Natural Language Understanding, Generation and Machine Translation
Coursework 2: Neural Machine Translation

The University of Edinburgh, School of Informatics
Andreas Neokleous, Sergios Xenides

July 15, 2019

Understanding the Baseline Model (Question 1)

Comment A

When self.bidirectional is to True, two LSTMs are used. This, allows the network to have both forward (left-to-right) and backward (right-to-left) information about the input sequence. The final hidden states of each LSTM (final_hidden_states) are concatenated. The final cell states (final_cell_states) of each LSTM are also concatenated using the same function (combine_directions).

The cell state is the memory of an LSTM and the hidden state is the output of that cell. The cell state has interactions with gates, that allow information to be added, removed or returned. The previous hidden state is used by these gates to make decisions. The cell state is the core idea of an LSTM that allows it to access long-term information about the data.

Comment B

The attention weights are calculated using the softmax of the attention scores (attn_scores). The attention context is calculated as the product (weighted average) of attention weights and the outputs of the encoder (source states). The result is "squeezed", to delete all the dimensions of size 1.

A mask is applied to the attention scores before calculating the weights to ensure that sentences of different lengths are handled.

Comment C

Luong et al. [3] examined different attention models (global, local-m, local-p) with different alignment/scoring functions (location, dot, general, concat). In this implementation of seq2seq, the global attention with general scoring function was used. The attention scores are calculated by projecting the encoder outputs to a feed-forward layer, and then multiplying it with the target input. This matrix multiplication (torch.bmm) aligns the encoder and decoder representations.

Comment D

Initializes hidden and cell states with zeros of the size (target_input, hidden_size). Cached state is none when incremental state is not set. Incremental state is mostly used when the model translates as the input is being typed, instead of taking the entire sentence as input at once [5]. Input feed's role is to take into consideration previous alignment decisions.

Comment E

Attention is one of the input for the decoder to calculate its final output by concatenating all attention weights from all time steps. Since it is sequential we need the previous hidden state to calculate the current one. Dropout randomizes the use of specific neurons in the training to avoid over-fitting by forcing the network to use different neurons at a time.

Comment F

Run model to get the output. Calculate the loss based on the output. Backward() calculates the derivative of the loss for every parameter x, dloss/dx. Normalizes the gradients preventing large updates on the model. Resetting the gradients to 0, preventing mixing the gradients between the mini-batches.

Understanding the Data (Question 2)

Figure 1 shows the distribution of sentence lengths in English and Japanese. We can see that the distributions for both of them are positive skewed. This is because of the rare long sentences in the data-set. From the graph we can also see that the majority of English sentences are between 3 and 14 words long, while the majority of Japanese sentences are between 5 and 20 words. Table 1 shows that the most common sentence length in the English data-set is 7 and in the Japanese data-set is 11. Thus, the Japanese data-set favours longer sentences. We can also see that the mean length of an English sentence is 9.3 words and the mean of Japanese sentence is 13.6 words.

Moreover, as shown in the table 1 there is a weak correlation (0.5807) between the length of the sentences of the two languages. For example a long Japanese sentence does not necessarily translate into a long English one and vice versa.

