國立成功大學
人工智慧科技碩士學位學程
碩士論文

以聯合語義語音詞嵌入強化中日文神經機器翻譯

Improving Chinese-Japanese Neural Machine Translation with Joint Semantic-Phonetic Word Embedding

研究生：王士杰　　Student : Shih-Chieh Wang
指導教授：賀保羅　　Advisor : Paul Horton

Master Degree Program on Artificial Intelligence,
National Cheng Kung University,
Tainan, Taiwan, R.O.C.
Thesis for Master of Science Degree
July, 2021

中華民國一百一十年七月

以聯合語義語音詞嵌入強化中日文神經機器翻譯

王士杰* 賀保羅†

國立成功大學人工智慧科技碩士學位學程

摘要

中文版簡介。手動換行會自動變成下一段文字區塊。

關鍵字： 關鍵字1、關鍵字2、關鍵字3

*學生
†指導教授

Improving Chinese-Japanese Neural Machine Translation with Joint Semantic-Phonetic Word Embedding

Shih-Chieh Wang* Paul Horton†

Master Degree Program on Artificial Intelligence
National Cheng Kung University

Abstract

Add your abstract here.

Keywords: Keyword1, Keyword2, Keyword3

*Student
†Advisor

Acknowledgements

Add your acknowledgements here.

Shih-Chieh Wang

CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter 1

Introduction

Over the past few years, the field of neural machine translation (NMT) between Chinese and Japanese is still an unresolved problem. Recent studies in Chinese-Japanese NMT have used specific methods such as sub-character level features to improve the translation quality. This is due to the lack of parallel corpus and the difference between logogram (a character or symbol that represents a word) and alphabet (a set of letters used when writing in a language) writing systems. This research explores phonetic information as an additional feature for improving the quality of Chinese-Japanese NMT systems.

1.1 Background

NMT is a popular area of natural language processing (NLP), has been proposed by using an end-to-end model which transforms a source sentence into a latent space and decodes it directly into a target sentence [Sutskever et al., 2014,Cho et al., 2014]. The model is called the encoder-decoder model or sequence-to-sequence model, and they are widely used by large technology companies such as Google, Facebook, Microsoft, and DeepL.

1.1.1 Progress of Neural Machine Translation

The progress of NMT and NLP are inseparable. The development of models, tokenization methods, embeddings, and the solutions to less or no parallel data, all involved in the progress of NMT.

Recurrent neural networks (RNNs), attention mechanisms, and transformers have been proposed sequentially throughout the progress of encoder-decoder models. RNN that recursively passes states in its networks was first applied in the encoder-decoder model [Cho et al., 2014]. Attention-mechanism had addressed the problem of insufficient information in the latent space between encoder and decoder based on RNN [Bahdanau et al., 2014]. Transformers had replaced the RNN structure with full attention-mechanism (i.e., self-attention) to achieve better results and used widely in NMT tasks [Vaswani et al., 2017].

Tokenization is one of the most important parts of any NLP task. It determines how a sentence will be tokenized, and it will generate different meanings to a sentence with different algorithms. Besides word-level

and character-level tokenization, several subword-level tokenization algorithms had become the mainstream. For example: Byte-Pair Encoding (BPE) [Sennrich et al., 2016b], Unigram Language Model [Kudo, 2018], WordPiece [Schuster and Nakajima, 2012], and SentencePiece [Kudo and Richardson, 2018]. This paper will utilize BPE, SentencePiece [Sennrich et al., 2016b,Kudo and Richardson, 2018] and two word-level tokenizer (*Jieba*[1] and *Janome*[2]) as tokenization methods.

The concept of embeddings, also known as distributed representations, was first proposed by [Hinton et al., 1986,Bengio et al., 2003], but was difficult to implement due to hardware limitations. With the development of parallel computing and GPU, many embedding implementations have been proposed, such as Word2Vec [Mikolov et al., 2013a], GloVe [Pennington et al., 2014], and fastText [Bojanowski et al., 2017]. The contextualized word embedding is another concept that obtains context-dependent word embedding from the whole sentence, meaning that the same word with a different position can obtain different embedding through the model. The representative ones are ELMo [Peters et al., 2018] and BERT [Devlin et al., 2019]. This paper will select Word2Vec [Mikolov et al., 2013a] as the tool for creating word embeddings because of its simplicity, rapidity, and convenience of analysis.

Several fields have been studied to solve the problems like low-resources and noisy parallel data in NMT tasks. Back-translation [Sennrich et al., 2016a] is a data augmentation method that uses monolingual data of the target language to generate source data and offset the imbalance between encoder and decoder. Parallel corpus filtering was examined for a large number of NMT tasks [Koehn et al., 2018], using pre-filtering rules and scoring functions to retain good sentence pairs can effectively reduce the corpus size and obtained better translation results. This paper will practice corpus filtering to retain quality training data and reduce corpus size to increase experimental efficiency.

1.1.2 Chinese-Japanese Neural Machine Translation

NMT system has gained a lot of improvement in translating between English and other languages by utilizing the techniques described in section 1.1.1. However, the improvement in translating between Chinese and Japanese is limited. The main reasons are the inadequacy of the corpus and the differences in the writing systems of Chinese, Japanese, and Western languages.

Many studies have focused on improving the Chinese-Japanese (zh-ja) NMT system. In addition to us-

[1] https://github.com/fxsjy/jieba
[2] https://mocobeta.github.io/janome

ing the methods [Imamura et al., 2018,Chu et al., 2017,Zhang et al., 2020] described in section 1.1.1, many feature engineering techniques have been proposed to utilize the features in Chinese Characters (*Hanzi*) and Japanese *Kanji*. For example, a character-level zh-ja NMT system had been improved by using radicals as character feature information [Zhang and Matsumoto, 2017]. Furthermore, the use of decomposed sub-character level information such as ideographs and strokes of Chinese characters, also improved the results [Zhang and Komachi, 2018].

1.1.3 Phonetic Information

Phonetic information is another feature that had been applied to NMT systems. [Khan and Xu, 2019] had suggested that a phonetic representation usually corresponds to semantically distinct characters or words. [Liu et al., 2019] had pointed out that phonetic information can effectively resist the homophone noises generated by typographical mistakes in Chinese sentences. Both papers had improved the performance of the NMT system between Chinese and other Western languages.

This paper attempts to use *Bopomofo* and *Hiragana* as Chinese and Japanese phonetic information to improve the performance of the zh-ja NMT system. Bopomofo also named *Zhuyin* (注音), is located in the Unicode block in the range U+3100–U+312F. It consists of 37 characters and 4 tone marks to transcribe all possible Chinese characters. Although it is the main component of Mandarin Chinese, it usually does not appear in Chinese sentences. That is, the machine loses some of the phonetic information when reading Chinese sentences. Hiragana (平仮名, ひらがな) is a component of Japanese, along with *Katakana* and Kanji. It consists of 46 base characters and is located in the Unicode block in the range U+3040–U+309F. Compared to Bopomofo, Hiragana is often found in Japanese sentences with Katakana and Kanji, forming mixed writing of Kanji and Kana (仮名交じり文). However, Hiragana disappears after forming Kanji, just like Bopomofo forms Hanzi. Therefore, the machine cannot obtain the phonetic information directly from Japanese sentences.

1.2 Objective

This paper aims to determine whether the use of phonetic information can help improve the performance of the zh-ja NMT system. We will use embedding, which is commonly used to represent semantics, to represent the features of phonetic information. The *gensim* library [3] will be utilized to implement Word2Vec [Mikolov et al.,

[3] https://radimrehurek.com/gensim/index.html

2013a] to extract both semantic and phonetic embedding. The embeddings will be trained on a small corpus (less than 1 million lines of sentences) to see if they are useful for the subsequent NMT task.

Combining the findings from other studies [Liu et al., 2019,Khan and Xu, 2019] described in section 1.1.3, we hypothesize that embeddings with the combination of semantics and phonetics (joint embedding) can improve the performance of the zh-ja NMT system more effectively than embeddings with only semantics or phonetics.

We will perform a series of experiments to test the hypothesis. First, we examine whether the joint embedding can improve the results of the zh-ja NMT system with different tokenization methods. Second, we conduct NMT tasks under four conditions: without any pre-trained embedding, with pre-trained semantic embedding, with pre-trained phonetic embedding, and with joint semantic-phonetic embedding. Third, we analyze the changes that occur when phonetic information is added to the general semantic embedding. The analyses include analogy reasoning, outlier detection, word similarity, and the influence on both homonyms and heteronyms.

