Bachelor thesis

literal Japanese to **English** translation

Nikolaj K. Bjerregaard (nbje@itu.dk)
Supervisor: Rob van der Goot (robv@itu.dk)
May 2020

IT UNIVERSITY OF COPENHAGEN

In this report, "I/me/my" refers to the author of the report and developer of the *Literal Japanese* system: Nikolaj K. Bjerregaard.

1 Abstract

Japanese sentence construction is very different compared to English. This difference is not captured in normal translations, since the words are reordered to fit the target language. For Japanese learners, this is an issue since they will not learn how Japanese sentences are constructed and deducing which words correspond to each other may be difficult. This project defines a new style of translation whose main paradigm is translation on a word-by-word basis, which preserves the Japanese word-order. During the course of this project, I have developed a system called "Literal Japanese" that seeks to automatically make such translations.

The translation process consists of two main parts: tokenization of the input sentence and translation of these tokens. For the tokenization, I compare different tokenizers and describe how I adapted these to fit the requirements of the system. These adaptations are merging endings of words and merging multiple tokens into a single token, if the combination is a single word or expression in English. For the translation, the main method is looking up words in a dictionary. Here, I use information such as part-of-speech tags to improve the selection of translations. In addition to this, I have developed different approaches for some types of tokens, such as word-endings, names, numbers, etc.

For testing the system, I use a set of sentences translated manually according to the translation style specification and compare these to the systems output. Some translations done by the system are so close in meaning to the ideal translation, while not being identical, that they could be considered correct. I have implemented methods such as lookups in a paraphrase database and calculation of word-vector distance, that enable the system to detect words and expressions with synonymous meanings. All aspects of the system have been thoroughly evaluated. The tokenization has an F1 score of 0.91 and the translation has a token error rate of approximately 0.3. Tests using different settings and optimizations are presented in the evaluation section of this report.

The system is available at: https://github.itu.dk/nbje/LiteralJapaneseTranslation

2 Terms

Term	Meaning
POS	Part Of Speech. Refers to the categorization of words. e.g. noun, verb or adjective.
Natural translation	A 'normal' translation, a translation that changes the grammar to fit the target language. As opposed to the translation style described in this report.
Script	Refers to a Python (.py) file. The implementation of *Literal Japanese* is entirely written in Python.
Token	Part of a sentence, usually a single word.
Repository	The GitHub repository for the *Literal Japanese* implementation.
Gold data	The ideal output of the program. Used for testing the performance of the system.
F1 score (Rijsbergen 1979)	A metric used in statistical analysis that combines precision: $\frac{true\ Positives}{true\ positives+false\ positives}$ and recall: $\frac{true\ positives}{true\ positives+false\ negatives}$ using the formula: $2 \cdot \frac{precision \cdot recall}{precision+recall}$
Kana	Refers to hiragana and katakana, two alphabets used in Japanese writing.

3 Problem definition

Japanese is regarded as being very difficult to learn for English speakers. The US Foreign Service Institute has compiled a list of the expected time used to attain proficiency in different languages. Here, Japanese is listed in the most difficult category and is even marked as being more difficult than the other languages in this category. (Foreign Service Institute 2007)

The main reason for this is that the grammar and writing of Japanese is very different from English. One way that this is apparent is the ordering of words in a sentence. Consider the following sentence:

ここから旭岳までのバスはありますか

And its English equivalent:

Is there a bus from here to Asahidake?

If we translate the Japanese words one by one, it looks like this:

ここ	から	旭岳	まで	の	バス	は	あります	か
here	from	Asahidake	to	<possessive>	Bus	<topic>	exist	?

(grammatical elements for which there is no English equivalent are marked <as so>)

Compare this to doing the same thing with Danish instead of Japanese:

er	der	en	bus	fra	her	til	Asahidake	?
is	there	a	bus	from	here	to	Asahidake	?

English word ordering, and more generally sentence construction, is almost identical to Danish. This means that learning Danish is mainly a task of accumulating vocabulary, whereas learning Japanese includes a step of learning to construct sentences. Machine translation systems today focus on 'natural' translations, that is, translations that reorder and even remove or add words to make the sentence fit the target language. This type of translation is good at conveying the meaning of the sentence. However, language learners lose the aspect of Japanese sentence construction. At best, it means looking up every word of the Japanese sentence in addition to seeing the translation. At worst, the learner may mistakenly think that some Japanese word corresponds to an English word, even though they are not related.

To illustrate this, look at the following screenshots from the TV-series "Giri/Haji". The subtitles do not match what the characters are saying. The actual translations are given in the captions.

Figure 1 - Japanese audio: "Until now has never been seen"

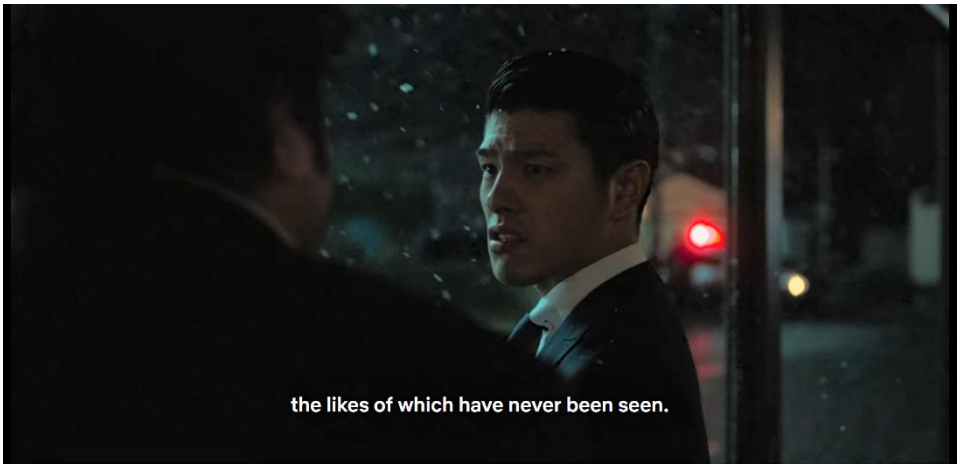

Figure 2 - Japanese audio: "wave of havoc and destruction unleash intention I have."
Source: Giri/Haji, production: Sister Pictures, distributers: BBC/Netflix

The mismatch of voice and subtitles is due to the word-order. Mainly, the two parts of the sentence are reversed. Clearly the subtitles form a more coherent English sentence, but they don't accurately reflect the Japanese.

4 A short introduction to the Japanese language

A basic knowledge of the Japanese language will help better understand certain concepts in this report.

4.1 Alphabets

Written Japanese has three alphabets.

4.1.1 Kanji

This alphabet consists of characters imported from Chinese in ancient times. It is made up of ideographic characters. This means that the characters represent a meaning and not a sound.

Examples: 猫(cat) 犬(dog) 駅(train station)

There are thousands of these characters, but only a little over 2000 are commonly used.

4.1.2 Hiragana

This alphabet consists of phonetic characters, that is, characters that represent sounds. This alphabet works similarly to the English alphabet, with one difference: Each hiragana represents a syllable.

Examples: ま(ma) き(ki) ぎ(gi) う(u)

4.1.3 Katakana

This alphabet is identical to hiragana, the characters are just written differently. Katakana is mainly used for writing foreign words.

Examples: バナナ(banana = banana) アイスクリーム(aisukuriimu = ice-cream)

4.1.4 Writing style

When writing Japanese, a combination of all three alphabets is used. Kanji is commonly used for writing word roots and nouns, while hiragana is used for grammatical elements. For example, it is very common for a verb to be written as a kanji, followed by hiragana representing its inflections: 食べたくなかった. This is the word 'to eat', root: 食べ, in its "want-to-do"(たく), negative(なかっ) and past(た) forms. It corresponds to the English: "I didn't want to eat". Having a lot of the meaning in a sentence represented as inflictions of a verb is a common phenomenon in Japanese.

Japanese is written with no spaces between words or sentences, which makes tokenization a non-trivial task. (see section 6.4)

4.2 Grammar

4.2.1 Inflections (conjunctions, endings)

In Japanese, both verbs and adjectives can be inflected. This means that words such yellow, beautiful and long have past and negative forms just like verbs.

Inflections change based on the level of politeness of the sentence. For example, the casual present form of 'to eat' is 食べる, while the polite present form is 食べます.

In this report, everything after the root of a word is referred to as the ending of the word.

4.2.2 Particles

The concept of particles is an important part of the language. These are words that are used for grammatical purposes. An example is the 'topic marker' は, which is used to indicate the 'topic' of a sentence, 'X は …' has a meaning that is close to 'as for X, ...'.

A more specific type of particles are the sentence final particles. These are particles that, when put at the end of a sentence, change the meaning or mood of the sentence. Examples are か that turns a sentence into a question and ね that seeks agreement with the listener.

For more information, wasabi-jpn.com has good guides that explain the different particles.[1]

[1] https://www.wasabi-jpn.com/category/japanese-grammar/japanese-particles/

5 Definition of task

The input to the program is a sentence in Japanese. The output of the program should be a list of English translations for each token in the input sentence. The translations should be made such that one could patch them together and read them as a sentence. This means that there should be no long explanations about the meaning words in the translation. Some words in Japanese simply do not have an equivalent in English. An example of this is the は particle mentioned earlier. For such particles the translation will be the pronunciation of the particle enclosed in diamond braces: <wa>. Another element to consider is the translation of word endings. The translation of these should be the root of the word with the endings added to the word using hyphens. E.g. "**sleep**-polite-past".

The following box shows an example of such a translation. The first line is the input to the program, a Japanese sentence. The second line is a natural English translation of the sentence. The lines not prefixed by # is the desired output, a tokenized sentence with English translations.

```
#jp 私はそのスピーチに深い感銘を受けた。
#en I was greatly impressed by the speech.
私 I
は <wa>
その that
スピーチ speech
に <ni>
深い greatly
感銘 impressed
を <o>
受け feel
た -past
。 .
```

Figure 3 - Example of translation

For more examples, see the sentences_dev.txt and sentences_test.txt documents.[2] For a more thorough description of the translation style, see the translation style specification in the appendix.

