Sequence to Sequence Japanese Neural Lemmatisation: doweneedsegmentation?

B117474

Word Count: 9964

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 these two types of models for 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 the best pipeline analyser and the 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) ∧ (why):

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

λ (my group members) ∧ (kind help and useful feedback)

λ (TAs, Sameer Bansal and Clara Vania) ∧ (solid support all the time)

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

λ (Arseny Tolmachev from Kyoto University) ∧ (insightful perspectives on Japanese morphological analysis)

λ (my parents) ∧ (sending me abroad)

λ (friends) ∧ (patient enough to bear with me bitching about stress)

λ (open-source community; Nematus, Lematus, etc.) ∧ (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*

Word Count: 9964)

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*. But for some languages such as Japanese, where word boundaries are not indicated, the answer is 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	.

Regarding segmentation, languages in the world can be roughly categorised into those that are segmented and those that are non-segmented. For different types of languages, the ways that computers process them vary. A related task in Natural Language Processing (NLP) is morphological analysis. It represents a series of approaches of processing natural language data by computers. Generally, 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. Segmentation is to split a chunk of text into words. 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 related morphological processes can analyse and return the morphological variants of the words in users' queries, which enlarges the search scope, thereby finding 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. pipeline models. A common approach adopted by some existing Japanese morphological analysers is to first segment the text by using language model (LM) 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 by dictionary look-up (Morita et al., 2015) (Kurohashi, 2018)[1]. These 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 optimised separately. Although segmentation is necessary as a pre-requisite for lemmatisation in pipeline models, we still question the necessity of such independence assumption in general Japanese morphological analysis (JMA), or cases where we discard the pipeline idea and carry out both tasks simultaneously. We call it joint or end-to-end.

To conduct both tasks simultaneously, we use the sequence to sequence (*seq2seq*) model with an encoder-decoder architecture proposed for MT (Sutskever et al., 2014). A *seq2seq* model maps one sequence of variable length to another, similar to a string transducer. An existing *seq2seq* framework equipped with attention called Nematus (Sennrich et al., 2017), is adopted to lemmatisation as Lematus (Bergmanis and Goldwater, 2018), which lemmatises 20 languages by mapping the inflected word with context to the corresponding lemma. Inspired by Lematus, we adopt Nematus to Japanese segmentation and lemmatisation, mapping, for example in segmentation, a sequence of characters (as a non-segmented chunk of text) to a sequence of the same characters with word boundary delimiters. We expect the joint models to be more efficient and perform better than the pipeline models.

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

1.2 Objective

So, *doweneedsegmentation*? Specifically, 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? Are segmentation and lemmatisation facilitating each other when learned jointly?

To answer them, we build 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 and its interplay with lemmatisation (Question 2), we construct neural segmentation models and combine them with lemmatisation models to make pipeline analysers. In contrast, we also build joint end-to-end models. We then compare and discuss their pros and cons, and draw the conclusion of which is the more preferred way for Japanese morphological analysis (Question 3).

1.3 Contributions

We have made the following contributions to non-segmented language processing:

- The validation of previous work in *seq2seq* neural lemmatisation on Japanese. Japanese differs itself from most Indo-European languages as it is a non-segmented language with inflectional morphology.
- The first attempt of building joint end-to-end models of Japanese segmentation and lemmatisation, which are *seq2seq* models with encoder-decoder architecture, in contrast to the existing pipeline models adopting non-neural methods. 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 small, and the superiority of the pipeline models might be insignificant.

- Detailed discussion about segmentation and lemmatisation models for Japanese and critical analysis about their pros and cons.
- A different research angle of non-segmented language processing. This could potentially boost related research in neural morphology, IR, MT, and corpus linguistics.

Chapter 2

Related Studies

We provide enough background knowledge for readers without experience in morphological analysis or neural networks (NNs) to understand the big picture.

2.1 Japanese morphological analysis

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

Segmentation, compared with lemmatisation, seems more attractive and challenging to researchers. Various solutions have been proposed: rule-based methods (Kurohashi and Nagao, 1998) (Kurohashi and Nagao, 2003), Hidden Markov Models (Matsumoto et al., 2000), CRFs (Kudo et al., 2004), and SVM (Neubig et al., 2011). From using n-gram LM 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 methods suffer from strong statistical independence assumptions and rely on manual feature engineering.

Apart from those, there has been a tendency of utilising deep learning on segmentation. For example, segmentation is still treated as sequence labelling or character tagging, but differently. BiRNN-CRF (Bidirectional Recurrent Neural Network Conditional Random Field) (Shao et al., 2017)[1] is newly proposed 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 is POS-tagged segmented words. The proposed model achieves 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 sheds light on a new perspective of JMA. It becomes our external motivation of jointly learning Japanese segmentation and lemmatisation.

2.2 Sequence to sequence neural networks with attention

Neural networks are essentially weighted nodes that operate matrix multiplication and activation functions. The weights of the nodes are optimised through backpropagation, tuning the weights by subtracting the gradients towards the direction of the steepest descent. Among various kinds of NNs, *seq2seq* NNs, 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 RNNs (Mikolov et al., 2010), i.e. encoding, then decode it into a sequence also of variable length through RNNs (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 source language is transduced into a sequence in target language. However, RNN has a problem called 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 well learned. 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. Similarly, there is gra-

[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; 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).

dient 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, attention mechanism (Bahdanau et al., 2014) between encoder and decoder is proposed to allow models 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 summation of attention, i.e. 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 the end[4]. Attention maps are a useful way to check what part of the encoder sentence the model is given attention to and how much.

For readers without a deep learning background but have been exposed to computer science, it helps to imagine *seq2seq* models as string transducers. For readers from linguistics background, it suffices to think of *seq2seq* models as black-box models which turn one sequence into another. In MT, it could be from an English sentence to its translation in Japanese. This, of course, can be extended to other NLP tasks dealing with sequences.