1.3 Related Work

The core technique in this paper is to enhance the ability of word embeddings by utilizing feature engineering on phonetic information. Therefore, we are going to review some studies that use additional features to improve word embeddings. The review will be carried out from two perspectives, one is to improve embedding with the features of Chinese characters such as radicals and strokes, and the other is to improve embedding with the features of phonetics. The concept of Word2Vec [Mikolov et al., 2013a] is commonly used in reviews, and we will mention it in section 2.3.1 of Chapter 2.

1.3.1 Chinese Word Embedding

We will review some studies which suggested that the rich, decomposed information of Chinese characters can be exploited to train the word embeddings with words or characters jointly. Some of the most popular studies in this field are: *CWE* [Chen et al., 2015], *MGE* [Yin et al., 2016], *JWE* [Yu et al., 2017], and *cw2vec* [Cao et al., 2018]. The following sections review the approaches from JWE and cw2vec because their concepts are relatively new and their performance is better.

Joint Learning Word Embedding Model (JWE)

The authors of JWE [Yu et al., 2017] proposed a model that combines word, character, and sub-character components to learn embeddings. They implemented the model (Figure 1.1) based on the Continuous Bag of Words (CBOW) proposed in Word2Vec [Mikolov et al., 2013a], and changed the default input words in CBOW by further adding the characters and sub-character components of input words.

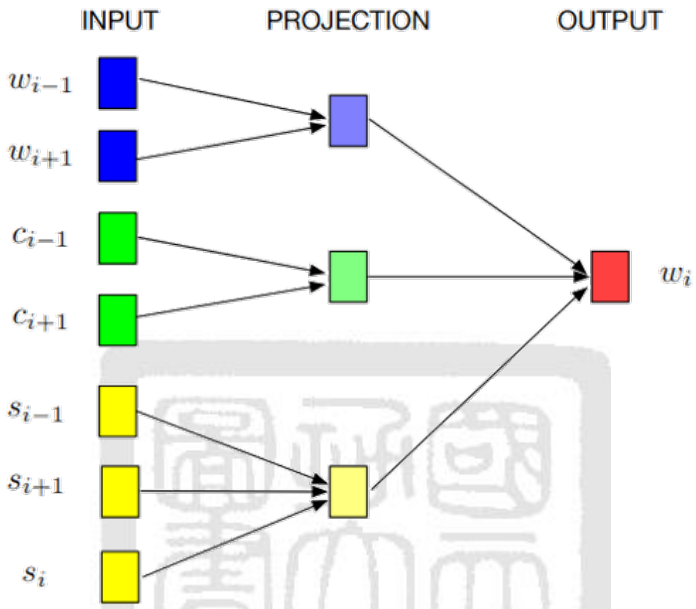


Figure 1.1: An illustration of JWE model proposed in [Yu et al., 2017]

As shown in the Figure 1.1 above, The JWE model is trained to predicts the present word from the context words in the same way as CBOW. The symbol w represents words, w_i is the current word, w_{i-1} and w_{i+1} are the context words. The symbol c represents the characters of context words, c_{i-1} is the characters of $wi-1$, and c_{i+1} is the characters of c_{i+1}. The symbol s represents the sub-characters of context words. s_i is the sub-character components of w_i, and s_{i-1}, s_{i+1} are the components of w_{i-1} and w_{i+1}. For example, a word 智能 (intelligence) has two characters 智 (wisdom) and 能 (able), and each has sub-character components: 知, 日 and 厶, 月, 匕, 匕. The JWE model aims to maximize the sum of log-likelihoods of 3 conditional probabilities that come from the context words (h_{i1}), characters (h_{i2}), and sub-characters (h_{i3}).

$$L(w_i) = \sum_{k=1}^{3} \log P(w_i \mid h_{ik}) \tag{1.1}$$

cw2vec Model

Inspired by fastText [Bojanowski et al., 2017], the authors of cw2vec [Cao et al., 2018] proposed an n-gram feature based on the strokes of Chinese characters. They first split and transformed the text into stroke information, and mapped the strokes into 5 corresponding stroke ids, then merge the ids and generate n-gram features from these stroke ids. The diagram in the paper (Figure 1.2) shows the complete process.

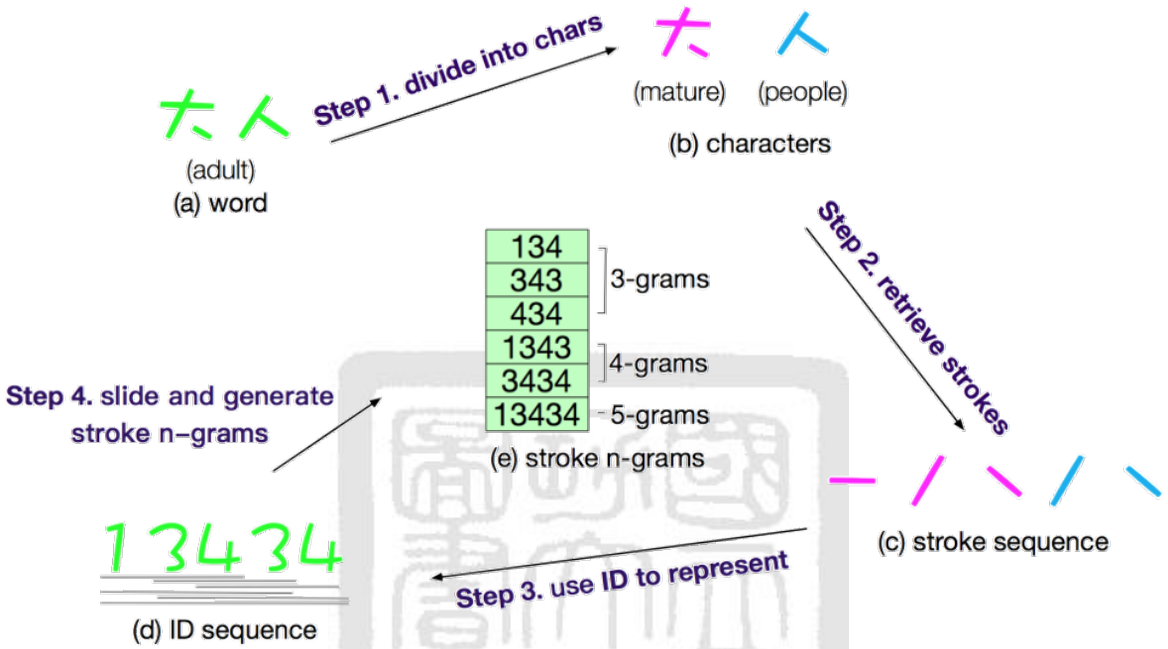


Figure 1.2: The Process of generating stroke n-grams shown in [Cao et al., 2018]

The authors used Skip-Gram, a second method other than CBOW proposed by Word2Vec [Mikolov et al., 2013a], as the base model, and replaced word inputs with stroke n-grams (Figure 1.3). It is mentioned in the paper that the final embeddings come from the contextual word vectors.

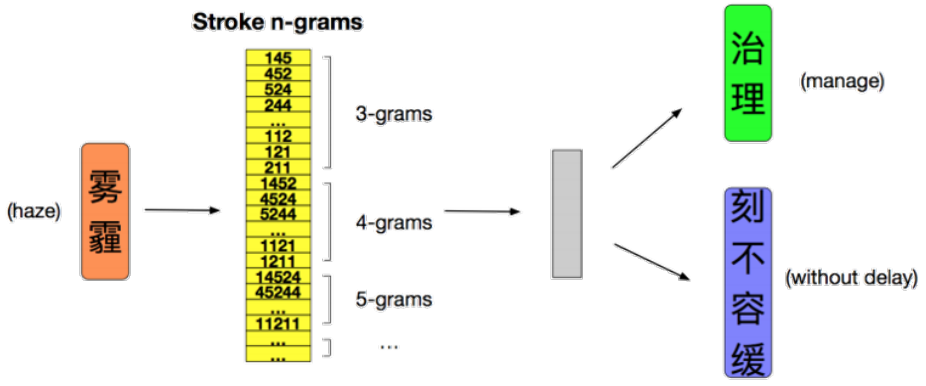


Figure 1.3: An illustration of cw2vec model proposed in [Cao et al., 2018]

These word embeddings designed for Chinese all utilized the radicals or strokes in Chinese characters. No

research in Chinese word embeddings has so far used phonetic information extracted from Chinese or Japanese as features.

1.3.2 Phonetic Word Embedding

We will review studies that use phonetic information to construct word embeddings. These studies are closely related to our research, and we will learn from their practical differences and use them as the cornerstone for our experiments.

