1. (10 points) Copy your code comments here.

NB: Chainer Version 3.3.0

A. 1. Define the structure of the encoder, initialize the bi-lstm layers based on the number of the encoder hidden layers. To be more specific, firstly, define a list structure to store each encoder layers based on the depth of the encoder, which includes the name of each layer. Next, use loop to build the bi-lstm structure of the encoder, each layer is a LSTM structure with n_units input and output.
2. add_link() function is to add layers for neural model (it is deprecated for version 2.0.0, currently replaced by 'chainer.Chain.init_scope'). It is used to define the links between adjacent layers. Added links can be accessed by calling `self.link_name` or `self['link_name']`.
3. Two loops correspond to bi-direction LSTM encoder, left-to-right, and right-to-left respectively. Each loop is used to add LSTM encoder layers in one direction.

B. - L.EmbedID(vsize_dec, 2*n_units) $\rightarrow (3713, 200)$
- L.LSTM(2*n_units, 2*n_units) $\rightarrow (200, 200)$
- L.Linear(2*n_units, vsize_dec) $\rightarrow (200, 3713)$
Because bi-directional lstm model has two directions in the encoder part, and the last states in two directions are concatenated and then passed to decoder layers. Accordingly, the layer embed_id process the input with vocabulary_size dimension and its output is 2*n_units dimension. After that, the LSTM decoder has `2*units` dimension input and output. Then, linear layer after LSTMs receives 2*n_units input processed by LSTM layers, and converts it to vocabulary size.

C. Concatenate the final states in two directions of the last layer bi-lstm (forward and reverse LSTM) together, both last cell state (c) and last hidden state (h). After that, set the concatenated c_state and h_state as the first cell state and hidden state of the first decoder layer respectively.

D. Feed the data to the forward and reverse direction LSTMs and encode sentences in forward and reverse direction with one step at each time. It should be noticed that the `encode` function is to feed the data into the structure of bi-lstm respectively (the forward direction and the reverse direction).

E. The formula of cross entropy is: cross entropy $= -\sum_{i=1}^{M} y_i \log(p_i)$, where y is the probability of predicted words (with vocabulary size dimension) and q is the one-hot vector of target word (value 1 at the target word index, otherwise 0), i is the index of vocabulary. Cross entropy measures the divergence between the predicted probability distribution among vocabulary and target probability distribution. In code, we calculate the sum of accumulated cross entropy value for each predicted next word `predicted_out` and label word `next_word` iteratively. Specifically, `predicted_out` is the predicted probability with full vocabulary-size length, while `next_word` is the target vocabulary ID of the target word.

F. Register `GradientClipping` as hook functions (hook function will be called right after the gradient computation) for the Adam and SGD optimization method. `GradientClipping` function scales all gradient values to fit the defined L2 norm threshold (the threshold of is set to 5 here). Specifically, the scale rate parameter `rate` is equal to the threshold value divided by L2 norm value. If `rate` is less than 1, the gradient will be scaled by multiplying the scale rate. For the first 4 epochs in training loop we use Adam optimization and switch to SGD afterward.

2. (10 points) Examine the parallel data and answer questions. (Plots may appear in the appendix.)

1. As Figure 1, the sentence length distribution of Japanese and English are different. Firstly, the top 3 most counts of English sentence length are 8,7,9 with counts of 1660, 1554,1485 word tokens respectively whilst those of Japanese are 12, 11, 13 with counts of 1098, 1038, 1006 word tokens respectively. The mode (values that appears most often) of Japanese sentence length is larger than the English sentences. It means the corresponding translated English sentence length is likely to be shorter than source Japanese sentence length. Secondly, English is more productive and has richer polysemy than Japanese on word-level, which means that individual English words can express more meanings than Japanese. The **Pearson correlation coefficient** between given Japanese and English text is **0.7663**, which means these length has positive linear correlation (see Figure 2). When the length of source sentence increased, the corresponding target sentence length is very likely to increase linearly.
2. There are 97643 tokens in English and 143580 tokens in Japanese.
3. There are 7211 word types in English and 8251 word types in Japanese.
4. There are 3384 in English and 4171 word tokens in Japanese replaced by `_UNK`.
5. **Sentence length** Due to the fact that the mismatch between the sentence length in Japanese and English, there could be difficult and inaccurate for NMT to translate, because NMT system could miss some words in Japanese when try to translate them into English, and the alignment between Japanese

and English will be complicated.

Type/Token ratios The type/token ratio (TTR) in Japanese and English are 5.75% and 7.39% respectively. A high TTR value means that the language has a high `lexical variation` (Thomas 2005). So, English has a higher lexical variation than Japanese. When NMT system translate Japanese to English, it may select the candidate predicted words from a wider range since English is more productive compared with Japanese. It may increase the difficulty level for our translation task.

Unknown words There are more `UNK`s in Japanese than English. When the NMT system translate Japanese to English and dealing with unknown words, it is definitely more Japanese `UNK`s that can not be translated to English correctly. The out-of-vocabulary (OOV) problem of rare words may influence the performance of the NMT system.

3. (10 points) Answers to questions about decoding

1. 1-best greedy search decoding will pick out the most probable single word at each time step and choose the following candidate words based on the previous output sequence, which could cause the `garden-path problem`. It would be some cases that the ideal prediction consists of a phrase sequence while the first word of the phrase seems to have less probability. Taking the phrase 'pieces of cake' for example, the first word of 'pieces' is not relevant of meaning 'easy' at all, but the full phrase will prove to be better in the latter decoding step. In this situation, there is no way to change the previous output when we find it is wrong with greedy decoding strategy, and some correct translation could be missed. In general, once there is a bad word (the bad word means even though it has the highest probability, it is not very useful to connect with a meaningful sentence) selected during the 1-best greedy search, following words will continually be influenced by that word and then generate a wrong sentence.
2. Beam search keeps K-best candidate hypothesis (beam) scored by the probability. We use each beam to predict the next word at each step and compute the total probability for next beam by multiplying the probability of each beam and its candidate word. Then the top K best beams with the highest probability will be selected. Finally, we choose the beam with the highest probability by backtracking pointers.
3. Decoders are more likely to output sentences with a shorter length. It is mainly because the total probability is calculated by multiplying the probability of candidate word(whose value between 0 and 1) with the probability of its previous beam at each step. Therefore, the longer the sentence length, the lower probability the hypothesis might get (Wu et al. 2016). Length normalization is dividing log probability of each K-best beam search hypothesis by its sentence length(Graves 2012). The usage of length normalization explicitly encourages to translate longer sentences. However, this may make the translation sentences "incoherent, redundant and contradictory" (Shao et al. 2017). There is no strict linear correlation among probabilities of different sentence lengths. Therefore, dividing different sentences by its original length (token number) might make the normalized probability not fair enough, which could make sentences with different length become unequal to be selected. Wu et al. (2016) proposed an refinement that replace $length$ with $length^{\alpha}$ ($\alpha \in (0, 1)$), where α is optimized by the validation set.

