Sequence to Sequence Japanese Neural Lemmatisation: doweneedsegmentation?

B117474

Master of Science

Speech and Language Processing

School of Philosophy, Psychology and Language Sciences

University of Edinburgh

2018

Abstract

Pipeline models and joint end-to-end models are two contrasting models in morphological analysis. The biggest difference comes from the independence assumption of the steps or modules, which is by default in pipeline models but not true in joint end-to-end models. In this dissertation, we construct pipeline models and joint end-to-end models which handle segmentation and lemmatisation, two important tasks in Japanese morphological analysis. By making use of Nematus, a sequence-to-sequence neural network framework with an encoder-decoder architecture, we model the two tasks as sequence-to-sequence mapping problems. By comparing our best pipeline analyser and our joint end-to-end analyser, we come to the conclusion that, although the pipeline analyser outperforms the joint analyser by a little bit, as it is time-consuming to separately optimise segmentation and lemmatisation, and joint learning of both does not worsen the performance a lot, we would still prefer the joint analyser. Only if there is enough data for either task to learn would we appreciate the pipeline idea.

Keywords: sequence to sequence, joint end-to-end, segmentation, Japanese morphological analysis, Nematus/Lematus

Acknowledgements

I would like to express my sincere gratitude to λ (who) $\wedge$ (why):

λ (my supervisor, Prof. Sharon Goldwater) $\wedge$ (patient and inspiring during every weekly group meeting session and individual supervision)

λ (my group members) $\wedge$ (kind help and useful feedback)

λ (TAs, Sameer Bansal and Clara Vania) $\wedge$ (solid support all the time)

λ (MSc Speech and Language Processing, Prof. Simon King) $\wedge$ (funding for purchasing the data; organising such as an amazing programme)

λ (Arseny Tolmachev from Kyoto University) $\wedge$ (insightful perspectives on Japanese morphological analysis)

λ (my parents) $\wedge$ (sending me abroad)

λ (friends) $\wedge$ (patient enough to bear with me bitching about stress)

λ (open-source community; Nematus, Lematus, etc.) $\wedge$ (without which I could not have been able to finish such a dissertation)

Declaration

I declare that this thesis was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification except as specified.

(*B117474*)

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

Doweneedsegmentation?

For English, the answer is not, as sentences in English naturally come with space as word boundary delimiters. We do not write English sentences in this way: *NaturalLanguageProcessingisinteresting* (Natural Language Processing is interesting). But for some other languages such as Japanese, where word boundaries are not explicitly indicated, the answer is completely different. The same sentence would look like:

自然言語処理が面白いです。

The correspondence between the words in the English and the Japanese sentence is:

自然	言語	処理	が	面白い	です	。
natural	language	processing	(subject case)	interesting	is	.

(“Natural Language Processing is interesting.”)

Regarding segmentation, languages in the world can be roughly categorised into those that are segmented and those that are not segmented. For different types of languages, the ways that computers process them vary a lot. A related task in Natural Language Processing (NLP) is called morphological analysis. It represents a series of systematic approaches of dealing with natural language data and let the data make sense for computers. In general, it includes Part-of-Speech (POS) tagging, lemmatisation, etc. But for some specific languages, there includes more. For non-segmented languages, segmentation is viewed as the first step of morphological analysis. Segmentation is to split a chunk of text into words or other morphologically meaningful units such as morphemes. Following segmentation, lemmatisation is indispensable for languages with inflectional morphology. It returns the canonical or dictionary form of a word. Morphological analysis is important in downstream NLP tasks such as Machine Translation (MT) (Fraser et al., 2012), Information Retrieval (IR), etc. For instance, in web search, lemmatisation and its related morphological process can analyse and return the morphological variants of the words in users' queries, which enlarges the search scope, making it easier to find more related web pages.

Japanese is a non-segmented language with morphological inflection. Various ways have been proposed to segment and lemmatise Japanese text in a step-by-step fashion, i.e. a pipeline model. A common approach adopted by some existing Japanese morphological analysers is to first segment the text by using language model scores (Kurohashi and Nagao, 1998), conditional random fields (CRFs) (Kudo et al., 2004), or support vector machine (SVM) (Neubig et al., 2011), then lemmatise the segmented units into their corresponding lemmas by dictionary look-up (Morita et al., 2015) (Kurohashi, 2018)[1]. These traditional methods work well on limited specific corpora but lack flexibility as they have to load dictionaries compiled by human beings. Besides, pipeline analysers have a strong independence assumption between segmentation and lemmatisation. Under such assumption, the two tasks are often optimised separately. Although it is undoubted that segmentation is necessary and important as a pre-requisite for lemmatisation in pipeline models, we still question the necessity of such independence assumption in general Japanese morphological analysis, or cases where we completely discard the pipeline idea and instead carry out segmentation and lemmatisation at the same time. To distinguish it from the pipeline idea, we call it joint or end-to-end.

To conduct segmentation and lemmatisation at the same time, we make use of the sequence to sequence model with an encoder-decoder architecture originally proposed for MT (Sutskever et al., 2014). A sequence to sequence model can map one sequence of variable length to another, similar to the idea of a string transducer. An existing sequence-to-sequence framework called Nematus (Sennrich et al., 2017), is adopted to the lemmatisation task as Lematus (Bergmanis and Goldwater, 2018), which has lemmatised 20 languages by mapping the inflected word with its context as input to the corresponding lemma as output. Inspired by Lematus, we adopt Nematus to our Japanese segmentation and lemmatisation task, mapping, for example in the segmen-

[1]Although the actual process of lemmatisation is not explained in the paper, we contacted with the authors and are informed so

tation task, a sequence of characters (as a non-segmented chunk of text) to a sequence of the same characters in the input with word boundary delimiters. We expect the joint models to be more efficient and perform better than the pipeline models.

1.2 Objective

Let us refresh our memory of the very first question raised in the title: *doweneed-segmentation*? More specifically, in joint end-to-end Japanese neural lemmatisation, do we first need explicit segmentation? To answer this, we must address three sub-questions:

1. Can neural models lemmatise segmented Japanese text?
2. How well can Japanese neural segmentation models work for pipeline analysers (constituted by segmentation models and lemmatisation models)?
3. Can neural models lemmatise un-segmented Japanese text? Or, are segmentation and lemmatisation facilitating each other when learned jointly?

To answer the above sub-questions, we build various models for segmentation, lemmatisation and both. First we compare variants of lemmatisation models to evaluate the performance of neural lemmatisation (Question 1). To explore the role segmentation plays in Japanese morphological analysis and figure out its interplay with lemmatisation (Question 2), we construct neural segmentation models and combine them with lemmatisation models to make pipeline analysers. In addition to pipeline models, following the spirit of end-to-end, we also build end-to-end models for joint segmentation and lemmatisation. Comparing the pipeline models with the joint models, we discuss their pros and cons, and come to the conclusion of which is the more preferred way of Japanese morphological analysis (Question 3).

1.3 Contributions

Some main contributions we have made to non-segmented language processing are:

- The validation of previous work in sequence to sequence neural lemmatisation on a new and special language, Japanese. Japanese differs itself from most Indo-European languages as it is a non-segmented language with inflectional morphology.

- The first attempt of building a joint end-to-end model of Japanese segmentation and lemmatisation, which is a sequence-to-sequence model with encoder-decoder architecture, in contrast to the existing pipeline models adopting traditional machine learning algorithms. Our empirical results show that the pipeline models outperform the joint models when there is only the same data for both segmentation and lemmatisation, but the gap between the two types of models is very close, and the superiority of the pipeline models might be insignificant.

- Detailed discussion about segmentation and lemmatisation models specifically for Japanese. The critical analysis about the pros and cons of the models provide fresh perspectives on Japanese morphological analysis.

- A different research angle of non-segmented language processing. This could potentially boost related research in neural morphology, web search, machine translation, and corpus linguistics.

Chapter 2

Related Studies

In this chapter, we provide enough background knowledge for readers without experience in morphological analysis or neural networks to understand the big picture.

2.1 Japanese morphological analysis

As introduced in Section 1, Japanese morphological analysis normally includes: segmentation, lemmatisation, POS tagging, dependency parsing, etc. A range of studies mainly discuss segmentation and POS tagging (Kurohashi and Nagao, 1998) (Kudo et al., 2004) (Neubig et al., 2011). It is understandable that segmentation matters for Japanese NLP as it is special and challenging; but POS tagging could have achieved better results if lemmatisation was accomplished as the pre-requisite. Lemmatisation could be helpful to POS tagging as they can return the dictionary form of a word which usually goes through the process of part-of-speech disambiguation. However, we rarely witness works specifically targeting lemmatisation. Very often, lemmatisation is viewed as a non-trivial task which can be tackled by dictionary look-ups (Morita et al., 2015) (Kurohashi, 2018). This motivates us to take a closer look at Japanese lemmatisation - is it really so simple a task? What other approaches can we explore about it?

Segmentation, compared with lemmatisation, seems more attractive and challenging to researchers. Various solutions have been proposed, such as: rule-based methods (Kurohashi and Nagao, 1998) (Kurohashi and Nagao, 2003), Hidden Markov Models (Matsumoto et al., 2000), Conditional Random Fields (Kudo et al., 2004), and Support Vector Machines (Neubig et al., 2011). From using n-gram language modelling scores to estimate the probabilities of each possible segmentation (Kurohashi and Nagao, 1998), to treating segmentation as sequence labelling (Kudo et al., 2004) (Neubig et al., 2011), these segmentation methods suffer from strong statistical independence assumptions and relies on sophisticated feature engineering, usually by hand.

Besides the application of traditional machine learning algorithms on segmentation, there has been a tendency of utilising deep learning. For example, segmentation is still treated as sequence labelling or character tagging, but in a different way. BiRNN-CRF (Bidirectional Recurrent Neural Network Conditional Random Field) (Shao et al., 2017)[1] is a newly proposed model for Chinese[2] word segmentation. The RNN part extracts global numerical features, which are fed into the CRF part; the output from the CRF layer will be segmented words with POS tags. The proposed model could achieve robust good performance on datasets of different sizes, genres and annotation schemes. The idea of jointly learning segmentation and another task has been witnessed in various studies of Chinese morphological analysis, and this can shed light on a new perspective of Japanese morphological analysis. This becomes our external motivation of applying joint learning on Japanese segmentation and lemmatisation.

2.2 Sequence to sequence neural networks with attention

Neural networks are essentially weighted nodes that can operate matrix multiplication and activation functions. The weights of the nodes can be optimised through backpropagation, tuning the weights by subtracting the gradients towards the direction of the steepest descent. Among various kinds of neural networks, sequence to sequence (*seq2seq*) neural networks, which are variants equipped with an encoder-decoder architecture, can be used in NLP tasks related to sequence. The core idea is to learn a vector representation of fixed size for a sequence of variable length through a recurrent neural network (RNN) (Mikolov et al., 2010), i.e. encoding, then decode it back into a sequence also of variable length through another RNN (Cho et al., 2014b). This is initially applied in MT (Cho et al., 2014b) or Neural Machine Translation (NMT) (Sutskever et al., 2014) where a sequence in the source language is transduced into a sequence in the target language. However, RNN has a problem called

[1]**A joint Chinese segmentation and POS tagger based on bidirectional GRU-CRF** https://github.com/yanshao9798/tagger *retrieval date: 30/July/2018*

[2]Chinese is also a non-segmented language which requires segmentation as a step of morphological analysis; Chinese morphological analysis is a rich research field which includes many significant works (Jiang et al., 2008) (Kruengkrai et al., 2009) (Sun, 2011) (Ma and Hovy, 2016).

recency bias and suffers from long-distance catastrophic forgetting. As it encodes the input at each time step and backpropagates the error through time (BPTT: BackPropagation Through Time), the embeddings of the input far away from the end of each sequence might not be learned as well as those close to the end. This is also known as gradient vanishing, i.e. gradients smaller than one becomes increasingly small and close to zero at the end, thus impeding the training. Similar to that, there is gradient exploding where gradients larger than one would be exponentially increasing through BPTT. Gradient-related issues are mitigated by variants of RNN: Long Short-Term Memorys (LSTMs) (Hochreiter and Schmidhuber, 1997) and the light-weight version of LSTMs, Gated Recurrent Units (GRUs) (Cho et al., 2014a). They are different from normal RNNs in that they both add a linear memory cell which allows for unhindered information flow across timesteps, thus improving the effectiveness of long-distance BPTT.