2.3 Sequence to sequence morphological analysis

One recent application of *seq2seq* models to NLP tasks other than MT is morphological analysis, including segmentation (Shao et al., 2017), lemmatisation (Bergmanis and

[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 (verbs underlined), *I do not eat apples* -> 私はりんごを食べない

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, because of its compounding rules where several nouns can be fused together into a compound. This leads to the data sparsity problem. As a downstream NLP task, lemmatisation could mitigate this problem. 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. As context is believed to be the key of better performance, Lematus makes use Nematus (Sennrich et al., 2017), to conduct context-sensitive lemmatisation (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 for most languages. However, Lematus lacks solid justification for supporting the hypothesis that, context is the key in improving the performance. For some languages (e.g. Table 1 from Lematus), context-free models even outperform the context-sensitive lemmatisers or are on par with them - context becomes uninformative in some cases. Now that Nematus is equipped with attention, it could have been more ideal if attention maps were demonstrated. Based on Lematus, we 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 non-trivial for non-segmented languages. By tradition, multi-class classification is the most popular view of segmen-

tation. Given a sequence of characters, segmentation models classify the characters into the beginning, the end or the middle of a word. The models 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 of these approaches, they all require sophisticated feature engineering. Inspired by the sequence modelling idea and thanks to the advent of *seq2seq* models, the idea of conducting *seq2seq* segmentation comes up to us. We model segmentation as to predict the probabilities of word breaks between adjacent characters (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 eases the data sparsity problem which the previous sequence labelling models suffer from, as we do not require feature-labelled data.

Chapter 3

Experimental Design

3.1 Data

3.1.1 Corpora and pre-processing

We use the Japanese GSD[1] corpus from Universal Dependencies (UD)[2] (hereinafter UD_Japanese).

As pre-processing, we remove Arabic numbers and symbols tagged as `PUNCT` (punctuation). After pre-processing, the number of word&lemma pairs in each dataset is: 141108 in the training set; 10093 in the development (dev) set; 11085 in the test set.

3.1.2 Statistics

To understand the corpus better, we compute the percentage of ambiguous words and word-lemma-copying (Table 3.2)[3]. Ambiguous words are words with multiple possible lemmas in the corpus (Table 3.1). Word-lemma-copying is the phenomenon that a word has itself as its lemma. The percentage of word-lemma-copying informs us of how accurate a baseline which simply copies the words as their lemmas (**base-copying**) could be. We convert the percentage into the accuracy of the baseline by subtracting the percentage from 100%.

[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] 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-existent 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 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 the expected accuracy of **base-copying** (copying words as their lemmas).

3.1.3 Basic units

In later experiments, we use three types of basic units: character, byte pair, and *wordpheme*.

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; *kanji*[6], initially used as *ateji* ("ruby"), can replace some characters in a word written in *hiragana*. See Appendix A.1 for a sample Japanese sentence. These character types contribute to the variety and the complexity of Japanese writing system, as well as the large size of a character-level lexicon. Our Japanese character lexicon contains 2818 elements (Table 3.6), while an English one may only have 150.

[4]some examples of *hiragana*: せ, い, か
[5]some examples of *katakana*: セ, イ, カ
[6]some examples of *kanji*: 清香

3.1.3.2 Byte pair

Byte Pair Encoding (BPE) is for compressing data (Gage, 1994). Byte pairs are obtained by recursively merging the most frequent byte pairs - character pairs, into new symbols. In NMT, frequent byte pairs are replaced with a symbol of the bytes combined, for example, `n g -> ng` and `i ng -> ing`. Then words such as `doing` would likely be represented as `d o 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). For efficiency, byte pairs are only learned within words[7]. The times of merging operation should be optimised according to languages - in Lematus, 500 for all 20 languages. We follow Lematus' decision, although further optimisation 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 outcome[8].

Our BPE lexicon has 500 byte pairs, 472 of which are *unique lexicon entries*[9]. Table 3.3 shows some representative byte pairs in ranked order of frequency[10]. The most frequent byte pairs are mostly inflectional morphemes: する (1st), いる (2nd), れる (3rd), ない (4th), しい (23rd), and られる (32nd). However, there are also some pairs that do not have morphological meanings, 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), に対して (*to, for*, a frequent preposition), 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 are even more deviated from the objective of extracting morphologically meaningful byte pairs in Japanese.

The BPE lexicon should be constituted by the characters in the character lexicon, plus the byte pairs. However, as we 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 the decoder side that do not appear at the encoder side. So the BPE lexicon size is: $2818 + 472 - 12 = 3278$.

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

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

[9] 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

[10] 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 3.3: Examples of byte pairs learned from UD_Japanese training set in the order of frequency (including inflectional morphemes, frequent prepositions, nouns, etc).

3.1.3.3 Worpheme

Worpheme is made up of *word* and *morpheme*, indicating its being a morpheme-like word unit. Worpheme approximates to morpheme in English, and it is also the basic segmentation unit in UD_Japanese. For example, the word 遊んだ (*"played"*) is segmented into two worphemes: 遊ん and だ, the former being the verb stem and the latter the morpheme of verb past tense inflection.

3.2 Models

We organise the experiments to answer the three sub-questions in Section 1.2. There are three types of models for: lemmatisation (lemmatisers), segmentation (segmentisers), and both (segmatisers[11]). Each model is a *seq2seq* model, implemented by Nematus. Table 3.4 displays the models, and the relationship between them. The best lemmatiser and the best segmentiser are selected among their variants; they constitute the best pipeline segmatiser. The best pipeline segmatiser is then compared with a joint segmatiser, the result of which gives us the better segmatiser. It is worth noting that, we only compare the performance scores of different models to select the *best* models. Although there are a lot more factors worth considering, e.g. efficiency and

[11] **segmatiser** is made up of *segmentiser* and *lemmatiser*

flexibility, here we focus more on the scores for easier comparison. The naming of the sub-models will be explained later.