4. (10 points) Experiment with more layers. (Diagram may go in the appendix.)

1. Compared with the baseline, the model with 2 encoders and 3 decoders got BLEU of 11.18, mean loss of 5.11 and dev-set perplexity of 37.18 at epoch 12.
2. The diagram can be seen at figure 4 of the appendix. The encoder consists of 2 bi-directional LSTMs, where the forward LSTM and reverse LSTM encode the sentence in two direction. After that, the cell state and hidden state in the last LSTM of each direction are concatenated and set as the initial states of decoder LSTM. Then the decoder of 3 LSTM layers pass the output into a linear layer, mapping the size of `2*units` to `out_vocabulary_size` followed by a softmax layer. Finally, select out (using `argmax`) and output the candidate word with highest probability in each decoding time step.
3.As it could be seen in figure 6, when we train the 2-encoder and 3-decoder architecture, the dev-set perplexity and BLEU perform worse than the baseline at epoch 12. The possible reason for the decrease of dev-set performance might be because of the increment of encoder-decoder architecture complexity. Even though the architecture is deeper and more complicated, other hyper-parameters still be the same as the baseline, such as hidden dimension and max_predict_length. These hyper-parameters may decrease the prediction ability of the new architecture. Moreover, a more complicated architecture need more training epoch to train the optimal hyper-parameter, at epoch 12, the model of 2-encoder and 3-decoder is still under-fitting, which could also influence its predict ability on dev set.

In terms of the worse results on the training set, it is caused by the deeper layers of encoder and decoder layers. Because the complicated new architecture requires more training epoch to train the optimal hyper-parameter. After we run the model with 2-encoder and 3-decoder, as figure 4, the convergence trend of mean training loss curve is slower than the baseline. It causes the fact that the results perform worse than baseline at epoch 12.

Both of the performance on dev set and training set get worse. However, the difference on the training set between dev set and training set has slightly decreased while the difference on the training set is significantly reduced. It is because the baseline has the simple architecture and more easily to converge(nearly converge at epoch 12). From the figure 6, it is obvious that the perplexity of the baseline system increased after 12 epoch while the perplexity of the complicated 2-3 architecture are still going down. However, since whether the model converged or not has no influence on the training set, the difference in training set has a similar decrease trend.

5. (10 points) Implement dropout, and explain the results.

1. Zaremba et al. (2014) added dropouts before each encoder and decoder LSTM layer (including dropout for embedding layer), with suggested dropout ratio of 0.2. Therefore, we add the dropout in the `feed_lstm` function(both embed_id layer and each LSTM layer) with dropout rate of 0.2. Dropout only works in training process, which means the dropout function takes no effect in the validation and test process.
2. As illustrated as figure 5 and figure 6 , models with dropout spend longer time to converge and the BLEU and perplexity improved slower than models without dropout. Compared with baseline, 1-encoder and 1-decoder model with dropout got 0.09 lower BLEU, 0.21 higher mean loss and 1.12 higher dev-set perplexity at epoch 12.

The improvement on the dev-set results from the fact that dropout could help to mitigate over-fitting problem. By randomly dropping some states, dropout could alleviate complicated co-adaptations, make the NMT model less sensitive to training data and thereby improve its generalization ability on the unseen data(more robust). The decrease of the mean loss on the training set is because of the limited training time (12 epoch). we do not train the dropout model until convergence and at the beginning of the training process, dropout may hurt the performance but could get lower training loss after convergence. BLEU is an automatic evaluation method to compare the machine translation results and references. As mentioned above, model with dropout at epoch 12 did not converge, therefore the performance is not as good as model without dropout. When dropout randomly masks some non-recurrent branches of the network, some particular words may be masked and not be translate correctly at this time. As the example below, after applying the dropout, the source word 彼女 will not be translated into 'he'. The performance was hurt by dropout at epoch 12.

source	彼女 が 最後 に ゴール し た 人 で す 。
reference	she was the last to cross the finishing line .
2-3 dropout	he is a to the . ._EOS
2-3 baseline	she has the to to the the . . _EOS

3. Sometimes BLEU and perplexity have the inverse trend , which might be caused by these different evaluation metrics. Firstly, BLEU pays more attention on averaging n-gram methods and can not capture the meaningful changes of candidate predictions. Due to the fact that there is no unique answer for target prediction and one sentence may have various different translations, BLEU score might not reflect the performance precisely. Besides, perplexity is "log-averaged inverse probability on unseen data"(Hofmann 2001). It measures the uncertainty and complexity of the candidate translation. Therefore, BLEU and perplexity are two different methods which evaluate the translation from different perspectives. There is no consistent correlation between BLEU and perplexity metrics when evaluating the NMT system.

6. (20 points) Implement attention. This question will be evaluated on the basis of your code.

7. (10 points) Explain the results of implementing attention.

As figure 5 and figure 1, 1-encoder and 1-decoder models with attention outperform baseline, acquiring perplexity on dev-set at 32.14, BLEU at 14.91 and training mean loss at 4.65. The augmentation of attention speeds up the convergence curve of perplexity and BLEU on dev-set whilst the mean training loss also outperform the baseline. The improvements on all the training set, dev set and BLEU are caused by the effects of attention mechanism. By computing the association between each decoder state and all encoder states, it could figure out the different weights on each input word and find the most suitable alignment between the source language and target language, which could enhance the model performance.

8. (5 points) Explain what you observe with attention. (Figures may appear in an appendix.)

The predicted sentences are partly reasonable. It could be seen from figure 7-11 that only parts of the sentences are correctly translated and some long spelling words are missing. However, it should be noticed that the beginning of the sentences is correctly translated, so it is still acceptable.

source	私 は あなた に あえ て うれし い 。
reference	i 'm glad to see you .
1-1 attention	i 'm like to you you . _EOS
baseline	i 'm to you you . _EOS

Compared with the baseline, attention model can produce better translations than the baseline model although some of them seems not reasonable enough. As above example shows, 'glad' has the similar meaning with the candidate word 'like', so the 1-1 attention model is better than the baseline model. Attention model calculates the weights for each word and finds the suitable alignment words between two languages. The results we report is based on the model with 12 training epoch, which did not reach its maximum performance (see top right graph in figure 5) and its BLEU score continues increasing after epoch 12.