Based on RNNs, an attention mechanism (Bahdanau et al., 2014) between encoder and decoder is proposed to allow a model to softly search for parts of the encoder side sentence that are most helpful in mapping it to the decoder side sentence. At each decoding step, its hidden state is dependent on the previous state, the previous decoder prediction, and the context vector. The context vector is the product of attention, which is the weighted sum of the attention scores of the encoder hidden states. The attention scores are essentially alignment scores, which show how well the inputs around the current encoder state match with the output at the current decoder state. Taking an example from NMT, when translating English into Japanese, as the word order is almost reversed[3], the model has to give more attention to the beginning of the English sentence when the translation in Japanese comes to an end[4]. Attention maps are a useful way to check what part of the encoder sentence the model is giving attention to and how much.

For readers without a deep learning background but have been exposed to computer science, it is helpful to imagine *seq2seq* neural networks as string transducers. For readers from linguistics background, it suffices to think of *seq2seq* neural networks as blackbox models which turn one sequence into another. In MT, it could be from an English sentence to its corresponding translation in Japanese. This, of course, can be

[3]the general sentence structure of English is Subject-Verb-Objective (SVO) while in Japanese it is SOV

[4]because of the difference in the general sentence structure, when translating the verb in a English sentence into Japanese, attention of the end of the Japanese sentence should be given to the first half part of the English sentence. for example (<u>verbs</u> underlined), *I <u>do not eat</u> apples* -> 私はりんごを<u>食べない</u>

extended to other NLP tasks dealing with sequences.

2.3 Sequence to sequence morphological analysis

One of the most recent application of *seq2seq* models to NLP tasks other than MT is morphological analysis, including segmentation (Shao et al., 2017), lemmatisation (Bergmanis and Goldwater, 2018), POS tagging (Plank et al., 2016), dependency parsing (Zhang et al., 2016), etc. Among them, lemmatisation receives least attention, as it is not always a barrier for NLP of most languages, let alone those without inflectional morphology. But inflectional morphology is exactly the biggest challenge lemmatisation faces. A language with rich inflectional morphology means the language has productive formation of words. For instance, German is known as very productive, especially because of its compounding rules where several nouns can be fused together into a compound noun. This leads to the data sparsity problem, a challenge posed to NLP. As a downstream task in NLP, lemmatisation could mitigate the problem to some extent. Lemmatisation on morphologically rich languages can effectively reduce the vocabulary size of a language, save computation, and represent rare or unseen words by linking them with their lemmas which at least appear in dictionaries.

Neural lemmatisation was first implemented as Lematus (Bergmanis and Goldwater, 2018), a neural lemmatiser tested on 20 languages. As context is believed to be the key of better lemmatisation, Lematus makes use of an existing *seq2seq* framework Nematus (Sennrich et al., 2017), to conduct context-sensitive lemmatisation: taking as input a word to be lemmatised with its context from both sides, and outputting the lemma (an example of Lematus' input and output in Table 2.1). Lematus experiments with 20 topologically varied languages, with different types of context representation units (character, byte pair and word) and context lengths (from 0 to 25). The empirical results from Lematus show that encoding 20 characters from both sides of the centre word works best than the rest for most languages, especially on unseen and ambiguous words. However, Lematus lacks solid justification for supporting the hypothesis that, context is the key in improving the performance[5]. For some languages (e.g. Table 1 from Lematus), context-free models even outperform the context-sensitive lemmatisers or are on par with them, so it seems that context becomes uninformative in some cases. Now that Nematus is equipped with attention, it could have been more ideal if attention maps were shown to either prove or overturn the hypothesis. Based on Lema-

[5]now that attention is employed in Nematus, it is sensible to show attention maps

tus, we would experiment with a new language with completely different topology, Japanese, try out varied context units and lengths, and see whether it is context that brings decisive progress.

Input:	`h r e e <s> <lc> b a s e s <rc> a r e <s> t`
Output:	`b a s i s`

Table 2.1: An example of the input and output for model **5-Char** in Lematus

Besides lemmatisation, segmentation is also a non-trivial task for non-segmented languages such as Chinese, Japanese, etc. By tradition, segmentation is treated as sequence labelling or character tagging. Classification is by far the most popular view of segmentation. Given a sequence of characters, the segmentation models classify the characters into the beginning, the end or the middle of a word, as multi-class classification. The most common approach of word segmentation is to extract useful features learned from data, then use sequence labelling models to tag the sequences (Jiang et al., 2008) (Kruengkrai et al., 2009) (Sun, 2011) (Wang et al., 2011) (Shao et al., 2017). Despite the promising performance these approaches have achieved, they almost all require sophisticated feature engineering. Inspired by the sequence modelling idea from tradition and thanks to the advent of *seq2seq* models, the idea of conducting *seq2seq* segmentation naturally comes up to us. Instead of sequence tagging, we model segmentation as to predict the probabilities of word breaks between adjacent characters (details in Section 4.2), i.e. the joint probability of the current character closing the current word and the next character starting a new word. This would ease the data sparsity problem which the previous sequence labelling models suffer from, as we do not rely on feature-labelled data.

Chapter 3

Experimental Design

3.1 Data

3.1.1 Corpora and pre-processing

The corpus we use is the Japanese GSD[1] corpus from the Universal Dependencies (UD) project[2] (hereinafter UD_Japanese).

As pre-processing, we remove all the Arabic numbers and symbols tagged as `PUNCT`[3], following what Lematus did. The same pre-processing is applied to all three datasets: training, development (dev) and test. After pre-processing, the number of word&lemma pairs in each dataset is as follows: 141108 in the training set; 10093 in the dev set; 11085 in the test set.

Besides UD_Japanese, we have a more well-recognised and popular dataset called The Balanced Corpus of Contemporary Written Japanese (BCCWJ) (Maekawa, 2007) (Maekawa et al., 2014), used by various previous studies on Japanese morphology (Neubig et al., 2011) (Yamamoto et al., 2015) (Uchiumi et al., 2015). Due to unexpected delays obtaining the corpus, we are unable to make use of it this time, but will include it in our future work (Section 5.2).

3.1.2 Statistics

To understand the corpus better, for each dataset (training, dev and test), we compute the percentage of ambiguous words and the percentage of word-lemma-copying.

[1]**Universal Dependencies Japanese GSD corpus** https://github.com/UniversalDependencies/UD_Japanese-GSD *retrieval date: 16/July/2018*

[2]**Universal Dependencies** http://universaldependencies.org *retrieval date: 16/July/2018*

[3]`PUNCT`: punctuation

Ambiguous words are defined as words with multiple possible lemmas in the corpus. Some examples of them, roughly categorised into three groups, are shown in Table 3.1. Word-lemma-copying refers to the phenomenon that a word has itself as its lemma, for example, the word *unicorn* and its lemma *unicorn*. The percentage of word-lemma-copying informs us of how accurate a baseline which simply copies the words as their lemmas could be. We convert the percentage into the accuracy of the baseline by subtracting the percentage from 100%. Table 3.2 displays the percentage of ambiguous words, and the expected accuracy of copying the words as their lemmas in each dataset.

Notice that although we are getting access to the data statistics of the test set, we keep it untouched until test time.

Category	**Example**	
	Word	**Lemmas**
lemmas of different POS tags	勝ち	勝ち (*win*, noun)
		勝つ (*win*, verb)
incorrect or inconsistent annotation	同じ	同じ (*the same*, correct)
		同じる (*a non-existing word*, incorrect)
lemmas of the same POS tags	行っ	行く (verb; *to go, to walk*)
		行う (verb; *to hold, to execute*)

Table 3.1: Three categories of ambiguous words in the UD_Japanese training set.

Dataset	**Percentage of ambiguous words**	**Accuracy of base-copying**
training	23.91%	85.66%
dev	16.95%	85.38%
test	18.15%	84.13%

Table 3.2: Percentage of ambiguous words in UD_Japanese and expected accuracy of model **base-copying** (copying words as their lemmas).

3.1.3 Basic units

In later experiments, we mainly apply three types of sub-word units: character, byte pair, and *worpheme*. We explain them one by one.

3.1.3.1 Character

Characters in Japanese include *hiragana*, *katakana*, and Chinese characters *kanji*. Generally, *hiragana*[4] is more used in traditional Japanese words; *katakana*[5] is used in loanwords from foreign languages; *kanji*[6], initially used as *ateji*, i.e, ruby, can replace some characters in a word written in *hiragana*. The three main character types contribute to the variety and the complexity of Japanese writing system, as well as the large size of its character-level lexicon. Compared with an English character-level lexicon which normally would not include more than 150 characters, a Japanese one could contain more than 2500 elements (the lexicon size of the current study in Table 3.7).

In normal Japanese writing, the three character types (sometimes *romaji*, equal to Roman character, is also witnessed; the same for numbers) can appear in the same sentence. Table 3.3 shows an example of a Japanese sentence randomly selected from the Internet[7], with annotation of different writing elements.

大型の台風5号は11日(月)3時現在、紀伊半島の沖を東北東に進んでいます。(ウェザーニュース)
大型 *kanji* の *hiragana* 台風 *kanji* 5 *alphanum* 号 *kanji* は *hiragana* 11 *alphanum* 日(月)*kanji* 3 *alphanum* 時現在 *kanji*、 紀伊半島 *kanji* の *hiragana* 沖 *kanji* を *hiragana* 東北東 *kanji* に *hiragana* 進 *kanji* んでいます *hiragana*。(ウェザーニュース) *katakana*
(“Since 3 a.m. on 11th, the 5th large-scale typhoon of this year has been moving off the Northease east of the Kii Peninsula. (Weather News)”)

Table 3.3: An example of Japanese sentence containing *kanji*, *hiragana*, *katakana*, and *alphanum* (with annotation and translation in English).

3.1.3.2 Byte pair

Byte Pair Encoding, or BPE is an approach of data compression (Gage, 1994). Byte pairs are obtained by recursively merging the most frequent byte pairs - character pairs, into new symbols. In neural machine translation (NMT), frequent byte pairs are replaced with a symbol of the bytes combined, for example, `n g -> ng` and `i ng ->`

[4]some examples of *hiragana*: せ, い, か

[5]some examples of *katakana*: セ, イ, カ

[6]some examples of *kanji*: 清香

[7]**Yahoo Japan News 大型の台風5号** 八丈島の南を通過見込み 関東に最も近づくのは午後に https://news.yahoo.co.jp/pickup/6285732 *retrieval date: 10/June/2018*

`ing`. Then words such as `doing` would likely be represented as `d o ing` or `do ing` in a character-level model. By BPE, the lexicon size is largely reduced through the mapping of byte pairs onto the words, thus mitigating the data sparsity problem (Sennrich et al., 2015). To increase the efficiency of learning and mapping byte pairs, they are only learned within words, without crossing the word boundaries[8]. The number of byte pair merging operation should depend on and be optimised according to languages - in Lematus, 500 for all 20 languages. In our study, we choose to follow Lematus' decision, although further exploration of a more number of operation times is worth discussing for Japanese. The byte pairs are jointly learned from the words in the input and the reference output in the training data to ensure better learning outcome[9].

Our BPE lexicon ends up with 500 byte pairs, 472 of which are *unique lexicon entries*[10]. Table 3.4 shows some representative byte pairs learned from the UD_Japanese corpus training set in ranked order of frequency[11]. The most frequent byte pairs are mostly inflectional morphemes, including する (1st), いる (2nd), れる (3rd) which are common inflectional morphemes making up verbs, ない (4th, an inflected negation suffix of verbs or adjectives), しい (23rd, an inflectional suffix of adjectives), and られる (32nd, an inflectional suffix of verbs), etc. However, among the most frequent byte pairs, there are also some pairs that do not make sense regarding morphology, such as リー (33rd) and ラン (34th) which happen to appear frequently in words. Besides, some whole words are also learned as byte pairs: によって (*according to*, a frequent preposition) and に対して (*to, for*, a frequent preposition), and 現在 (*now*, a frequent noun) and 使用 (*use*, a frequent verb). Apart from Japanese words, some byte pairs in English are also witnessed: `in`, `er`, `am`, etc. These English byte pairs are even more deviated from the objective and expectation of extracting morphologically meaningful byte pairs in Japanese.

