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

A. Create `nlayers_enc` encoding layers using `add_link`. There are two loops for adding layers because we create one encoder to pass over the sentence in order, and one to pass over the sentence in reverse order.
B. The encoding of a sentence is a matrix, each column of which is the concatenation of the hidden states of the forward and the backward encoder LSTMs at that position. Since both the LSTMs are sized `n_units`, the concatenation of these two hidden states is `2*n_units`.
C. The method `set_decoder_state` will feed the encoder output into the decoder. The initial implementation takes the cell states and hidden state from the final LSTM of the encoder, and feeds them into the first LSTM of the decoder.
D. We are performing two encode operations because we are stepping through the sentence forwards and backwards at the same time – see the call to `zip` above – and we are encoding the 'forwards' word using the forwards encoder, and the backwards word using the backwards encoder.
E. Let us assume we have a vector x of activations $[0.8, 0.6, 0.3]$, with each position in the vector corresponding to a word in the output vocabulary. Let us also assume we have a scalar t, say 2, which represents the true index of the word to be generated. We take the softmax of x to get a probability distribution, and we turn t into a one-hot vector (also a probability distribution). We then compute the loss as the cross-entropy over these two distributions: $-\sum_i x(i) \log t(i)$. In our running example, this would work out to be ± 1.39. This measures how close the network got to the true answer.
F. The `add_hook` function adds some code which is to be executed right after the computation of the gradient. The `GradientClipping` function will scale gradients down when their L2 norm goes over a certain threshold (here 5).

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

1. When looking at figures 1 and 2, it seems as if English is a more productive language (more 'meaning' with fewer symbols) when we measure sentence length in tokens, and Japanese is more productive when we measure it in characters. There explanations for both these observations, which I'll go into in Q3. What our graphs do make clear, however, is that there is a linear correlation between sentence lengths in English and Japanese. This means that we can expect sentence length to change linearly when we translate, either as $|f| \approx 1.42|e|$ (tokens) or as $|f| \approx 0.42|e|$ (characters).
2. In total, there are 97643 and 143581 tokens in the English and Japanese data, respectively.
3. There are 7217 and 8246 word types/unique tokens in the English and Japanese data, respectively.
4. In the default implementation, the vocabulary size is 3713 for English and 3949 for Japanese. This means that tokens of 3505 and 4298 token types will be replaced with `_UNK`, assuming `_UNK` is part of the vocabulary. These account for 3608 (3.82%) and 4413 (3.28%) tokens in the data.
5. A vocabulary of 3713 tokens is *really* small. When we look at the most common `_UNK` words – `users protest commerce irony hats` – we see that these are still quite common. While $\pm 3.5\%$ does not sound like a lot, it will probably cripple a large number of translations with `_UNK` words. Furthermore, this will probably make `_UNK` quite difficult for the network to translate, at it will occur in huge number of syntactic positions. The differences in sentence length seems linear enough not to be a big problem.

3. (10 points) What language phenomena might influence what you observed above?

Our two measures for sentence length seem to contradict one another. Let's look at Japanese to see why:

- *High productivity using characters.* Japanese uses logograms, and each logogram corresponds to one or more syllables. Thus, each character in Japanese can carry far more information than in English.
- *Low productivity using tokens.* Japanese also uses post-positional clitics to mark e.g. subjects and objects (Hinds, 1986), which are treated as separate tokens. Many uses of these clitics, e.g. topic, subject, object, are marked in English using word order and some vestigial cases, neither of which adds tokens. Making matters worse, the tokeniser for Japanese seems to treat each hiragana as its own token, meaning long and common verb endings such as '-します' are treated as three separate tokens (3A Corporation, 1998).

4. (10 points) Answers to questions about sampling, beam search, and dynamic programming.