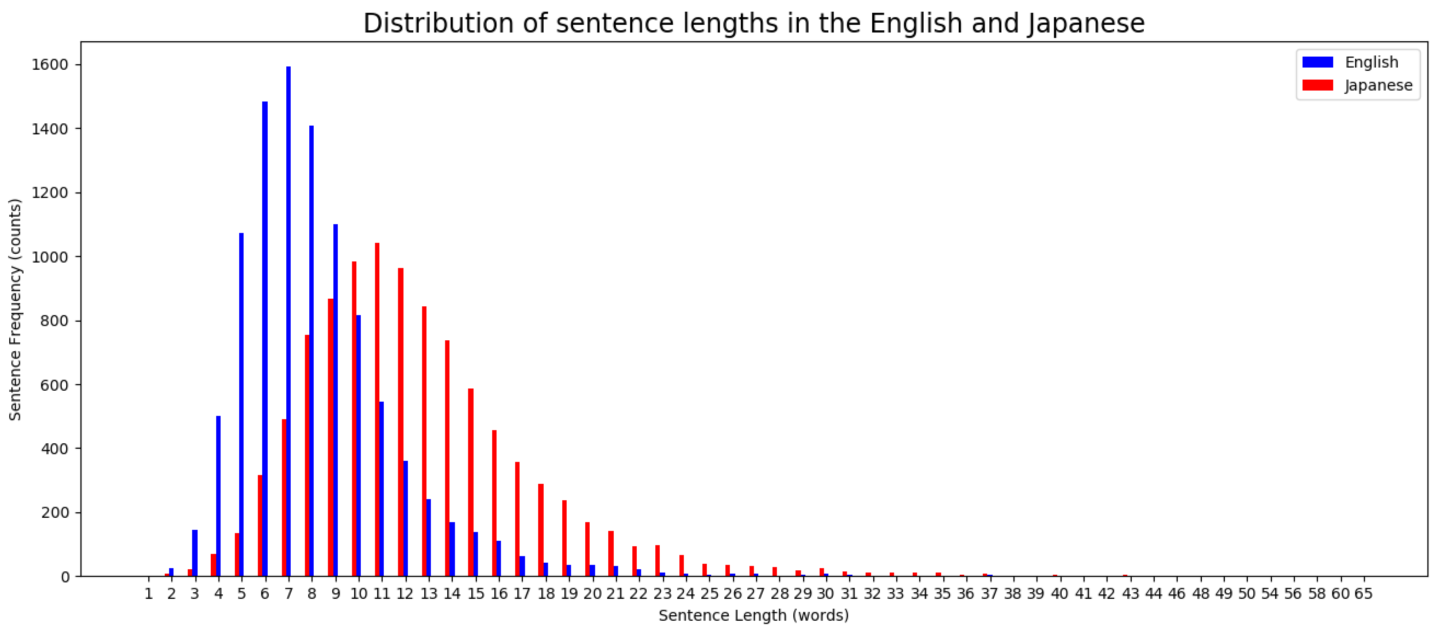

Figure 1: Distribution of sentence lengths in the English and Japanese data.

	English	Japanese
Mean	9.3086	13.6899
Median	9	13
Mode	7	11
Spearman Correlation	0.9070	
Pearson Correlation	0.5807	

Table 1: Statistical information of sentence length distributions.

As seen in Table 2 there are 93086 tokens in English and 136899 tokens in Japanese. This further confirms our findings from Figure 1 and Table 1 that the sentences in the Japanese data-set are longer, since there are more Japanese tokens in total. There are 7040 English word types and 8058 Japanese word types. The English data-set has 3331 word types that appear only once and the Japanese data-set has 4113.

From these observations we are expecting the translation of Japanese to English to be a hard task. The differences in length of the sentences, number of type/tokens and number of UNK words means that in most translations there will not be a one-to-one mapping between the source and target words. Apart from the observed problems, the syntactic structure of each language is different, resulting in different word ordering between the source and the target sentence.

	English	Japanese
Tokens	93086	136899
Types	7040	8058
UNK	3331	4113

Table 2: Counts of tokens, types and UNKs in the training set of English and Japanese.

Improved Decoding (Question 3)

Greedy decoding selects the word with the highest probability ignoring any alternatives that might give better translations. For example, consider the following translations.

Translation 1: I am driving to Germany in May.
Translation 2: I am going to be driving to Germany in May.
Given that:
$P(going|I, am) > P(driving|I, am)$

Greedy decoder is more likely to calculate Translation 2 since "going" is more likely to follow "I am" than "driving", which will produce a poorer translation.

With beam search we would keep track of k most probable candidate translations, instead of taking the most probable translation at each time step. At every time step, the score of each candidate sentence is updated, keep only the k best candidates (the beam). At the end, the translation with the highest score from each word is selected. To implement this, we would add a for-loop in the translate.py file, to go over all possible translations at each time step to and keep the k most probable translations based on the output probabilities of the model. The disadvantage of beam search is the increased decoding time. Freitag et al [1] proposed a method to speed up the decoder without losing translation quality. They achieved this by having a flexible size of candidates at each time step.

Larger sentences require more multiplications to find the best candidates, hence smaller log probability. Since we are looking for the highest log probability, the decoder would automatically favor the short translation for that reason. Although, length normalization can overcome this problem, it still suffers from problems such as under-translation and over-translation as explained in [2]. Since the sentence is normalize, hence bigger than it would supposed to be, some source words might get translated multiple times. To fix this, the coverage normalization is introduced by [6].