The BPE lexicon should be constituted by the characters in the character lexicon, plus the learned byte pairs. However, as we will use separate lexica for encoder and decoder (Section 3.2.1.2), they can be of different sizes. The difference comes from the byte pairs learned in each side, i.e. there are 12 byte pairs from decoder side that do not appear at encoder side. So the BPE lexicon size would be: $2818 + 472 - 12 = 3278$.

[8]**Subword Neural Machine Translation** https://github.com/rsennrich/subword-nmt *retrieval date: 17/July/2018*

[9]as suggested in https://github.com/rsennrich/subword-nmt *retrieval date: 17/July/2018*

[10]e.g. あ る and あ る`</w>`. The former is a byte pair appearing in the middle of a word while the latter at the end of a word, but they become the same entry in a lexicon as they have the same orthographical form

[11]each byte pair is made by merging the characters split by the space

Freq. rank	**Byte pair**	**Freq.**	**Freq. rank**	**Byte pair**	**Freq.**
1	す る<s>	6455	67	によ って<s>	244
2	い る<s>	3176		...	
3	れ る<s>	2832	81	現 在<s>	206
4	な い<s>	1882		...	
5	か ら<s>	1863	91	使 用	180
	...			...	
23	し い<s>	470	113	に対 して<s>	165
	...		319	i n	70
32	ら れる<s>	400		...	
33	リ ー	390	450	e r	52
34	ラ ン	388	500	a m	44

Table 3.4: Examples of byte pairs learned from the UD_Japanese corpus training set in the order of frequency (including inflectional morphemes, frequent prepositions, nouns, etc).

3.1.3.3 Worpheme

Worpheme is a word made up of *word* and *morpheme*, indicating its being a morpheme-like word unit. Worpheme approximates to what we call morpheme in English, but it is meanwhile also the basic segmentation unit (originally called *bunsetsu* in Japanese) in the UD_Japanese corpus, just like in English the basic segmentation is word-based. For example, the word *played* is translated into Japanese as 遊んだ, but according to the worpheme-based segmentation, 遊んだ is segmented into two worpheme units: 遊ん and だ, the former being the verb stem and the latter being the morpheme of verb past tense inflection.

3.2 Models

We build the models and organise the experiments to answer the three sub-questions in Section 1.2. There are three main types of models: lemmatisation models (lemmatisers), segmentation models (segmentisers), and joint end-to-end models (segmatisers[12]) which handle segmentation and lemmatisation at the same time. Each model is a *seq2seq* model with encoder-decoder architecture, implemented by Nematus. Table

[12]**segmatiser** is a word made up of *segmentiser* and *lemmatiser*

3.5 displays all the models, the relationship and the comparison between them. The best lemmatiser and the best segmentiser are selected among the lemmatisers and segmentisers; they constitute the best pipeline segmatiser. The best pipeline segmatiser is then compared with a fine-tuned joint end-to-end segmatiser, the result of which will give us the better segmatiser. It is worth noting that, we only compare the overall scores of the task of different models to select the *best* model among its variants. In addition to the overall score, there are a lot more other factors worthwhile considering when selecting the better or the best model, such as: efficiency (runtime), flexibility, applicability, etc. But here we focus more on models' performance on their tasks as it is easier to compare. The naming of all sub-models is explained in the rest of this section.

3.2.1 Lemmatisation models

Can neural models lemmatise segmented Japanese text? We explore different ways of lemmatising segmented Japanese text, regarding lemmatisation mode, context unit type, and context length.

3.2.1.1 Mode

There are two modes of lemmatisation: word-within-context (**word2word**) and sentence-to-sentence (**sent2sent**). In **word2word** models, as indicated by the name, the input is the to-be-lemmatised word in character representation (i.e. split into characters) with its context from both sides, and the output is the corresponding lemma of the centre word. Special symbols are used to indicate specific information for training (Table 3.6). This mode is adopted in Lematus (Bergmanis and Goldwater, 2018). It is worth noting that, regardless of its name, **word2word** is essentially still a *seq2seq* model encoding and decoding sequences - words represented by character. Unlike **word2word**, in **sent2sent** models, both input and output are full sentences[13], and the output is constituted of the corresponding lemmas of the input words; its full context is encoded for each sentence.

3.2.1.2 Context unit type

In **word2word** lemmatisation models, we use various types of context unit, including character, byte pair and worpheme. These choices are inspired by Lematus (Bergmanis

[13]even in the case when the training sample is only one word, it is still viewed as a sentence

Models	**Lemmatiser**		**Segmentiser**		**Segmatiser**
Sub-models	**word2word**	**sent2sent**	**sent2sent**		**sent2sent**
	base-copying	char	char	maxratio	char
	base0	bpe	bpe	lenpred	bpe
	char5				
	char10				
	char15				
	char20				
	char25				
	char30				
	bpe5				
	bpe10				
	bpe15				
	bpe20				
	bpe25				
	bpe30				
	worpheme5				
	worpheme10				
	worpheme15				
	worpheme20				
	worpheme25				
	worpheme30				
Comparison	**the best lemmatiser**		**the best segmentiser**		
	the best pipeline segmatiser				**a joint segmatiser**
	the better segmatiser				

Table 3.5: The complete list of our models: Lemmatisers, segmentiser, and segmatisers. The first half list all the models we construct, and the second half indicates how we compare between different models and select the best models; models in the same colour will be compared with each other.

and Goldwater, 2018), and the corresponding context extraction can be carried out on Japanese as well. Various context unit types are believed to capture different linguistics information thus perform differently in neural NLP tasks; their performance may also vary among languages (Vania and Lopez, 2017).

Symbol	Meaning
`<w>`	the start of a sequence
`</w>`	the end of a sequence
`<s>`	space
`<lc>`	(the end of the) left context
`<rc>`	(the start of the) right context

Table 3.6: Special symbols used in formatting input and reference output. In some of the later models, part of these symbols would only appear in the encoder lexicon, as there is no left and right context at the decoder side. This would explain the different sizes of some encoder and decoder lexica later in Table 3.7.

In **word2word** character context models, the model name indicates the context unit type, and the centre word is always split into characters.

Table 3.7 reports the encoder and the decoder lexicon size and the lexicon overlap. Models in each context group share the same encoder and decoder lexicon. The decoder lexicon for all models should be the same, and the decoder lexicon is a subset of the encoder lexicon. In later experiments, we use separate lexicon for encoders and decoders to save computation; otherwise, the huge difference between the lexica in **worpheme** models, specifically, would cause computation waste.

Context unit type	Encoder lexicon	Decoder lexicon	Overlap
char	2818	2811	2811
bpe	3278	2811	2811
worpheme	22954	2811	2811

Table 3.7: The encoder and decoder lexicon size and the amount of overlap vocabulary of the context models. Notice that the difference between the encoder and decoder lexicon **char** models come from the special symbols (`<s>, <lc>, <rc>, ''` (the last symbol is generated by us mistakenly forgetting to delete the last new line in the file)) used only at the encoder side, and some characters (𠀃, 辶, 戓) only appear at the encoder side. The **bpe** encoder lexicon size is already explained in Section 3.1.3.2. Both the **char** and the **bpe** lexica have a high percentage of overlapping vocabulary; the **worpheme** encoder lexicon has a much larger size.

3.2.1.3 Context length

Besides various context unit types, we also experiment with different context lengths. The length ranges from 5 to 30 units, counted from both sides of the to-be-lemmatised word in the centre. The choices of context length are also inspired by Lematus. With different amount of context encoded, we expect to see different results and behaviours of the models. Besides, we have a baseline model **base0** which does not encode any context as comparison with models that encode context.

As for some short sentences, bigger context window could not always capture longer context. For example, for the sentence 楽しみ です (“*I look forward to it.*”) which only contains two words, suppose the centre word is 楽しみ, the context it can encode in each group of context models should be the same, as the sentence is shorter than any context window size:

in **char** models:	`<w>`	`<lc>`	楽	し	み	`<rc>`	`<s>`	で	す	`</w>`
in **bpe** (byte pair) models:	`<w>`	`<lc>`	楽	し	み	`<rc>`	`<s>`	です		`</w>`
in **worpheme** models:	`<w>`	`<lc>`	楽	し	み	`<rc>`	`<s>`	です		`</w>`

For some models with bigger context window size, there is not enough context for them to encode. We conjecture that would be a factor of unfair comparison between models. So we compute the average length of encoded context for all **word2word** models as in Table 3.8. For models with smaller context window size, such as **char5 char10 bpe5 bpe10** and **worpheme5**, the context being encoded is close to the specified number; however, with the window size increasing, especially when it is 25 and 30, most models do not have enough context to encode.

3.2.2 Segmentation models

3.2.2.1 sent2sent segmentiser

We start from a **sent2sent** segmentiser which segments an un-segmented input by outputting space symbols between characters where it believes true. Both input and output are in character representation. An example of input and output is in Table 3.9. The performance of the segmentiser is based on sentence-level F1 score of the correctly segmented units over the whole corpus (details in Section 3.4.2).

Model	Average left context length	Average right context length
char5	4.53	4.48
char10	8.41	8.23
char15	11.66	11.34
char20	14.32	13.87
char25	16.48	15.91
char30	18.20	17.53
bpe5	4.47	4.36
bpe10	8.19	7.88
bpe15	11.18	10.67
bpe20	13.54	12.85
bpe25	15.36	14.52
bpe30	16.74	15.77
worpheme5	4.25	4.25
worpheme10	7.32	7.32
worpheme15	9.43	9.43
worpheme20	10.82	10.82
worpheme25	11.70	11.70
worpheme30	12.25	12.25

Table 3.8: Actual length of context being encoded in all **word2word** models. The length of context being encoded is not always proportionately increasing with the specified/expected context length, especially when it comes to **20**, **25** and **30**.

Input:	`<w> 命 が 大 事 で す </w>`
Output:	`<w> 命 <s> が <s> 大 事 <s> で す </w>`

Table 3.9: Example of input and output for **sent2sent** segmentisers

3.2.2.2 For better segmentation length matching

Besides fine-tuning the segmentisers, we also consider to put length constraints on output to improve their performance, if length mismatch turned out to be an issue that matters. The approaches we propose are: constructing a length predictor of the expected output length for each input sequence; constraining the maximum output length for each input sequence by setting a maximum ratio of output length to input length for the whole test set.

The length predictor is also a *seq2seq* model which takes as input a sequence of characters, and returns the predicted output length. Let us suppose the input sequence for the predictor is `<w>` 月 が 綺 麗 で す `</w>` (*"The moon is beautiful."*) and its correct segmentation is `<w>` 月 が 綺麗 です `</w>`. Then the actual output for segmentation (with space symbols) in character representation would be `<w>` 月 `<s>` が `<s>` 綺 麗 `<s>` で す `</w>`. So the expected output length should be `11` . The learning of mapping a sequence to its length is based on UD_Japanese as well.

The second approach does not rely on any external models, but puts length constraints on the output directly. The length constraint is set as the maximum ratio of output length to input length in the UD_Japanese training set, `max(|output|:|input|)` or `maxratio` used as the parameter name in Nematus. Table 3.10 reports the length statistics of the above-mentioned dataset when the sequence is split into characters or byte pairs. The average ratio for both unit types are around `1.5` and the `maxratio` close to `2`. So we decide to set `2` as the `maxratio` for validation and testing[14].

Unit type	**Average `|output|:|input|`**	**`max(|output|:|input|)`**
char	1.5123	1.8182
bpe	1.6076	1.9412

Table 3.10: `|output|:|input|`: Length statistics of the UD_Japanese corpus training set. The average ratio of output length to input length is from 1.5 to around 2.

3.2.3 Segmatisation models: segmentation & lemmatisation models

Segmatisation[15], as defined by us, means the process of both segmentation and lemmatisation. Similarly, a segmatiser is a model which does segmatisation.

A segmatiser could either be a pipeline or an end-to-end one, depending on its optimisation, which is also their main difference. If the optimisation is separately done for the two modules, i.e. segmentation and lemmatisation, then the segmatiser is a pipeline model. The best pipeline model would be the combination of the best segmentiser and the best lemmatiser. Unlike a pipeline model, an end-to-end joint segmatiser is optimised as an integrated system to lemmatise un-segmented text directly. It does not fine-tune each module one by one; instead, it is a big black box which handles both at

[14]during training time, the sequences will not be cut off according to the `maxratio` length constraints
[15]**segmatisation** is a word made up of *segmentation* and *lemmatisation*

once; indeed, we cannot inspect how it "segments" the text in order to lemmatise, or know if it "segments" at all.