<table>
<tr><th>Models</th><th colspan="2">Lemmatiser</th><th colspan="2">Segmentiser</th><th>Segmatiser</th></tr>
<tr><th>Sub-models</th><th>word2word</th><th>sent2sent</th><th colspan="2">sent2sent</th><th>sent2sent</th></tr>
<tr><td rowspan="20"></td><td>base-copying</td><td>char</td><td>char</td><td>maxratio</td><td>char</td></tr>
<tr><td>base0</td><td>bpe</td><td>bpe</td><td>lenpred</td><td>bpe</td></tr>
<tr><td>char5</td><td rowspan="18"></td><td rowspan="18"></td><td rowspan="18"></td><td rowspan="18"></td></tr>
<tr><td>char10</td></tr>
<tr><td>char15</td></tr>
<tr><td>char20</td></tr>
<tr><td>char25</td></tr>
<tr><td>char30</td></tr>
<tr><td>bpe5</td></tr>
<tr><td>bpe10</td></tr>
<tr><td>bpe15</td></tr>
<tr><td>bpe20</td></tr>
<tr><td>bpe25</td></tr>
<tr><td>bpe30</td></tr>
<tr><td>worpheme5</td></tr>
<tr><td>worpheme10</td></tr>
<tr><td>worpheme15</td></tr>
<tr><td>worpheme20</td></tr>
<tr><td>worpheme25</td></tr>
<tr><td>worpheme30</td></tr>
<tr><td rowspan="3">Comparison</td><td colspan="2">the best lemmatiser</td><td colspan="2">the best segmentiser</td><td rowspan="2"></td></tr>
<tr><td colspan="4">the best pipeline segmatiser</td><td>a joint segmatiser</td></tr>
<tr><td colspan="5">the better segmatiser</td></tr>
</table>

Table 3.4: The complete list of our models: Lemmatisers, segmentiser, and segmatisers. The top half list all the models we construct, and the bottom half indicates how we compare between different models and select the best models; models in the same non-black colour are compared with each other.

3.2.1 Lemmatisation models

Can neural models lemmatise segmented Japanese text? We explore different ways of lemmatising segmented Japanese text, regarding 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, the input and output follows Lematus[12]. Special symbols are used to indicate specific information for training (Table 3.5). Unlike **word2word**, in **sent2sent** models, both input and output are full sentences[13], and the output is a sequence of lemmas of the input; full context is encoded.

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

3.2.1.2 Context unit type

In **word2word** lemmatisers, we use these context units: character, byte pair and morpheme. Different unit types are believed to capture different linguistics information thus performing differently in NLP tasks; their performance also varies among languages (Vania and Lopez, 2017).

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

[12]regardless of its name, **word2word** is essentially still a *seq2seq* model encoding and decoding sequences - words represented by character

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

Table 3.6 reports the sizes of the encoder and the decoder lexica, and the lexicon overlap. Models in each context unit group share the same encoder and decoder lexicon. The decoder lexicon for all models is the same, and is a subset of the encoder lexicon. We use separate lexicon for encoders and decoders to save computation.

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

Table 3.6: The encoder and decoder lexicon size and the lexicon overlap of the context models. Notice that the difference between the encoder and decoder lexicon of **char** models comes 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 (且, 迄, 或) 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

The context length ranges from 5 to 30 units, counted from both sides of the to-be-lemmatised centre word. Besides, we have a baseline **base0** which encodes zero context as opposed to context-aware models.

However, for models with bigger window sizes, there is not always enough context to encode. We conjecture that would be a factor of unfair comparison between models. We compute the average length of encoded context for **word2word** models in Table 3.7. For models with smaller window sizes, the context being encoded is close to the specified number; however, with an increasing window size, especially for 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. An example of input

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 3.7: Actual length of context being encoded in **word2word** models. The length of context being encoded is not always proportionately increasing with the specified/expected context length.

and output is:

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

The segmentation performance is evaluated by F1 score of the correctly segmented units over the whole corpus (details in Section 3.4.2).

3.2.2.2 For better segmentation length matching

Besides fine-tuning the segmentisers, we also attempt to put length constraints on output for better performance, if length mismatch turned out to be a critical issue. Our

approaches are: constructing an output length predictor for each input sequence; constraining the maximum output length for each input sequence during decoding.

The length predictor is a *seq2seq* model which takes as input a sequence of characters, and returns the predicted output length. Suppose the input sequence is <w> 月 が 綺 麗 で す </w> 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 is <w> 月 <s> が <s> 綺 麗 <s> で す </w>. So the expected output length is 11 . We use UD_Japanese to learn the length prediction.

The second approach puts length constraints on the output directly. The length constraint is set as the maximum ratio of output length to input length in UD_Japanese training set. We call the ratio max(|output|:|input|) or maxratio as the parameter name in Nematus. Table 3.8 reports the length statistics of the training set when the sequence is split into characters or byte pairs. The average ratios are around 1.5 and the maxratio close to 2. So we set 2 as 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 3.8: |output|:|input|: Length statistics of UD_Japanese training set. The maximum ratio of output length to input length is around 2.

3.2.3 Segmatisation models: segmentation & lemmatisation models

Segmatisation is made up of segmentation and lemmatisation. Similarly, a segmatiser is a model which does segmatisation.

A segmatiser could either be a pipeline or a joint/end-to-end one. If the optimisation of segmentation and lemmatisation is separately done, it is a pipeline model. Unlike a pipeline model, a joint segmatiser is optimised as an integrated system to lemmatise un-segmented text. It is a big black-box which handles both tasks simultaneously; indeed, we cannot inspect how it "segments" the text in order to lemmatise, or know if it "segments" at all.

[14] during training time, the sequences will not be cut off according to the maxratio length constraints

We compare the best pipeline segmatiser with a joint segmatiser. Our best pipeline model is constituted of the best segmentiser and the best lemmatiser, achieving the highest scores. The output of the segmentiser is fed as 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**. Table 3.9 displays examples of 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 3.9: Examples of input and output for the **sent2sent** segmatisers.

