proceedings of the 53th Annual Meeting of the Association for Computational Linguistics (ACL) and the 7th Internationali Joint Conference on Natural Language Processing of the Asian Federation of Natural Language Processing, ACL 2015, July 26-31, 2015, Beijing, China, Volume 1: Long Papers* (2015), Association for Computational Linguistics, pp. 11–19.

[55] LUONG, T., SOCHER, R., AND MANNING, C. Better word representations with recursive neural networks for morphology. In *Proceedings of the Seventeenth Conference on Computational Natural Language Learning* (2013), pp. 104–113.

[56] MARCUS, M. P., MARCINKIEWICZ, M. A., AND SANTORINI, B. Building a large annotated corpus of English: The Penn treebank. *Computational Linguistics* (1993), 313–330.

[57] MATTHEWS, A., NEUBIG, G., AND DYER, C. Using morphological knowledge in open-vocabulary neural language models. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)* (2018), vol. 1, pp. 1435–1445.

[58] MI, H., WANG, Z., AND ITTYCHERIAH, A. Vocabulary manipulation for neural machine translation. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (ACL)* (2016), pp. 124–129.

[59] MIKOLOV, T., CHEN, K., CORRADO, G., AND DEAN, J. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781* (2013).

[60] MIKOLOV, T., SUTSKEVER, I., CHEN, K., CORRADO, G. S., AND DEAN, J. Distributed representations of words and phrases and their compositionality. In *Advances in Neural Information Processing Systems 26*. Curran Associates, Inc., 2013, pp. 3111–3119.

[61] NAKAZAWA, T., YAGUCHI, M., UCHIMOTO, K., UTIYAMA, M., SUMITA, E., KUROHASHI, S., AND ISAHARA, H. Aspec: Asian scientific paper excerpt corpus. In *Proceedings of the Ninth International Conference on Language Resources and Evaluation (LREC 2016)* (2016), European Language Resources Association (ELRA), pp. 2204–2208.

[62] NEUBIG, G. The Kyoto free translation task. http://www.phontron.com/kftt, 2011.

[63] NEUBIG, G. Travatar: A forest-to-string machine translation engine based on tree transducers. In *Proceedings of the 51th Annual Meeting of the Association for Computational Linguistics (ACL)* (2013), pp. 91–96.

[64] NEUBIG, G., MORISHITA, M., AND NAKAMURA, S. Neural reranking improves subjective quality of machine translation: NAIST at WAT2015. In *Proceedings of the 2nd Workshop on Asian Translation (WAT2015)* (Kyoto, Japan, October 2015).

[65] NEUBIG, G., NAKATA, Y., AND MORI, S. Pointwise prediction for robust, adaptable Japanese morphological analysis. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics (ACL)* (2011), pp. 529–533.

[66] NEUBIG, G., SPERBER, M., WANG, X., FELIX, M., MATTHEWS, A., PADMANABHAN, S., QI, Y., SACHAN, D. S., ARTHUR, P., GODARD, P., HEWITT, J., RIAD, R., AND WANG, L. XNMT: The extensible neural machine translation toolkit. In *Conference of the Association for Machine Translation in the Americas (AMTA) Open Source Software Showcase* (Boston, March 2018).

[67] NEUBIG, G., WATANABE, T., MORI, S., AND KAWAHARA, T. Machine translation without words through substring alignment. In *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics: Long Papers - Volume 1* (2012), pp. 165–174.

[68] OCH, F. J., AND NEY, H. A systematic comparison of various statistical alignment models. *Computational Linguistics* (2003), 19–51.

[69] PAPINENI, K., ROUKOS, S., WARD, T., AND ZHU, W.-J. Bleu: A method for automatic evaluation of machine translation. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics (ACL)* (2002), Association for Computational Linguistics, pp. 311–318.

[70] PARTEE, B. Compositionality and Coercion in Semantics: The Dynamics of Adjective Meaning. *Cognitive foundations of interpretation* (2007), 145–161.

[71] PASSBAN, P., LIU, Q., AND WAY, A. Improving character-based decoding using target-side morphological information for neural machine translation. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)* (New Orleans, Louisiana, June 2018), Association for Computational Linguistics, pp. 58–68.

[72] PEREYRA, G., TUCKER, G., CHOROWSKI, J., KAISER, Ł., AND HINTON, G. Regularizing neural networks by penalizing confident output distributions. *arXiv preprint arXiv:1701.06548* (2017).

[73] SAFFRAN, J. R., SENGHAS, A., AND TRUESWELL, J. C. The acquisition of language by children. *Proceedings of the National Academy of Sciences 98*, 23 (2001), 12874–12875.

[74] SENNRICH, R., HADDOW, B., AND BIRCH, A. Improving neural machine translation models with monolingual data. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (ACL)* (2016), pp. 86–96.

[75] SENNRICH, R., HADDOW, B., AND BIRCH, A. Neural machine translation of rare words with subword units. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)* (2016), vol. 1, pp. 1715–1725.

[76] SHEN, S., CHENG, Y., HE, Z., HE, W., WU, H., SUN, M., AND LIU, Y. Minimum risk training for neural machine translation. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics, ACL 2016, August 7-12, 2016, Berlin, Germany, Volume 1: Long Papers* (2016), pp. 1683–1692.

[77] SRIVASTAVA, N., HINTON, G., KRIZHEVSKY, A., SUTSKEVER, I., AND SALAKHUTDINOV, R. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research* (2014), 1929–1958.

[78] SUTSKEVER, I., VINYALS, O., AND LE, Q. V. Sequence to sequence learning with neural networks. In *Proceedings of the 28th Annual Conference on Neural Information Processing Systems (NIPS)*. Curran Associates, Inc., 2014, pp. 3104–3112.

[79] TAMCHYNA, A., WELLER-DI MARCO, M., AND FRASER, A. Modeling target-side inflection in neural machine translation. In *Proceedings of the Second Conference on Machine Translation* (Copenhagen, Denmark, September 2017), Association for Computational Linguistics, pp. 32–42.

[80] VANIA, C., AND LOPEZ, A. From characters to words to in between: Do we capture morphology? In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)* (2017), pp. 2016–2027.

[81] WAWRZYŃSKI, P. Real-time reinforcement learning by sequential actor–critics and experience replay. *Neural Networks 22*, 10 (2009), 1484–1497.

[82] WIETING, J., BANSAL, M., GIMPEL, K., AND LIVESCU, K. Charagram: Embedding words and sentences via character n-grams. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing* (Austin, Texas, November 2016), Association for Computational Linguistics, pp. 1504–1515.

[83] YAMADA, I., SHINDO, H., TAKEDA, H., AND TAKEFUJI, Y. Joint learning of the embedding of words and entities for named entity disambiguation. *arXiv preprint arXiv:1601.01343* (2016).

[84] YONATAN BELINKOV, Y. B. Synthetic and natural noise both break neural machine translation. In *Proceedings of the 6th International Conference on Learning Representations (ICLR)* (2018).