Phonetic Encoding

The study [Khan and Xu, 2019] extracted sentences from Chinese or Western languages into phonetic encodings [4], which used *Soundex*, *NYSIIS*, *Metaphone*, and *Hanyu Pinyin* as the extraction algorithm. After that, they applied BPE [Sennrich et al., 2016b] to tokenize the sentences and encodings, and trained separate embeddings from the sentences and each encoding (empty boxes in Figure 1.4). Lastly, they concatenated the embeddings and fed them into the NMT system.

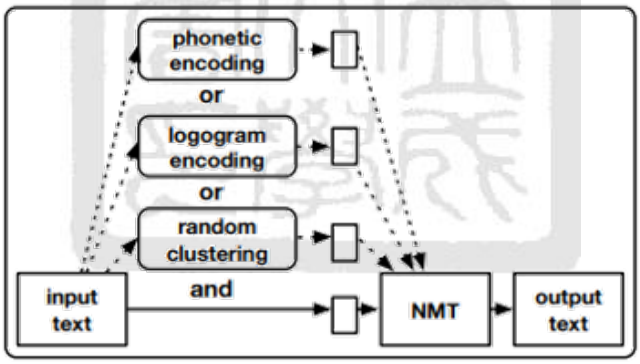


Figure 1.4: Implementation of phonetic encoding proposed in [Khan and Xu, 2019]

This paper did not explain the implementation of phonetic information in detail, but they had presented a hypothesis and verified it. That is, phonetics is a function that groups semantically distinct words. The words with the same pronunciation and spelling are usually distinguished by the different contexts.

[4]The logogram encoding is also used as a feature but is not explained here. Random clustering is used for comparison purposes.

Joint Textual and Phonetic Embedding

This paper [Liu et al., 2019] focused on using phonetic embedding to adjust the homophone noise problem, which is a frequent problem in a parallel corpus. For instance, when a single word in the source sentence is misplanted as a corresponding homophone (e.g., 有 (yǒu, to have) is wrongly replaced with 友 (yǒu, friend)), the model will read the wrong embedding and training in a wrong direction. The phonetic embedding can be seen as a feature to offset the errors that occur in semantic embeddings with homophone noise.

The paper trained the text (denoted by a) as semantic embedding (denoted by $\pi(a)$) and phonetic embedding (denoted by $\psi(a)$), and combined two embeddings with a configurable parameter (denoted by β) as follows:

$$\pi([a, \psi(a)]) = (1 - \beta) \times \pi(a) + \beta \times \pi(\psi(a)) \tag{1.2}$$

This study did not give the details of constructing embeddings, but they had mentioned that the best result occurs when the β is 0.95. That is, when they used 5% of semantic embedding and 95% of phonetic embedding, they obtained the best performance in their NMT system.

To summarize the findings in these related works. The phonetic encodings can emphasize the difference between semantically diverse sentences. The joint semantic-phonetic embedding also shows the robustness to noise in parallel corpora. All of these features can help improve the performance of the NMT system. However, the methods of using phonetic information as embeddings had been explored mainly in Chinese and Western languages. They also used *Pinyin* instead of Bopomofo as the component to decompose Chinese characters. According to our study, no research has been proposed to apply and analyze phonetic information as an additional feature in Chinese and Japanese NMT systems.

Chapter 2

Method

This paper attempts to improve the zh-ja NMT system by utilizing the phonetic information hidden in the sentences. Section 2.1 explains which tokenization methods we use to deconstruct sentences. Section 2.2 explains which phonetic extraction methods we use to transform plain sentences into phonetic encodings. Section 2.3 shows how we use Word2Vec as our algorithm to build and combine semantic and phonetic embeddings. Section 2.4 shows how we process data from the corpus to eliminate as much noise and reduce the size of the corpus as possible. Section 2.5 explains the details of two NMT models, which are the Attention-based GRU encoder-decoder Model and Transformer. Section 2.6 describes the methods we use to analyze the difference between semantic embeddings and joint semantic-phonetic embeddings.

2.1 Tokenization

This paper will examine the effectiveness of phonetic information under different tokenization methods. We use the *tokenizers* [1] library from the *huggingface* team as our main framework for tokenization. SentencePiece [2], Jieba, and Janome can be implemented as pre-tokenizers in Huggingface Tokenizers.

2.1.1 Huggingface Tokenizers

Huggingface Tokenizers has 5 components that allow users to customize their tokenization methods. These five components are normalizers, pre-tokenizers, models, post-processors, and decoders. Normalizers process an input string such as lower cases or remove spaces and symbols to make it normalized. Pre-tokenizers split an input string according to a set of rules, and pre-tokenizers are where we apply SentencePiece, Jieba, and Janome. Models are responsible for converting text into ids by using the rules learned in the corpus (e.g., WordPiece, BPE, Unigram). Post-processors help us to insert special tokens into the sentence, such as the start token *[BOS]* and the end token [EOS] in NMT tasks. Lastly, the job of Decoders is to reverse the ids to the original sentence.

[1] https://github.com/huggingface/tokenizers
[2] https://github.com/google/sentencepiece

2.1.2 Byte-Pair Encoding (BPE)

We use BPE as the model for merging Chinese and Japanese tokens. BPE builds a dictionary of all the words in the corpus and merges the most frequent words to generate new tokens until the maximum number of our dictionary is reached. We demonstrate the basic flow of BPE applied to Chinese through Table 2.1 and 2.2.

Frequency	Vocabulary	Dictionary
5	区 _	_, 区, 地, 经, 济
7	地 区 _	
3	地 区 经 济 _	
6	经 济 _	

Table 2.1: A simple dataset for demonstrating BPE tokenization

Words are first tokenized by pre-tokenizers and loaded into the BPE vocabulary, with the underscore (_) representing the end of the words.

Total Frequency	Merge	New Dictionary
12	(区, _)	_, 区, 地, 经, 济, 区_
9	(济, _)	_, 区, 地, 经, 济, 区_, 济_
9	(经, 济_)	_, 区, 地, 经, 济, 区_, 济_, 经济_
7	(地, 区_)	_, 区, 地, 经, 济, 区_, 济_, 经济_, 地区_

Table 2.2: The process of merging tokens in BPE tokenization

The merge begins with 区 and _, which appear most frequently in the vocabulary. After merging, 区_ will be added to the final dictionary and replaces (区 _) in the vocabulary. This process continues until the final dictionary reaches its maximum size.

2.1.3 SentencePiece

SentencePiece treats all text in the same Unicode format. It will escape the white space with a meta symbol '_' (U+2581). Therefore, the sentences in Chinese, Japanese, and English are considered to be in the same format, which achieving language independence.

SentencePiece is a purely data-driven method, which means it relies on the corpus to learn the tokenization. It is simple to implement SentencePiece in Huggingface Tokenizers. First, Normalization Form Compatibility Composition (NFKC) normalizes the sentence, for example, by converting a symbol or text in the full-width form to a normalized form. Second, Metaspace pre-tokenizer splits the sentence by white space and converts

the white space into the '_' symbol. Last, BPE with dropout is applied to train with the corpus file. The dropout method will improve the robustness and accuracy.

```
from tokenizers.normalizers import NFKC
from tokenizers import Tokenizer, pre_tokenizers, decoders, trainers

tokenizer = Tokenizer(BPE(dropout=dropout, unk_token="[UNK]"))
tokenizer.normalizer = NFKC()
tokenizer.pre_tokenizer = pre_tokenizers.Metaspace(replacement="_",
    add_prefix_space=True)
tokenizer.decoder = decoders.Metaspace(replacement="_", add_prefix_space=
    True)

trainer = trainers.BpeTrainer(vocab_size=vocab_size)
tokenizer.train(corpus, trainer=trainer)
```

2.1.4 Jieba

Jieba is a famous Chinese tokenization Python library that has more than 26,000 stars on Github currently. Jieba uses a prefix dictionary to store the words and calculates the longest path from the Directed Acyclic Graph (DAG) created by the sentences and dictionary to return the most likely tokenized words. In addition, Jieba uses Hidden Markov Model (HMM) and Viterbi algorithm to tokenized the unknown words in the prefix dictionary. There are four states (B, M, E, S) in the HMM model, which represent the beginning, middle, end, and single (the character can represent a word) of a character. The Viterbi algorithm takes all the words as observation and outputs the states of each character from the input sentence.