MOAR Layers! (Question 4)

Command used to train the deeper architecture: python train.py –encoder-num-layers 2 –decoder-num-layers 3

The model stopped on the 52nd epoch because the validation loss did not improve for 5 epoch. Stopping the training prevents over-fitting since the training loss kept improving.

From the Table 3 we can see that the deeper architecture produces worst results compared to the baseline. This might occur because, deeper networks can over-fit easier because they can find more features in the training set which do not reflect on the validation and test set. Hence, a deeper network is not necessarily a better one. The validation loss is worst than the training lost as expected. The validation (dev) loss is calculated using the validation set which is different from the training set. The training loss is lower because the same data is used to optimize the weights of the network over and over again. The given test set is the same as the validation (dev) set (contains the same sentences), therefore the test loss is the same as the validation loss.

	Baseline	Deeper Architecture
Training Loss	2.334	2.443
Validation Loss	3.1	3.32
BLEU on test	7.98	5.78
Dev set perplexity	22.4	27.6

Table 3: Comparing the results of the baseline and the deeper architecture.

Figure 2 shows the diagram of the deeper architecture.

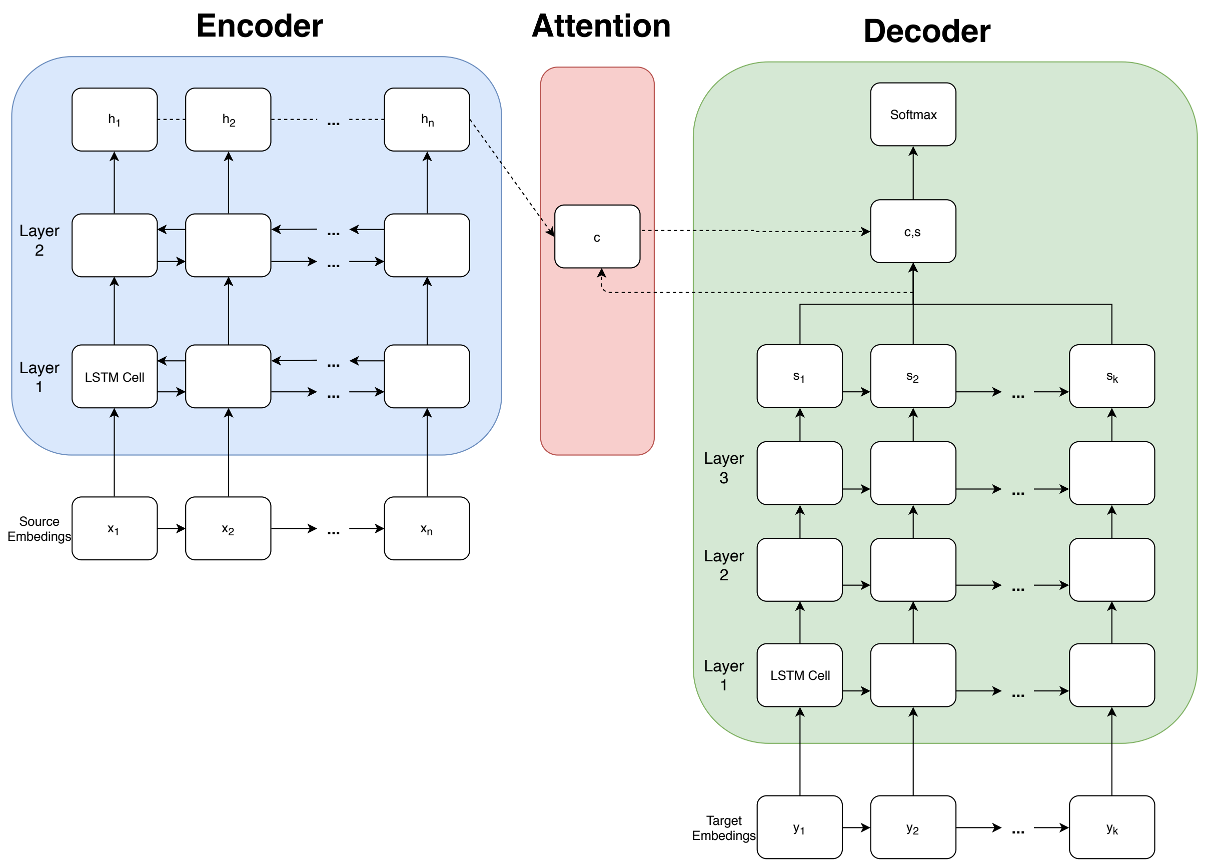

Figure 2: Architecture with 2 layer encoder and 3 layer decoder.