9. (15 points) Where does attention help?

1. **Hypothesis:** 1)Some high-frequency words or common words could be translated more accurate by the attention method. 2)The simple sentence structure `SVO`(Subject-Verb-Object) and declarative sentence are more likely to be improved. 3)Polysemy words in the source language are sometimes not consistent in the target language, which may lead to the mismatch between two languages, attention may help to mitigate this problem. 4)Some verb types such as auxiliary verb and the different tenses of verb may be translated more accurately. **Reason:** Theoretically, attention model we apply is global attention, which takes all the sentence tokens into consideration without bias. At each decode time step, attention mechanism simulates the focus on different encoding words, and the occurrence of these common words appears much more frequent. This enhances the attention of frequently-occurred words (such as 'to','the') and therefore the hypothesis is more easily to contain these words. Moreover, attention method may also help align and order different type of word pairs.

2&3. We calculate the difference between the performance of models with and without attention, and filter out all the predicted hypothesis whose precision and recall improved or degraded by 0.1 and 0.2 respectively. There are 54 sentences whose precision and recall both improved by 0.1, where 10 out of 54 is interrogative sentence, one is imperative sentence and the rest 43 sentences belong to declarative sentence. In addition, there are 4 sentences whose recall and precision increased by 0.2 at the same time among all 500 sentences on dev set. We find that all of them are declarative sentence.

1) We found some high-frequency `personal pronouns` such as 彼女(which means 'her' in English) and 私(meaning 'I' in English), could be translated more accurately than baseline system. 2) Among the predicted sentences where the difference between models with and without attention mechanism, we sort their perplexity gaps in descending order and found that 20 of 54 are `SVO` structures and this could support our previous hypothesis. 3) Plus, attention model performs worse when tackling polysemy words, such as 好き(which means 'like' or 'love' in English). As the following example shows, the word 好き could not be translated correctly in attention method because of its multiple meanings.

source	私 は あなた を 好き に な ら ず に は い られ な い 。
reference	i cannot help falling in love with you .
1-1 attention	i cannot 't not to you . . _EOS
baseline	i cannot not like to to . . _EOS

4) In addition, some auxiliary verbs such as は (means 'is' or 'are' in English), could be more accurately translated by attention model. As the example shows, the attention model could translate the word 'am' correctly while the baseline couldn't.

source	私 **は** その しきたり を よく 知 っ て い る 。
reference	i `am` acquainted with the custom .
1-1 attention	i `am` the the the . . _EOS
baseline	i `have` the the the the . . _EOS

In contrast, attention model did not perform better when dealing with polysemy words. For example, 何 in Japanese is hard to translate into English individually. The composition of 何時 corresponds to `when` in English. This may be because global attention is limited to capture some intrinsic connection between words whilst local attention with reinforcement learning might solve it.

Optional: you may include an appendix after the line above. Everything above the line must appear in the first four pages of your submission.

References

Graves, A. (2012), 'Sequence transduction with recurrent neural networks', *arXiv preprint arXiv:1211.3711* .

Hofmann, T. (2001), 'Unsupervised learning by probabilistic latent semantic analysis', *Machine learning* **42**(1-2), 177–196.

Shao, Y., Gouws, S., Britz, D., Goldie, A., Strope, B. & Kurzweil, R. (2017), Generating high-quality and informative conversation responses with sequence-to-sequence models, *in* 'Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing', pp. 2210–2219.

Thomas, D. (2005), 'Type-token ratios in one teacher's classroom talk: An investigation of lexical complexity', *United Kingdon: University of Birmingham* .

Wu, Y., Schuster, M., Chen, Z., Le, Q. V., Norouzi, M., Macherey, W., Krikun, M., Cao, Y., Gao, Q., Macherey, K. et al. (2016), 'Google's neural machine translation system: Bridging the gap between human and machine translation', *arXiv preprint arXiv:1609.08144* .

Zaremba, W., Sutskever, I. & Vinyals, O. (2014), 'Recurrent neural network regularization', *arXiv preprint arXiv:1409.2329* .

Appendix

Table 1: Comparison for Results of different NMT models

Model				Perplexity	BLEU	Avg Training set loss during lass epoch
decoder layer num	decoder layer num	Dropout	Attention			
1	1	×	×	32.49	13.65	4.74
1	1	√	×	31.37	13.56	4.95
1	1	×	√	32.14	14.91	4.65
1	1	√	√	30.24	14.88	4.86
2	3	×	×	37.18	11.18	5.11
2	3	√	×	36.63	10.61	5.29
2	3	×	√	34.84	13.78	4.84
2	3	√	√	32.09	12.33	5.00

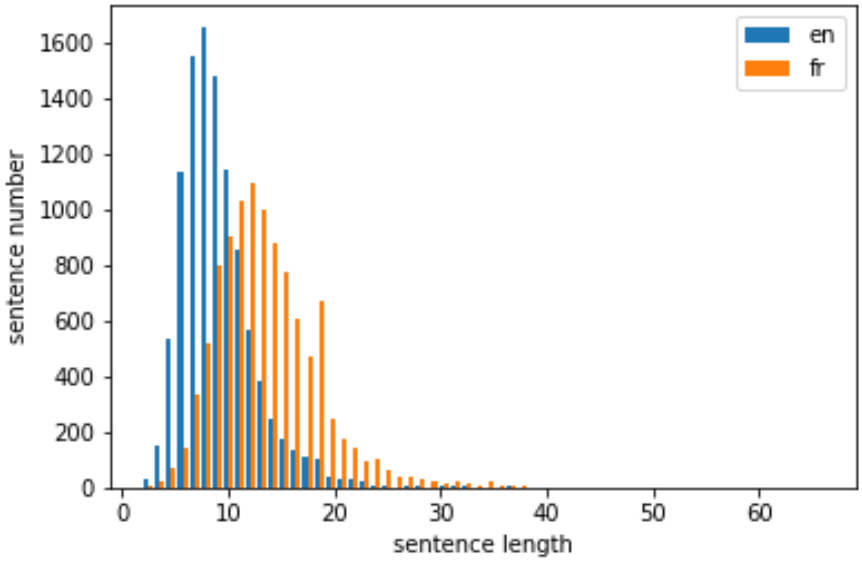

Figure 1: Comparison between English sentence and Japanese in term of sentence length and sentence number