A single line of code `jieba.tokenize(sentence_str)` can obtain the tokenized words from Jieba. We inserted it into Huggingface Tokenizers as a pre-tokenizer and trained the Chinese dictionary using BPE.

```
    class JiebaPreTokenizer:
    def jieba_split(self, i: int, normalized_string: NormalizedString) ->
         List[NormalizedString]:
        splits = []
        for _, start, stop in jieba.tokenize(str(normalized_string)):
            splits.append(normalized_string[start:stop])
        return splits

    def pre_tokenize(self, pretok: PreTokenizedString):
         pretok.split(self.jieba_split)
```

2.1.5 Janome

Janome is a Japanese tokenization Python library that currently has 600 stars on Github. It applied the Japanese dictionary of another famous tokenization library, mecab [3]. For the methodology, Janome used the Minimal Acyclic Subsequential Transducer (MAST) as the internal dictionary data structure and the Viterbi algorithm to calculate the probability of tokenized words.

We inserted the Janome tokenizer as a pre-tokenizer to Huggingface Tokenizers and trained the Japanese dictionary using BPE.

```
ja_tokenizer = janome.tokenizer.Tokenizer()

class JanomePreTokenizer:
    def janome_split(self, i: int, normalized_string: NormalizedString)
        -> List[NormalizedString]:
        splits = []
        i = 0
        for token in ja_tokenizer.tokenize(str(normalized_string).strip()
            , wakati=True):
            splits.append(normalized_string[i: i+len(token)])
            i += len(token)
        return splits

    def pre_tokenize(self, pretok: PreTokenizedString):
        pretok.split(self.janome_split)
```

2.1.6 Comparison of Tokenizers

A sample of tokenized sentences using three tokenizers is listed in Table 2.3. Jieba and Janome can tokenize the words more precisely than SentencePiece. The better performance of Jieba and Janome over SentencePiece can be considered to be due to the small size and domain specificity of the corpus.

Input (Chinese)	平成１５年进行的研究内容如下。
SentencePiece	[_平成, _15, _年, 进行的研究, 内容, 如下, _。]
Jieba	[平成, 15, 年, 进行, 的, 研究, 内容, 如下, 。]
Input (Japanese)	平成１５年度に行なった研究内容は次の通りである。
SentencePiece	[_平成, _15, _年度, に行, なった, 研究, 内容, は次の通りである, _。]
Janome	[平成, 15, 年度, に, 行なっ, た, 研究, 内容, は, 次, の, 通り, で, ある, 。]

Table 2.3: Tokenized sentences using three tokenizers

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

2.2 Phonetic Data Extraction

Dragonmapper [4] is used to extract information from Chinese sentences. Pykakasi [5] is used to extract information from Japanese sentences.

2.2.1 Dragonmapper

Dragonmapper is a Python library that can convert between Chinese characters, Bopomofo, Pinyin, and International Phonetic Alphabet (IPA). When we call **`dragonmapper.hanzi.to_pinyin(chinese_str)`**, it will convert the Chinese sentence to Bopomofo tokens based on *CC-CEDICT* [6] and *Unihan* [7] database. Table 2.4 shows the result of phonetic extraction in Chinese sentences. The extraction will be applied to the tokenized sentences to achieve better accuracy.

Input	数据详细计量对于节能推进的重要性
Tokenized	[数据, 详细, 计量, 对于, 节能, 推进, 的, 重要性]
Extracted	[ㄕㄨˋ ㄐㄩˋ, ㄒㄧㄤˊ ㄒㄧˋ, ㄐㄧˋ ㄌㄧㄤˋ, ㄉㄨㄟˋ ㄩˊ, ㄐㄧㄝˊ ㄋㄥˊ, ㄊㄨㄟ ㄐㄧㄣˋ, ㄓㄨㄥ ㄧㄠˋ ㄒㄧㄥˋ]
Input	分析项目是１９９７－２００１年是砷、镍、锰、铬、铍
Tokenized	[分析, 项目, 在, 1997, -, 2001, 年, 是, 砷, 、, 镍, 、, 锰, 、, 铬, 、, 铍]
Extracted	[ㄈㄣ ㄒㄧ, ㄒㄧㄤˋ ㄇㄨˋ, ㄗㄞˋ, 1997, -, 2001, ㄋㄧㄢˊ, ㄕˋ, ㄕˋ, ㄧˋ, ㄋㄧㄝˋ, 、, ㄇㄥˇ, 、, ㄍㄜˋ, 、, ㄆㄧ]
Heteronym Test	长大很快乐，音乐很长久
Tokenized	[长大, 很, 快乐, ,, 音乐, 很, 长久]
Extracted	['ㄓㄤˇ ㄉㄚˋ, ㄏㄣˇ, ㄎㄨㄞˋ ㄌㄜˋ, ,, ㄧㄣ ㄩㄝˋ, ㄏㄣˇ, ㄔㄤˊ ㄐㄧㄡˇ]

Table 2.4: Chinese Phonetic Extraction using Dragonmapper library

2.2.2 Pykakasi

Pykakasi is a Python library that implements algorithms in *kakasi* [8] library. It is able to convert Kanji characters to Hiragana, Katakana, and Romaji through the SKK Japanese dictionary [9] and UniDic [10]. Table 2.5 shows the result of phonetic extraction in Japanese tokenized sentences.

[4] https://github.com/tsroten/dragonmapper
[5] https://github.com/miurahr/pykakasi
[6] https://cc-cedict.org/wiki/
[7] http://www.unicode.org/charts/unihan.html
[8] http://kakasi.namazu.org/index.html.en
[9] https://github.com/skk-dev/dict
[10] https://unidic.ninjal.ac.jp/

Input	技術以外にも，政策、法律、社会的側面の課題がある
Tokenized	[技術, 以外, に, も, ,, 政策, 、, , 法律, 、, , 社会, 的, 側面, の, 課題, が, ある]
Extracted	[ぎじゅつ, いがい, に, も, ,, せいさく, 、, , ほうりつ, 、, , しゃかい, てき, そくめん, の, かだい, が, ある]
Input	冠状動脈レントゲンでＴＩＭＩ血流３級が示された
Tokenized	[冠状, 動脈, レントゲン, で, TI, MI, 血流, 3, 級, が, 示さ, れ, た]
Extracted	[かんじょう, どうみゃく, れんとげん, で, TI, MI, けつりゅう, 3, きゅう, が, しめさ, れ, た]
Heteronym Test	一生，芝生で生ビールを飲む
Tokenized	[一生, ,, 芝生, で, 生, ビール, を, 飲む]
Extracted	[いっしょう, ,, しばふ, で, なま, びーる, を, のむ]

Table 2.5: Japanese Phonetic Extraction using Pykakasi library

2.3 Embedding

This paper uses Word2Vec as an embedding framework. Embedding is a distribution representation of words. Compared to using sparse vectors such as one-hot representations to represent words, embedding can represent words as high-dimensional vectors of real numbers. These vectors can be used to obtain similarity between words by computing distances such as cosine similarity.

First, We will explore the details of Word2Vec. Second, we explain how to train a general word embedding with semantics from a corpus. Third, we examine how to manipulate the phonetic information to build a phonetic embedding. Finally, we combine two embeddings to a joint embedding which can have both semantic and phonetic information.

2.3.1 Word2Vec

Word2Vec has two classic papers. The first one [Mikolov et al., 2013a] contributes two approaches to constructing an embedding, namely CBOW and Skip-gram. The second one [Mikolov et al., 2013b] contributes the improved methods of embedding construction, such as Hierarchical Softmax and Negative Sampling. We will use Skip-gram and Negative Sampling to generate word embeddings in this paper.