Implementing the Lexical Model (Question 5)

The implementation of this part can be found in the lstm.py file. For each time step the *step_attn_weights* are multiplied using *torch.bmm* with the *src_embeddings*. Then, the sum of the 0 dimension is calculated and tanh function is applied. The output of this is projected to a feed forward layer. Tanh is applied to the result of the projection and then added with the output of the first tanh function. The final output is appended to the *lexical_context* list. This is repeated for each time step. Finally, the list is projected to a linear layer and added to the *decoder_output*.

Effects on Model Quality (Question 6)

	Baseline	Lexical Translation
Training Loss	2.334	1.67
Validation Loss	3.1	2.99
BLEU on test	7.98	11.14
Dev set perplexity	22.4	20.2

Table 4: Comparing the results of the baseline and lexical translation model.

From the table above (table 4) we can see that adding the lexical translation has produced better results giving lower training loss and perplexity and also +3.16 BLEU score. The model stopped at 48th epoch due to early stopping.

Effects on Attention (Question 7)

An indication that the new model works better given the visualizations of the attention heat-maps is the matching of ”EOS” between the two sentences. The pre-trained model has misaligned the ”EOS” and period(.). On the other hand, the extended lexical model matched both ”EOS” and period(.) correctly as shown in Figure 4.

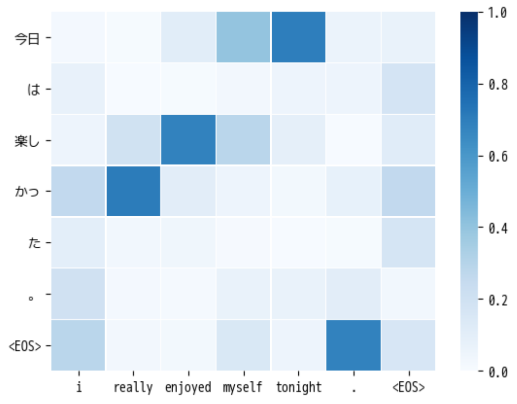

Figure 3: Base model attention heat-map on sentence 8.

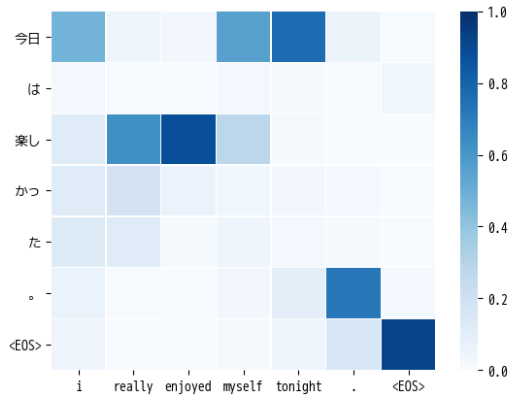

Figure 4: Extended model attention heat-map on sentence 8.

Using Google Translate we can verify some other translations from the above figures. ”Tonight” matched with the first Japanese word which according to Google translate, means ”today” hence we consider it to be a good match for the attention. Also, ”really” and ”enjoyed” matched with the third Japanese word which translates to ”fun” which is another decent match.

Another observation is that Japanese words 2,4 and 5 did were almost ignored by the translation. Removing words 2 and 4 we get a similar translation ”I enjoyed it today”. Removing word 5 loses the tense of the sentence, which neither the baseline nor our lexical model took into consideration.

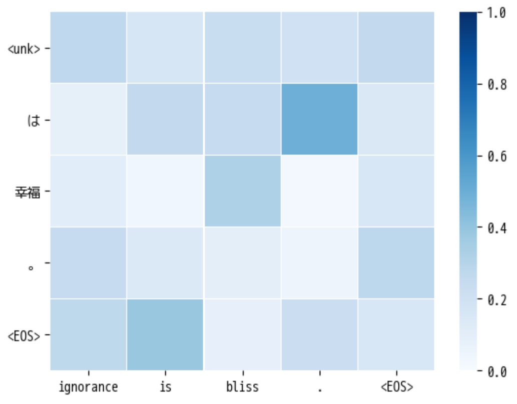

Figure 5: Base model attention heat-map on sentence 0.