[2] Located in the repository's data folder

6 Implementation

The first two sections discuss some approaches that turned out not to be applicable for this task. The remaining sections describe the current system.

6.1 Choice of methodology

Today, the field of machine translation is dominated by machine learning paradigms, more specifically deep learning (Barrault, et al. 2019). Deep learning replaced the classical approach of rule-based machine translation. Rule based approaches failed as natural languages proved too complex to be represented as a set of rules (Marta and Shanahan 2019). This project mainly focuses on rule-based approaches. The main reason for this is that in order to train a deep learning model, huge amounts of data are needed. Since this project defines a new translation style, there are no datasets that fit the requirements. There are many Japanese-English datasets of translations in the natural style, but none using a word for word translation approach. That being said, this only implies that it is not possible to pose the entire task as a machine learning problem. For sub-tasks machine learning can be used to solve specific problems.

6.2 On alignment

Upon encountering the problem of translating using the style proposed in this project, one might come up with the following solution: use existing machine translation to translate the sentence, then, use an alignment algorithm to restore the word order to Japanese ordering. Following this idea, I created a model for aligning natural translations. To align sentences, I used the fast_align word aligner (Dyer, Chahuneau and Noah 2013). As training data for the aligner, I tried two English-Japanese parallel corpora. One was the JParaCrawl corpus (Morishita, Suzuki and Nagata 2019), the other was the Kyoto Free Translation Task (Neubig 2011). To test the model, I used sentences from the Literal Japanese test data.[3] Looking at the results, it was clear that the alignments were quite faulty. Including alignments such as '誰か (someone)' = 'reexamine', '殺し(kill)' = ',' or '他(other)' = 'maybe'. The full list of alignments is available in the repository.[4] To quantify these results, I tested the alignments against the gold translations. The results can be seen in evaluation section 7.6.

It is a known problem that alignment of Japanese and English is problematic, as described by (Ding, Utiyama and Sumita 2015). In short, the two languages are simply too different for alignment to work well. For many words, there is no direct counterpart in the other language and the different word orderings serve to further obscure the results.

To illustrate these problems, see the manual alignment below.

[3] See section 7.1

[4] Under data/FastAlignResults

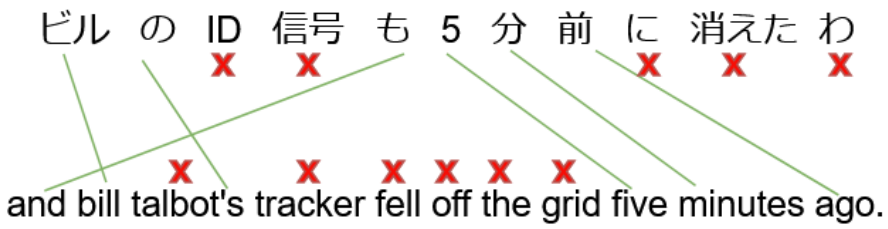

Figure 4 - Alignment of a Japanese sentence

In figure 5, the red Xs indicate that a word does not exist in the corresponding sentence. It's no wonder that the alignment algorithm performs poorly when you consider that almost half the words have no equivalent.

Trying to align the entire sentence does not work, but could alignment be used to find the definitions for a subset of the words? That is, is it possible to take the alignments that have a high likelihood of being correct and incorporate these into a translation? When fast_align picks the alignments, it will loop through all the possible alignments for each word, then pick the one with the highest score. The score represents how confident the system is that a given alignment is correct. fast_align provides an option for printing the score of the entire sentence, but does not provide a mechanism for getting the scores of individual alignments. To get these, I modified the original fast_align source code to also write the scores for each alignment. Unfortunately, this did not pan out either, high scores did not seem to favor correct alignments to a useable degree. Even when setting a cutoff on the scores that threw away most of the alignments, there were still about as many incorrect alignments as there were correct ones. In the printout of the alignments, the scores for each pair can also be seen. In this case, the alignment and scores are based on the forward alignment.

With this, I abandoned the idea of using alignment. Some improvements might be possible for this approach. However, this would still not bring it anywhere near the accuracy of other approaches, namely the one implemented as the final version of *Literal Japanese*.

6.3 Overview of final system

On the next page is an overview of the dataflow in the *Literal Japanese* system. The squares represent python files, the thick arrows represent dataflow and the thin arrows show dependencies.

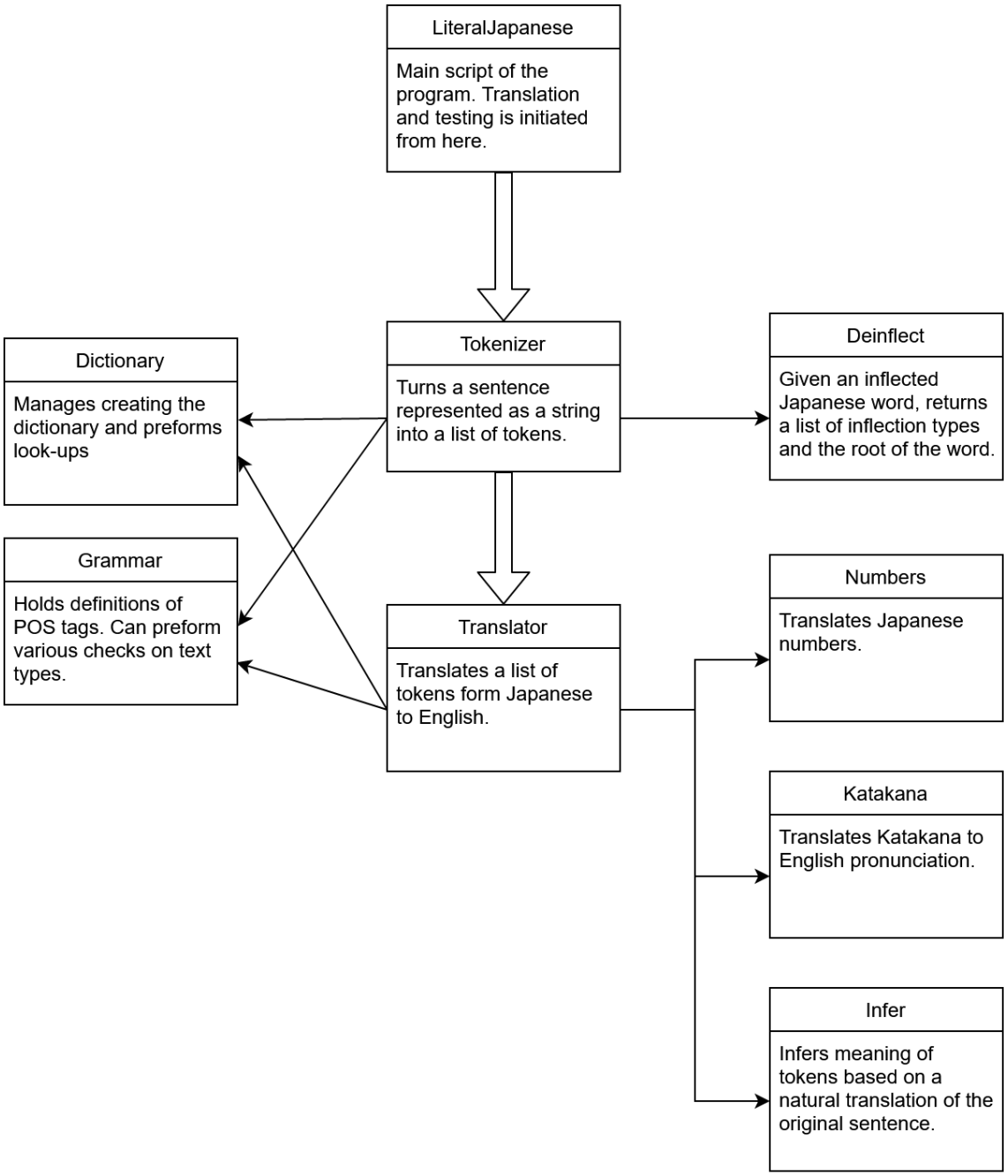

Figure 5 - Overview of dataflow in Literal Japanese

The program is initiated from LiteralJapanese.py, this script calls Tokenizer and Translator to translate the input text. The Tokenizer and Translator files will use other scripts to accomplish specific tasks. Two important files are Grammar, which contains the pos tags used in the system, and functions for detecting different kinds of text. The other is Dictionary which creates a representation of the Japanese-English dictionary used throughout the system.

The repository contains a file called Tests, which is responsible for evaluating the system. The folder called scripts contains code that is not used in the main system, but for testing and to manipulate data in various ways.

For instructions on how to use the system, see the readme located at the root of the repository.

6.4 Tokenization

Tokenization is a bigger problem in Japanese than in languages like English, since Japanese does not use spaces to sperate words. Whereas tokenizers for English usually have an F1 score of over 99, Japanese tokenization only reaches 94 and for some datasets the score is as low as 75. (Universal Dependencies 2018)

6.4.1 Choice of tokenizer

Given that tokenization is different for Japanese than for western languages, it makes sense to use a tool specialized for Japanese tokenization. I selected two libraries for further evaluation; Sudachi (python version)[5] and Nagisa.[6] Table 1 gives an overview of the differences.

	Sudachi	Nagisa
pos tagging	yes	yes
model training	no	yes
custom dictionary	yes	no
normalization	yes	no
word roots	yes	no

Table 1 - Sudachi and Nagisa differences

I chose Sudachi as the primary tokenizer. This was mainly due to the feature that enables getting the roots of words from the tokens, as this made the translation step much easier. I also made an implementation using Nagisa, which is described in section 6.4.4.

6.4.2 Tokenization process

The first step of the process is tokenizing using the tokenizers described above. After this, some additional post-processing is applied to the raw tokens.