Skip-gram model uses a neural network with hidden layers to predict the likelihood of context words $w_{t-2}, w_{t-1}, w_{t+1}, w_{t+2}$ of an input word w_t. The input word and context words will be paired, such as $(w_t, w_{t-2}), (w_t, w_{t-1})$ to perform supervised training. The model can be expressed as the following equation, where θ represents hidden states of the input word w_{center}, and $w_1, w_2, \cdots, w_C$ represents the context words of w_{center} within the window size C.

$$\underset{\theta}{\operatorname{argmax}}\, p\left(w_1, w_2, \ldots, w_C \mid w_{\text{center}}; \theta\right) \tag{2.1}$$

Skip-gram uses the softmax model for context word classification. h is the hidden layer word vector for the input word w_{center} from the input embedding matrix, and W_{output} is a row vector for a context word from the output embedding matrix. $W_{\text{output}_{(c)}}$ corresponds to the row vector of the context word in (input, context) pair, while $W_{\text{output}_{(i)}}$ corresponds to the row vector of each word in the corpus of size V.

$$p\left(w_{\text{context}} \mid w_{\text{center}}\right) = \frac{\exp\left(W_{\text{output}_{(c)}} \cdot h\right)}{\sum_{i=1}^{V} \exp\left(W_{\text{output}_{(i)}} \cdot h\right)} \in \mathbb{R}^1 \tag{2.2}$$

The goal is to obtain the conditional probability distribution of each unique word observed in the corpus given an input word.

$$\begin{bmatrix} p\left(w_1 \mid w_{\text{input}}\right) \\ p\left(w_2 \mid w_{\text{input}}\right) \\ p\left(w_3 \mid w_{\text{input}}\right) \\ \vdots \\ p\left(w_V \mid w_{\text{input}}\right) \end{bmatrix} = \frac{\exp\left(W_{\text{output}\ (c)} \cdot h\right)}{\sum_{i=1}^{V} \exp\left(W_{\text{output}\ (i)} \cdot h\right)} \in \mathbb{R}^V \tag{2.3}$$

In machine learning, it is a convention to minimize the loss function. The log is added to the equation to simplify and accelerate the calculation, and the formula is rewritten as minimizing a negative log-likelihood instead of maximizing a positive log-likelihood.

$$J\left(\theta; w^{(t)}\right) = -\log \prod_{c=1}^{C} \frac{\exp\left(W_{\text{output}\ (c)} \cdot h\right)}{\sum_{i=1}^{V} \exp\left(W_{\text{output}\ (i)} \cdot h\right)} \tag{2.4}$$

$$= -\sum_{c=1}^{C} \log \frac{\exp\left(W_{\text{output}\ (c)} \cdot h\right)}{\sum_{i=1}^{V} \exp\left(W_{\text{output}\ (i)} \cdot h\right)} \tag{2.5}$$

$$= -\sum_{c=1}^{C} \left(W_{\text{output}\ (c)} \cdot h\right) + C \cdot \log \sum_{i=1}^{V} \exp\left(W_{\text{output}\ (i)} \cdot h\right) \tag{2.6}$$

Figure 2.1 shows the Skip-gram model. we will learn word embedding vectors of size N given the vocabulary size V. The model uses one input word at a time for learning to predict one context word (output).

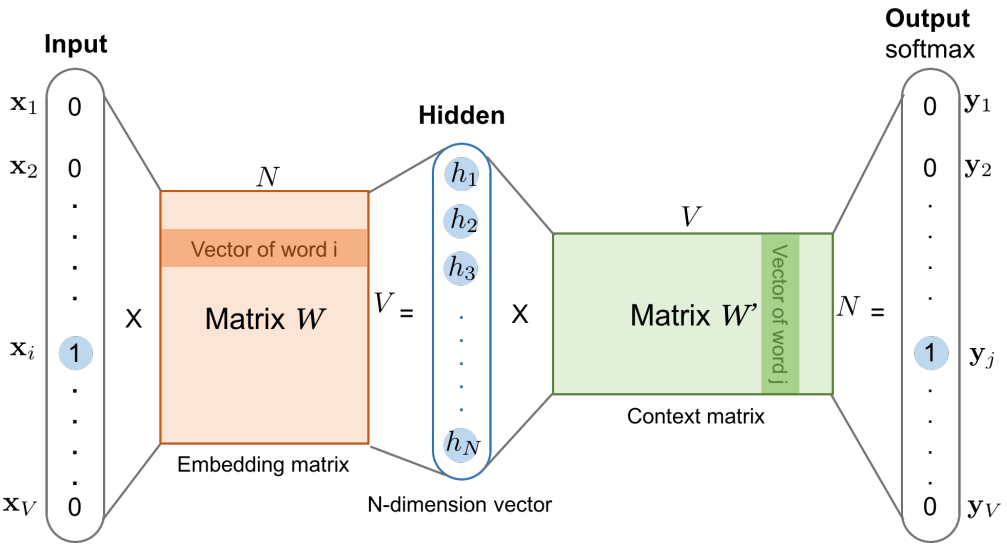


Figure 2.1: The Skip-gram Model[11]

Since the softmax model requires a lot of computational space and time, we also use negative sampling with the Skip-gram model. Negative sampling will convert the multi-classification problem that the model is designed to solve into a binary-classification problem. First, the method will randomly select K negative samples from the corpus that are irrelevant to the input word. K is a hyper-parameter, usually between 5 and 20. The model will update $(K+1) \times N$ parameters using the sigmoid function, where N is the dimension of the hidden layer h, and $+1$ accounts for the positive sample.

The probability that a word (c) appears within the context of the input word (w) can be defined as the following formula. The goal is to determine whether c is in the context window of w ($D = 1$) or not ($D = 0$) based on the input-context pairs (w, c).

$$p(D = 1 \mid w, c; \theta) = \frac{1}{1 + \exp\left(-\bar{c}_{\text{output}\ (j)} \cdot w\right)} \in \mathbb{R}^1 \tag{2.7}$$

2.3.2 Semantic Embedding

We use gensim to implement Skip-gram with negative sampling from Word2Vec. We put the tokenized sentences generated by different tokenizers into gensim to train the general word embedding with semantics. The following are the main settings of the parameters: *max_vocab_size* is 32000, *vector_size* is 300, *epoch_number* is 13, *window_size* is 5, the number of negative samples is 5, ignoring the words with total frequency lower than 5.

[11] Source: https://lilianweng.github.io/lil-log/2017/10/15/learning-word-embedding.html

Some words may not be trained in the model, such as special tokens and low-frequency words. Therefore, the size of the trained embedding matrix and the tokenizer vocabulary is not identical, resulting in different coverage (Table 2.6). We calculate the mean and standard deviation of the whole embedding and randomly assign the empty word vectors using the normal distribution.

Language	SentencePiece	Jieba	Janome
Chinese	95% (30368/32000)	93% (29613/32000)	-
Japanese	97% (30940/32000)	-	93% (29801/32000)

Table 2.6: The coverage of semantic embedding in vocabulary

2.3.3 Phonetic Embedding

The training method for phonetic embedding is mostly the same as semantic embedding. First, we convert tokenized sentences into phonetic encodings using the phonetic extraction techniques described in Section 2.2. Second, we adopt the same training methods and parameters from semantic embedding. And last, we share the same vector of characters or words with the same pronunciation or spelling in the embedding matrix. Therefore, the coverage (Table 2.7) will be even higher than the semantic embedding. The empty vectors are also randomly assigned using the normal distribution with mean and standard deviation from the whole embedding.

Language	SentencePiece	Jieba	Janome
Chinese	99% (31753/32000)	97% (31068/32000)	-
Japanese	99% (31698/32000)	-	96% (30809/32000)

Table 2.7: The coverage of phonetic embedding in vocabulary

2.3.4 Joint Embedding

Several ways have been proposed to combine multiple embeddings, such as concatenation, training and blending separate embedding using the same model, and meta-embedding. Meta-embedding is a very vague term, but the core idea is to accept any kind of pre-trained embedding and fuse them into one (meta) embedding. Many studies on meta-embedding have been proposed [Kiela et al., 2018,Yin and Schütze, 2015,Muromägi et al., 2017]. The methods of meta-embedding can range from very complicated to very straightforward. This paper uses a simple averaging method [Coates and Bollegala, 2018] to merge semantic and phonetic embedding.

2.4 Corpus Filtering

We define custom rules and apply alignment scores from *fast_align* [Dyer et al., 2013] to perform corpus filtering to reduce noise, dataset size, and resource burden during training in the NMT system. The original dataset had 672,315 sentence pairs, which were reduced to 557,685 using rules and 462,582 using alignment scores.

2.4.1 Pre-filtering rules

We apply the following rules to exclude the pairs of sentences that do not match:

1. Removing sentences that are too large in proportion to their length.
2. Removing sentences that are too short or too long.
3. Removing identical sentences.
4. Removing sentences that cannot be correctly identified as Chinese or Japanese by the fastText identification model.
5. Removing sentences with more English and numeric characters than Chinese and Japanese.
6. Removing sentences that have more than one counterpart.

2.4.2 Scoring functions

fast_align is a tool based on *IBM alignment model 2* for training statistical machine translation and alignment model. The score is the probability of the source sentence given the target sentence and its alignment. The algorithm iteratively estimates the probability of being each other translation for source-target pairs and optimal alignment given the word-to-word translation probabilities.

In theory, the score should correlate with the degree of parallelism of the sentences. We remove the sentence pairs that are below a certain probability score threshold as shown in Table 2.8.