In our experiments, we compare a pipeline segmatiser with a joint segmatiser. The pipeline segmatiser simply combines the best segmentiser with the best lemmatiser, both achieving the highest F1 score or accuracy among the above segmentisers (Section 4.2) and lemmatisers (Section 4.1). The output of the segmentiser is fed as the input to the lemmatiser. The joint segmatisers take un-segmented input and lemmatise it into segmented units. There are two joint segmatiser variants, with different basic units: **char** and **bpe**, i.e. character and byte pairs respectively. Table 3.11 displays some example input and output of the joint segmatisers.

Model		**Input & Output**									
char	**input**	`<w>`	き	っ	と	勝	つ	で	す	`</w>`	
	output	`<w>`	き	っ	と	`<s>`	勝	つ	`<s>`	だ	`</w>`
bpe	**input**	`<w>`	き	っと	勝	つ	です	`</w>`			
	output	`<w>`	き	っと	`<s>`	勝	つ	`<s>`	だ	`</w>`	

Table 3.11: Examples of input and output for the **sent2sent** segmatisers.

3.2.4 Baseline models

We use both internal and external models as baselines for specific tasks. For lemmatisation, we have a context-free model **base0** as the internal baseline to be compared with the context-aware models.

Meanwhile, we also have three existing Japanese morphological analysers, **MeCab**[16], **JUMAN**[17] and **JUMAN++**[18], as external baseline models. They are compared with our models in the segmentation task and the segmatisation task, but they cannot be used as pure lemmatisers.

Table 3.12 compares the three external baselines regarding dictionaries, which play the main role in Japanese morphological analysis as they provide the segmentation and lemmatisation criteria. In later experiments, we use the same dictionary Jumandic for all three analysers for a fair comparison, as different dictionaries usually have divergent segmentation and lemmatisation criteria. Those differences would usually cause

[16]**MeCab** http://taku910.github.io/mecab/ *retrieval date: 15/July/2018*
[17]**JUMAN** http://nlp.ist.i.kyoto-u.ac.jp/index.php?JUMAN *retrieval date: 15/July/2018*
[18]**JUMAN++** http://nlp.ist.i.kyoto-u.ac.jp/index.php?JUMAN++ *retrieval date: 15/July/2018*

systematic errors during test time.

	Available dictionaries
MeCab	IPAdic[19]/Jumandic[20]/Unidic[21]
JUMAN	Jumandic
JUMAN++	Jumandic

Table 3.12: Baseline models: three existing non-neural (not completely neural) Japanese morphological analysers and their dictionaries.

Regarding the algorithms adopted by the baselines, **MeCab** and **JUMAN** adopt CRF and mainly rule-based approaches respectively. But **JUMAN++**, the latest Juman family member, interpolate n-gram language modelling (LM) scores which are used by **JUMAN** when computing the probabilities of segmentation hypotheses, with RNNLM scores, to take into account semantic plausibility of word sequences. As a result, **JUMAN++** works better than **JUMAN** in some ambiguous cases.

3.3 Training

3.3.1 Framework

We utilise an existing *seq2seq* framework Nematus (Sennrich et al., 2017), which is also adopted by Lematus (Bergmanis and Goldwater, 2018). The implementation we use is the Theano version (the same as in Lematus), although there is the latest one built by TensorFlow. There are a lot of tunable parameters: encoder/decoder layers, dimensions of word embeddings, dimension of hidden layers, batch size, etc. The details of parameter tuning goes to the next section.

As a *seq2seq* neural model, Nematus has an encoder side and a decoder side. An attention mechanism is employed at the decoder side when the context vector is computed, as the weighted sum of each encoding step, i.e. when decoding the current step, how much attention is given to each encoding step. All encoder and decoder layers we use are Gated Recurrent Units (GRUs), variants of Long-Short Term Memory (LSTM).

With Nematus, we only need to change input and output into the formats that are suitable for training *seq2seq* neural networks.

The loss function in Nematus is sentence-level softmaxed cross-entropy. The loss is optimised by back propagation.

For inference, decoding is based on beam search with a beam sized 12.

3.3.2 Hyper-parameters

We tune hyper-parameters for the models for better performance. Table B.1, Table B.2 and Table B.3 in Appendix B list the hyper-parameters for the lemmatisers, the segmentisers and the segmatisers respectively. The naming follows the Table 3.5.

For most of the hyper-parameters of the **word2word** lemmatisers, we follow the choices of Lematus. The only thing we have changed is the maximum input length - from 75 to 150, which is not a big deal, as we want to include more training sentences.

Based on the hyper-parameters for the lemmatisers, we find that it helps achieve better performance by increasing the dimensions of word embeddings and the hidden layers for the segmentisers. This matches our expectation of segmentation being more complicated than lemmatisation (i.e. needs more parameters to model well). Besides, we make the encoder layers bidirectional which is believed to be able to encode context from both sides better. This also brings increase in the performance.

For the segmatisers, which carry out both tasks, by increasing the number of encoder and decoder layers from 2 to 3, there is a consistent increase in the performance in several pairs of comparison. With more parameters, the models can learn meaningful patterns well enough that are useful for both tasks.

Notice that in each group of models where there are sub-models using different unit types (e.g. **sent2sent-char** and **sent2sent-bpe** segmentisers), we use the same hyper-parameter setting for them. We expect it would bring unfairness in the comparison between the sub-models. In later experiments, we select the best models mainly based on the performance scores. But when the gap between the sub-models is small, we do not draw strong conclusions about models' superiority or inferiority.

3.3.3 Computational resources

The workstation we use is Scientific Linux 7.4, on Distributed Informatics Computing Environment (DICE) machines (Intel Core^{TM} i5-6500 CPU @ 3.20GHz x 4, 64-bit) in School of Informatics, the University of Edinburgh. The Graphic Processing Units (GPUs) we use for the deep models include: GeForce GTX 1060 6GB and Tesla K80. For reference, the average speed of both GPUs, for example on our **word2word-base0** lemmatiser, is around 3000 words per second.

3.4 Evaluation

For different tasks and models, there are specific evaluation metrics.

3.4.1 Accuracy: token-level

Token-level accuracy is used to evaluate the performance of the lemmatisers, as there is one-to-one correspondence between the input token and the output lemma. The computation of accuracy is to divide the number of correctly lemmatised words by the total word count in the test set.

3.4.2 F1 score: sentence-level

Sentence-level F1 score over the whole test set is used to evaluate the other tasks where the reference length and the actual output length might be different, including segmentation, **sent2sent** lemmatisation and segmatisation.

F1 score is a balanced accuracy of precision and recall. Precision is the amount of correct output tokens divided by the amount of reference tokens; recall is the amount of correct output tokens divided by the amount of output tokens. F1 score takes both into account and is computed as follows:

$$F1 = (2 \times Precision \times Recall) \div (Precision + Recall)$$

However, F1 score faces the problem of free word order. We thus employ modified F1 score which constrains the word order by aligning uni-grams between reference and output. Here is an example of computing constrained F1 score for segmentation (Table 3.13). In this example, the second を (in red) in the output is a mistake and should not be counted as a correct label for the first を (in green) in the input. By constrained F1 score computation, this extra を is instead viewed as a wrong output label as it does not align with anything in the reference at the corresponding position.

In addition, although we treat our tasks as MT and use Nematus which is a sequence-to-sequence model originally built for MT, we do not think BLEU score, a well-recognised evaluation metric for MT can be applicable as an alternative. BLEU score is a balanced score which takes into account percentages of n-gram overlapping, normally with n ranging from one to four, computed on the whole corpus. As BLEU score is only informative at corpus-level, it does not inform us of how well each sentence is processed. This property of BLEU would make sentence-level error analysis almost

Input	夢 を 叶 え る た め に エ ン ジ ン バ ラ 大 学 に 入 学 し た
Output	夢 <s> を <s> 叶 え る <s> た め <s> に <s> エ ン ジ ン バ ラ <s> 大 学 <s> に <s> を <s> 入 学 <s> し <s> た
Merged output	夢 を 叶える ため に エンジンバラ 大学 に を 入学 し た
Reference	夢 を 叶える ために エンジンバラ 大学 に 入学 し た
Precision	$11 \div 12 = 0.9167$
Recall	$11 \div 11 = 1$
F1	$(2 \times 0.9167 \times 1) \div (0.9167 + 1) = 0.9565$

Table 3.13: Example of computing F1 score for segmentation.

impossible to be carried out. Instead, by using sentence-level F1 score, we can easily pick up sentences that we concern based on their F1 score.

Chapter 4

Results and Discussion

4.1 Lemmatisers

4.1.1 word2word lemmatisers

Token-level accuracies are computed on the test set for two baseline lemmatisers **base-copying** and **base0**, and all **word2word** lemmatisers. The results are displayed in Figure 4.1. The accuracy of **base-copying** (from Table 3.2, 84.13%) is not plotted due to limited space.

From Figure 4.1, the **char** lemmatisers are almost always outperforming the **bpe** and the **worpheme** counterparts, except when the context window size is 10. Regardless of the general superiority of the **char** lemmatisers, the performance of the **char** lemmatisers is close to that of the **bpe** lemmatisers all the time. After inspecting the **char** lexicon and the **bpe** lexicon, we conjecture that it is due to the high overlap between the two lexicons, that they achieve similar results. The **char** lexicon is exactly a subset of the **bpe** lexicon after 500 times of byte pair merging operation (representative byte pairs in Section 3.1.3.2) (Table 4.1). In the **bpe** lexicon, in addition to the frequent byte pairs which are more likely to be better represented in the neural space, there are as many infrequent characters in it as in the **char** lexicon, whose neural representation cannot be well learned in either case.

Regarding the context length, there is no clear relationship between the context length and the overall performance. For other languages such as most of the languages which Lematus has experimented with, context window size 20 is a sweet spot (Bergmanis and Goldwater, 2018), as it contains enough context without leading to sequences too long for the neural networks. However, in our case, 20 does not

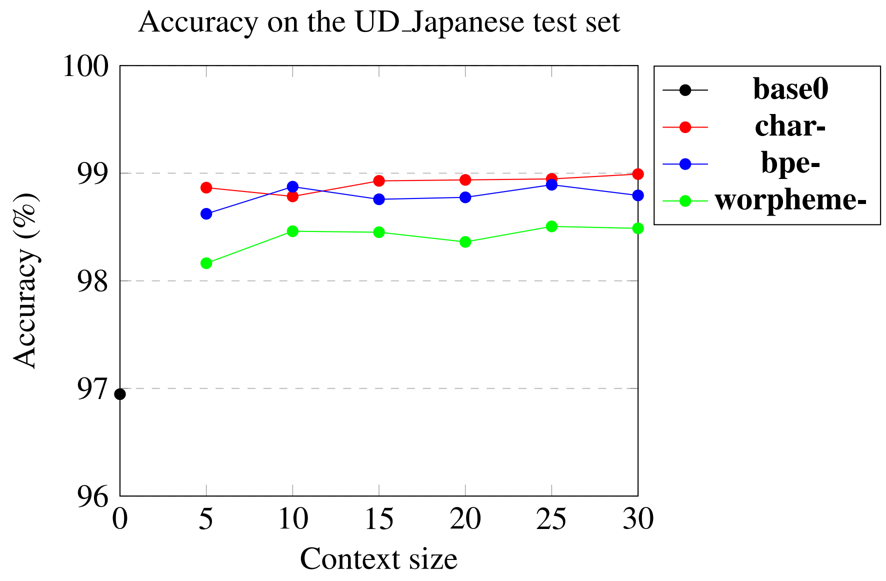

Figure 4.1: The **word2word** lemmatisers: Accuracy on the UD_Japanese test set. The **char** lemmatisers are almost consistently outperforming the other two kinds of lemmatisers; the results of the **char** lemmatisers are always very close to those of the **bpe** lemmatisers. All context-aware lemmatisers outperform the context-free lemmatiser **base0** (96.95%) and **base-copying** (84.13%) which copies the words as their lemmas.