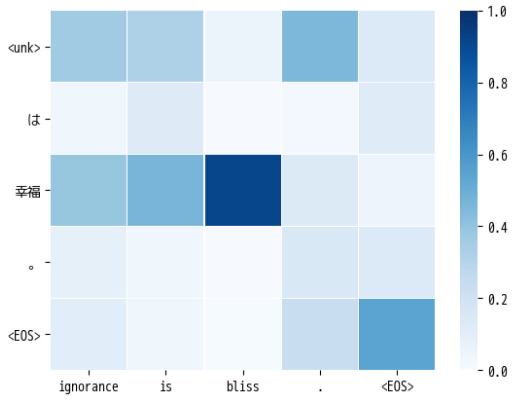

Figure 6: Extended model attention heat-map on sentence 0.

The same improvement can be seen in the second example (Figures 21, 22). The lexical model has improved the attention where the baseline seems to have no impact on the translation. The lexical model has matched correctly bliss with the 3rd Japanese word. It also matched the ”EOS”. Although, there was a misalignment of the period(.) and ”ignorance” which is unknown in the Japanese dictionary.

The baseline has translated the above Japanese sentence to ”i’m sorry” where the lexical model has translated a better sentence, ”happy is happiness”. Therefore, we can conclude that a better attention produces better translations.

The visualizations of the attention weights for the rest of the sentences are shown in the Appendix section at the end of this document.

Analyse Effects on Translation Quality (Question 8)

The lexical model improves the translation of less frequent words compared to the base model [4]. To find evidence for this hypothesis, we implement a code that stores sentences with rare words (words that appear in less than 49 sentences) from the test set and the corresponding sentences in the translation output of the base model and lexical model. We then calculate the BLEU score between those files.

In separate files, we store sentences with more common words (words that appear in 50-200 sentences) and calculate their BLEU score.

To support the hypothesis, we need to compare the difference of the BLEU scores between the base model and the lexical model. If the difference between the rare-word sentences is higher than the common ones, it means that the lexical model worked better for the rare words, hence the hypothesis holds.

	Baseline	Lexical	Difference
Rare	6.65	9.62	2.97
Common	7.77	9.45	1.68

Table 5: Comparing the results of the baseline and lexical translation model.

As shown in the table the difference between the base and the lexical model is bigger than the common with 1.29 BLEU difference. Therefore the hypothesis where the lexical model improves the translation of less frequent words holds.

References

[1] Markus Freitag and Yaser Al-Onaizan. "Beam Search Strategies for Neural Machine Translation". In: *CoRR* abs/1702.01806 (2017). arXiv: `1702.01806`. URL: `http://arxiv.org/abs/1702.01806`.

[2] Yanyang Li et al. "A Simple and Effective Approach to Coverage-Aware Neural Machine Translation". In: *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*. Vol. 2. 2018, pp. 292–297.

[3] Minh-Thang Luong, Hieu Pham, and Christopher D Manning. "Effective approaches to attention-based neural machine translation". In: *arXiv preprint arXiv:1508.04025* (2015).

[4] Toan Q Nguyen and David Chiang. "Improving lexical choice in neural machine translation". In: *arXiv preprint arXiv:1710.01329* (2017).

[5] Baskaran Sankaran, Ajeet Grewal, and Anoop Sarkar. "Incremental decoding for phrase-based statistical machine translation". In: *Proceedings of the Joint Fifth Workshop on Statistical Machine Translation and MetricsMATR*. Association for Computational Linguistics. 2010, pp. 216–223.

[6] Yonghui Wu et al. "Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation". In: *CoRR* abs/1609.08144 (2016). arXiv: `1609.08144`. URL: `http://arxiv.org/abs/1609.08144`.