6.4.2.1 Merging of word-endings

The first pre-processing is to merge word-endings. With any tokenizer and in datasets containing tokenized Japanese, the endings of words are split into multiple tokens. This is in accordance with formal Japanese definitions of how endings are constructed. The desired output is a list of inflections[7] that modify the word before the ending. But the splits that are commonly used are not necessarily based on the inflections. For example, in other tokenizers, the ending ている is split as て|いる. ている means that

[5] https://github.com/WorksApplications/SudachiPy

[6] https://github.com/taishi-i/nagisa

[7] By inflections, I mean morphemes that indicate things like 'past', 'polite', 'ongoing' or 'negative'.

an action is ongoing, this is split into て and いる since it is technically the conjunctive 'te-form' (て), followed by the auxiliary verb いる(to be). This means that when parsing て, we would have to look at the next token in order to infer what inflection is used. Generally, when you want to add a new inflection to a word, you have to change the original ending to accommodate the new addition. For example, if you wanted to make ている into a polite ending, you would change it to ています, removing the る in the process. This means that it is somewhat unnatural to split up an ending since the different parts are interdependent. The easiest way to handle this situation is to merge all tokens that are part of the ending and translate it as a whole. This is also the way word-endings are taught in most textbooks and other resources on Japanese grammar.
An example of this process could be:

食べ|て|い|ない -> **食べ**|ていない = "**eat**-ongoing-negative" (progressive negative form)

In order to archive this, I use a process of de-inflection. This is based on the algorithm used in Rikaichamp, which is a browser addon that shows the definitions of Japanese words. For inflected words, it also shows what inflections modify the word.[8] I translated the script called deinflect.ts to Python. [9] This script takes an inflected word as input and outputs a list of possible roots and the inflections that modify them. There is nothing in this script that ensures that the returned words are valid Japanese words, they are simply all the possible roots that could lead to an inflection like the one from the input. To overcome this, I implemented a function that filters away words that are not in the dictionary.

The de-infliction algorithm is used as follows for merging tokens. The system will loop through the tokens. When it finds a token that could possibly be inflected, it looks at the next token and checks if the two tokens together are a valid inflection of the word. This continues until a token that cannot be part of an ending is reached, then, the last match is split into a root and an ending. This archives three goals: One, it merges the endings. Two, it finds the root of inflected word. This has the benefit of correcting any errors in the tokenization done by Sudachi. For example, the following is a single token according to Sudachi: 行け. This is the word 'to go (somewhere)' in its imperative form. This should be translated as '**go**-imperative', which means it should be split into two tokens. This is done during the merging phase. Lastly, de-inflecting finds the inflections that modify the word, which saves a step in the translation process.

6.4.2.2 Dictionary based merging of tokens

The second post-processing is merging words based on the Japanese-English dictionary. The reasoning for this step is that sometimes, multiple words in Japanese may correspond to a single word in English. E.g. the word 'anyone' is written as 誰も in Japanese. Consisting of the words 誰(who) and も(also). 誰も will not show up as a single word in a Japanese dictionary, and will be split into two tokens in datasets such as Universal Dependencies.[10] The difficulty of this step lies in deciding when words should be merged, merging every time a combination exists in the dictionary will result in many incorrect merges.

[8] https://addons.mozilla.org/en-US/firefox/addon/rikaichamp/

[9] https://github.com/birtles/rikaichamp/blob/master/src/deinflect.ts

[10] https://universaldependencies.org/

Deciding when merges should happen can be fine-tuned using the pos tags of the merged word or the type of tokens being merged. While the previous step is required for the system to work, this step is optional. The difference in performance with dictionary-based merging turned on and off, can be seen in evaluation section 7.4.2.

6.4.3 Alternative merging of tokens

Before switching to the de-inflection algorithm, I implemented a simpler algorithm for merging endings, this method works as follows. During tokenization, whenever a possibly inflected word is reached, the algorithm will consider the following tokens and merge them if they are endings. Whether a token is an ending can be deduced by its pos tag, which will usually be auxiliary verb, in addition to some special cases. In the translation step, a list of all strings that contribute inflections in endings is used. The system will then select the longest matching inflection from this list and add it's meaning to the translation. For example, the character た contributes the meaning of past tense, unless the following character is ら, in which case the たら ending is matched, which means 'if' or 'when'. This continues until all of the ending has been parsed.

The reason I chose de-inflecting over this simpler method was that de-inflecting is more robust, as it does not rely on the tokens produced by external libraries being correct or to fit the tokenization needed in this project. Additionally, the de-inflection approach is likely to be more general, as it does not rely on my (limited) knowledge of Japanese inflections or on having all inflections covered in the test data.

6.4.4 Differences in Nagisa implementation

Nagisa allows for the tokenizer to be trained using a tokenized and pos-tagged dataset. For the implementation in this project, the Universal Dependencies Japanese GSD dataset was used (Asahara, et al. 2018). This was chosen since it was the largest free dataset in Japanese from Universal Dependencies. It also contained xpos tags[11] which were richer and more specific for Japanese than the universal pos tags used in other data sets.

Nagisa does not provide the roots of words. This means that only the de-inflection approach works for this implementation as it provides the roots.[12]

[11] xpos: Language specific part-of-speech tags. Unfortunately, there was no definition of these tags anywhere. The tags were not very descriptive, having names such as XS, PS or VV, so it was not possible to infer their meaning based on the names alone. I wrote to one of the members of the team responsible for the corpus. He confirmed that the tags were indeed undocumented. He also explained that they would be replaced with descriptive Japanese tags in an upcoming update. For now, I created my own mapping of the current tags by looking at how they were used in the corpus, but once the new tags are released it would be better to use those.

[12] I developed the de-inflection approach specifically for the Nagisa implementation. But it worked so well that I decided to use it in the Sudachi version as well.

6.5 Translation

The following sections present and discuss the different techniques used during translation.

The overall approach works as follows: The translator will try one translation type, if this does not give a translation it will try the next type and so on. This means that the order matters, e.g. special cases such as particles should be translated before trying a lookup in the dictionary, because otherwise a definition would be found in the dictionary and the special translation of particles would never be used. The precedence is as follows:
endings > special cases > inferred meanings > dictionary lookup > non-Japanese names > English text > numbers

6.5.1 Word endings

With the current de-inflection approach, the endings are already available and only need to be converted into text.

6.5.2 Special cases

For some tokens, the system dictates the translation directly. These special cases belong to three categories.

Particles

Japanese particles often do not have an English counterpart. They are used for grammatical purposes, but do not provide meaning in themselves. In a natural translation, these would simply be dropped. But since the goal of this system is to show Japanese grammar, they are kept. Their translation will in most cases be the pronunciation of the particle enclosed in diamond braces.

Symbols

Symbols do not exist in the dictionary. Some symbols are kept the way they are, while others are translated, such as the Japanese quotes 「 and 」.

Name suffixes

In Japanese, names are often followed by a suffix. This suffix indicates the speaker's relation to the named person. As there is no English equivalent for most of these, they are translated as the pronunciation of the suffix. This practice is also common in natural translations of Japanese where the audience is thought to be familiar with Japanese culture.

6.5.3 Meanings inferred from machine translation

While the main translation mechanism is using dictionary lookup, this is sub-optimal. The translation will always just be the first definition of the word. The inference system seeks to improve this and works as follows: do a natural machine translation of the sentence. Then, for each token that needs to be translated, look in the natural translation for words that match definitions of the word currently being translated. If such a word exists, use that as the translation. The advantage of this is that it builds on statistical machine translations' ability to find the correct definitions of words. Initially, I had imagined that alignment could be used here, alignments with high certainty could be used as translations. However, when the alignment approach fell through (see section 6.2), other techniques were needed.

One issue is unavoidable for any implementation of an inference/alignment algorithm. As discussed in section 6.2, Many Japanese words are either dropped or replaced with different expressions in a natural English translation.

The current implementation is a naïve approach: For every token, find all definitions, if one of these definitions exist in the natural translation, pick that as the translation for the token. This method creates another problem. The system can't tell if a word found in the natural translation is a valid candidate. For example, the word 上げる has 27 definitions in the dictionary, if any of these exists anywhere in the sentence, it will match. This means that the system will often be translating a token in one part of the sentence, only to find a matching definition in an entirely different part of the sentence. This is especially a problem for particles, as these correspond to many common English words. E.g. に can mean at, to, for, by, from, per, and, etc. In the sentence "I was **at** the office **and** was sitting **by** the window", there are three candidates for the translation of に. What is needed is some way to tell that 'at' is connected to 'office'. Using this information, you could infer that the translation for に is 'at' if に is also connected to office(事務所) in the Japanese sentence. This could be done by using a tagger trained to produce dependency trees.[13] However, this approach builds on some big assumptions. To begin with, it uses circular reasoning, since it would rely on a translation (e.g. of the word 'office') in order to make a translation of another word. Also, it assumes that 1. The machine translation is correct, 2. The dependency tagging of the Japanese sentence is correct, 3. The tagging of the English sentence is correct, 4. The two relation-taggings are directly comparable. With this many layers of possible errors, I decided not to pursue this approach any further.

I did implement a slightly simpler approach that considered the distance to nouns in order to find the location of a word in the sentence. The reason this works is that nouns are typically much more 'stable' and easier to translate compared to particles. This proved to work well for some particles, but I did not find it general enough to use it in the final implementation.

In spite of these problems, the naïve inference system actually improves the translation slightly. I tested this using different natural translations. One surprising result is that a machine translation produces better results from the inferences than a human translation! More on this in evaluation section 7.5.5.

6.5.4 Dictionary

The dictionary used is the JMdict (Breen 2004).[14] A few measures are taken for the program to select the correct translation.

First, the system will attempt to match the pos of the word. This means that if the token is identified as being a verb, all matches that are not verbs will be filtered away.

Secondly, the system will match the writing of the word being kana (hiragana or katakana) or kanji. In Japanese, all words could potentially be written in kana, which would be writing the pronunciation of the word. However, in practice there are some words that are usually written in kana and some that are not.