Alignment Score Range	Number of sentence pairs
$[-3, -81]$	173,100
$[-81, -160]$	287,882
$[-160, -238]$	93,134
$[-238, -317]$	3,540
$[-317, -395]$	29

Table 2.8: Distribution of Sentence Pair Alignment Scores

We sampled some sentence pairs corresponding to alignment scores in Table 2.9. The findings were that the lower the score, the longer the sentences tended to be, the more words were not translated, and the higher the proportion of proper nouns became.

Score	-52
Chinese	残留性化学物质的物质循环模型的构建和再利用·废弃物政策评价的应用
Japanese	残留性化学物質の物質循環モデルの構築とリサイクル　廃棄物政策評価への応用
Score	-151
Chinese	使用传统的轴组装工法的2层木制住宅,进行了室内甲醛浓度的测量,同时也研讨了上下楼层的甲醛浓度和换气量的关系。
Japanese	在来軸組工法木造2階建住宅を用いて,室内ホルムアルデヒド濃度の測定を行うとともに,上下階のホルムアルデヒド濃度と換気量の関係も検討した。
Score	-254
Chinese	当我尽快编写了PID控制程序并运行后发现,不仅温度不能很好地稳定,还出现了“不规则振荡”现象(重复出现大幅度偏离控制值的振动),控制彻底失败了。
Japanese	早速,PID制御のプログラムを書いて実行したところ,うまく温度が安定化しないどころかいわゆる“ハンチング”現象(制御値から大きく外れた振動を繰り返す)を起こしてしまい,見事,制御に失敗した
Score	-338
Chinese	(’关于葛根、黄芩、黄连汤之证,在『伤寒论』34条中叙述:“太阳病,桂枝证,医反下之,利遂不止,脉促者,表未解也,喘而汗出者,葛根黄芩黄连汤主之”。
Japanese	’葛根黄芩黄連湯証について,『傷寒論』34条に,「太陽病,桂枝の証,医反ってこれを下し,利遂に止まず,脈促のものは,表いまだ解せざるなり,ぜんして汗出づるものは,葛根黄芩黄連湯これを主る」という。

Table 2.9: Samples of Sentence Pairs with Alignment Scores

2.5 NMT Model

We will train and evaluate the NMT tasks in two classical models, the attention-based RNN model proposed by [Bahdanau et al., 2014], and the Transformer model proposed by [Vaswani et al., 2017]. We will explain the structure of models in this section. Furthermore, The use of parameters is presented in Section 3.3, the metric is explained in Section 3.4, the results are listed in Section 3.5, and the case study is conducted in Section 4.1.

The two models and the following illustrative diagrams are derived from *bentrevett/pytorch-seq2seq* [12]. We utilize the source code and optimize the details, such as accelerating the decoding efficiency of the decoder.

2.5.1 Attention-based GRU encoder-decoder Model

This model (Seq2Seq) has three main components, namely Encoder, Attention, and Decoder. We used the `nn.Module` provided by PyTorch to implement three components and PyTorchLightning to combine and train the entire model.

Encoder

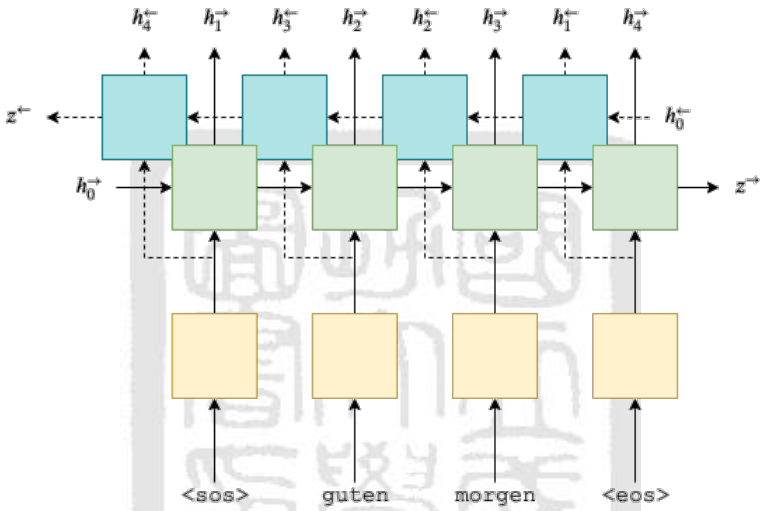


Figure 2.2: The Encoder of Seq2Seq Model

The structure of the encoder (Figure 2.2) can be represented by the following equations. In eq. 2.8, we enter the embedded tokens into the bi-directional GRU model to calculate the forward and backward hidden states h.

$$h_{\overrightarrow{t}} = \overrightarrow{\text{EncoderGRU}}(\text{emb}(x_{\overrightarrow{t}}), h_{\overrightarrow{t-1}}), h_{\overleftarrow{t}} = \overleftarrow{\text{EncoderGRU}}(\text{emb}(x_{\overleftarrow{t}}), h_{\overleftarrow{t-1}}) \tag{2.8}$$

The output H (in eq. 2.9) represents the combination of hidden states in the last layer of the model and will be used to calculate the attention.

$$\text{outputs}(H) = \{h_1, h_2, \cdots, h_T\}, h_i = [h_{\overrightarrow{i}}; h_{\overleftarrow{i}}] \text{ for } i \in \{1, \cdots T\} \tag{2.9}$$

The obtained hidden states $\overrightarrow{z}$ and $\overleftarrow{z}$ (in eq. 2.10) will be used as the initial hidden states of the decoder.

[12] https://github.com/bentrevett/pytorch-seq2seq

$$\overrightarrow{z} = h_{\overrightarrow{T}}, \overleftarrow{z} = h_{\overleftarrow{T}} \tag{2.10}$$

Since the decoder is not bidirectional, we enter the hidden states $\overrightarrow{z}$ and $\overleftarrow{z}$ into a linear layer g and a activation function $tanh$ (in eq. 2.11) to get the condensed context vector z, also called s_0.

$$z = \tanh(g(\text{cat}(\overrightarrow{z}, \overleftarrow{z}))) = s_0 \tag{2.11}$$

Attention

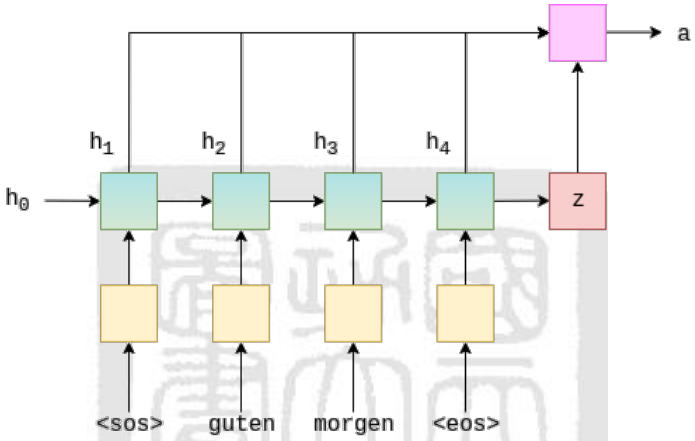


Figure 2.3: The Attention of Seq2Seq Model

The attention layer (Figure 2.3) generates an array of the same length as the source sentence, representing how much attention was given to each token in the source sentence when predicting the next token $\hat{y}_{t+1}$. The attention layer is a linear layer, and each time it takes the previous hidden state s_{t-1} of the decoder (the first time is the z or s_0 from encoder), and the output H from encoder to compute an energy value E_t with $tanh$ activation function (in eq. 2.12).

$$E_t = \tanh(\text{attn}(s_{t-1}, H)) \tag{2.12}$$

Because the shape of the energy value E_t is (*src_len* [13], *hid_dim* [14]), it will enter into a linear layer v with the shape (*hid_dim*, 1) (in eq. 2.13) to get the attention sequence $\hat{a}_t$ with the shape (*src_len*). The parameters learned in the linear layer v can be imagined as the weight of the energy value E_t for each token in the source

[13] The length of source sentence
[14] The dimension of hidden layer

sentence.

$$\hat{a}_t = v(E_t) \tag{2.13}$$

The attention sequence $\hat{a}_t$ will be passed to the softmax layer (in eq. 2.14) so that all the attention values add up to 1. The process combines with the mask to zero out the attention of the [PAD] tokens.