Appendix

Visualizations of the rest sentences for Question 7

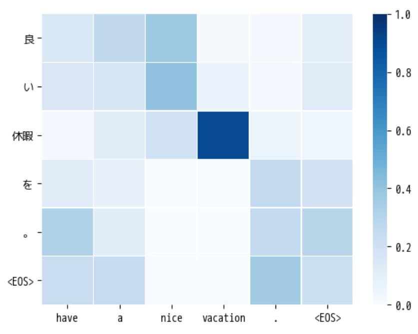

Figure 7: Base model attention heat-map on sentence 1.

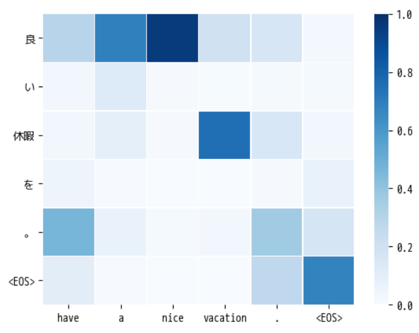

Figure 8: Extended model attention heat-map on sentence 1.

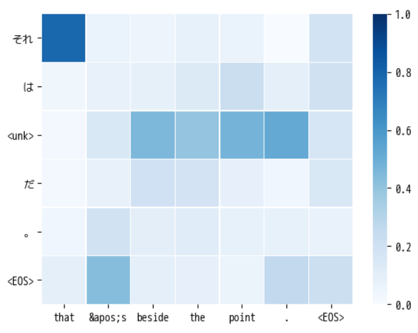

Figure 9: Base model attention heat-map on sentence 2.

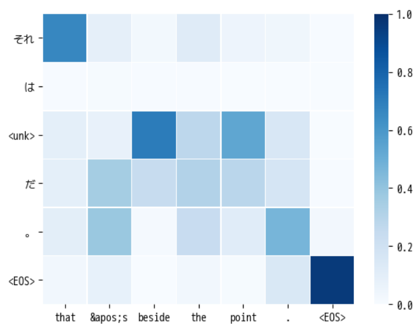

Figure 10: Extended model attention heat-map on sentence 2.

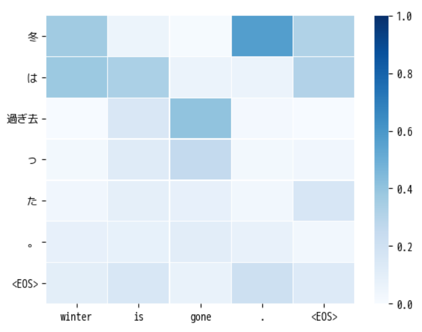

Figure 11: Base model attention heat-map on sentence 3.

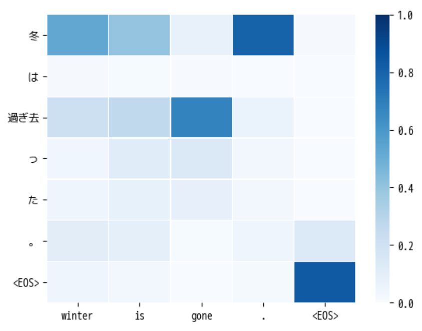

Figure 12: Extended model attention heat-map on sentence 3.

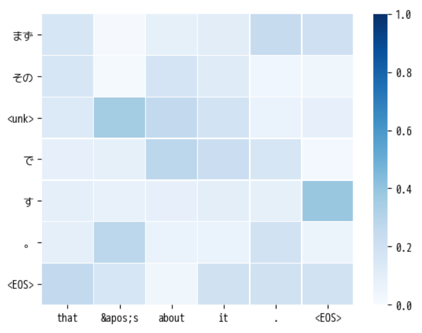

Figure 13: Base model attention heat-map on sentence 4.

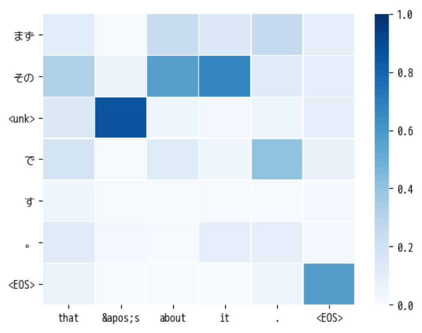

Figure 14: Extended model attention heat-map on sentence 4.