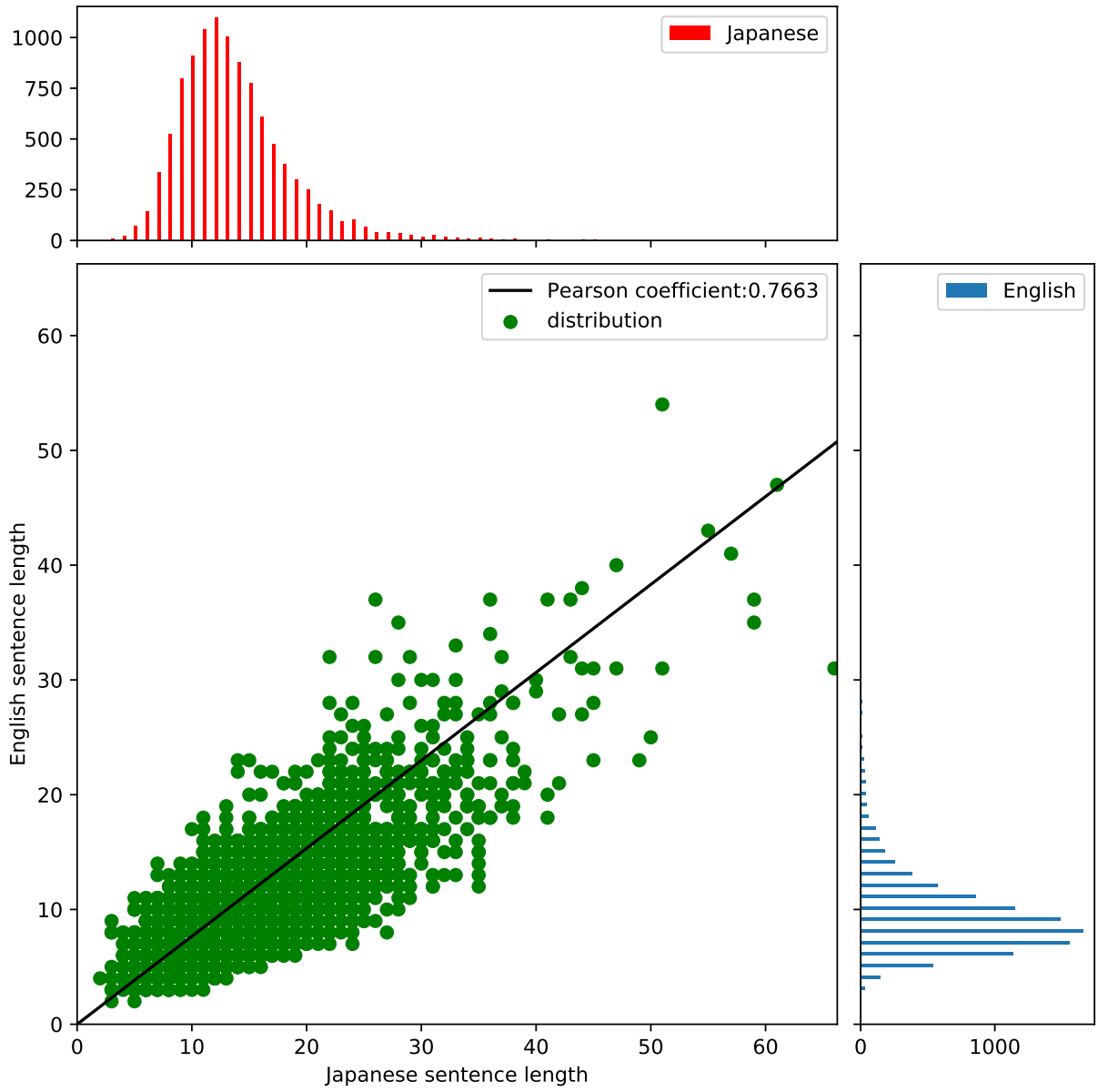

Figure 2: Correlation and distribution of English and Japanese sentences.

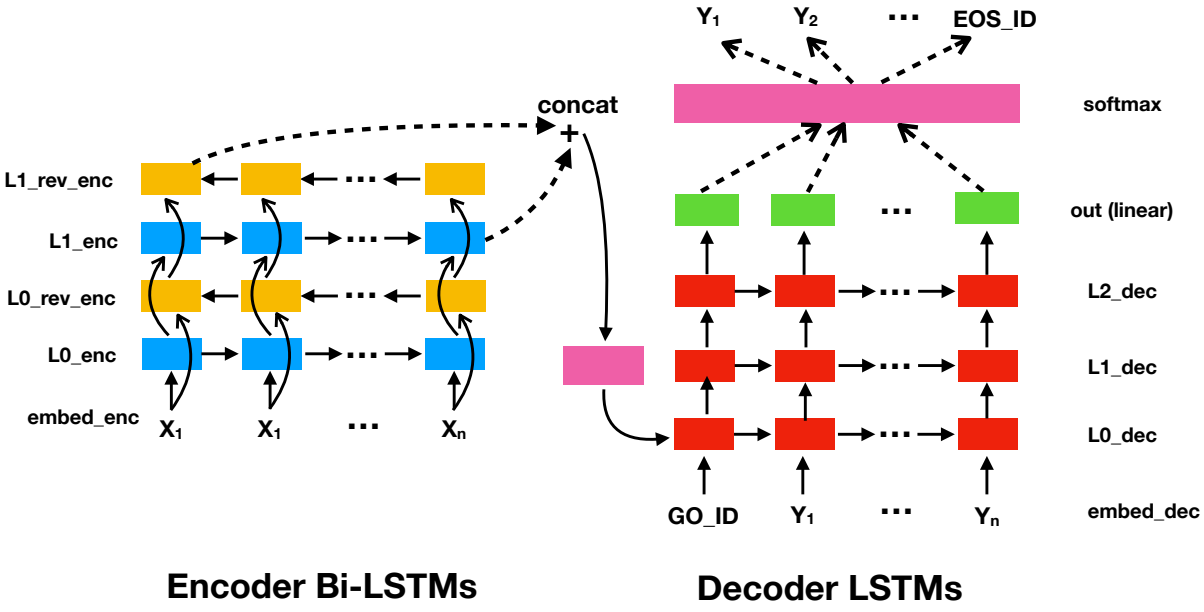

Figure 3: Question4.2, Architecture of 2 encoder and 3 decoder, where the source sentence $(X_1, X_2, ..., X_m)$ has the length of m tokens whilst the target sentence has the length of n tokens, and $+$ means `concat` operation here.