[13] For example: https://github.com/Hyperparticle/udify

[14] http://www.edrdg.org/wiki/index.php/JMdict-EDICT_Dictionary_Project#INTRODUCTION

For example, the word りんご(apple) is almost always written in hiragana even though it has a writing in kanji (林檎). Other words are almost never seen written in kana, e.g. 日本 (Japan). This means that matching words written in kana with words that are usually written in kana is important, since this will avoid matching some obscure writing of a word.

Lastly, the remaining translations are sorted by their frequency of use and the most common word is picked. This is based on a frequency ranking used in the dictionary. However, this ranking seems to be somewhat incorrect. For example, the word 生る(naru / to bear fruit) is ranked higher the extremely common word 成る(naru / to become). Looking at lists of Japanese word frequency based on datamining might reveal the reason why. Both words are usually written なる(naru). This is indeed one of the most common words in Japanese. However, it is not possible to know from a corpus of text which meaning of the word is implied (成る or 生る). One attempt at solving this could be to consider the cases where the word is written in kanji. However, here we run into problems. 生る turns out to be more common than 成る. This does not mean that 生る is the more common meaning. The reason is simply that 成る is almost always written as なる.[15] [16]

6.5.4.1 Picking the right definition

The problem of picking the right definition for a word is closely related to the task of word sense disambiguation. This task deals with finding the right meaning for words written the same way. This is known to be a very difficult problem, so much as to be considered an AI-complete[17] task e.g. by (Navigli 2009). When humans consider a sentence, they rely heavily on the knowledge of the context of the sentence. This knowledge is not available when entering a sentence into a program. Nevertheless, the sentence itself might reveal which definition is correct, but this still requires advanced algorithms from the fields of AI and NLP, see (Raganato, Camacho-Collados and Navigli 2017) and (Yuan, et al. 2016). There is one big problem with any algorithm that picks definitions: Simply picking the most common definition of a word is such a powerful heuristic that changing it will often result in a deterioration in performance. The following two sections describe my attempts and findings in improving the selection of definitions.

Currently, the system will select the first sense of the first word, from the list of matches in the dictionary. This first sense is usually the most common sense for a word with this writing. As stated above, this is a good heuristic, but not always correct. One possible solution to this could be to use information about the English language to select definitions. The approach was implemented as follows: For every lookup in the dictionary, save the list of all matches. At the end of the translation procedure, you will have a list containing lists of different senses, one list of senses for each word. Now it is possible to apply an algorithm that looks at the permutations of these lists and finds the best combination of word-senses. In practice, there is one problem for every implementation of this approach. The amount of possible permutations is huge, for some sentences there are 80 million possible combinations. This means that

[15] https://github.com/hingston/japanese/blob/master/44492-japanese-words-latin-lines-removed.txt

[16] http://gfahl.github.io/japanese-word-frequency/word_frequency_wikipedia.html

[17] AI-complete: as difficult as solving the central problem of making AI that thinks like humans.

testing each permutation is not possible. To solve this, I used a simple local search algorithm. The algorithm starts at the permutation that corresponds to taking the first sense of every word. Then it will do hill-climbing until no neighbors have a higher score. The highest scoring permutation should be at a local maximum close to the "select first sense" permutation since this is a good heuristic. For evaluating the score of a permutation, I tried two methods.

Co-occurrence

A co-occurrence matrix is a matrix where the rows and columns correspond to words and the intersections specify how many times those two words appear together. For the purpose of this system, we're interested in how often words appear in sentences together. I created such a matrix from 2 million lines of text taken from Wikipedia, which was the biggest matrix that would fit in memory on my machine.[18] The algorithm calculates the score by considering the co-occurrence between every word in the sentence and dividing this by how frequent the word is in the data-set. The formula is shown in formula 1.

$$\sum_{w1 \in words} \sum_{w2 \in words} \frac{co\_occurrence(w1, w2)}{frequency(w1)}$$

Formula 1 - Co-occurrence permutation score

Word vector similarity

Word embeddings are models that represent words as vectors. The similarity of two words can be found by calculating the distance between their vectors. For more information on this see (Mikolov, et al. 2013) which is the basis for Googles implementation word2vec.[19] The model used in this system is the Google News Vectors – Negative 300. Which is a pre-trained model containing 3 million words and phrases trained on the Google News dataset. The library gensim is used for operations on this model.[20] This approach is similar to co-occurrence except that the vector similarity is used to approximate how related two words are.

Conclusion

Both methods ended up increasing the amount of errors slightly, which meant that they were not included in the final implementation. The results can be seen in evaluation section 7.5.6.

6.5.4.2 Parsing the dictionary – less is more

The dictionary contains a lot of information about each entry, this information could be used for improving selection of definitions. Indeed, that is what I have done by parsing the pos tags in the dictionary. But there is more information than that. One such thing is that the dictionary will sometimes note that a certain sense of the word is bound to one writing of the word, or that some pos only applies for some senses of the word. I added this information to the parser so that these restrictions were upheld.

---

[18] This was done on a PC with 16gb ram. The matrix was represented as a sparse matrix consisting of a dictionary of dictionaries. I compared this representation to sparse matrix implementations is the SciPy library and found that it used less memory than any of these.

[19] https://code.google.com/archive/p/word2vec/

[20] https://radimrehurek.com/gensim/

However, the results were quite mixed. The usage of these tags is not that consistent and somewhat ambiguous. For example, the dictionary reads: "In general where there are multiple senses in an entry, the part-of-speech of an earlier sense will apply to later senses unless there is a new part-of-speech indicated." The problem is that there is no definition of which tags are "incompatible", some tags clearly apply in addition to the tag defined in the first entry, while others override it.

Applying these new restrictions resulted in some strange errors. For example, for the entry on 中国, the dictionary specifies that the sense 'China' only applies for the writing ちゅうごく. This is not correct as China is almost always written as 中国. Maybe the author meant that ちゅうごく is the only correct pronunciation, but then again, there are no other pronunciations for the word 中国. This restriction results in 中国 being translated as 'province of the second lowest rank' (a term from the ancient Ritsuryō system of law), since this sense does not have any restrictions, even though it should be China. In similar manner, the word だ ends up not having a POS tag at all. だ has the very common meaning of is/be, but was now translated as 'has'.

In the final implementation I kept these restrictions after I, with some tweaking, managed to get the same score as when not having the restrictions. Including them should mean that if the inconsistencies are fixed, the system will get a boost in performance. This example shows that adding more complexity, rather than picking the first results (is/China), does not always result is better scores.

6.5.5 Non-Japanese names

When a non-Japanese name is written in Japanese, it is transliterated to the Japanese alphabet. This means that the translator will find a series of Japanese letters whose pronunciation is as close as possible to the original name. Since the Japanese alphabet is quite different than western alphabets, the pronunciation will often be very different. For example, the name Brent is written as ブレント in Japanese, and is pronounced 'Burento'.

Clearly, translating the Japanese writing directly will not result in the original name. I tried solving this by training a model to restore the original name based on the Japanese writing. I used the sequence to sequence approach to translate the names. Specifically, the seq2seq module in TensorFlow[21] is used, which is based on the original sequence to sequence paper by (Sutskever, Vinyals and Le 2014). The code for generating the model was adapted from a project that did translation from roman letters to Japanese, I simply reversed the dataset.[22] The model was trained on around 100.000 names acquired by crawling Wikipedia page titles. However, the results were less than satisfactory. The model had an error rate of 82% when testing on the Wikipedia list.

Upon further inspection, there were multiple issues that could contribute to this result. First off, the spelling of western names in Japanese is not consistent. In the dataset, the name Garvey was found to be spelled ガーヴェイ(gaavei), ガーヴィー(gaavii), ガーベイ(gaabei) and ガービー(gaabii) in different places. Furthermore, when transliterating to the Japanese alphabet, a lot of information is lost. This is

[21] https://github.com/tensorflow/nmt#introduction

[22] https://medium.com/@wanasit/english-to-katakana-with-sequence-to-sequence-in-tensorflow-a03a16ac19be

because Japanese has fewer possible syllables compared to western languages. This means that for example the names Nikolaj, Nicolai, Nikolai and Nicolaj would all be written as ニコライ. From this we can conclude that, even for a human, there is no way to translate the Japanese writing of a western name without knowing the context in which it is used. That is, you would have to know what thing or person is being mentioned.

In the current version, the program uses a dictionary of proper nouns and the list of names from Wikipedia to translate western names. Failing to find the name, the system will output the Japanese pronunciation. Note that even if the name exists in the dictionary, this does not guarantee that the translation will be correct because of the ambiguities discussed above.

6.5.6 English text

When a token is identified as being English text, it is not translated, but simply passed on as is.

6.5.7 Numbers

In Japanese, numbers are constructed using slightly different rules than in English. For example, 50 thousand is written 5 万, which translates to 5 ten-thousands. Just like in English, numbers can be written using Arabic numerals, words or a combination of the two (as above). To translate Japanese numbers to English, the system first translates the Japanese to an Arabic numeral. Then, the prefixes million, billion etc. are added. This is done using the libraries japanese_numbers [23] and millify.[24]

6.5.8 Unknown words

Any token that fails to be translated using the logic described above will be left as is.

[23] https://github.com/takumakanari/japanese-numbers-python

[24] https://github.com/azaitsev/millify

7 Evaluation

These sections present and discuss the results of evaluating the system. Sections 7.4 and 7.5 contain the test results. Section 7.1 describes the test data, section 7.2 the test metrics. Section 7.3 explains how the testing handles system translations that are synonyms of the gold translation. 7.6 contains the results of aligning natural translations using fast_align. Finally, section 7.7 categorizes different types of errors.