3.2.4 Baseline models

We have internal and external models as baselines. For lemmatisation, a context-free model **base0** as the internal baseline, is compared with the context-aware models.

Meanwhile, we have three existing Japanese morphological analysers, **MeCab**[15], **JUMAN**[16] and **JUMAN++**[17], as external baselines. They are compared with our segmentisers and segmatisers, but cannot be used as pure lemmatisers.

Table 3.10 compares the external baselines regarding dictionaries, which provide segmentation and lemmatisation criteria. Different dictionaries usually have divergent segmentation and lemmatisation criteria, and those differences would cause systematic errors.

MeCab and **JUMAN** adopt CRFs and n-gram LM scores respectively. **JUMAN++** interpolates n-gram LM scores with RNNLM scores, to take into account semantic plausibility of word sequences. Consequently, **JUMAN++** works better in some ambiguous cases (Morita et al., 2015).

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

	Available dictionaries
MeCab	IPAdic[18]/Jumandic[19]/Unidic[20]
JUMAN	Jumandic
JUMAN++	Jumandic

Table 3.10: Baseline models: three existing non-neural (or not completely neural) Japanese morphological analysers and their dictionaries.

3.3 Training

3.3.1 Framework

We utilise the Theano implementation of Nematus (Sennrich et al., 2017). There are lots of tunable parameters: encoder/decoder layers, dimensions of word embeddings and hidden layers, batch size, etc. The details of parameter tuning go to the next section.

As a *seq2seq* framework, Nematus has an encoder-decoder architecture. We use GRUs for the encoder and the decoder. Attention mechanism is employed in between to compute the context vector.

The loss function is sentence-level softmaxed cross-entropy. 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 the hyper-parameters for better performance. Table C.1, Table C.2 and Table C.3 in Appendix C list the hyper-parameters for our models.

For most hyper-parameters of **word2word** lemmatisers, we follow the choices of Lematus. The only thing we change is the maximum input length - from 75 to 150, as we want to include more training sentences.

Based on the hyper-parameters for the lemmatisers, we find it helps achieve better performance for the segmentisers, by increasing the dimensions of word embeddings and hidden layers. This matches our expectation of segmentation being more complicated and requiring more parameters to model well. Besides, we make the encoder layers bidirectional to better encode context from both sides. This also increases the performance.

For the segmatisers, 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 better 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-parameters. We expect it might bring unfairness when comparing sub-models. In later experiments, we select the best models based on the performance scores. But when the gap between sub-models is small, we do not intend to draw strong conclusions about the models' superiority or inferiority.

3.4 Evaluation

3.4.1 Accuracy: token-level

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

3.4.2 F1 score: sentence-level

Sentence-level F1 score over the whole data set is used to evaluate other models (**sent2sent** lemmatisers, segmentisers, and segmatisers), where the reference and the actual output might be of different lengths.

F1 score is a benchmark balancing 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 is then computed as:

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

However, F1 score faces the problem of free word order. We modify F1 score to constrain the word order by aligning uni-grams between reference and output. Here is an example of computing constrained F1 score for segmentation (Table 3.11). The second を in the output is a mistake and should not be counted as a correct label for the first を in the input. By constrained F1 score computation, this extra を is

viewed as a wrong output label as it does not align with anything in the reference at the corresponding position.

Input	夢 を 叶 え る た め 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> 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 エンジンバラ 大学 に を 入学 し た
Reference	夢 を 叶える ため エンジンバラ 大学 に 入学 し た
Precision	$10 \div 11 = 0.9091$
Recall	$11 \div 10 = 1.1$
F1	$(2 \times 0.9091 \times 1.1) \div (0.9091 + 1.1) = 0.9955$

Table 3.11: Example of computing constrained F1 score for segmentation. The second を in the output is not counted as a correct label for the を in the reference

Chapter 4

Results and Discussion

4.1 Lemmatisers

4.1.1 word2word lemmatisers

Token-level accuracies are computed on the test set for baselines **base-copying** and **base0**, and all **word2word** lemmatisers. The results are 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, **char** lemmatisers almost always outperform **bpe** and **worpheme** counterparts, except for window size 10. Regardless of their superiority, their performance is close to that of **bpe** lemmatisers. After inspecting the **char** and the **bpe** lexicon, we conjecture it is due to the high overlap between the two lexica, that they achieve similar results. The **char** lexicon is a subset of the **bpe** lexicon after 500 times of merging operation (Table 4.1). In the **bpe** lexicon, in addition to the byte pairs which are more likely to be better learned in the neural space, there are as many infrequent characters as in the **char** lexicon, whose neural representation cannot be well learned in either case.

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

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

Regarding context length, there is no clear relationship between context length and performance. For most languages Lematus experimented with, window size 20 is a

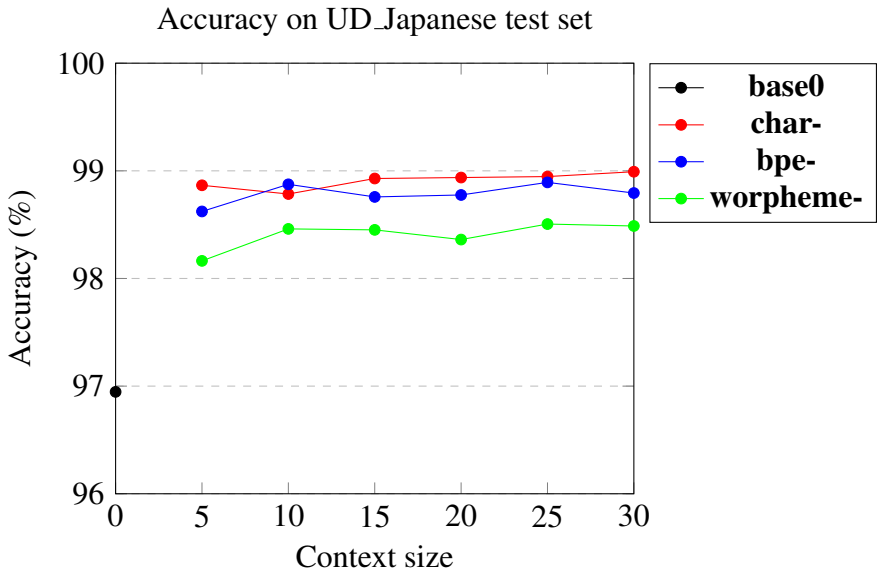

Figure 4.1: The **word2word** lemmatisers: Accuracy on UD_Japanese test set. **char** lemmatisers are almost consistently outperforming the rest; the results of **char** lemmatisers are always very close to those of **bpe** lemmatisers. All context-aware lemmatisers outperform the context-free lemmatiser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