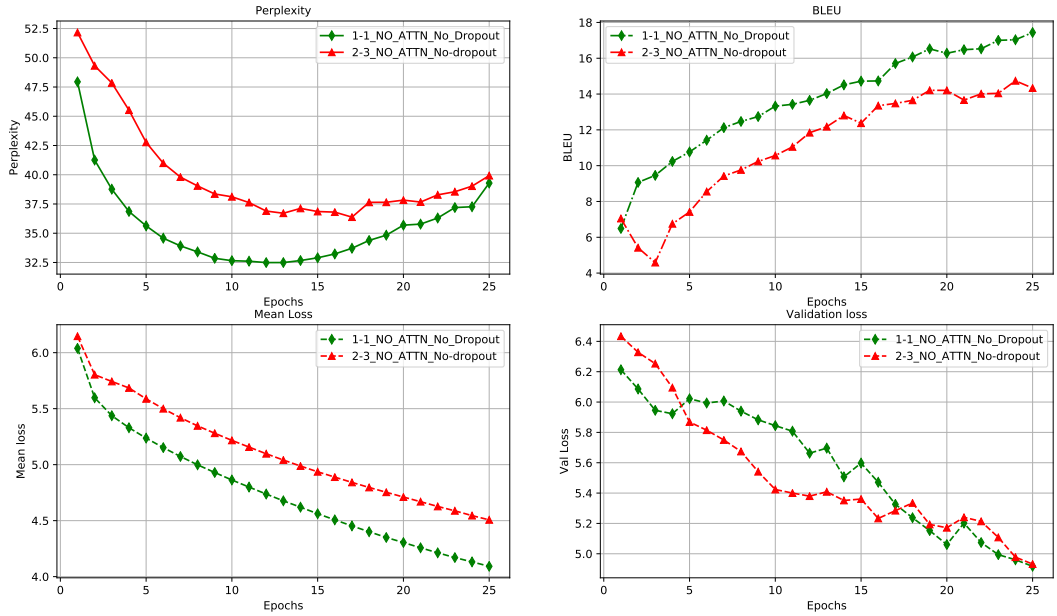

Figure 4: Compare model with 2 encoders and 3 decoders with baseline (1-encoder_ 1-decoder).

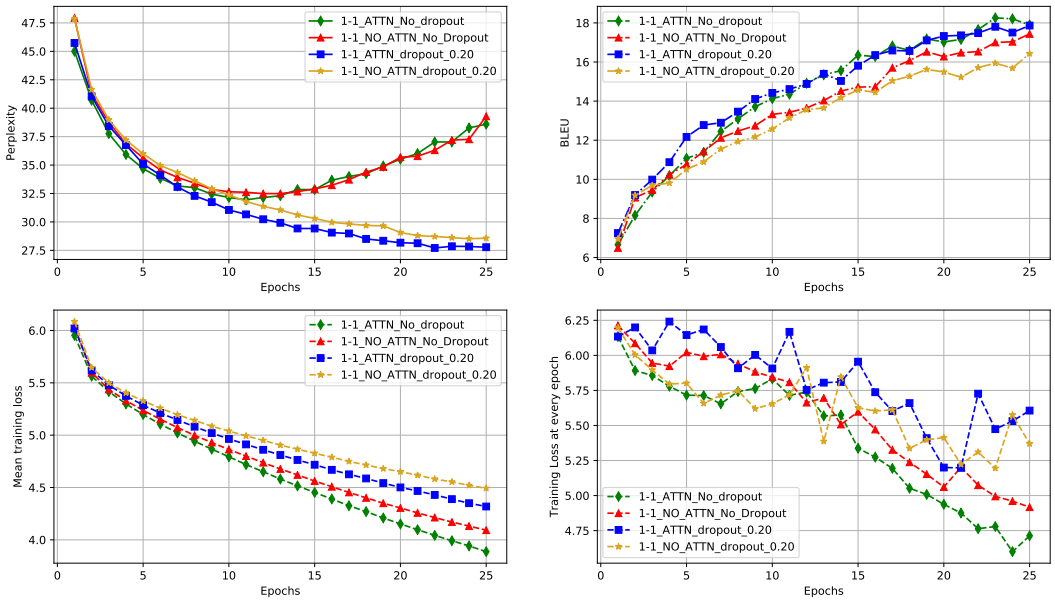

Figure 5: Comparison for results of model of 1 encoder and 1 decoder with different settings.

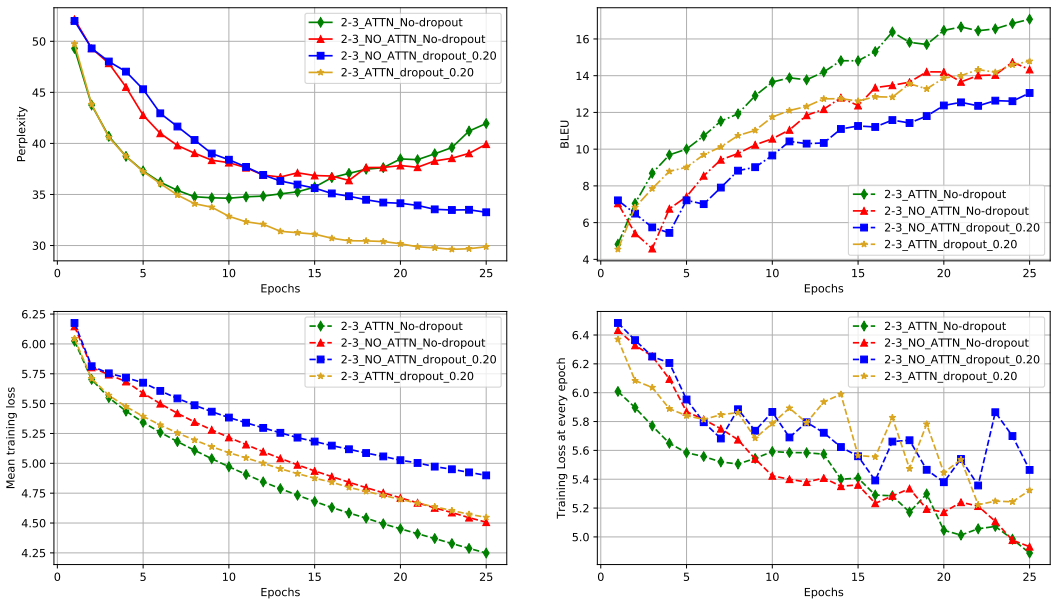

Figure 6: Train models of 2 encoders and 3 decoders with 25 epochs and plot its trends.