Model	Encoder lexicon size	Overlapping vocabulary
word2word-char	2818	2818
word2word-bpe	3278	

Table 4.1: Encoder lexicon statistics of the **word2word char** and **bpe** lemmatisers. The **bpe** lexicon includes the **char** lexicon.

guarantee the best performance in any case. The best lemmatisers categorised by context unit type are: **char30**, **bpe25**, and **worpheme25**, all with context length longer than 20. It seems that Japanese lemmatisation can learn more and better from longer context. Nonetheless, as displayed in Table 3.8, the length of the actual context being encoded does not necessarily increase with the specified context length proportionately. For example, the amount of context in **char20** is not exactly, or even close to, twice of that in **char10**. The unfairness caused by the sentence length distribution could potentially explain the close results of the lemmatisers with bigger context size windows.

Compared with the non-neural baseline lemmatiser **base-copying**, all neural lemmatisers are doing better. They achieve an increase of at least 12% in accuracy than copying and pasting the words as their lemmas (84.13%). Among the neural lemmatisers, all context-aware models outperform the context-free model, by 1.65% on average. This encourages the idea of context-aware neural lemmatisation.

Although the context-aware neural lemmatisers achieve very high accuracies, we are still curious about the mistakes they make. There are three main types of mistakes. First, for unseen *characters*, the output is always the least frequent characters, and different models may choose different characters. An example is displayed in Table 4.2.

Input 兎				**Reference** 兎	
char5	儁	**bpe5**	犁	**worpheme5**	[illegible]
char10	麾	**bpe10**	埠	**worpheme10**	攝
char15	咳	**bpe15**	恵	**worpheme15**	櫓
char20	惇	**bpe20**	瀑	**worpheme20**	梃
char25	-	**bpe25**	峯	**worpheme25**	攝
char30	梱	**bpe30**	畑	**worpheme30**	槌
		base0 兎			

Table 4.2: An example of how different models lemmatise an unseen *character* by outputting different least frequent characters. Only the **base0** lemmatiser is able to output the correct answer by simply copying and pasting the word as its lemma.

In contrast, **base0** is good at copying and pasting the unseen *characters*, especially when there is only one single character to be lemmatised as the above example.

Second, while the context-free lemmatiser simply copies and pastes the words as their lemmas mostly, the context-aware lemmatisers seem to be able to disambiguate between different lemmas for the same word form, to a limited extent though. Some obvious examples are the following frequent ambiguous words with their possible lemmas in the brackets: だ (に, だ, で), で (で, でる), し (し, する), さ (さ, する), etc. The correct disambiguation of them largely accounts for the increase in the accuracies of the context-aware lemmatisers. Instead of *blindly* outputting the most frequent lemma for each word, the context-aware lemmatisers all seem to have something else to rely on to make their judgments. As one of the main objectives of the neural lemmatisation experiments is to see whether neural models are making use of the context, we plot out the attention maps for some correctly and incorrectly lemmatised ambiguous words by models of different context unit types or varying lengths of context (See

Figure A.1 vs. Figure A.2 in Appendix A). Even though some models can correctly lemmatise some test examples, their attention maps are not informing any use of context. At each decoding step, almost 100% attention is attended to part of the centre word; negligible attention is distributed to the rest of the sentence.

Third, **base0** is confused a lot by words in the same form but with different POS tags, and the context-aware lemmatisers do not seem to bring obvious improvement on ambiguous words regarding POS tags. Some examples include 楽しむ -> 楽しむ/楽しみ where both 楽しむ and 楽しみ share the same stem 楽し but have different inflectional suffixes which indicate their POS (む: verb; み: noun). All models, regardless of the context units and context lengths, make mistakes in lemmatising POS-ambiguous words.

4.1.2 sent2sent lemmatisers

The two **sent2sent** lemmatisers, **sent2sent-char** and **sent2sent-bpe**, are evaluated in terms of sentence-level F1 score of the correctly lemmatised words. We prefer F1 than accuracy here because, even though the segmentation information is provided (by adding space symbols <s> between groups of characters that make up words), there is no guarantee that the neural models could produce exactly the same number of words as in the input (or reference output). This concern is also supported by our empirical results (Table 4.4 & Table 4.5, explained later). In case of length mismatch between input (or reference output) and actual output, F1 score can still ensure the smooth progress of evaluation.

Models	F1	Encoder lexicon size	Overlapping vocabulary
sent2sent-char	0.9853	2818	2818
sent2sent-bpe	0.9796	3278	

Table 4.3: The **sent2sent** lemmatisers: F1 score on the UD_Japanese test set and encoder lexicon statistics. The two models achieve close results as the **bpe** lexicon includes the **char** lexicon, and the byte pairs are not bringing too much difference.

Both **sent2sent** lemmatisers achieve an F1 score around 0.98, and the gap between them is negligible 0.0057 (Table 4.3). As pointed out in Section 4.1.1, the **bpe** lexicon learned from the current corpus shares most entries with the **char** lexicon. The fact that the **bpe** lexicon actually includes the whole **char** lexicon, might be contributing to their close F1 scores here as well. Although the **char** lemmatiser still outperforms the

bpe lemmatiser (by less than 0.01), rather than believing this is a systematic weakness of the latter, we would attribute this subtle difference to hyper-parameter tuning.

As a lemmatiser given perfect segmentation, it is supposed to give an output of the same length as the corresponding input. However, it remains a question whether our **sent2sent** lemmatisers are actually producing the correct length of output, and this information is not informed by the overall F1 scores. So we have to check whether the output length matches the input length for each test sentence. Here we take the sentences with mismatched length from both lemmatisers (Table 4.4 for the **char** lemmatiser and Table 4.5 for the **bpe** lemmatiser). Among the 558 test sentences, only 4 (**char**) or 5 (**bpe**) of them mismatched in length. Due to limited space, we only show part of the sentences in Table 4.4 and Table 4.5. We mark the incorrectly lemmatised units in the output red, and mark the missing output in the reference green. For both lemmatisers, the biggest length match comes from sentence **511**, 40 (**char**) and 34 (**bpe**) words respectively. The average length mismatch for the two lemmatisers is 25.5 (**char**) and 15.4 (**bpe**) words. The **char** lemmatiser always terminate the output halfway; for the **bpe** lemmatiser, three of the sentences (**335**, **463** and **511**) are not finished but the rest two are output more than the reference: the red extra character at the end of sentence **33** is mistakenly output, and the red part in sentence **85** is a disordered repeating of part of the sentence. Coincidentally, the output lengths of the last three sentences in **char** are all around 70, and those of **bpe** around 80. We conjecture that this might be due to the inability of neural networks to handle long sequences: the information flow becomes increasingly weak with the increasing sequence length. Besides, as the sequences in **bpe** are shorter than those in **char**, the **bpe** lemmatiser is more likely to output sequences of longer lengths (close to the actual lengths of the reference or input), as the signal is stronger even after having gone through a long distance.

4.1.3 The best lemmatiser

The lemmatiser with the best performance on the test set[1] is selected as the best lemmatiser, among the **word2word** lemmatisers in Section 4.1.1 and the **sent2sent** lemmatisers in Section 4.1.2. Although, the two groups of lemmatisers are evaluated by

[1]it makes more sense to choose the best model based on the validation performance, but in the current and the following model selection, the model selected by validation performance is the same as the one selected by test performance. As we do not include the validation scores anywhere in this dissertation, we just select models based on test performance.

Sentence id		Reference & Output	Length
246	Ref	... タクシー が ずーーーっと 待つ て ます ので 急ぐ て くださる と 言う て 来る ます た	39
	Out	... タクシー が ずーーーっと 待つ て 来る ます た	31
335	Ref	... これ まで 不足 する て いる た 公園 緑地 の 確保 による 都市美 化 従来 の 不規則 だ 街区 整理 を 目的 に 着実 だ 着工 する	89
	Out	... これ まで 不足 する て	68
463	Ref	... 日常的 だ 行う れる たり 日本列島 を 指す 島国 という 言葉 が 劣等 だ 未開 という 意味 で 使う れる て 駐日韓国大使 が テレビ の インタビュー で 使う まで に なる て いる こと から も 伺う *可能*	102
	Out	... 日常的 だ 行う れる た	69
511	Ref	... 山地 の 傾斜 面 を 切り開く て 棚田 を つくる まで に 至る ない た ところが 多い 棚田 は あまり つくる れる ない か つくる れる た 場合 だ 畔 や 土手 は 傾斜 が 緩やか だ 土盛り と なる 西日本 と は 対照的 だ 棚田 風景 と なる た	113
	Out	... 山地 の 傾斜 面 を 切り開く て 棚田 を	73

Table 4.4: Test sentences with mismatched length in the **sent2sent-char** lemmatiser. The reference outputs all terminate halfway, three of which with a length around 70.

different metrics, i.e. token-level accuracy and sentence-level F1 score, as F1 is a benchmark for accuracy which balances precision and recall, the comparison is valid.

Averaging the scores by all **word2word** lemmatisers and the scores by the two **sent2sent** lemmatisers, we come to an intermediate comparison result that, in general, the **word2word** (average accuracy: 98.70%) lemmatisers outperform the **sent2sent** (average F1 score: 0.9825) lemmatisers by 0.0045. The inferiority of the **sent2sent** lemmatisers could be partly explained by the length mismatch problem, which is never

Sentence id		**Reference & Output**	**Length**
33	Ref	愛人　が　いる　たら　怒る　だ	6
	Out	愛人　が　いる　たら　怒る　だ ょ	7
85	Ref	... だ　ので　司法解剖　によって　わかる　類　の 因果関係　と　ある　はず　も　ある　ます　ない	33
	Out	... だ　ので　司法解剖　によって　わかる　類　の 因果関係　と　ので　司法解剖　によって　わかる 類　の 因果関係　と　ある　はず　も　ある　ます　ない	41
335	Ref	... これ　まで　不足　する　て　いる　た　公園　緑地 の　確保　による　都市美　化　従来　の　不規則 だ　街区　整理 を　目的　に　着実　だ　着工　する	89
	Out	... これ　まで　不足　する　て　いる　た　公園　緑地 の　確保 による　都市美　化　従来	78
463	Ref	... 日常的　だ　行う　れる　たり　日本列島　を　指す 島国　という　言葉　が　劣等 だ　未開　という　意味 で　使う　れる　て　駐日韓国大使　が　テレビ　の インタビュー　で　使う　まで　に　なる　て　いる こと　から　も　伺う　*可能*	102
	Out	... 日常的　だ　行う　れる　たり　日本列島　を　指す 島国　という　言葉　が　劣等 で	79
511	Ref	... 山地　の　傾斜　面　を　切り開く　て　棚田 を　つくる　まで　に　至る　ない　た　ところが 多い　棚田　は　あまり　つくる　れる　ない　か つくる　れる　た　場合　だ　畔　や　土手　は 傾斜　が　緩やか　だ　土盛り　と　なる　西日本 と　は　対照的　だ　棚田　風景　と　なる　た	113
	Out	... 山地　の　傾斜　面　を　切り開く　て　棚田 を　つくる まで　に　至る　ない　た　とこ	79

Table 4.5: Test sentences with mismatched length in the **sent2sent-bpe** lemmatiser. Two of the sentences output more than the reference; the other three sentences terminate halfway, with a length of around 80.

a problem for the **word2word** lemmatisers. Although regarding the overall scores, the **sent2sent** lemmatisers are not preferred more, they do not encode context for each word, resulting in a much smaller training set, much shorter runtime, and a more efficient training process. As long as the length of the test sentences can be controlled, the length mismatch problem could be largely got rid of, thus improving the overall performance of the **sent2sent** lemmatisers.

Among all 21 neural lemmatisers, **word2word-char30** achieves the highest accuracy 98.99%, thus selected as the best lemmatiser, and utilised in the following experiments.

It is worth noticing that the scores are very close between the best lemmatiser and its followers such as **word2word-char25** and **word2word-char20** (some of the possible reasons already discussed in Section 4.1.1), and the superiority of **word2word-char30** could be inconsistent under different settings of hyper-parameters.

4.2 Segmentisers

4.2.1 sent2sent segmentisers

The **sent2sent** segmentisers are evaluated in terms of F1 score of correctly segmented words. F1 score takes into account the issues of length mismatch between the reference output and the actual output. We compare the F1 scores of our own segmentisers with those of the three existing Japanese morphological analysers: **MeCab**, **JUMAN** and **JUMAN++**. As introduced in Section 3.2.4, these three analysers can segment an un-segmented sequence. As the baselines all heavily rely on their own dictionaries to look up entries during segmentation, and Jumandic is the only dictionary available to all three analysers, we use Jumandic for a fairer comparison; meanwhile, we also load **MeCab** with its recommended dictionary IPAdic. The F1 scores of the **sent2sent** segmentisers **char** and **bpe** with characters and byte pairs as the basic unit and the baselines are shown in Table 4.6.