7.1 Description of test data

The primary test data used in this project were 200 tokenized and translated sentences. The sentences contain a line with the original Japanese sentence, a line with a natural English translation and lastly one or more lines containing the Japanese tokens and their English translations.[25] Only the tokens are required, the first two lines are only present to improve readability.[26] When I mention translations in this section, I'm referring to the translations of the tokens. The sentences were divided into two sets of 100 each, one for development and one for testing. The testing set was only used during the final testing of the program, in order to show any bias towards the data used during development. The tokenized sentences had to be hand-written specifically for this project. This is the reason for the relatively low amount of test data, as it is quite labor intensive to make. Having at least a couple of hundreds of sentences in the test and dev sets would have been ideal, as there might be constructions not covered by the current tests. Half of the sentences were created by me. The other half was created by Japanese translator Jiro Osuga.[27] I revised the translations done by Osuga after completion. This is another potential issue. As I had read the test set, any major issues present in this set would be known to me during development, even though the program was not tested using this dataset during development. In general, Osuga and I were in agreement about the translations. I ended up changing about one token per sentence in the translations done by Osuga. The changes were mostly corrections in word endings and changing the translation of particles to an actual translation, instead of the pronunciation in diamond braces, as I wanted to have as few untranslatable particles as possible.

The test data comes from three sources. The three sources are from different domains, which ensures that the system is robust when faced with different kinds of text.

The first source is the Tatoeba corpus.[28] The sentences were picked randomly using the website's own search tool. These sentences are characterized by being quite simple and straight forward. They are intended to be complete, that is, they are not part of some larger context.

[25] For examples, refer to section 5 or the sentences in the repository's data folder.

[26] Except that the natural English translations have been used for testing the inference system, refer to section 7.5.5

[27] https://www.peopleperhour.com/freelancer/translation-tutorials/jiro-osuga-translator-japanese-to-english-and-znnxjjy

[28] https://tatoeba.org/eng/downloads

The second source is the Universal Dependencies Japanese PUD dataset.[29] The sentences in this dataset were taken from News articles and Wikipedia pages. These sentences are quite formal in nature. This is an important factor in Japanese, where the wording is quite different when writing formally.

The last source is the Japanese-English Subtitle Corpus by (Pryzant, et al. 2018). Being subtitles, these sentences are representative of spoken language, and include many contractions, incomplete sentences and instances of slang. The sentences also contained spaces. This is not usually seen in Japanese, even in informal text. Furthermore, the spaces were even put in the middle of expressions, such as after a particle binding a noun and a verb together, something that a Japanese person would definitely never do. The spaces are most likely due to pauses in the dialogue of the source material. I chose to keep these spaces to ensure that the program would be able to handle spaces, even if they are not common.

A comparison of the scores for the different corpora can be seen in section 7.5.2.

7.2 Evaluation metrics

7.2.1 Tokenization

For tokenization, I use the precision, recall and F1 scores (Rijsbergen 1979), based on the number of tokens with correct start/end indices. This is in alignment with the metric used in Universal Dependencies[30] e.g. (Zeman, et al. 2018). The F1 score is calculated based on the following formula, where s is a sentence.

$$2 * \frac{correct(s)}{|GoldTokens(s)| + |SystemTokens(s)|}$$

Formula 2 - Tokenization F1 score

For the final result, the macro average of all sentences in the test set is used. This means that the score for all sentences are calculated, then the average of these scores is the final result.

7.2.2 Translation

For testing translations, I use token error rate (TER): the rate of tokens that are wrongly translated. This is a metric specifically designed for this project; however, it is very close to word error rate. The only difference is that tokens may be multiple words in a short expression. To calculate TER, I use the Levenshtein Distance algorithm on the lists of tokens (Levenshtein 1966). I changed this algorithm so that it outputs how many times each operation (insertion/deletion/replacement) was used. The total of these, divided by the total number of tokens, correspond to the TER. The reason for using this metric is that each token is translated individually and so we are interested in how many of these translations were

[29] https://universaldependencies.org/treebanks/ja_pud/index.html

[30] It should be noted that some publications choose to multiply F1 score by 100 for readability. In this report, I have chosen to leave the F1 score as a value between 0 and 1.

done correctly. Formula 3 shows how the TER is calculated, s is the system tokens being tested and g is the gold tokens.

$$\frac{\text{LevenshteinDistance}(s, g)}{|GoldTokens(s)|}$$

Formula 3 - Token Error Rate

Same as for tokenization, the macro average is used.

7.3 Handling of synonyms

Synonyms is a very prevalent type of error in the system. Often, the system translation is similar to the gold translation, but has slightly different wording. This includes pairs like 'film/movie', 'henceforth/from now on' or 'stamp/postage stamp'. In fact, these constitute the majority of the errors in the program. When tests are run with the '-p' argument, the system will try to detect such pairs and count them as correct. The original scores are still important, since the choice of which synonym to use might be important in some cases, but knowing how many errors are due to synonyms is a good distinction to make. In order to find synonyms, multiple techniques are applied.

The first technique is using a database of paraphrases. The database used is the PPDB by (Pavlick, et al. 2015). This contains a large amount of phrase/paraphrase pairs. In the system, a translation may then be counted as correct if it is a paraphrase of the gold translation. The database comes in multiple sizes. Choosing the correct size is important, as the larger versions will contain pairings that would not be considered legitimate paraphrases, while the smaller versions might not find all paraphrases in the translations. Section 7.5.4.1 compares the scores when using different sizes.

The second technique is word-vector similarity, using the word2vec model. This is the same model described in section 6.5.4.1. For cases where both the system translation and the gold translation are single words, the similarity is used. If one of the translations consists of more than one word, the n_similarity function is used, this calculates the cosine similarity between the two sets of words. The algorithm first finds the mean of the word-vectors in the two expressions, then it finds the distance between these two means. The most important variable when using this approach is the cutoff at which two translations are considered close enough to be counted as correct. Different cutoffs are compared in section 7.5.4.2.

The last technique is simply to check if the gold translation is a substring of the system translation or vice versa. The tests for this are in section 7.5.4.3.

An alternative to the methods above is described by (Wang and Merlo 2016). The metric consists of using word-vectors to give a score based on how similar the translations are, on a scale ranging from 0 to 1. Tests using this metric are shown in section 7.5.4.4.

7.4 Tokenization

7.4.1 Sudachi vs. Nagisa

The scores are shown in table 2. Nagisa generally makes more errors in tokenization. Specifically, it sometimes produces tokens that are not real Japanese words, which Sudachi does not as it is based on a dictionary and only makes tokens that are in this dictionary. This could also mean that Sudachi would not be able to handle words that are not in the dictionary, but as the results indicate and according to my observations, this did not happen, at least in the sentences from the test data.

Note that there might be a slight bias towards the Sudachi implementation, as the majority of the development time was spent using this implementation.

	Sudachi	Nagisa (default)	Nagisa (UD-GSD trained)
Precision	0.900	0.887	0.888
Recall	0.930	0.923	0.901
f1	0.914	0.903	0.893

Table 2 - Scores for Sudachi and Nagisa tokenizer implementations. Nagisa was tested both with its default tokenization and with the model trained on the UD-GSD corpus.

7.4.2 Dictionary based merging

Table 3 shows the difference in score when enabling the merging of tokens based on the Japanese-English dictionary. The merging improves the score a good amount. Especially the merging of WH-questions contributes to the higher score, since these are two words in Japanese but not in English.

	No merging	With merging
Precision	0.875	0.900
Recall	0.922	0.930
f1	0.896	0.914

Table 3 - Scores for dictionary based merging

7.5 Translation

In the results, the first four rows are the average amount of errors in each sentence. The last four are the averages, but where the amount of errors in a sentence have been divided by the number of tokens in the sentence (the error rate). The most interesting result is usually the error rate (as described in section 7.2.2). Unless mentioned otherwise, the tests are based on the dev sentence set.

7.5.1 Main results

As table 1 shows, substitution is the most common operation, which means that tokenization is not a big issue. This makes sense given the relatively high tokenization scores. Deletion rate is higher than insertion rate. This means that often the system translation has too many tokens, i.e. it is “too tokenized”. This is why I focused on trying to merge tokens, so that the level of tokenization done by the system is more in line with the desired output.

Averages	Dev	Test
errors	4.019	4.030
deletions	0.709	1.040
insertions	0.097	0.120
substitutions	3.214	2.870
error rate	0.285	0.347
deletion rate	0.050	0.124
insertion rate	0.008	0.009
substitution rate	0.227	0.214

Table 4 - Results of testing the system, with the current standard settings, using the dev and test sets respectively. This includes all optimizations except for the inference system.

7.5.2 Scores by corpus

Table 5 shows scores that are made by running tests on sentences from each corpus individually. Since the number of tests is already quite small, these tests used sentences from both the dev and test set. As the results indicate, the UD-GSD corpus has the lowest error rate. This is likely due to the fact that these sentences are quite formal, and thus use proper grammar, which is easier for the program to understand. The subtitle corpus has a lower error rate than Tatoeba, which I find to be surprising given that the subtitle corpus is quite noisy and the Tatoeba corpus on the other hand contains very simple sentences. However, it should be noted that none of the differences are big enough to be considered statistically significant.

Averages	All	GSD	Subtitles	Tatoeba
errors	4.025	5.824	4.220	3.080
deletions	0.872	1.118	0.800	0.800
insertions	0.108	0.235	0.100	0.050
substitutions	3.044	4.471	3.320	2.230
error rate	0.316	0.289	0.319	0.332
deletion rate	0.087	0.057	0.061	0.116
insertion rate	0.008	0.014	0.009	0.005
substitution rate	0.221	0.218	0.249	0.210

Table 5 - Scores for the different corpora

7.5.3 Wrong definition or wrong word

One interesting property of the errors is whether they are due to the program selecting the wrong English definition of a Japanese word or selecting the wrong Japanese word, i.e. a word with the same writing, but a different meaning. The approaches one would use to solve these issues could be quite different, so knowing which one is most prominent is important. To test this, I used the following procedure: evaluate

all translated tokens, if the Japanese tokens match, but the translations are different, consider this pair in the statistic. If the gold translation exists in the definitions of the selected word, the error is counted as being due to a wrong definition. If the gold translation exists in the definitions of another word with the same writing, the error is counted as being due to the wrong word being selected. In many cases, neither is true, this is counted as "other issue". Also, particles obscure the statistic since they are special cases not translated using the dictionary, so they are given their own category. Table 6 shows the results.