sweet spot (Bergmanis and Goldwater, 2018), as it contains enough context without leading to sequences too long for NNs. However, 20 does not guarantee the best performance in any case for us. The best lemmatisers categorised by context unit are: **char30**, **bpe25**, and **worpheme25**. Japanese lemmatisers seem to learn more from longer context. Nonetheless, as in Table 3.7, the length of the actual context being encoded does not necessarily increase proportionately with the specified number. The unfairness caused by the sentence length distribution might explain the close results of the lemmatisers with bigger context windows.

Compared with the non-neural baseline **base-copying**, all neural lemmatisers outperform. They achieve an increase of at least 12% in accuracy than **base-copying** (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 lemmatisation.

Although the context-aware lemmatisers achieve high accuracies, we are curious about the mistakes they make. They make three main types of mistakes. First, for unseen *characters*, the output is always the least frequent characters, and different models choose different characters. An example is in Table 4.2. In contrast, **base0** is

good at copying and pasting unseen *characters*, especially when the input is only one character.

Input 兎				**Reference** 兎	
char5	僊	**bpe5**	墾	**worpheme5**	嬢
char10	麾	**bpe10**	埠	**worpheme10**	攝
char15	咳	**bpe15**	患	**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 different models lemmatising an unseen *character* by outputting different least frequent characters. Only **base0** outputs the correct answer by copying and pasting the word itself.

Second, while the context-free lemmatiser copies and pastes the words as their lemmas mostly, the context-aware lemmatisers seem to disambiguate between different lemmas for the same word form. Some obvious examples are (possibles lemmas in the brackets): で (で, でる), し (し, する), さ (さ, する), etc. The correct disambiguation of them largely accounts for the increased accuracies of the context-aware lemmatisers. Instead of *blindly* outputting the most frequent lemma, the context-aware lemmatisers seem to rely on something else to make judgments. Is context the answer here? We plot out the attention maps for some correctly and incorrectly lemmatised ambiguous words by models of different context unit types or context lengths (Figure B.1 vs. Figure B.2 in Appendix B.1). The attention maps do not inform any use of context for correct disambiguation. At each decoding step, almost 100% attention is attended to the centre word; negligible attention is distributed to the rest of the sentence.

Third, **base0** is confused by the same word forms with different POS-tags, and the context-aware lemmatisers do not bring obvious improvement. An example is 楽しむ -> 楽しむ/楽しみ where 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 make mistakes in lemmatising POS-ambiguous words.

4.1.2 sent2sent lemmatisers

The two **sent2sent** lemmatisers, **sent2sent-char** and **sent2sent-bpe**, are evaluated by sentence-level F1 score. We prefer F1 than accuracy, as even though segmentation information is provided (by putting space symbols <s> between groups of characters that make up words), there is no guarantee that neural models could produce exactly the same number of words as the input. This concern is supported by our empirical results (Table 4.4 & Table 4.5). In case of length mismatch problems, F1 score ensures 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 UD_Japanese test set and encoder lexicon statistics. Two models achieve close results; the byte pairs are not bringing 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 fact that the **bpe** lexicon includes the **char** lexicon, might contribute to their close F1 scores here as well. The learning of byte pairs does not help.

As a lemmatiser given perfect segmentation, it is supposed to give an output of the same length as the input. However, this is not always true in reality and F1 score is not informative. We evaluate the length mismatch problem of both lemmatisers (Table 4.4 for **char** & Table 4.5 for **bpe**). Among 558 test sentences, only 4 or 5 mismatched in length. We mark the incorrectly lemmatised units in the output red, and the missing output in the reference green. For both lemmatisers, the biggest length mismatch comes from sentence **511**. The average length mismatch is 25.5 (**char**) and 15.4 (**bpe**) words. The **char** lemmatiser always terminate the output halfway; for the **bpe** lemmatiser, three sentence are not finished but the rest two output too much. Interestingly, the output lengths of the last three sentences are all around 70 in **char**, and around 80 in **bpe**. We conjecture this might be due to the inability of NNs to handle long sequences: the information flow becomes increasingly weak with the increasing sequence length. Besides, as the input in **bpe** is shorter than that in **char**, the **bpe** lemmatiser is more likely to output sequences closer to the actual reference, as the information is stronger even after having gone through a long distance.

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.

4.1.3 The best lemmatiser

The lemmatiser with the best performance on the test set[1] is selected as the best lemmatiser. Although, **word2word** and **sent2sent** lemmatisers are evaluated by different metrics, as F1 is a balanced benchmark for accuracy, the comparison is valid.

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

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 too much; the other three terminate halfway, with a length of around 80.

On average, **word2word** (98.70%) lemmatisers outperform **sent2sent** (0.9825) lemmatisers. The inferiority of **sent2sent** lemmatisers could be partly explained by the length mismatch problem, which is never a problem for **word2word** lemmatisers. Although regarding overall scores, **sent2sent** lemmatisers are not preferred more, they do not encode context for each word, resulting in a smaller training set and more efficient training. As long as the length of the test sentences can be controlled, the length mismatch problem could be largely got rid of.

Among all neural lemmatisers, **word2word-char30** achieves the highest accuracy 98.99%, thus selected as the best lemmatiser.