All three baselines, when using the same dictionary Jumandic, perform similarly regarding the overall F1 score. However, **MeCab** with IPAdic achieves a much higher F1 score, 0.9067, than the other models, achieving 0.7697 on average. This supports our hypothesis in Section 3.2.4 that the inconsistent use of dictionary might cause systematic errors in morphological analysis. To figure out the inconsistency and the incompatibility between different dictionaries, we look at the actual output of the above

Model		Dictionary	F1 score
baseline	**MeCab**	IPAdic	0.9067
	MeCab	Jumandic	0.7667
	JUMAN	Jumandic	0.7708
	JUMAN++	Jumandic	0.7716
sent2sent-	**char**	N/A	0.9208
	bpe	N/A	0.9223

Table 4.6: Segmentation F1 scores of the baselines and the **sent2sent** segmentisers. **MeCab** with IPAdic works the best; with the same dictionary Jumandic, all three baselines achieve close results. The two **sent2sent** segmentisers perform similarly.

models. There are four main types of inconsistency in segmentation and the resultant inconsistency in lemmatisation between IPAdic and Jumandic (examples in Table 4.7). First, they deal with inflected verbs in different ways: IPAdic tends to split the stem and the inflectional suffix but Jumandic does not. Second, for nouns with prefixes, IPAdic treats them as a word but Jumandic separates the prefixes from the noun. Third, a compound preposition consisting of a preposition and a verb is split by Jumandic but not by IPAdic. Fourth, IPAdic segments adjectival verbs[2] with preposition into two parts; while Jumandic does not perform. These differences in segmentation lead to divergence in lemmatisation, as the number of units to be lemmatised are different. However, it remains unknown how much of the segmentation or lemmatisation by the baselines is effected by using different dictionaries; to figure it out, a great amount of manual correction and inspection is necessary, which is impossible to be carried out in this dissertation. From our observation, UD_Japanese matches more with the annotation style of IPAdic. This explains the high score of **MeCab** equipped with IPAdic.

We move on to explore the behaviour of **JUMAN** and **JUMAN++** when they both use Jumandic. The F1 scores of **JUMAN** and **JUMAN++** are similar, with **JUMAN++** being a bit higher. Interestingly, the interpolation of RNNLM with n-gram LM (Section 3.2.4) in **JUMAN++**, does not bring obvious improvement compared with **JUMAN**, although it claims to work better in *some* ambiguous cases as RNNLM integrates semantic context (Morita et al., 2015). From our observation, **JUMAN++** seems to

[2]adjectival verbs, 形容動詞, are verbs which can be used as if they were adjectives; they become adverb-like when appended with prepositions

Category	Example	Seg & Lem	IPAdic	Jumandic
Inflected verbs	表立って	seg	表立っ　て	表立って
		lem	表立つ　る	表立つ
Prefixed nouns	お話	seg	お話	お　話
		lem	お話	お　話
Combined preposition	について	seg	について	に　ついて
		lem	について	に　つく
Adjectival verbs + preposition	明らかに	seg	明らか　に	明らかに
		lem	明らかに	明らかだ

Table 4.7: Examples of inconsistency in segmentation and lemmatisation between IPAdic and Jumandic (seg: segmentation; lem: lemmatisation).

be especially good at handling the following types of ambiguous cases: short nouns, names, and words written in the same character types. For instance, while **JUMAN** segments 政党と柴犬[3] (political parties and Shiba Inu[4]) into 政党　と　柴　犬 (political parties and firewood dog), **JUMAN++** can correctly segment it into 政党　と　柴犬 (political parties and Shiba Inu). See more ambiguous examples in Table 4.8.

From now on, for a fairer comparison between the baselines and our segmentisers, we only use the results from the baselines equipped with the same dictionary Jumandic.

Compared with the baselines, both **sent2sent** segmentisers achieve much higher F1 scores, with **bpe** surpassing **char** by 0.0062. As the byte pairs are learned within words without crossing word boundaries, the byte pairs are providing the correct segmentation to the **bpe** segmentiser, which is not the case for the **char** segmentiser. For example, in the sequence of byte pairs 美　味　しい　です　ね, the true segmentation is 美味しい　です　ね. As we already know that しい, as a byte pair, must be grouped together, the segmentiser does not need to judge whether to output a space symbol between し and い or not; however, in the **char** segmentiser, it faces し and い as two characters to be either grouped or split. In this sense, byte pairs are offering important prior information about segmentation, which benefits the segmentiser ("cheating") to some extent.

Besides F1 scores, we also check the length match of both segmentisers. Among

[3]the original sentence was: 政党と柴犬を同列に扱うというのもなかなかです
[4]**Shiba Inu** or 柴犬, a type of dog, literally means *firewood* and *dog*

Example	Output
政党と柴犬	JUMAN: * 政党　と　柴　犬 (political parties and firewood dog) JUMAN++: 政党　と　柴犬 (political parties and Shiba Inu)
小林香菜	JUMAN: * 小林　香　菜 (family name + split first name) JUMAN++: 小林　香菜 (family name + first name)
橋下知事	JUMAN: * 橋　下　知事 (Bridge down governor) JUMAN++: 橋下　知事 (Hashimoto governor)
劣等国有史以来	JUMAN: * 劣等　国有　史　以来 (inferior state-owned history since) JUMAN++: 劣等　国　有史　以来 (inferior country history since)
つくられないかつくられた	JUMAN: * つくら　れ　ない　かつ　くら　れた (cannot make and warehouse could) JUMAN++: つくら　れ　ない　か　つくら　れた (cannot be made or be made)

Table 4.8: **JUMAN** vs. **JUMAN++**: Segmentation of ambiguous cases.

the 558 test sentences, the **char** segmentiser outputs 272 (error rate: 48.75%) of them with wrong lengths, and the **bpe** segmentiser makes length mismatch mistake in 251 (error rate: 44.98%) of them. Almost half of the sentences are not segmented into correct length by both segmentisers. The idea of constraining output length naturally comes to us. As described in Section 3.2.2.2, we experiment with building an external length predictor as the first approach, and putting length constraints during validation and test time as the second approach.

As almost 50% of the sentences are segmented into wrong lengths, we only adopt a length predictor if its length prediction accuracy is above 50%; otherwise, it might even worsen the segmentation models by providing inaccurate length information. For our length predictor, the input is always a sequence of characters without segmentation information, but we format the reference output in two ways: numbers as digits (e.g. `45` as `4 5`) to generate unseen numbers flexibly, or numbers as complete numbers themselves (e.g. `45` as `45`). As results, although both versions of the predictor obtain high accuracies (99.61%) on the dev set, neither of them could get a decent accuracy on the test set (3.40% & 1.25%), which would only add noises on the segmentisers if adopted. Therefore, our **sent2sent-lenpred** model is failed. The failure of the length predictor is totally not surprising. After all, the symbols indicating lengths cannot convey ordinal information, and are treated as normal vocabulary by the neural networks.

Our second approach is more straightforward as it directly constrains the output length. We experiment with the best segmentiser so far, the **sent2sent-bpe** segmentiser, by constraining the output length up to twice the input length[5], while keeping the other parameters unchanged. The F1 score of the model with length constraints **sent2sent-bpe-maxratio2** is 0.9270, higher than that of **sent2sent-bpe**, 0.9223. Although the error rate of length mismatch of **sent2sent-bpe-maxratio2** is 44.98%, on par with that of **sent2sent-bpe**, the total amount of mismatched length is decreased by 49 units from 413 units. See Table 4.9 for a detailed comparison of length mismatch error rate and length mismatch unit counts among the **sent2sent** segmatisers. To conclude, putting length constraints during decoding is useful to improve the performance of the **sent2sent** segmentisers.

Model	Mismatch error rate	Mismatch unit count
char	48.75%	457
bpe	44.98%	413
bpe-maxratio2	44.98%	364

Table 4.9: Length mismatch error rate and mismatch unit count for the **sent2sent** segmentisers. Adding output length constraint does mitigate the length mismatch problem to some extent, especially regarding the unit count.

4.2.2 The best segmentiser

By simply comparing the F1 scores of the above models including the baselines and our **sent2sent** segmentisers, we select **sent2sent-bpe-maxratio2** as the best segmentiser.

4.3 Segmatisers

4.3.1 Baseline segmatisers

The three baseline segmatisers are: **MeCab**, **JUMAN** and **JUMAN++**. As we already know the issues of inconsistent segmentation from Section 4.2.1, we infer that there must be inconsistency in their lemmatisation as well, because they rely on dictionary look-ups to lemmatise the segmented units. If the sequence can be segmented into various ways, the lemmatisation would also vary. All three baselines use the same

[5]in Nematus, we set `max-ratio` to `2` to constrain the output length up to twice the input length

dictionary Jumandic to segmatise the test set. Table 4.10 displays their F1 scores of correctly lemmatised segmented units (**MeCab** using IPAdic included).

Model	Dictionary	F1 score
MeCab	IPAdic	0.8792
MeCab	Jumandic	0.7952
JUMAN	Jumandic	0.8081
JUMAN++	Jumandic	0.8144

Table 4.10: Segmatisation F1 scores of the baseline segmatisers. **MeCab** with IPAdic works the best; with the same dictionary Jumandic, all baselines achieve similar F1 scores.

It is clear to tell from Table 4.10 that, the inconsistency issue of using different dictionaries still exist, as expected. When using the same dictionary Jumandic, all three baselines perform similarly regarding F1 score, with **MeCab** falling behind the two Juman family members only by 0.1605 on average. Between the two Juman family analysers, **JUMAN++** outperforms **JUMAN** only by 0.063. This superiority should be partly accounted as the better segmentation performance of **JUMAN++** (Section 4.2.1).

4.3.2 Pipeline segmatisers

Combining any lemmatiser in Section 4.1 with any segmentiser in Section 4.2, we obtain a pipeline segmatiser. The core assumption of a pipeline model is that, the steps or the modules in the model are independent of each other and optimised separately. In our case, where the output of segmentation is used as the input for lemmatisation, we are assuming that for non-segmented Japanese, segmentation and lemmatisation could be independently optimised. However, there is no guarantee that separate optimisation would bring the optimal overall result. So, does this drawback, resulting from the core assumption of pipeline models, also exist in our pipeline segmatiser?

We obtain our best pipeline segmatiser by combining the best segmentiser in Section 4.2, the **sent2sent-bpe-maxratio2** segmentiser, and the best lemmatiser in Section 4.1, **word2word-char30**. We take the segmentation results from the **sent2sent-bpe-maxratio2** segmentiser, and format it as input for the **word2word-char30** lemmatiser. For example, suppose **sent2sent-bpe-maxratio2** segments the 1st following sentence into the 2nd line, but the input it actually accepts (byte pairs underlined) and its output are the 3rd and the 4th line (the first four lines on the next page):

```
1 | 高い点数を取りたい (“I want to achieve high scores.”)
2 | 高い 点数 を 取り たい
3 | 高 い 点 数 を 取 り たい
4 | 高 い <s> 点 数 <s> を <s> 取 り <s> たい
```

The output then has to be formatted appropriately as the input for **word2word-char30**, such as breaking the byte pairs into characters when they are the centre word to be lemmatised (the 5th line on the next page):

```
5 | <w> 高 い <s> 点 数 <s> を <s> 取 り <s> <lc> た い <rc> </w>
```

The accuracy of our best pipeline segmatiser is 92.46%.

The pros and cons of pipeline Japanese segmatisers are discussed based on our observation. First, as pipeline segmatisers assume independence between segmentation and lemmatisation, the fine-tuning for both tasks take a long time. In our case, we attempt with twenty-one (**word2word**) + two (**sent2sent**) variants of lemmatisers and two (**sent2sent**) variants of segmentisers experimented with two approaches to improve length matching. Training alone has taken almost one and a half months, let alone the time for fine-tuning. Second, when we have a large amount of training data and use the same data for training the lemmatisers and the segmentisers in parallel, it is time-consuming, and we would naturally come to the idea of training them together if possible, as long as the overall performance is not worsened. On the other hand, the pipeline segmentisers could work reasonably well if a large amount of annotated data for either task could be collected. For example, for our segmentisers, we can make use of any segmented text from other corpora. We believe separate optimisation allows the separate models to learn from more data, as long as there is a lot of data for either task to ensure their better training outcome.