First, the most prominent error is neither wrong definition or wrong word. This means that many gold translations are not ones that exist in the dictionary, which means that there is no way to correct these errors using the approach of dictionary lookup. An example of this is the word 会社. In one sentence it is specified that this should be translated as "office". However, the closest dictionary definition is "workplace". The "office" translation can only be made given a context where it is known that the workplace in question is an office.

Back to the question of wrong definition vs. wrong word, wrong definition is more common. This shows that the techniques for selecting words based on pos etc. is working quite well and that better selection of definitions, as described in section 6.5.4.1, has a higher potential for improving the score.

wrong definition	75
wrong word	26
other issue	125
particle	56

Table 6 - Distribution of error types

7.5.4 Detection of synonyms

These are the results for the different methods for detecting synonyms. For all tests in this section, only the method being tested is enabled. The score when enabling all three is the one shown in the main results.

7.5.4.1 PPDB

As table 7 indicates, the difference in score between the sizes is quite small. Indeed, even the large version fails to find paraphrases such as: 'hurt/painful', 'discovery/find' and 'now and then/sometimes'. On the other hand, during testing, I found that the medium and large versions were quite lenient in their definitions of paraphrases, containing equalities such as: hurt = caused, tears, exposed, effected. In order to be conservative, the small version is used in the final program.

Averages	PPDB disabled	Small	Medium	Large
errors	5.350	4.845	4.767	4.592
deletions	0.709	0.718	0.718	0.718
insertions	0.097	0.107	0.107	0.107
substitutions	4.544	4.019	3.942	3.767
error rate	0.380	0.344	0.339	0.328
deletion rate	0.050	0.051	0.051	0.051
insertion rate	0.008	0.009	0.009	0.009
substitution rate	0.322	0.285	0.279	0.268

Table 7 - Comparisons of the different sizes of the PPDB (Pavlick, et al. 2015).

7.5.4.2 Word vectors

Gensim[31] allows one to specify the number of word-vectors loaded. Loading all of the vectors takes a lot of time, for this reason the library allows the user to limit the number of vectors to be loaded. Table 8 shows the difference in score when loading different numbers of vectors. The Google News vectors are used here.[32] As table 8 indicates, the score stops changing after around 100,000 vectors loaded. This is because the vectors are sorted by how common the word/phrase is, so once all the common words have been loaded, more vectors won't make a big difference.

Averages	1,000	10,000	100,000	1,000,000	3,000,000 (all)
errors	5.010	4.408	4.252	4.243	4.243
deletions	0.718	0.709	0.709	0.709	0.709
insertions	0.107	0.097	0.097	0.097	0.097
substitutions	4.184	3.602	3.447	3.437	3.437
error rate	0.355	0.312	0.301	0.300	0.300
deletion rate	0.051	0.050	0.050	0.050	0.050
insertion rate	0.009	0.008	0.008	0.008	0.008
substitution rate	0.295	0.253	0.243	0.242	0.242

Table 8 - Comparison of errors when loading different numbers of vectors

[31] https://radimrehurek.com/gensim/

[32] https://code.google.com/archive/p/word2vec/

With 100,000 vectors loaded, I tested multiple cutoffs for when to consider a pair to be equal.

Table 9 shows the results, but on its own, this does not say much, as a lower error rate may be due to false positives. When selecting the default cutoff, I looked at the output of the program and found that 0.4 was the highest setting with which there were few to no false positives.

This approach is not without problems. The definition of two words being related in word2vec is a bit different than them being similar in meaning. Two words are related in word2vec if they belong to a similar category or are used in the same contexts. For example, a query on the word2vec model shows that the words 'red' and 'blue' are very closely related. This is because they are both colors and are related in that respect. This does not mean that 'red' would be an acceptable translation, if the intended translation is 'blue'.

Averages	word vectors disabled	0.6	0.5	0.4	0.3	0.2
errors	5.350	4.883	4.563	4.252	3.825	3.437
deletions	0.709	0.709	0.709	0.709	0.718	0.748
insertions	0.097	0.097	0.097	0.097	0.107	0.136
substitutions	4.544	4.078	3.757	3.447	3.000	2.553
error rate	0.380	0.344	0.323	0.301	0.272	0.243
deletion rate	0.050	0.050	0.050	0.050	0.051	0.053
insertion rate	0.008	0.008	0.008	0.008	0.009	0.011
substitution rate	0.322	0.285	0.264	0.243	0.212	0.179

Table 9 - Number of errors given different cutoffs for when to consider words to be equal

7.5.4.3 Substrings

Table 10 shows the difference in score when counting a translation as correct if it is a substring of the gold translation or vice versa. As the table indicates, this simple approach improves the score quite a lot. Usually this is due to the dictionary definitions being a bit more verbose than the gold translation, such as 'stamp/postage stamp'.

Averages	Substrings disabled	Substrings enabled
errors	5.350	4.786
deletions	0.709	0.709
insertions	0.097	0.097
substitutions	4.544	3.981
error rate	0.380	0.339
deletion rate	0.050	0.050
insertion rate	0.008	0.008
substitution rate	0.322	0.280

Table 10 - Number of errors when counting substrings as correct

7.5.4.4 Vector weighted scores

Table 11 contains the scores when using the alternative approach of giving translations scores based on the similarity of word-vectors. This is a reasonably accurate score. However, the problem with it is that it is not directly comparable with the other methods, since these return a clear true/false answer i.e. a score of 1 or 0, rather than a number *between* 0 and 1. One way that this is an issue is that when using the weighted scores, all word-pairs will get at least some points. Since the distance between vectors is never infinite, their similarity will always be above 0, no matter how different they are. The same goes the other way around, two different expressions will never have a similarity of 1. The latter is more fair, since it makes sense to score two highly related expressions higher than two expressions that are related but only loosely.[33]

Averages	Weights disabled	Weights enabled
errors	5.350	4.251
deletions	0.709	0.709
insertions	0.097	0.097
substitutions	4.544	3.446
error rate	0.380	0.301
deletion rate	0.050	0.050
insertion rate	0.008	0.008
substitution rate	0.322	0.243

Table 11 - Vector weighted scoring

[33] To clarify, a higher score means a lower error rate in the results.

7.5.5 Inference

Table 12 is a comparison of scores using different solutions for making natural translations. All scores are based on the implementation where particles are not translated, except for the last column. The inference of particles is based on the approach of measuring the distance from the particle to a noun.

As can be seen in table 12, machine translations do better than the human translations, likely because they already more 'literal' than the human translations. Inferring particles improves the translation very slightly, but it is not general at all. In practice, it just fixes one common error, while creating many new errors in the process. The JParaCrawl machine translation is based on a model I trained myself using the JParaCrawl corpus by (Morishita, Suzuki and Nagata 2019). It based on the example provided by the JParaCrawl creators.[34] It performs better than the human translation, but is slightly out-performed by Google Translate[35], owing to Google Translate's superior translation model.

Averages	No inference	Inference: human translations	Inference: Google Translate	Inference: JParaCrawl (PyTorch)	Inference: Human translations (+inference of particles)
errors	4.019	3.864	3.816	3.854	3.854
deletions	0.709	0.709	0.709	0.709	0.709
insertions	0.097	0.097	0.097	0.097	0.097
substitutions	3.214	3.058	3.010	3.049	3.049
error rate	0.285	0.273	0.271	0.272	0.272
deletion rate	0.050	0.050	0.050	0.050	0.050
insertion rate	0.008	0.008	0.008	0.008	0.008
substitution rate	0.227	0.215	0.212	0.214	0.214

Table 12 - Scores when using different natural translations in the inferrence system

[34] https://github.com/MorinoseiMorizo/jparacrawl-finetune

[35] https://translate.google.com/intl/en/about/

7.5.6 Word-definition selection

In table 13, "Select First Definition" refers to the default selection heuristic used in all other sections. Due to the higher error rate when using the two algorithms for selecting definitions, the final program selects the first definition in the dictionary.

Averages	Select First Definition	Co-Occurrence	Word Vector Similarity
errors	4.019	4.466	4.330
deletions	0.709	0.709	0.718
insertions	0.097	0.097	0.107
substitutions	3.214	3.660	3.505
error rate	0.285	0.324	0.316
deletion rate	0.050	0.050	0.051
insertion rate	0.008	0.008	0.009
substitution rate	0.227	0.265	0.256

Table 13 - Comparison of different methods for selecting definitions of words

7.6 Alignment

Table 14 shows the scores when testing the program using the alignments given by fast-align. The tests were done using fast_align trained on the KFTT corpus. I also tested fast align trained on one million sentences from JParaCrawl and with different tokenizations. However, due to the tremendously bad scores for any configuration, I have chosen not to include more in-depth comparisons.

Averages	Fast Align (KFTT)
Errors	10.961
deletions	0.058
insertions	5.058
substitutions	5.845
error rate	0.799
deletion rate	0.007
insertion rate	0.354
substitution rate	0.437

Table 14 - Number of errors given the alignments produced by the fast_align model

7.7 Categorization of errors

In the following sections, I describe different categories of translation errors that are currently produced by the system.

7.7.1 Insignificant differences

As described in section 7.3, I implemented multiple tools to mitigate these errors. Still, there are many equalities that the current system does not detect. A type that the system does not detect is errors in tokenization. The system regards 実現可能性調査 as one token meaning 'feasibility study'. However, the

gold translation specifies that the expression should be two tokens, 実現可能性(feasibility) and 調査(study). Both are acceptable, so ideally the system would count both as correct.