$$a_t = \text{softmax}(\hat{a}_t) \tag{2.14}$$

Decoder

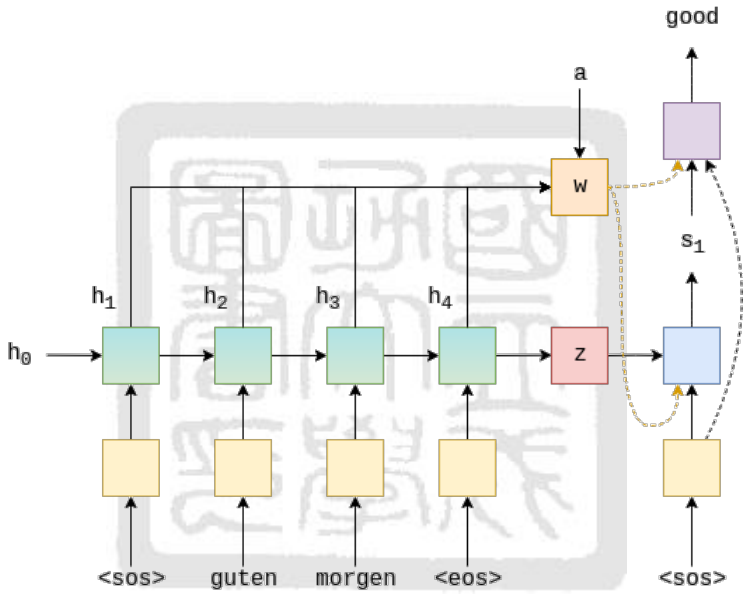


Figure 2.4: The Decoder of Seq2Seq Model

The decoder (Figure 2.4) first uses the previous hidden state s_{t-1} and the encoder output H to compute the attention vector a_t for the current time-step t. The attention vector a_t will perform a matrix multiplication with H (in eq. 2.15) to produce the weighted sum w_t that represents the attention to each token in the source sentence.

$$w_t = a_t H \tag{2.15}$$

The decoder then takes the embedded target token $d(y_t)$, the weighted source vector w_t, and the previous hidden state s_{t-1} into the one-directional GRU model to compute the current hidden state s_t (in eq. 2.16).

$$s_t = \text{DecoderGRU}(d(y_t), w_t, s_{t-1}) \tag{2.16}$$

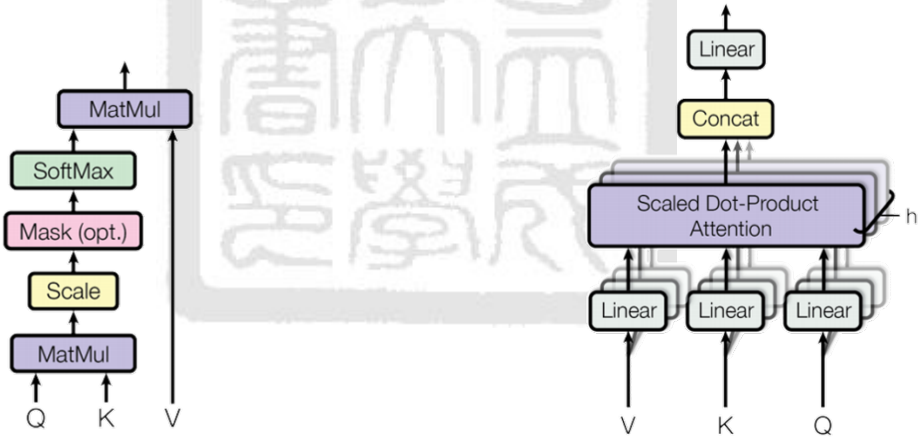
Finally, the decoder uses $d(y_t)$, w_t, s_t and a linear layer f to predict the next token $\hat{y}_{t+1}$ (in eq. 2.17). We will apply the teacher forcing technique, which has a 0.5% chance of using the predicted token and a 0.5% chance of using the ground truth token as the next target input token.

$$\hat{y}_{t+1} = f(d(y_t), w_t, s_t) \tag{2.17}$$

2.5.2 Transformer

The Transformer is an attention-driven model that uses the attention mechanism to process data in either encoder or decoder. We first describe the attention mechanism and then introduce it to the encoder and decoder.

Attention



(a) Scaled Dot-Product Attention

(b) Multi-Head Attention

Figure 2.5: The Multi-head Attention of Tranformer

In the Transformer, Attention layer requires three elements, namely query (Q), key (K), and value (V), as shown in Figure 2.5a. To begin with, the energy value is generated by taking the dot product between query Q and key K. It is then scaling by the square root of *head_dim* [15] ($\sqrt{d_k}$) to avoid gradient vanishing. After that, the energy value is fed into the softmax layer to get the attention sequence and finally multiplied with the value V to get the weighted sum (in eq. 2.18).

[15] The hidden state dimension of each head in Multi-Head Attention

$$\text{Attention}(Q, K, V) = \text{Softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V \tag{2.18}$$

The Transformer model splits query Q, key K, and value V into multiple heads with smaller dimensions and trains them separately (Figure 2.5b). They are divided into heads by the linear layers (W_i^Q, W_i^K, W_i^V). Each has the size that equals the dimension of the hidden state divided by the number of heads.

$$\text{head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V) \tag{2.19}$$

After all the heads pass through the attention layer (in eq 2.19), they are re-concatenated and pass through the last linear layer W^O to get the final weighted sum (in eq 2.20).

$$\text{MultiHeadAttention}(Q, K, V) = \text{Concat}(\text{head}_1, \cdots, \text{head}_h)W^O \tag{2.20}$$

Encoder and Encoder Layers

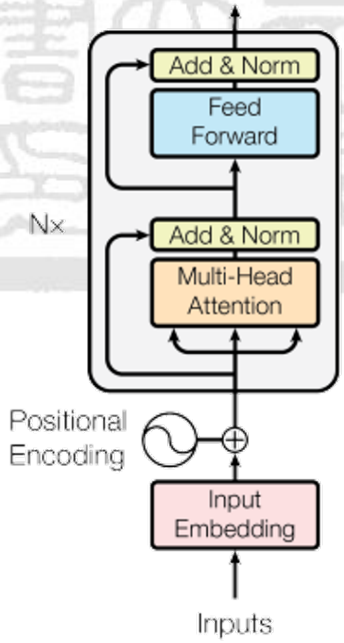


Figure 2.6: The Encoder of Transformer

Unlike the RNN model, the input tokens will generate the corresponding number of context vectors directly after passing through the encoder. For example, if the input has 5 tokens $X = (x_1, x_2, x_3, x_4, x_5)$, the encoder will generate 5 context vectors $Z = (z_1, z_2, z_3, z_4, z_5)$ for the decoder to predict.

The tokens are passed to the word embedding and retrieved the position information by the positional embedding. The size of the positional embedding is equal to the length of the sentence; if the sentence length is 100, then the size of the positional embedding will be 100. The word embedding is added to the positional embedding to obtain the text and position information. However, the word embedding is multiplied by a scaling factor $\sqrt{d_{\text{model}}}$, which is used to reduce the variance of the embedding.

An encoder will have multiple encoder layers. Each of them contains an attention layer and a position-wise feedforward layer (which can be considered as a large linear layer). In addition to these two components, there are also residual connection and layer normalization, which aim to generalize all parameters and prevent gradients from disappearing, making it easier to train networks with a large number of parameters. Because the attention is calculated from the source sentence and itself, it is also called self-attention.

Decoder and Decoder Layers

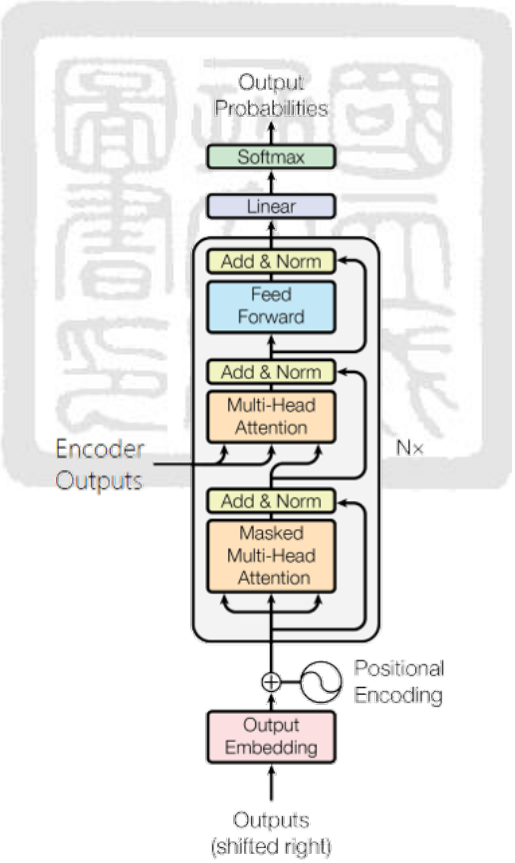


Figure 2.7: The Decoder of Transformer

The architecture of the decoder is almost identical to that of the encoder. The decoder feeds the embedded-target-tokens into several decoder layers, and predicts the next target token $\hat{y}$ with source context vector Z. Following this, it computes the loss by the predict token $\hat{y}$ and the answer token y, and updates all the weights

of the network.