4.3.3 sent2sent segmatisers

For **sent2sent** segmatisers, there are two variants, **char** and **bpe**, using character and byte pairs as the basic units respectively. Their performance is evaluated by F1 score of correctly lemmatised segmented units.

Model	F1	Encoder lexicon size	Overlapping vocabulary
char	0.9208	2818	2818
bpe	0.9198	3278	

Table 4.11: The **sent2sent** segmatisers: F1 score on the UD_Japanese test set and encoder lexicon statistics. The two models achieve close F1 scores as their lexica share a lot of entries.

From Table 4.11, both joint segmatisers achieve an F1 score as high as around 0.92. The tiny gap of 0.001 between them might be explained by the same hypothesis we have made in Section 4.1.2 that, the high overlap between the character lexicon and the bpe lexicon accounts for the close results of the corresponding models. Besides, the **bpe** segmatiser has the benefits from the byte pairs which provide some ready answer for segmentation as discussed in Section 4.2.1, but it lacks the flexibility that **char** has.

More than numbers, we inspect the output by the joint segmatisers, looking for significant patterns. As we care about the length mismatch problems witnessed in neural segmentisers in Section 4.2.1, we also check the length mismatch of the output from the **sent2sent** segmatisers here (Table 4.12, two new segmatisers **char-maxratio2** and **bpe-maxratio2** with length constraints `maxratio=2` also included). The segmatisers got wrong lengths for almost half of the test sentences; putting length constraints by setting `maxratio` to `2` is only helpful to a limited extent.

Model	F1	Mismatch error rate	Mismatch unit count
char	0.9208	45.88%	467
char-maxratio2	0.9218	45.52%	372
bpe	0.9174	45.52%	409
bpe-maxratio2	0.9198	45.88%	384

Table 4.12: Length mismatch error rate and mismatch unit count for the **sent2sent** segmatisers. Adding output length constraints works for both models, regarding mismatch unit count.

Some types of segmentation mistakes by the joint segmatisers are listed as follows. First, they are bad at segmenting/recognising long words (characters/byte pairs which make up a long word), and tend to break them into units and lemmatise them correspondingly. Some examples are (in each example, the 1st line is the un-segmented text; the 2nd line is the reference segmatised output; the 3rd line is the actual segmatised output; the 4th line is the corresponding segmentation inferred from the segmatisation):

1	とんでもない	机上の空論
2	とんでもない	机上の空論
3	とん だ も ない	机上 の 空論
4	とん で も ない	机上 の 空論

Meanwhile, they also tend to combine adjacent words together into a long word, such as the below examples (the first line: un-segmented text; the second line: the reference segmatised output; the third line: the actual segmatised output; the fourth line: the corresponding segmentation inferred from the segmatisation):

1	多少左右	カナダメキシコ	もちろんまじめ
2	多少 左右	カナダ メキシコ	もちろん まじめ
3	多少左右	カナダメキシコ	もちろんまじめ
4	多少左右	カナダメキシコ	もちろんまじめ

The elements in these combined words are mostly written in the same character types. In the examples, the first is combining two words written in *kanji*; the second two words in *katakana*; the third two words in *hiragana*.

4.3.4 The better segmatiser

To remind the readers of the best pipeline segmatiser and the joint segmatiser, we compare them regarding F1 score and mistakes they make (Table 4.13).

Model	**F1**
pipeline: sent2sent-bpe-maxratio2 + word2word-char30	0.9246
joint: sent2sent-char-maxratio2	0.9218

Table 4.13: F1 score of the pipeline segmatiser vs. the joint segmatiser. The **joint** model is not worse than the **pipeline** model a lot.

Simply from the above F1 scores, the **pipeline** segmatiser seems more preferred. However, it only surpasses the **joint** segmatiser by tiny 0.0028. Given perfect segmentation, the lemmatiser **word2word-char30** that is part of the pipeline model can lemmatise 98.99% of the test tokens correctly. In contrast, when given the segmentation results from the segmentiser **sent2sent-bpe-maxratio2**, which has a high length mismatch error rate 44.98% and an F1 score of 0.9270, the overall performance of the pipeline segmatiser is thus largely adversely influenced by the performance of the segmentiser.

Regarding segmentation, its performance in the pipeline segmatiser can be informed by the F1 score, but its performance in the joint segmatiser cannot be directly measured, even though the F1 score still implicitly indicates to some extent. Another direct and fair yet imperfect metric is the length mismatch error rate. The length mismatch error rate of the **sent2sent-bpe-maxratio2** segmentiser is 44.98% (364 units), and the **sent2sent-char-maxratio2** segmatiser 45.52% (372 units). It seems that overall, jointly learning segmentation and lemmatisation is not helping too much to eliminate the length mismatch problem.

One thing we have noticed from the pipeline segmatiser is that, once the segmentiser has made any mistakes of wrong segmented units, it is impossible for the lemmatiser to correct them. This is understandable because for each word, the lemmatiser must always output its corresponding lemma. Unlike the pipeline segmatiser, we expect to see segmentation and lemmatisation facilitating each other in the joint segmatisers: segmentation is improved as lemmatisation could help decide word boundaries; meanwhile lemmatisation can be helped by segmentation accurately segmenting units that can be encoded as useful context. However, we witness the failure of outputting the correct characters in the joint segmatiser, such as some names containing infrequent *kanji*[6]. Infrequent characters could not even be *blindly* copied and pasted by the segmatiser, but they are not a problem as serious to the standalone segmentiser as to the joint segmatiser. Therefore, in the case of infrequent characters, the pipeline segmatiser is more likely to get them correct. However, neither segmatisers are able to handle *unseen characters*. This is hard to imagine in some other languages such as English, as the character lexicon is usually small and finite - whatever new words are made up, they must be constituted by the characters in the lexicon. Unlike that, the Japanese character lexicon is unable to cover all possible characters[7], based only on a corpus with a very limited coverage of characters, let alone the more infrequent ones.

[6]the joint segmatiser could not correctly handle names such as 石川悦男 which contains infrequent Chinese characters 悦 and instead alters the string as 石川豹男; the same phenomenon is also seen in other types of words where infrequent characters appear, e.g. 湘南 色 altered as 纏 南色 in the joint segmatiser

[7]besides the finite set of *hiragana* and *katakana*, there is a much large vocabulary of *kanji*, which is supposed to be finite as well, contributing to the explosion of the character lexicon size

Chapter 5

Conclusion

We explore the application of sequence to sequence models on Japanese lemmatisation, segmentation, and joint segmentation and lemmatisation - segmatisation. We are now able to answer the three questions raised in Section 1.2:

1. We extend neural lemmatisation to Japanese. Among all the variants of lemmatisers we build, the one encoding context of 30 characters from both sides perform the best. Although context-aware lemmatisers all work better than the context-free lemmatiser, it remains unknown whether the context is the key to attribute to.

2. By modelling Japanese word segmentation as a sequence labelling problem of predicting the probabilities of word breaks in a sequence of characters, we succeed in constructing a sequence to sequence Japanese segmentiser. Our segmentiser which uses byte pairs as basic units and under output length constraints perform the best among the variants. However, segmentation turns out to be more challenging than lemmatisation for Japanese: the latter has hard alignment between the words and the lemmas; while the former faces complicated Japanese writing system and large-sized lexica.

3. Neural pipeline analysers outperform joint end-to-end analysers. However, the pipeline models still suffer from the high time cost of separate optimisation; and the joint models are only short of the pipeline counterparts by a very small gap. There is weak evidence that segmentation and lemmatisation can facilitate each other in the joint models, and it is almost impossible for the lemmatiser to correct incorrect segmentation in the pipeline models. When there is not enough training

data for both parts and enough time for separate fine-tuning, we would prefer the joint models as they are more efficient.

5.1 Limitations

5.1.1 Data

The corpus we deal with is the UD_Japanese corpus which is not a popular choice and not used by any previous studies, so it is hard for us to compare our work with the others directly[1]. If we had the opportunity to work with another corpus, we could have been able to test the robustness of our analysers. After all, solving a problem does not equal to achieving state-of-the-art on some specific corpora (Lipton and Steinhardt, 2018). We hope our proposal of joint *seq2seq* Japanese segmentation and lemmatisation could achieve satisfying results on the mainstream corpora as well, and shed some light on non-segmented language processing.

5.1.2 Sub-word units

Byte pair is an important sub-word unit in our models. We experiment with BPE as it improves the performance of NMT on morphologically complex German by largely reducing data sparsity and increasing the learning of rare and unseen words (Sennrich et al., 2015). However, BPE is not the panacea to every language. It is inappropriate for Japanese as the language has completely different features from languages such as German: short morphemes, large sized character lexicon, tons of unseen *characters*, etc. 500 times of byte pair merging operation is still too many for Japanese. The results of BPE models are actually very close to those of the character-level models. In future studies, we would not experiment with BPE anymore and would not recommend using it on languages such as Chinese.

5.1.3 Training and computational resources

Due to the time limit of each job submitted to the GPU cluster based in the University of Edinburgh, we are forced to reload partially trained models again and again until they are early stopped. So it is possible that some of the models are sub-optimal.

[1]but this is the only reliable corpus we could get access to for free at the point of starting this dissertation

5.2 Future work

To test the robustness of our models (lemmatisers, segmentisers and segmatisers), we would repeat at least the joint end-to-end segmatisation experiments on one of the mainstream corpora BCCWJ. In addition to Japanese, it would be interesting to see how our works could be transferred to Korean, which shares almost the same language topology with Japanese.

The most interesting and challenging task we find is Japanese segmentation. Based on the error analysis of our segmentisers, we would propose to integrate the character type information with our model, as words with the same character type are much more likely to be in one word.

Even though *seq2seq* does show the power of neural models on traditional NLP tasks, we do not deny the effectiveness of the existing Japanese morphological analysers which we use as baselines. As long as the dictionaries they are equipped with are open and updated, the performance of the analysers should not be worsened.

Bibliography

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

Bergmanis, T. and Goldwater, S. (2018). Context sensitive neural lemmatization with lematus. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, volume 1, pages 1391–1400.

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

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

Fraser, A., Weller, M., Cahill, A., and Cap, F. (2012). Modeling inflection and word-formation in smt. In *Proceedings of the 13th Conference of the European Chapter of the Association for Computational Linguistics*, pages 664–674. Association for Computational Linguistics.

Gage, P. (1994). A new algorithm for data compression. *The C Users Journal*, 12(2):23–38.

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

Jiang, W., Huang, L., Liu, Q., and Lü, Y. (2008). A cascaded linear model for joint chinese word segmentation and part-of-speech tagging. *Proceedings of ACL-08: HLT*, pages 897–904.

Kruengkrai, C., Uchimoto, K., Kazama, J., Wang, Y., Torisawa, K., and Isahara, H. (2009). An error-driven word-character hybrid model for joint chinese word segmentation and pos tagging. In *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP: Volume 1-Volume 1*, pages 513–521. Association for Computational Linguistics.

Kudo, T., Yamamoto, K., and Matsumoto, Y. (2004). Applying conditional random fields to japanese morphological analysis. In *Proceedings of the 2004 conference on empirical methods in natural language processing*.

Kurohashi, A. T. S. (2018). Juman++ v2: A practical and modern morphological analyzer.

Kurohashi, S. and Nagao, M. (1998). Building a japanese parsed corpus while improving the parsing system. In *Proceedings of The 1st International Conference on Language Resources & Evaluation*, pages 719–724.

Kurohashi, S. and Nagao, M. (2003). Building a japanese parsed corpus. In *Treebanks*, pages 249–260. Springer.

Lipton, Z. C. and Steinhardt, J. (2018). Troubling trends in machine learning scholarship. *arXiv preprint arXiv:1807.03341*.

Ma, X. and Hovy, E. (2016). End-to-end sequence labeling via bi-directional lstm-cnns-crf. *arXiv preprint arXiv:1603.01354*.

Maekawa, K. (2007). Kotonoha and bccwj: development of a balanced corpus of contemporary written japanese. In *Corpora and Language Research: Proceedings of the First International Conference on Korean Language, Literature, and Culture*, pages 158–177.

Maekawa, K., Yamazaki, M., Ogiso, T., Maruyama, T., Ogura, H., Kashino, W., Koiso, H., Yamaguchi, M., Tanaka, M., and Den, Y. (2014). Balanced corpus of contemporary written japanese. *Language resources and evaluation*, 48(2):345–371.