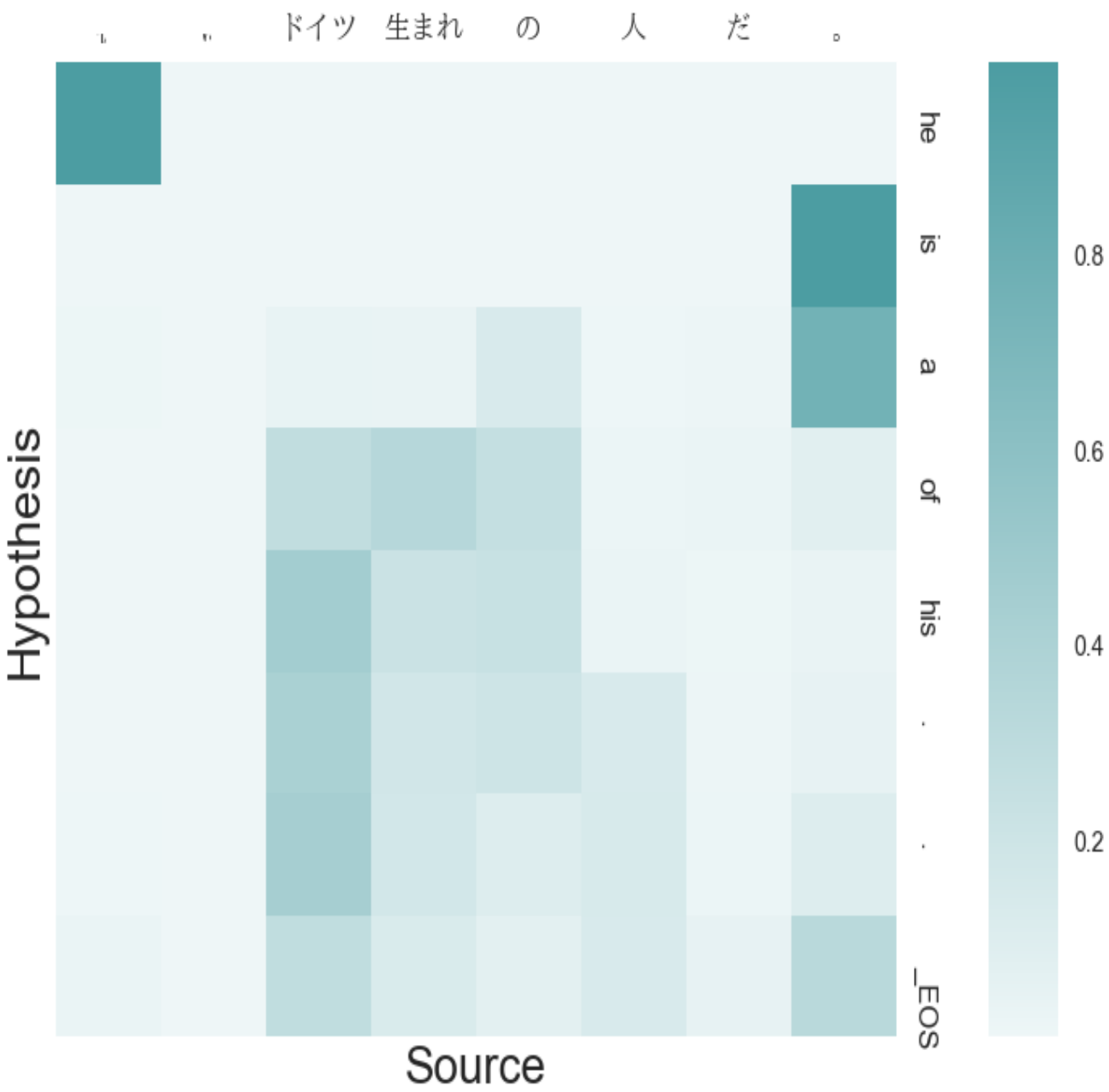

Figure 7: Q8 plot(1). Reference:"he is a german by origin ."