There are three main components in a decoder layer. First, the decoder has a masked multi-head attention layer, in which the target sentence will perform self-attention with itself and a special mask to prevent the decoder from seeing the future words in advance. Second, the multi-head attention layer is calculated with the context vector from the masked multi-head attention layer as the query Q and the encoder context vector as the key K and value V. Third, the last position-wise feedforward layer receives the results of the previous layer and outputs the final context vector to predict the next token $\hat{y}$.

2.6 Embedding Analysis

2.6.1 Analogy Reasoning

2.6.2 Outlier Detection

2.6.3 Word Similarity

2.6.4 Homonym and Heteronym

Chapter 3

Experiment and Result

3.1 Dataset

3.2 Environment

3.2.1 PyTorch Lightning

3.2.2 wandb

3.3 Parameter

3.4 Metric

3.5 Result

Chapter 4

Discussion

4.1 Case Study

4.2 Embedding Analysis

4.2.1 Analogy Reasoning

4.2.2 Outlier Detection

4.2.3 Word Similarity

4.2.4 Homonym and Heteronym

Chapter 5

Conclusion and Future Work

5.1 Conclusion

5.2 Future Work

REFERENCES

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

[Bengio et al., 2003] Bengio, Y., Ducharme, R., Vincent, P., and Janvin, C. (2003). A neural probabilistic language model. *The journal of machine learning research*, 3:1137–1155.

[Bojanowski et al., 2017] Bojanowski, P., Grave, E., Joulin, A., and Mikolov, T. (2017). Enriching word vectors with subword information. *Transactions of the Association for Computational Linguistics*, 5:135–146.

[Cao et al., 2018] Cao, S., Lu, W., Zhou, J., and Li, X. (2018). cw2vec: Learning chinese word embeddings with stroke n-gram information. In *Proceedings of the AAAI Conference on Artificial Intelligence*.

[Chen et al., 2015] Chen, X., Xu, L., Liu, Z., Sun, M., and Luan, H. (2015). Joint learning of character and word embeddings. In *Twenty-Fourth International Joint Conference on Artificial Intelligence*.

[Cho et al., 2014] Cho, K., van Merrienboer, B., Gulcehre, C., Bougares, F., Schwenk, H., and Bengio, Y. (2014). Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*.

[Chu et al., 2017] Chu, C., Dabre, R., and Kurohashi, S. (2017). An empirical comparison of domain adaptation methods for neural machine translation. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 385–391.

[Coates and Bollegala, 2018] Coates, J. and Bollegala, D. (2018). Frustratingly easy meta-embedding – computing meta-embeddings by averaging source word embeddings. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Short Papers)*, pages 194–198, New Orleans, Louisiana. Association for Computational Linguistics.

[Devlin et al., 2019] Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. (2019). BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Vol-*

ume 1 (Long and Short Papers), pages 4171–4186, Minneapolis, Minnesota. Association for Computational Linguistics.

[Dyer et al., 2013] Dyer, C., Chahuneau, V., and Smith, N. A. (2013). A simple, fast, and effective reparameterization of ibm model 2. In *Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 644–648.

[Hinton et al., 1986] Hinton, G. E. et al. (1986). Learning distributed representations of concepts. In *Proceedings of the eighth annual conference of the cognitive science society*, volume 1, page 12. Amherst, MA.

[Imamura et al., 2018] Imamura, K., Fujita, A., and Sumita, E. (2018). Enhancement of encoder and attention using target monolingual corpora in neural machine translation. In *Proceedings of the 2nd Workshop on Neural Machine Translation and Generation*, pages 55–63.

[Khan and Xu, 2019] Khan, A. R. and Xu, J. (2019). Diversity by phonetics and its application in neural machine translation. *arXiv preprint arXiv:1911.04292*.

[Kiela et al., 2018] Kiela, D., Wang, C., and Cho, K. (2018). Dynamic meta-embeddings for improved sentence representations. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 1466–1477, Brussels, Belgium. Association for Computational Linguistics.

[Koehn et al., 2018] Koehn, P., Khayrallah, H., Heafield, K., and Forcada, M. L. (2018). Findings of the wmt 2018 shared task on parallel corpus filtering. In *Proceedings of the Third Conference on Machine Translation: Shared Task Papers*, pages 726–739.

[Kudo, 2018] Kudo, T. (2018). 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)*, pages 66–75, Melbourne, Australia. Association for Computational Linguistics.

[Kudo and Richardson, 2018] Kudo, T. and Richardson, J. (2018). SentencePiece: A simple and language independent subword tokenizer and detokenizer for neural text processing. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 66–71, Brussels, Belgium. Association for Computational Linguistics.

[Liu et al., 2019] Liu, H., Ma, M., Huang, L., Xiong, H., and He, Z. (2019). Robust neural machine translation with joint textual and phonetic embedding. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 3044–3049, Florence, Italy. Association for Computational Linguistics.

[Mikolov et al., 2013a] Mikolov, T., Chen, K., Corrado, G., and Dean, J. (2013a). Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*.

[Mikolov et al., 2013b] Mikolov, T., Sutskever, I., Chen, K., Corrado, G., and Dean, J. (2013b). Distributed representations of words and phrases and their compositionality. *arXiv preprint arXiv:1310.4546*.

[Muromägi et al., 2017] Muromägi, A., Sirts, K., and Laur, S. (2017). Linear ensembles of word embedding models. *arXiv preprint arXiv:1704.01419*.

[Pennington et al., 2014] Pennington, J., Socher, R., and Manning, C. D. (2014). Glove: Global vectors for word representation. In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543.

[Peters et al., 2018] Peters, M. E., Neumann, M., Iyyer, M., Gardner, M., Clark, C., Lee, K., and Zettlemoyer, L. (2018). Deep contextualized word representations. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 2227–2237, New Orleans, Louisiana. Association for Computational Linguistics.

[Schuster and Nakajima, 2012] Schuster, M. and Nakajima, K. (2012). Japanese and korean voice search. In *2012 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 5149–5152.

[Sennrich et al., 2016a] Sennrich, R., Haddow, B., and Birch, A. (2016a). Improving neural machine translation models with monolingual data. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 86–96, Berlin, Germany. Association for Computational Linguistics.

[Sennrich et al., 2016b] Sennrich, R., Haddow, B., and Birch, A. (2016b). 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)*, pages 1715–1725, Berlin, Germany. Association for Computational Linguistics.

[Sutskever et al., 2014] Sutskever, I., Vinyals, O., and Le, Q. V. (2014). Sequence to sequence learning with neural networks. In *Advances in Neural Information Processing Systems*, pages 3104–3112.

[Vaswani et al., 2017] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L. u., and Polosukhin, I. (2017). Attention is all you need. In Guyon, I., Luxburg, U. V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., and Garnett, R., editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc.

[Yin et al., 2016] Yin, R., Wang, Q., Li, P., Li, R., and Wang, B. (2016). Multi-granularity chinese word embedding. In *Proceedings of the 2016 conference on empirical methods in natural language processing*, pages 981–986.

[Yin and Schütze, 2015] Yin, W. and Schütze, H. (2015). Learning meta-embeddings by using ensembles of embedding sets. *arXiv preprint arXiv:1508.04257*.

[Yu et al., 2017] Yu, J., Jian, X., Xin, H., and Song, Y. (2017). Joint embeddings of chinese words, characters, and fine-grained subcharacter components. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 286–291.

[Zhang et al., 2020] Zhang, B., Nagesh, A., and Knight, K. (2020). Parallel corpus filtering via pre-trained language models. *arXiv preprint arXiv:2005.06166*.

[Zhang and Matsumoto, 2017] Zhang, J. and Matsumoto, T. (2017). Improving character-level japanese-chinese neural machine translation with radicals as an additional input feature. In *2017 International Conference on Asian Language Processing (IALP)*, pages 172–175.

[Zhang and Komachi, 2018] Zhang, L. and Komachi, M. (2018). Neural machine translation of logographic language using sub-character level information. In *Proceedings of the Third Conference on Machine Translation: Research Papers*, pages 17–25, Brussels, Belgium. Association for Computational Linguistics.