7.7.2 Incorrect tokenization

The tokenization is not perfect, as can be seen from the tests. This contributes to errors in the translation. Most errors involving tokenization are due to the tokens being too short, that is, two or three tokens should really have been one larger token. This is especially prevalent in expressions that were once multiple words or a small sentence, but in modern Japanese have merged into a single word, e.g.ごめんなさい (sorry), すみません (excuse me) or just instances where a single word in English corresponds to multiple words in Japanese, such as 背の高い (stature <no> high), which simply means 'tall' (of a person). As described in section 6.4.2.2, I implemented measures to merge these, but the current implementation only works for some types of words.

7.7.3 Wrong definition or word being picked

Japanese words are generally less ambiguous than western languages, this is mainly due to the use of Kanji. The usage of Kanji means that even if two words are pronounced the same way, they will be written differently. Still, many ambiguous words do exist. For one, words may be written using only hiragana, in which case all words pronounced this way would be matched. Some words written in Kanji are also ambiguous, such as 問題 which can mean question or problem. I mitigated some of these errors using the filtering of words described in section 6.5.4 and inferring the meanings of words as described in section 6.5.3.

7.7.4 Inconsistencies in test data

Some errors are due to faults in the test data, not in the output of the program. This is mainly due to two problems. One, being a new style of translation, the rules for the translation were not completely fixed. This means that it is up to the translator to figure out a translation, where the guidelines do not apply. The style of translation has also been revised during the project, which runs the risk of introducing inconsistencies in the test data. Secondly, my knowledge of Japanese is limited. This means that some constructions may have been misinterpreted. This is especially prevalent in cases of complex grammar or contracted expressions. I have attempted to fix these problems by carefully comparing the system translation with the gold translation in cases where they disagree and studying texts on Japanese language to find out which one is correct. Yet, a few of these errors may still exist in the current version. Additionally, there seems to be disagreements about some aspects of the language. For example, 来ないでくれ means "don't come". Some sources say that 来ないで is the negative te-form of 'to come', while others state that it's not, making で the particle 'de'.[36] [37] [38] Also, whether くれ should be viewed as an ending to the verb or a separate word (to do for one) is not clearly defined, (Makino and Tsutsui, A Dictionary of Basic Japanese Grammar 1986) conflicts with other resources e.g. Kanshudo[39]. The

[36] https://www.sljfaq.org/cgi/verb-inflect.cgi?verb=kuru

[37] https://www.tanoshiijapanese.com/dictionary/conjugation_details.cfm?entry_id=52892

[38] https://www.alc.co.jp/jpn/article/faq/03/2.html (in Japanese, states that ないで and なくて are both te-forms)

[39] https://www.kanshudo.com/grammar/%E3%81%AA%E3%81%84%E3%81%A7%E3%81%8F%E3%82%8C

translations in this project generally tries to stick to Japanese grammatical definitions. In cases like the one above, I have selected the translation I feel best explains how the sentence is constructed, but having no strict rules for all cases leaves the translations vulnerable to inconsistencies.

7.7.5 Counters

For an in-depth explanation of Japanese counters refer to the Tofugu guide.[40] Counters are words used after a number to indicate what type of thing is being counted. The difficulty of translating these lies in the fact that they are sometimes omitted in an English translation and sometimes not. To illustrate this, consider the following expression: この 58 人の患者の話 (The stories of these 58 patients). Here the counter used is 人, the counter for people. Japanese uses the construction "58 peoples of patients", just like you would write "58 slices of bread" in English. In this case, the counter does not appear in the translation because the people are already identified as being patients. However, were we to remove 'patients' from the sentence, it would be: この 58 人の話 which should be translated as "The stories of these 58 people", where is counter is translated as 'people'.

7.7.6 Potential or passive

In Japanese, the potential and passive forms are written and pronounced exactly the same. This means that, barring an analysis of the context, it is not possible to know which one is implied. In the RikaiChamp de-inflection algorithm, the inflection is called "potential or passive", which is also the case in *Literal Japanese*.

[40] https://www.tofugu.com/japanese/japanese-counters-guide/

8 Discussion

8.1 Future work

8.1.1 Alignment

While current alignment algorithms do not seem up for the task of aligning English and Japanese, this does not mean that it would be impossible. I believe that in order to get good alignments, a specialized algorithm would be needed. One of the major problems I encountered while using fast_align was that the system seemed to be quite confused about the Japanese particles. As mentioned in this report, they do not have English counterparts. This means that they probably have a negative impact on the final result as the system will try to make connections even though none exist. A specialized alignment algorithm might include:

- Filtering of particles
- Reordering as in (Ding, Utiyama and Sumita 2015)
- Methods for handling 'missing' words. E.g. Japanese often omits who is doing a verb, while English does not.

8.1.2 Word sense disambiguation

As mentioned in section 6.5.4.1, word sense disambiguation (WSD) deals with determining the sense of ambiguous words. It is an important part of machine translation systems, whether explicitly or implicitly. State of the art algorithms reach F1 scores of up to 79.7, this is perhaps less impressive when you consider that picking the most common sense reaches a score of 65.6. These results are taken from a paper by (Lecouteux, Schwab og Vial 2019). It should also be noted that these scores are reached on English text, so the results may not be comparable with Japanese. Still, WSD *has* been applied to Japanese, for example, (Bond, et al. 2012) have created a sense-tagged corpus for training Japanese WSD models. Even If a good WSD model was implemented, there remains the problem of connecting it to the dictionary used in *Literal Japanese*. A WSD model might return a definition in either a Japanese or English, but going from this definition to an entry in the dictionary would be a non-trivial task. Nevertheless, it would be worthwhile to do further research into how WSD could be implemented as a part of *Literal Japanese*.

8.2 Alternatives to the per token translation

At the beginning of this project, the goal was to make a translation, that while using Japanese word ordering would still produce a readable, easy to understand, English sentence. This would be based on translating each word of the Japanese sentence to an English counterpart. However, over the course of this project, it has become increasingly clear that when making these kinds of translations, even when doing them by hand, the resulting English sentence can look quite strange.

Consider this sentence:

すみませんが、駅へ行く道を教えていただけませんか。

That according to the specification should be translated as:

Sorry but, station to go route <o> tell-te give-polite-negative?.

This is a good translation if one is interested in learning the exact sentence construction used in Japanese, since it is very literal. This is especially true if viewed side by side with the Japanese counterpart, but it certainly requires a thorough reading to understand.

I see two interesting alternatives to this.

A natural-literal midpoint

This would be a softer version of the current approach, with more focus on creating immediately readable English sentences. For example, the sentence above could be translated as:

Sorry but, the route to the train station, could you tell it to me?

Another example:

百歳の誕生日の数日前になくなった。
100 years birthday, a few days prior to that, he died.

This would require a different approach from the one described in this report and would require heavier processing of the English translation. It would also be much harder for humans to agree on the "correct" translation given this style.

An advanced dictionary

This idea is closer to the current system. This would be a tool that for each word in a Japanese sentence would find the correct definition for that word. This is quite similar to current tools such as Jisho.org that allows the user to enter a sentence and look up the words in it. The only difference is that the advanced dictionary would also infer the correct definition of each word. This is what the website Satori Reader does, but there the meanings are selected manually and thus only work on the stories that the website provides.

Comparison

Both ideas have their strengths and weaknesses. The advanced dictionary would be good for outright studying of Japanese. The midpoint would be good for more passive learning, e.g. as a replacement for subtitles on videos or automatic translations of small pieces of text on the internet.[41] The current system lies somewhere in the middle and thus draws on both the strengths and weaknesses of these two ideas.

[41] Such as the feature many social networks provide to translate posts.

9 Conclusion

During this project, I developed a system that is able to translate Japanese sentences into English, on a word by word basis. The system keeps Japanese word order, can detect and translate word-endings and leaves most Japanese particles in the translation. Word alignment was shown to work very poorly with Japanese. The final implementation consists of two steps: tokenization and translation. For tokenization, the library Sudachi was found to provide the best results. In addition to this, I implemented algorithms for merging and finding the meaning of word-endings, which is based on the de-inflection approach used in RikaiChamp, and for merging tokens using a Japanese to English dictionary. The translation is mostly based on dictionary lookup, with matching of pos and kana providing better selections in cases of ambiguous words. The system can be made to detect synonyms in translations using word-vectors, a paraphrase database and sub-strings. The system has an error rate of around 0.3 errors per token or about 4 errors per sentence. This makes the system, in its current state, unsuited for serious usage as a tool for learning Japanese. However, it is a good baseline considering that it's the first program to make this kind of translation. As with any machine translation task, the task defined in this project is extremely difficult to perfect. These difficulties are described in great detail by (Madsen 2009) and it is by many experts considered to be an AI-complete task (Yampolskiy 2012). Additionally, word sense disambiguation, a sub-task of machine translation, is also described as being AI-complete (Navigli 2009). As described by (Läubli, et al. 2020), the problem of machine translation, of any kind, is still an open problem at the time of writing.

10 References

Asahara, Masayuki, et al. "Universal Dependencies Version 2 for Japanese." *Proceedings of the Eleventh International Conference on Language Resources and Evaluation.* 2018.

Barrault, Loïc, et al. "Findings of the 2019 Conference on Machine Translation." *Proceedings of the Fourth Conference on Machine Translation (Volume 2: Shared Task Papers, Day 1).* 2019.

Bond, Francis, Timothy Balwin, Richard Fothergill, and Kiyotaka Uchimoto. "Japanese SemCor: A Sense-tagged Corpus of Japanese." 2012.

Breen, James. "JMdict: a Japanese-Multilingual Dictionary." *COLING Multilingual Linguistic Resources Workshop in Geneva.* 2004.

Ding, Chenchen, Masao Utiyama, and Eiichiro Sumita. "Imporving fast_align by Reordering." *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing.* 2015.

Dyer, Chris, Victor Chahuneau, and Smith A Noah. "A Simple, Fast, and Effective Reparameterization of IBM Model 2." *Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies.* 2013.