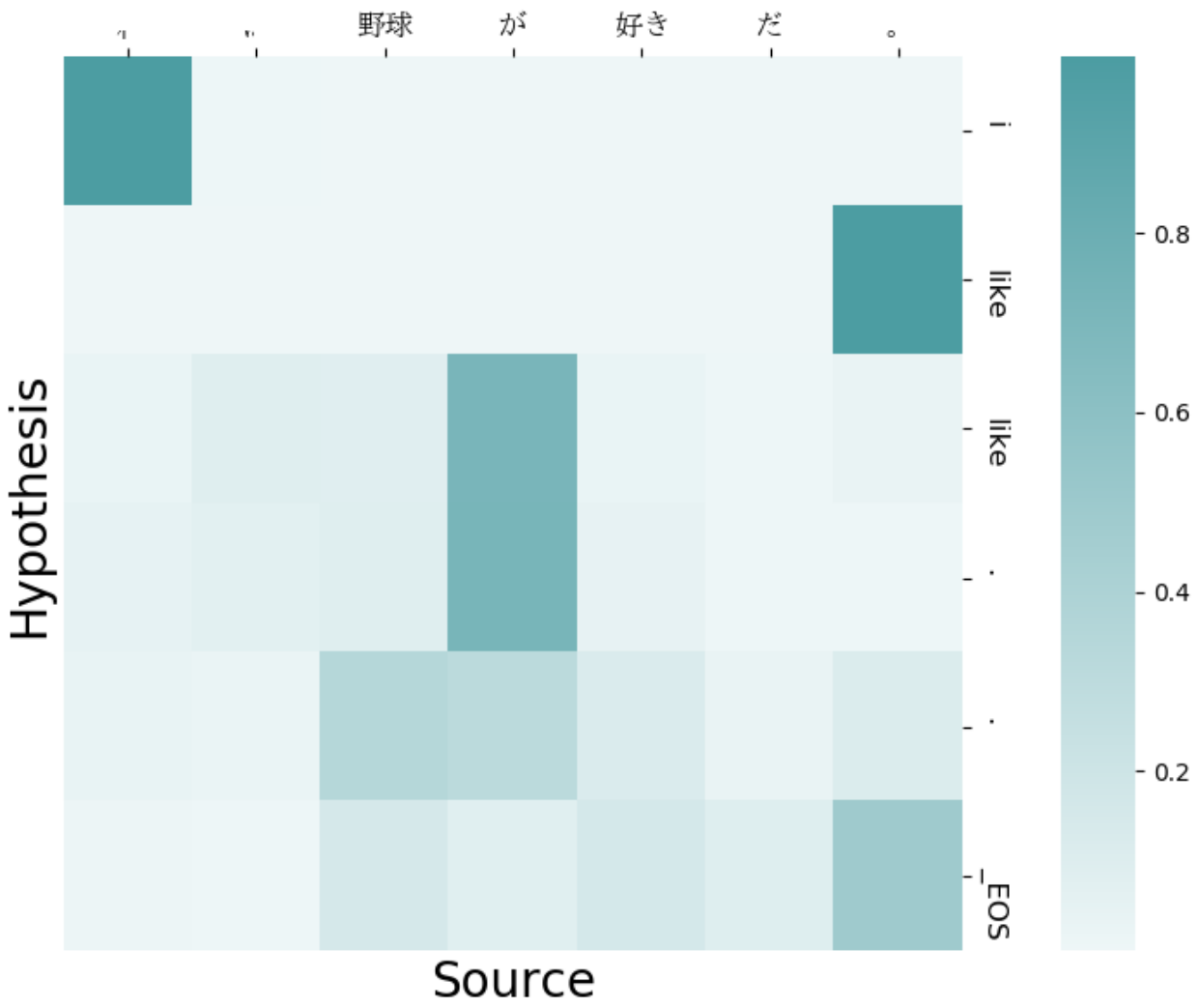

Figure 8: Q8 plot(2). Reference:"i like baseball . "

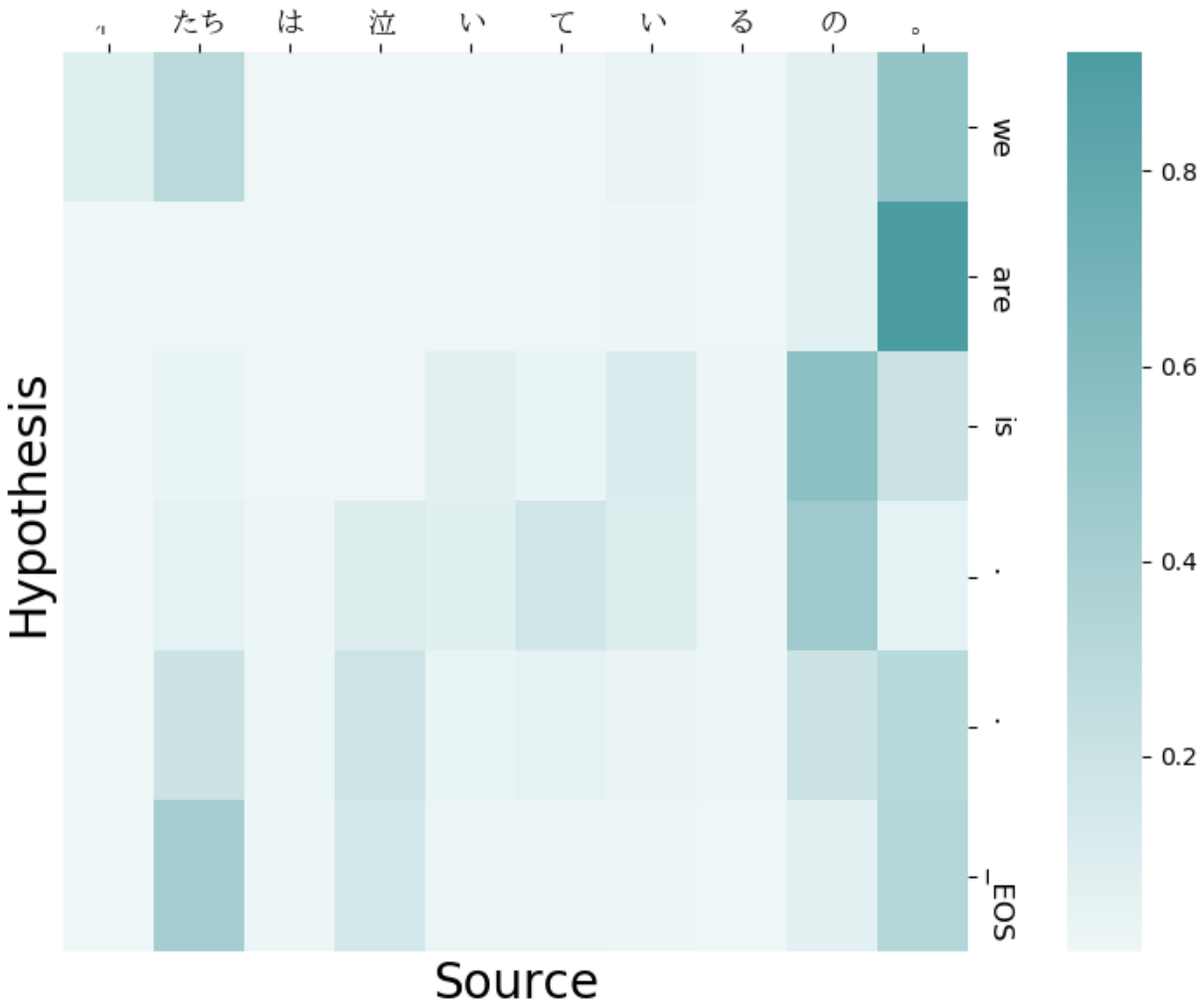

Figure 9: Q8 plot(3). Reference:" we are crying . "