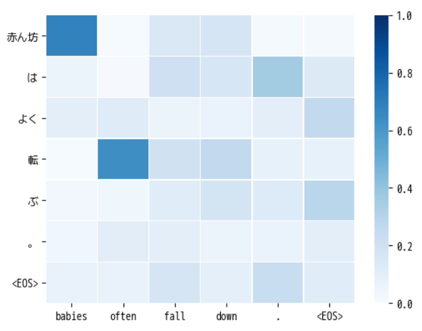

Figure 15: Base model attention heat-map on sentence 5.

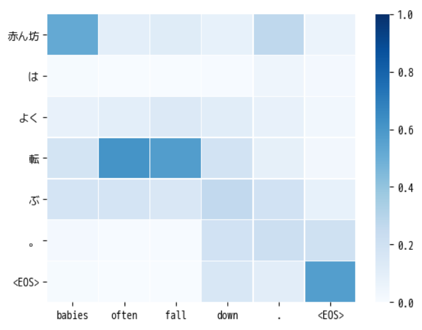

Figure 16: Extended model attention heat-map on sentence 5.

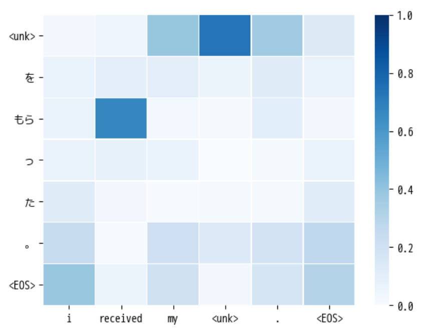

Figure 17: Base model attention heat-map on sentence 6.

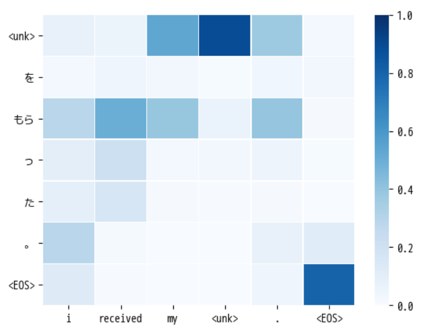

Figure 18: Extended model attention heat-map on sentence 6.

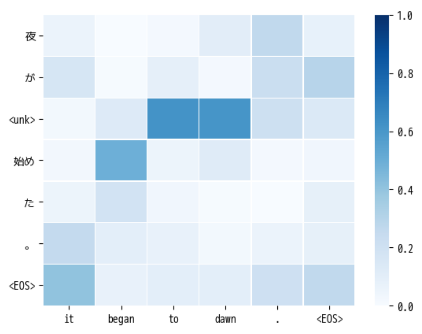

Figure 19: Base model attention heat-map on sentence 7.

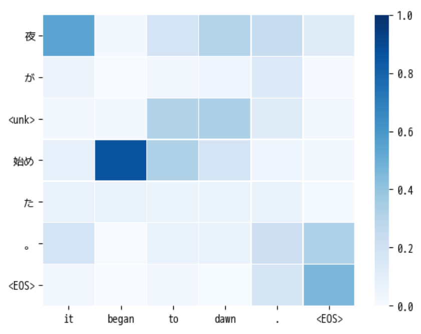

Figure 20: Extended model attention heat-map on sentence 7.

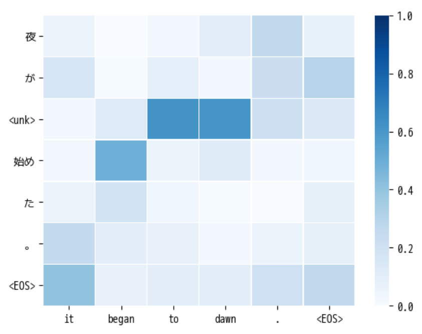

Figure 21: Base model attention heat-map on sentence 9.

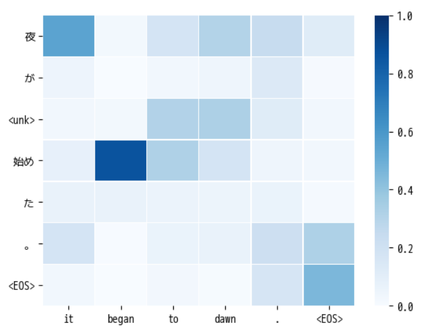

Figure 22: Extended model attention heat-map on sentence 9.