Foreign Service Institute. "National Virtual Translation Center." 2007. http://web.archive.org/web/20071014005901/http://www.nvtc.gov/lotw/months/november/learningExpectations.html.

Lecouteux, Benjamin, Didier Schwab, and Loïc Vial. "Sense Vocabulary Compression through the Semantic Knowledge of WordNet for Neural Word Sense Disambiguation." *Preprint*, 2019.

Levenshtein, V. I. "Binary Codes Capable of Correcting Deletions, Insertions and Reversals." *Soviet Physics Doklady*, 1966.

Läubli, Samuel, Sheila Castilho, Graham Neubig, Rico Sennrich, Qinlan Shen, and Antonio Toral. "A Set of Recommendations for Assessing Human–Machine Parity in Language Translation ." *Journal of Artificial Intelligence Research*, 2020.

Madsen, Mathias W. *The Limits of Machine Translation.* University of Copenhagen, 2009.

Makino, Seiichi, and Michio Tsutsui. *A Dictionary of Advanced Japanese Grammar.* The Japan Times, 2008.

—. *A Dictionary of Basic Japanese Grammar.* The Japan Times, 1986.

—. *A Dictionary of Intermidiate Japanese Grammar.* The Japan Times, 1995.

Marta, Garnelo, and Murray Shanahan. "Reconciling deep learning with symbolic artificial intelligence: representing objects and relations." *Elsevier*, 2019.

Mikolov, Thomas, Kai Chen, Greg Corrado, and Jeffrey Dean. "Efficient Estimation of Word Representations inVector Space." *Proceedings of the International Conference on Learning Representations.* 2013.

Morishita, Makoto, Jun Suzuki, and Masaaki Nagata. "A Large Scale Web-Based Japanese-English Parallel Corpus." 2019.

Navigli, Roberto. "Word Sense Disambiguation: A Survey." *ACM Computing Surveys*, 2009.

Neubig, Graham. "The Kyoto Free Translation Task." 2011.

Pavlick, Ellie, Pushpendre Rastogi, Juri Ganitkevitvh, Benjamin Van Durme, and Chris Callison-Burch. "PPDB 2.0: Better paraphrase ranking, fine-grained entailment relations,word embeddings, and style classification." *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 2: Short Papers).* 2015.

Pryzant, R., Y. Chung, D. Jurafsky, and D. Britz. "JESC: Japanese-English Subtitle Corpus." *Language Resources and Evaluation Conference (LREC)*, 2018.

Raganato, Alessandro, Jose Camacho-Collados, and Roberto Navigli. "Word Sense Disambiguation: A Unified Evaluation Framework and Empirical Comparison." *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 1, Long Papers.* 2017.

Rijsbergen, C. J. van. *Information Retrival.* London: Butterworths, 1979.

Sasaki, Yutaka. "The truth of the F-measure." *University of Manchester*, 2007.

Sutskever, Ilya, Oriol Vinyals, and Quoc V. Le. *Sequence to Sequence Learning with Neural Networks.* NIPS Proceedings, 2014.

Universal Dependencies. *CoNLL 2018 Shared Task: segmentation results.* 2018. http://universaldependencies.org/conll18/results-words.html.

Wang, Haozhou, and Paola Merlo. "Modifications of Machine Translation Evaluation Metrics by Using WordEmbeddings." *The COLING 2016 Organizing Committee*, 2016.

Yampolskiy, Roman V. *Turing Test as a Defining Feature of AI-Completeness.* Springer, 2012.

Yuan, Dayu, Julian Richardson, Ryan Doherty, Colin Evans, and Eric Altendorf. "Semi-supervised Word Sense Disambiguation with Neural Models." *Proceedings of COLING 2016, the 26th International Conference on Computational Linguistics: Technical Papers.* 2016.

Zeman, Daniel, et al. "CoNLL 2018 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies." *Proceedings of the CoNLL 2018 Shared Task: Multilingual parsing from raw text to universal dependencies.* 2018.

11 Appendix - Translation style specification

This document specifies how sentences should be translated for the project *Literal Japanese Translation*. For examples, see the sentences_dev.txt and sentences_test.txt documents located in the repository's data folder.
Each Japanese sentence should be split into tokens. The tokens can be said to represent a word in Japanese. If multiple words in Japanese correspond to one word in English, these should be grouped as one token. Endings(inflections) of verbs and adjectives should be one token. The translator should translate in a way where each token is viewed as being part of a sentence. This means that the translation of the tokens should be able to be put together to form a relatively coherent English sentence. This means that explanations of tokens should be avoided, single words or short expressions are preferred. Japanese particles that are purely grammatic should not be translated. Instead, the Translator should write the pronunciation of the particle in diamond braces <>. For sentence final particles, the fixed translation given below should be used with '(se)' indicating that it is a sentence end. Word endings should be translated as the meanings of the endings separated by lines -, see the Word endings section. The following sections will specify fixed translations that should always be followed.

Meanings given in this document are taken from Jisho.org or written by Nikolaj.

11.1 Particles

Word	Meaning	Possible translations
を	1. indicates direct object of action 2. indicates subject of causative expression 3. indicates an area traversed 4. indicates time (period) over which action takes place 5. indicates point of departure or separation of action 6. indicates object of desire, like, hate, etc.	<o>
が	1. indicates sentence subject (occasionally object) 2. But	<ga>
は	1. topic marker particle	<wa>

	2. indicates contrast with another option (stated or unstated) 3. adds emphasis	
に	1. at (place, time); in; on; during 2. to (direction, state); toward; into 3. for (purpose) 4. because of (reason); for; with 5. by; from 6. as (i.e. in the role of) 7. per; in; for; a (e.g. "once a month") 8. and; in addition to	<ni> one of the meanings
と	1. if; when 2. and 3. with 4. particle used for quoting (with speech, thoughts, etc.)	<to> one of the meanings
の	1. indicates possessive 2. nominalizes verbs and adjectives 3. substitutes for "ga" in subordinate phrases 4. (at sentence-end, falling tone) indicates a confident conclusion 5. (at sentence-end) indicates emotional emphasis 6. (at sentence-end, rising tone) indicates question	<no> ‘s (possessive) if at sentence end: explain (se)

も	1. too; also; in addition; as well; (not) either (in a negative sentence) 2. both A and B; A as well as B; neither A nor B (in a negative sentence) 3. even; as much as; as many as; as far as; as long as; no less than; no fewer than 1. 4. even if; even though; although; in spite of	<mo> one of the meanings
ので	that being the case; because of ...; the reason is ...; given that ...	so
で	1. at; inindicates location of action 2. at; whenindicates time of action 3. by; with 1. 4. and then; so	<de> one of the meanings
か	1. marks sentence as question 2. alternative item conjunction 3. whether	if indicating question: ? <ka>
と/って	Used for marking sentence as a quote	<to>
へ	indicates direction or goal (e.g. "to")	<e> to
よ	Used when presenting new information to the listener. Also used when making assertions.	you know? (se)
もの／もん	1. Added to end of some verbs, turn verbs into nouns 2. Casual feminine sentence ender	<mono> feminine sentence end
や	3. Used when making incomplete list of things (usually nouns)	And Or

わ	Establishes emotional connection, generally used by women.	feminine (se)
ね	Eh? Seeks agreement	right? (se)
な	Particle that follows na-adjectives when modifying a verb.	<na>
ぞ	adds emphasis	! (se)
ぜ	adds emphasis	! (se)
さ	used when trying to get someone's attention	hey (se)

Note that this list does not include all of the Japanese particles. All other particles should be translated using an English word or expression.

11.2 Prefixes

prefix	Example	Meaning	Translation
お	お早め	adds politeness	pol-[word]
ご	ご飯	adds politeness	pol-[word]

11.3 Word endings

11.3.1 General

Ending	Example	Meaning	Translation
Tara	食べたら	If, when	[word]-tara
Tai	食べたい	Want to do X	[word]-want
Miru	してみる	to try something	[word]-try
Ku	忙しく	Adverbial form of i-adjectives	[word]-adj
Nagara	食べながら	While	[word]-while
Ba	できれば	If, then	[word]-ba

The English translations of endings are written in the same order as their logically equivalent Japanese counterpart.
The non-past, negative and past forms apply for inflections of adjectives as well as verbs.

11.4 Verb/adjective forms

Ending	Example (to eat)	Meaning	Translation
Non-past	食べる	Dictionary form of verb	[verb]
Non-past negative	食べない	Negates the verb. Not doing verb	[verb]-negative
Non-past polite	食べます		[verb]-polite
Non-past polite negative	食べません		[verb]-polite-negative
Past	食べた	past form of verb	[verb]-past
Past negative	食べなかった	negative past form	[verb]-negative-past
Past polite	食べました		[verb]-polite-past
Past polite negative	食べませんでした		[verb]-polite-past-negative
Te-form	食べて	Used for linking phrases together. (gerund)	[verb]-te
Te-form negative	食べなくて		[verb]-negative-te
Potential	食べられる	To be able to X	[verb]-potential
Potential negative	食べられない		[verb]-potential-negative
Passive	食べられる	X was done. Used as in English when the person or thing that does the verb is not named.	[verb]-passive
Passive negative	食べられない		[verb]-passive-negative
Causative	食べさせる	to make/let someone do something.	[verb]-causative
Causative negative	食べさせない		[verb]-causative-negative
Causative Passive	食べさせられる	someone was made to do [verb]	[verb]-causative-passive
Causative Passive negative	食べさせられない		[verb]-causative-passive-not
Imperative	食べろ	Indicates a command. (impolite)	[verb]-imperative

Imperative negative	食べるな		[verb]-imperative-negative
Progressive	食べている	Indicates that an action is ongoing. Corresponds to the English '-ing'	[verb]-ongoing
Progressive negative	食べていない		[verb]-ongoing-negative
Progressive polite	食べています		[verb]-polite-ongoing
Progressive polite negative	食べていません		[verb]-ongoing-polite-negative