4.2 Segmentisers

4.2.1 sent2sent segmentisers

The **sent2sent** segmentisers are evaluated by F1 score. The baselines are **MeCab**, **JUMAN** and **JUMAN++**. As they all rely on dictionaries to look up entries during segmentation, and Jumandic is the only dictionary available to all of them, we use Jumandic for fairer comparison; meanwhile, we load **MeCab** with its recommended dictionary IPAdic. The F1 scores of **sent2sent** segmentisers **char** and **bpe** and the baselines are in Table 4.6.

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: F1 scores of the baselines and **sent2sent** segmentisers. **MeCab** with IPAdic works the best; with Jumandic, three baselines achieve close results. Two **sent2sent** segmentisers perform similarly.

All baselines perform similarly when using Jumandic. However, **MeCab** with IPAdic achieves a much higher score. This supports our hypothesis in Section 3.2.4 that using different dictionaries might cause systematic errors. The analysis of the in-

consistency is in Appendix A.2. 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 both using Jumandic. Their F1 scores are very close, with **JUMAN++** being only a bit higher. But we do observe **JUMAN++** good at handling limited ambiguous cases (discussion in Appendix A.3).

Compared with the baselines, both **sent2sent** segmentisers achieve higher F1 scores, with **bpe** surpassing **char** by 0.0062. As the byte pairs are learned within words, they are providing correct segmentation to the **bpe** segmentiser, which is not the case for the **char** segmentiser. For example, in the sequence of byte pairs おか　しい　です, the true segmentation is おかしい　です. As we already know that しい, as a byte pair, must be grouped together, the segmentiser does not need to judge whether to split し and い or not; however, the **char** segmentiser faces し and い as two characters to be either grouped or split. Thereby, byte pairs offer prior information about segmentation, which benefits the segmentiser ("cheating").

Besides F1 scores, we check the length mismatch of both segmentisers. Among the 558 test sentences, the **char** segmentiser outputs 272 (48.75%) of them with wrong lengths, and the **bpe** segmentiser makes length mismatch mistake in 251 (44.98%) of them. As their error rates are high, we attempt to constrain output length as proposed in Section 3.2.2.2.

The first model is **sent2sent-lenpred**, an output length predictor. As the error rates are almost as high as 50%, we only adopt a length predictor if its accuracy is above 50%. For the length predictor, the input is a sequence of characters, and we format the reference output in two ways: numbers as digits (e.g. `45` as `4 5`) to generate unseen numbers flexibly, or numbers themselves (e.g. `45` as `45`). As results, neither versions of the predictor could get a decent accuracy on the test set (3.40% & 1.25%). This approach is unsurprisingly failed. 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 **sent2sent-bpe**, by constraining the output length up to twice the input length. From Table 4.7, the F1 score of **sent2sent-bpe-maxratio2** is 0.9270, higher than that of **sent2sent-bpe**, 0.9223. Although the length mismatch error rate of **sent2sent-bpe-maxratio2** is 44.98%, on par with **sent2sent-bpe**, the total amount of mismatched length is decreased by 49 units. To conclude, putting length constraints during decoding does help **sent2sent** segmen-

tisation.

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

Table 4.7: Length mismatch error rate and mismatch unit count for **sent2sent** segmentisers. Adding output length constraint does mitigate the length mismatch problem, regarding the unit count.

4.2.2 The best segmentiser

Comparing the F1 scores of the baselines and our **sent2sent** segmentisers, we select **sent2sent-bpe-maxratio2** as the best segmentiser.

4.3 Segmatisers

4.3.1 Baseline segmatisers

The baselines are **MeCab**, **JUMAN** and **JUMAN++**. As we already know their inconsistent segmentation issues from Section 4.2.1, we infer there must be inconsistency in lemmatisation as well, as they rely on dictionary look-ups to lemmatise the segmented units. If the sequence can be segmented into various ways, the lemmatisation should also vary. All three baselines use the same dictionary Jumandic to segmatise the test set. Table 4.8 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 4.8: Segmatisation F1 scores of baseline segmatisers. **MeCab** with IPAdic works the best; with the same dictionary Jumandic, all baselines achieve similar F1 scores.

From Table 4.8, the inconsistency issue still exists, as expected. When using Jumandic, the baselines perform similarly, with **MeCab** falling behind two Juman family members only by 0.1605 on average. **JUMAN++** outperforms **JUMAN** by 0.063. This superiority should be partly accounted to 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 best pipeline segmatiser is thus obtained by combining **sent2sent-bpe-maxratio2** segmentiser, with **word2word-char30** lemmatiser. We take the segmentation results from **sent2sent-bpe-maxratio2**, and format it as input for **word2word-char30**.

The accuracy of our best pipeline segmatiser is **92.46%**.

We discuss the pros and cons of pipeline Japanese segmatisers based on observation. First, the separate fine-tuning for both tasks is time-consuming. We attempt with twenty-one (**word2word**) + two (**sent2sent**) lemmatisers and two (**sent2sent**) segmentisers experimented with two approaches to improve length matching. Second, as the same data is used for training lemmatisers and segmentisers in parallel, we would certainly prefer simultaneous training if possible, as long as the performance is not worsened. On the other hand, pipeline segmentisers could work even better if a large amount of annotated data for either task was prepared. For example, for the segmentisers, we can make use of segmented text from other corpora.

4.3.3 sent2sent segmatisers

There are two **sent2sent** segmatiser variants, **char** and **bpe**. Their performance is evaluated by F1 score. From Table 4.9, both segmatisers achieve an F1 score as high as around 0.92. The tiny gap between them might be explained by the same hypothesis about the lexicon overlap we have been repeating. Besides, **bpe** segmatiser has the benefits from the byte pairs which provide ready answers for segmentation as discussed in Section 4.2.1, but it lacks the flexibility that **char** has.

We also check the length mismatch problem here (Table 4.10, **char-maxratio2** and **bpe-maxratio2** with length constraints included). The segmatisers get wrong lengths for almost half of the sentences; putting length constraints is only slightly helpful.