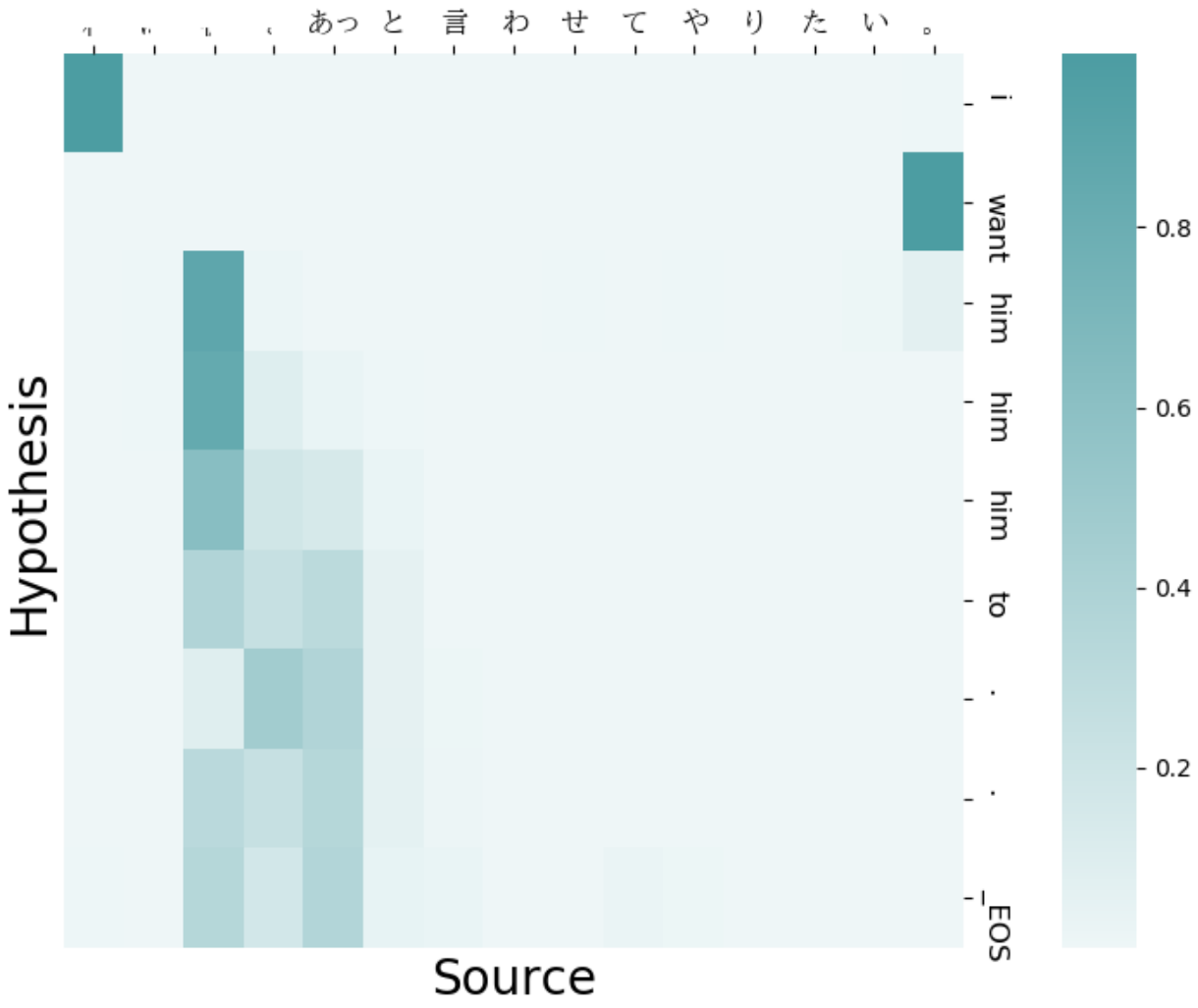

Figure 10: Q8 plot(4). Reference:"i want to surprise him . "

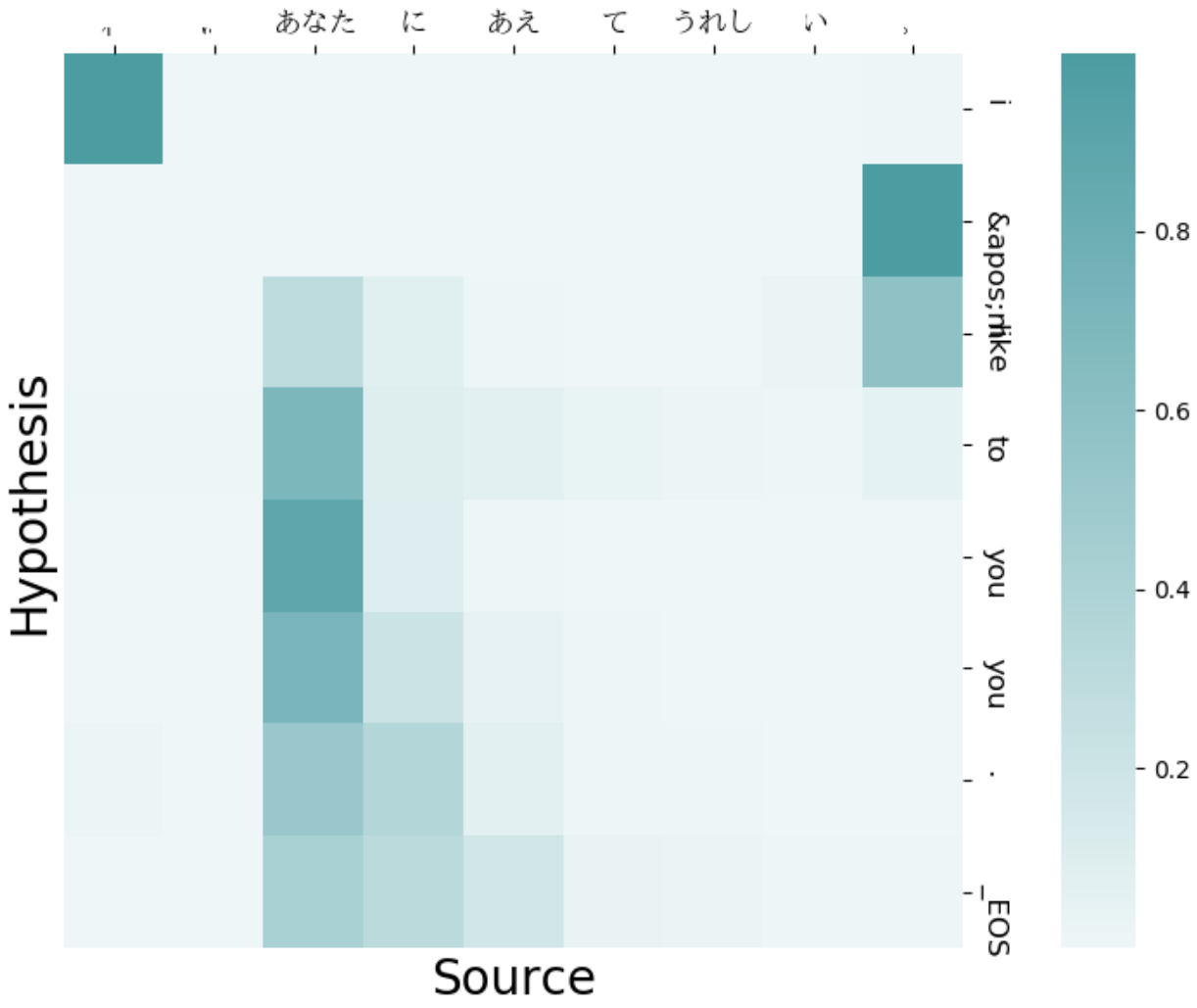

Figure 11: Q8 plot(5). Reference:”i 'm glad to see you . ”