Matsumoto, Y., Kitauchi, A., Yamashita, T., Hirano, Y., Matsuda, H., Takaoka, K., and Asahara, M. (2000). Morphological analysis system chasen version 2.2. 1 manual. *Nara Institute of Science and Technology*.

Mikolov, T., Karafiát, M., Burget, L., Černockỳ, J., and Khudanpur, S. (2010). Recurrent neural network based language model. In *Eleventh Annual Conference of the International Speech Communication Association*.

Morita, H., Kawahara, D., and Kurohashi, S. (2015). Morphological analysis for unsegmented languages using recurrent neural network language model. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 2292–2297.

Neubig, G., Nakata, Y., and Mori, S. (2011). Pointwise prediction for robust, adaptable japanese morphological analysis. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies: short papers-Volume 2*, pages 529–533. Association for Computational Linguistics.

Plank, B., Søgaard, A., and Goldberg, Y. (2016). Multilingual part-of-speech tagging with bidirectional long short-term memory models and auxiliary loss. *arXiv preprint arXiv:1604.05529*.

Sennrich, R., Firat, O., Cho, K., Birch, A., Haddow, B., Hitschler, J., Junczys-Dowmunt, M., Läubli, S., Barone, A. V. M., Mokry, J., et al. (2017). Nematus: a toolkit for neural machine translation. *arXiv preprint arXiv:1703.04357*.

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

Shao, Y., Hardmeier, C., Tiedemann, J., and Nivre, J. (2017). Character-based joint segmentation and pos tagging for chinese using bidirectional rnn-crf. *arXiv preprint arXiv:1704.01314*.

Sun, W. (2011). A stacked sub-word model for joint chinese word segmentation and part-of-speech tagging. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies-Volume 1*, pages 1385–1394. Association for Computational Linguistics.

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.

Uchiumi, K., Tsukahara, H., and Mochihashi, D. (2015). Inducing word and part-of-speech with pitman-yor hidden semi-markov models. 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), volume 1, pages 1774–1782.

Vania, C. and Lopez, A. (2017). From characters to words to in between: Do we capture morphology? *arXiv preprint arXiv:1704.08352*.

Wang, Y., Kazama, J., Tsuruoka, Y., Chen, W., Zhang, Y., and Torisawa, K. (2011). Improving chinese word segmentation and pos tagging with semi-supervised methods using large auto-analyzed data. In *Proceedings of 5th International Joint Conference on Natural Language Processing*, pages 309–317.

Yamamoto, K., Miyanishi, Y., Takahashi, K., Inomata, Y., Mikami, Y., and Sudo, Y. (2015). What we need is word, not morpheme; constructing word analyzer for japanese. In *Asian Language Processing (IALP), 2015 International Conference on*, pages 49–52. IEEE.

Zhang, X., Cheng, J., and Lapata, M. (2016). Dependency parsing as head selection. *arXiv preprint arXiv:1606.01280*.

Appendix A

Plots: attention maps for Section 4.1.1

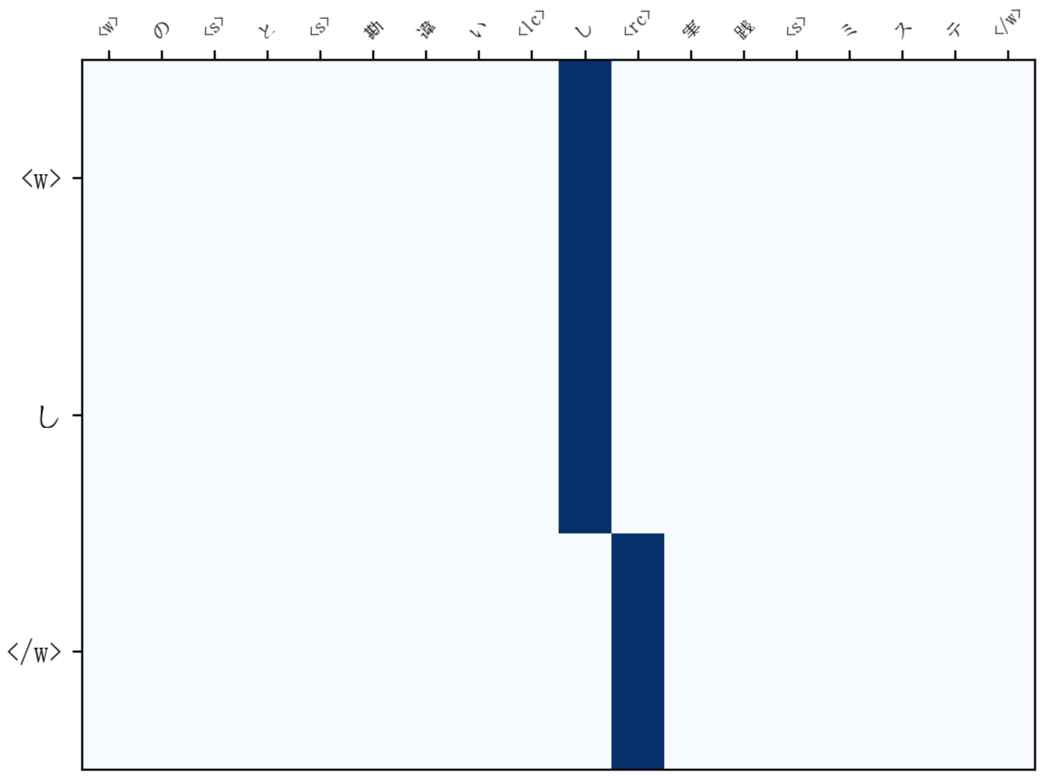

Figure A.1: Attention map for an incorrectly lemmtised example し -> する by **char5**. Most attention of the decoder side is given to the centre word of the encoder side. Attention given to the rest of the sentence is neglectable.

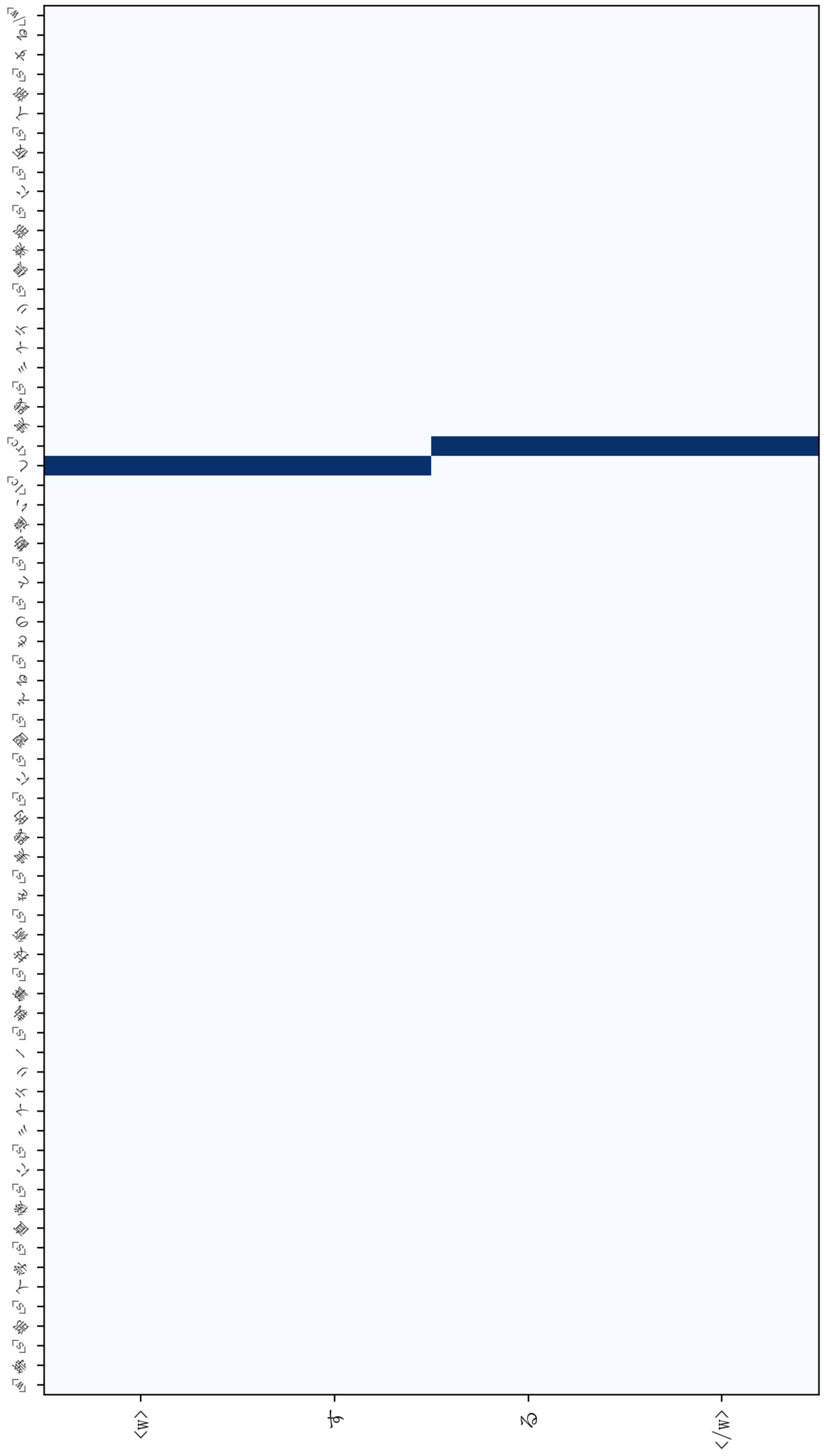

Figure A.2: Attention map for a correctly lemmtised example し -> する by **char30**.
Most attention of the decoder side is given to the centre word of the encoder side.

Appendix B

Training and hyper-parameter tuning

B.1 Hyper-parameters

		Lemmatisers	
Hyper-parameter	**Meaning**	**word2word**	**sent2sent**
`dim_word`	word embeddings dimension	300	300
`dim`	hidden layer dimension	100	100
`batch_size`	batch size	60	60
`optimizer`	optimiser	adadelta	adadelta
`maxlen`	maximum length of sentences	150	150
`weight_normalisation`	normalising weights of each hidden layer	True	True
`layer_normalisation`	batch normalisation; normalising the output of a previous activation layer	False	True
`use_dropout`	randomly drop out nodes in hidden layers	True	True
`dropout_ratio`	the ratio of nodes being dropped out	0.2	0.2
`enc_depth`	number of encoder layers	2	2
`dec_depth`	number of decoder layers	2	2
`enc_depth_bidirectional`	make this number of encoder layers bidirectional	0	2
`patience`	steps of waiting for updates in validation loss	10	10

Table B.1: Hyperparameters for the lemmatisers

		Segmentisers
Hyper-parameter	**Meaning**	**sent2sent**
`dim_word`	word embeddings dimension	512
`dim`	hidden layer dimension	256
`batch_size`	batch size	60
`optimizer`	optimiser	adadelta
`maxlen`	maximum length of sentences	150
`weight_normalisation`	normalising weights of each hidden layer	True
`layer_normalisation`	batch normalisation; normalising the output of a previous activation layer	True
`use_dropout`	randomly drop out nodes in hidden layers	True
`dropout_ratio`	the ratio of nodes being dropped out	0.2
`enc_depth`	number of encoder layers	2
`dec_depth`	number of decoder layers	2
`enc_depth_bidirectional`	make this number of encoder layers bidirectional	2
`patience`	steps of waiting for updates in validation loss	10

Table B.2: Hyperparameters for the segmentisers

		Segmatisers
Hyper-parameter	**Meaning**	**sent2sent**
`dim_word`	word embeddings dimension	512
`dim`	hidden layer dimension	256
`batch_size`	batch size	60
`optimizer`	optimiser	adadelta
`maxlen`	maximum length of sentences	150
`weight_normalisation`	normalising weights of each hidden layer	True
`layer_normalisation`	batch normalisation; normalising the output of a previous activation layer	True
`use_dropout`	randomly drop out nodes in hidden layers	True
`dropout_ratio`	the ratio of nodes being dropped out	0.2
`enc_depth`	number of encoder layers	3
`dec_depth`	number of decoder layers	3
`enc_depth_bidirectional`	make this number of encoder layers bidirectional	3
`patience`	steps of waiting for updates in validation loss	10

Table B.3: Hyperparameters for the segmatisers