Some types of segmentation mistakes by joint segmatisers are as follows. First,

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

Table 4.9: The **sent2sent** segmatisers: F1 score on UD_Japanese test set and encoder lexicon statistics. The two models achieve close scores as their lexica share a lot of entries.

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

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 (the 1st line: the un-segmented text; the 2nd line: the reference segmatised output; the 3rd line: the actual segmatised output; the 4th line: 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 into a long word. And the elements are mostly written in the same character type. For example (the 1st line: un-segmented text; the 2nd line: the reference segmatised output; the 3rd line: the actual segmatised output; the 4th 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
character type	*kanji*	*katakana*	*hiragana*

4.3.4 The better segmatiser

To remind the readers of the best pipeline and joint segmatiser, we compare them regarding F1 score in Table 4.11.

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

Table 4.11: F1 score of the pipeline segmatiser vs. the joint segmatiser. The **joint** model is not worse than the **pipeline** model a lot.

From the 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**, part of the pipeline model, can lemmatise 98.99% of the test tokens correctly. In contrast, when given the segmentation results from **sent2sent-bpe-maxratio2** segmentiser, which has a high length mismatch error rate 44.98% and an F1 score of 0.9270, the performance of the pipeline segmatiser is largely adversely influenced by the performance of the segmentiser.

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

One thing we have noticed from the pipeline segmatiser is, once the segmentiser has made any mistakes, it is impossible for the lemmatiser to correct them. This is understandable because for each word, the lemmatiser must always output a lemma. Unlike the pipeline segmatiser, we expect to see segmentation and lemmatisation facilitating each other in the joint segmatiser: segmentation is improved as lemmatisation

helps decide word boundaries; lemmatisation can be helped by segmentation accurately segmenting units that can be encoded as useful context. However, we witness the failure of even outputting incorrect characters in the joint segmatiser, such as some names containing infrequent *kanji*[2]. Infrequent characters are not even *blindly* copied and pasted, but this is not a problem as serious to the standalone segmentiser as to the joint segmatiser. For *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 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 known characters. But the Japanese character lexicon is unable to cover all possible characters[3], based on a corpus with a very limited coverage of characters, let alone the more infrequent ones.

[2]the joint segmatiser could not handle names such as 石川悦男 which contains infrequent character 悦, and instead alters the string as 石川豹男; the same phenomenon is seen in other types of words as well

[3]besides the finite set of *hiragana* and *katakana*, there is a much larger vocabulary of *kanji*, contributing to the explosion of the character lexicon size

Chapter 5

Conclusion

We apply *seq2seq* models on Japanese lemmatisation, segmentation, and segmatisation. Our answers to the three questions in Section 1.2 are:

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

2. By modelling word segmentation as predicting the probabilities of word breaks in a sequence of characters, we succeed in constructing *seq2seq* Japanese segmentisers. Our segmentiser which uses byte pairs as basic unit and has output length constraints performs the best. 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, pipeline models still suffer from high time cost of separate optimisation; and joint models are only short of the pipeline counterparts by a little bit. There is weak evidence that segmentation and lemmatisation can facilitate each other in joint models, and it is almost impossible for the lemmatiser to correct incorrect segmentation in pipeline models. When there is not enough training data for both tasks or enough time for separate fine-tuning, we prefer joint models as they are more efficient.

5.1 Limitations

5.1.1 Data

The corpus we use is not a popular choice, so it is hard 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 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.

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 (Sennrich et al., 2015). However, BPE is not the panacea to every language. It is inappropriate for Japanese as Japanese has completely different features from languages such as German: short morphemes, large-sized character lexica, tons of *unseen*characters, etc. 500 times of merging operation is still too many for Japanese. The results of BPE models are 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 similar languages such as Chinese.

5.2 Future work

To test the robustness of our models, we would like to repeat the experiments on at least one of the mainstream corpora, The Balanced Corpus of Contemporary Written Japanese (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). 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, but with a less complicated writing system.

The most interesting and challenging task we find is Japanese segmentation. Based on the error analysis of our segmentisers, we propose to include the character type

[1] but this is the only free corpus we could use when starting this dissertation

information, 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. 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

Extra examples

A.1 An example of Japanese sentence

In normal Japanese writing, the three character types (sometimes *romaji*, equal to Roman character, and Arabic numbers are also witnessed) can appear in the same sentence. Table A.1 shows an example of a Japanese sentence randomly selected from the Internet[1], with annotation of different character types.

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 A.1: An example of Japanese sentence containing *kanji*, *hiragana*, *katakana*, and *alphanum* (with annotation and translation in English).

[1] **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*

A.2 Inconsistency of IPAdic and Jumandic in Section 4.2.1

There are four main types of inconsistency in segmentation and the resultant inconsistency in lemmatisation between IPAdic and Jumandic (examples in Table A.2). 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.

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 A.2: Examples of inconsistency in segmentation and lemmatisation between IPAdic and Jumandic (seg: segmentation; lem: lemmatisation).

[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

A.3 The superiority of JUMAN++ over JUMAN

JUMAN++ is especially better at handling the following types of words: short nouns, names, and words written in the same character type.

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 A.3.

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) つくら　れ　ない　か　つくら　れた JUMAN++: (cannot be made or be made)

Table A.3: **JUMAN** vs. **JUMAN++**: Segmentation of ambiguous cases.

[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*

Appendix B

Plots

B.1 Attention maps for Section 4.1.1

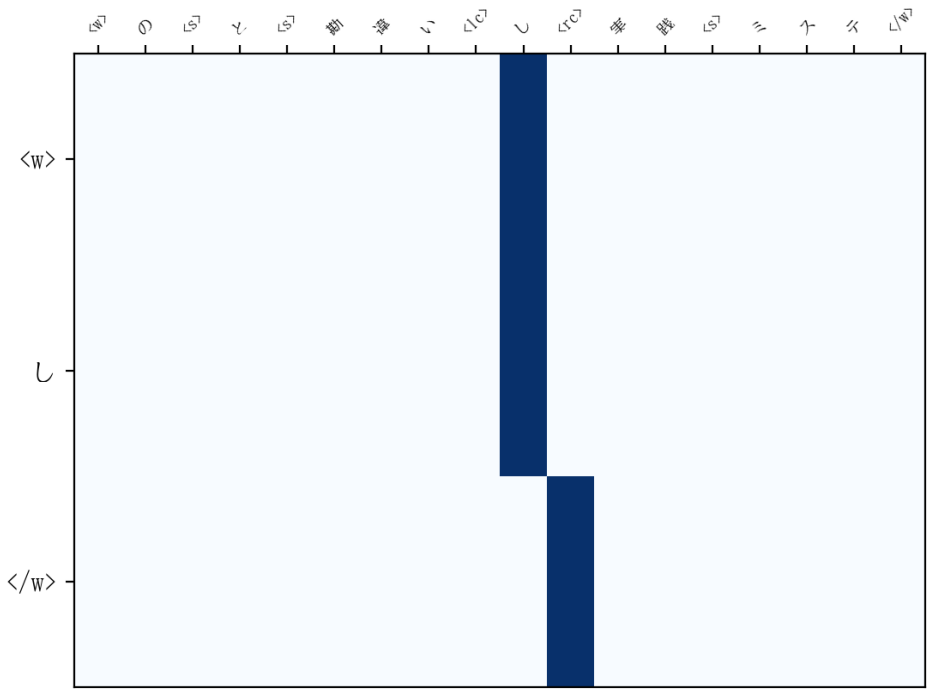

Figure B.1: Attention map for an incorrectly lemmtised example し -> する by **char5**. Most attention of the decoder side is given to the centre word of the encoder side. Attention given to the rest of the sentence is negligible.

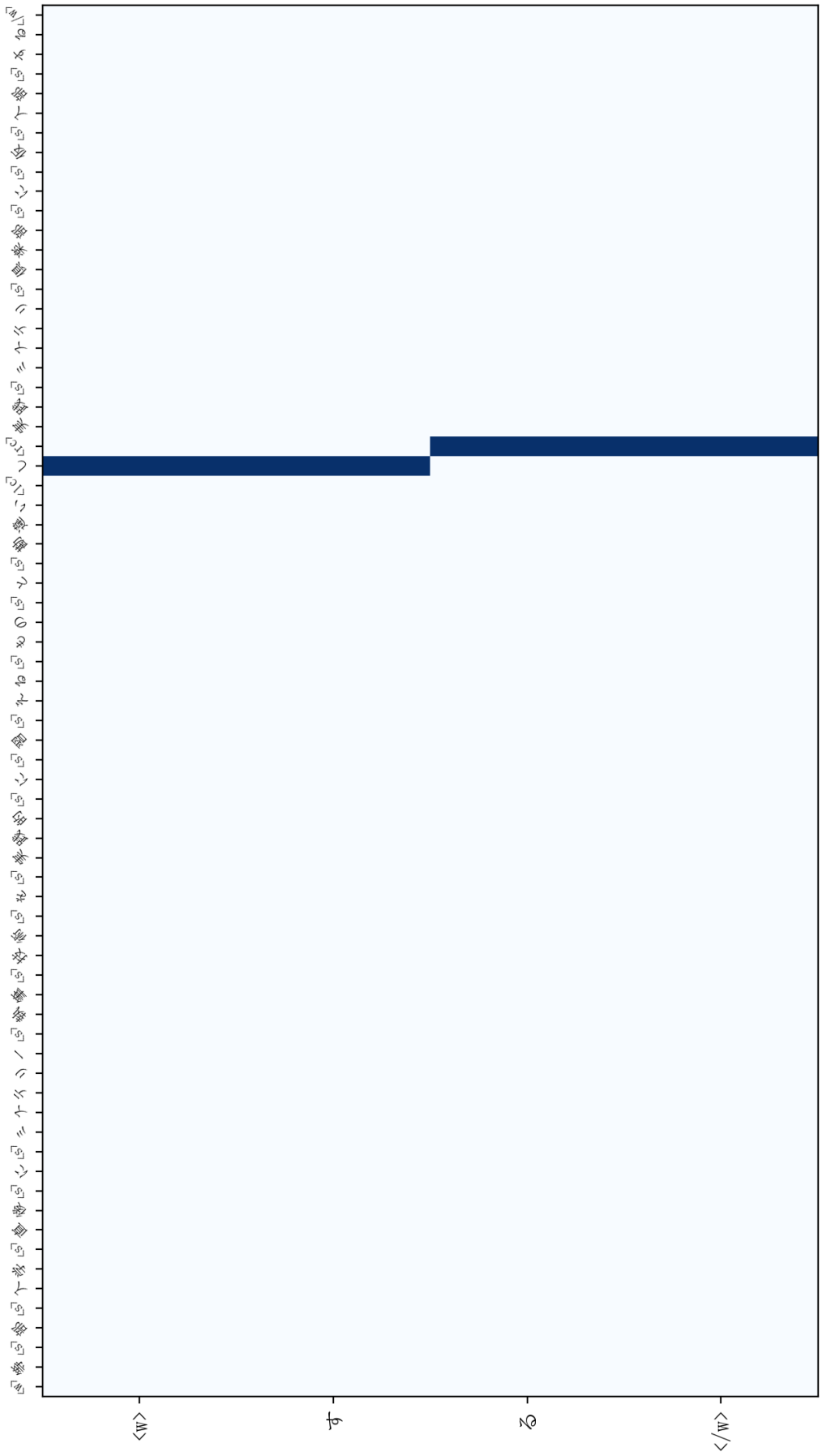

Figure B.2: Attention map for a correctly lemmtised example し -> する by **char30**. Most attention of the decoder side is given to the centre word of the encoder side. Attention given to the rest of the sentence is negligible, even though the model outputs the correct word.

Appendix C

Hyper-parameter tuning

C.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 C.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 C.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 C.3: Hyperparameters for the joint segmatisers