1. It seems that the argmax version has a bias towards generating more common words. For instance, have a look at the translation below:
   source 皮肉 な 笑い を 浮かべ て 彼 は 私 を 見つめ た 。
   reference he stared at me with a satirical smile .
   baseline i gave him him he him him him . _EOS
   The sampling version "solves" this problem to the extend that it is a lot less common. Unfortunately, this does not mean its translations are better.
   sampling he gave him from his coming he much . _EOS
   sampling i came him him to borrow him him . _EOS
   sampling i gave him him for they died job _EOS
2. How did we implement beam search for the phrase-based model? The idea was that each phrase in the output would correspond to a phrase on the input, and that these may be permuted in some way. However, our NMT model no longer *really* has this property. It simply consumes a bunch of words, and then produces out a bunch of words, with no obvious relation between an output phrase and an input phrase.
   So, what can we do? We could use beam search to search all possible output phrases. When decoding, I have access to a list of all possible next words (namely, the vocabulary) and a score for each possible next word (namely, the log of that word's probability according to the network's output distribution). A naive implementation could simply branch, trying each possible word as the next word, and scoring the generated sentences using the sum of the log-probabilities. From here, we can add the beam search heuristic, e.g. at each step, we only keep the k most promising sentences. We simply keep expanding the k best sentences until they produce an `_EOS` token (or run into the `MAX_PREDICT_LEN` setting).
3. We cannot implement dynamic programming for the above model, because we have no notion of when translations are sharing work. This was also discussed above: we don't know what phrases in the source language correspond to which phrases in the target language. We only know that the NMT model has chosen to generate some phrase. Therefore we cannot say that some partial translation is better than another partial translation which translates the same source words, because we don't know which source words a partial translation translates.
   Furthermore, discounting the `MAX_PREDICT_LEN` setting, there is no way to know when translations are done, or how long the translations will be, because we are basically waiting for the `_EOS` token to be predicted, and there is no knowing when that will happen.

5. (10 points) Experiment with changes to the model, and explain results.

For this experiment, I trained a model with a 2-layer encode, and a 2-layer decoder. I compare this models after 10 epochs of training, to the provided baseline, as well as throughout their training with two baselines which I trained specifically for this purpose. See figure 5.

This model did not do well. It managed to find new lows in its BLEU score, and new highs in perplexity. It is hard to compare the translations generated by the baselines with those generated by the new model as both are, in fact, terrible. However, some of the translations that *almost* worked under our original baseline, bear no resemblance to the reference translation under the 2-layer model:

source	宿題 は もう 終わ っ た の で す か 。
reference	have you finished your homework already ? _EOS
baseline	have you finished your homework homework ? _EOS
2-layer	where did you rain ? ? _EOS

However, the model still learns some very basic features of Japanese. For instance, it consistently translates the question particle か as a question mark, but does not translate other occurrences of か as question marks.

6. (10 points) Implement dropout, and explain the results.

For this experiment, I trained two models with dropout as described by Zaremba et al. (2014), with ratios of 0.5 (the default value) and 0.2 (the recommended value). Once again, I will compare these models to the original baseline and the two which I trained myself. See figure 5.

Dropout is a technique which randomly zeros out some connections, and therefore these two experiments might simply have been unlucky. That being said, we found that both our models using dropout perform worse on the BLEU score, losing between 4.6 and 0.26 points. As a minor consolation, both models with dropout maintain a fairly consistent perplexity throughout training, where the perplexity of the baselines skyrockets, almost doubling in the span of 10 epochs.

Once again, it is hard to compare the translations generated by the baselines with those generated by the models with dropout, as both are still terrible. However, if looking at my favorite translation, we find that again, the models with dropout mess up the only thing the baseline had going for it:

source	宿題 は もう 終わ っ た の で す か 。
reference	have you finished your homework already ? _EOS
baseline	have you finished your homework homework ? _EOS
dropout(0.5)	where did you you finish your ? ? _EOS
dropout(0.2)	where did you you a ? ? ? _EOS

Remarkably, the dropout(0.5) model *does* still generate the word "finish". Furthermore, looking at the remainder of the sentences, the dropout(0.5) model still accurately translates the question particle か. Worryingly, the dropout(0.2) model no longer translates all occurrences of か correctly. When looking at the scores, the seemingly better performance of dropout(0.5) looks like a quirk of these examples, though I have found no reason to support this decision. Really, though, both are pretty bad.

(While it is irrelevant, it is interesting to note that both the 2-layer model and the dropout model choose to translate the source using the phrase "where did you".)

7. (20 points) Implement attention. This question will be evaluated on the basis of your code.

8. (10 points) Explain the results of implementing attention.

For this experiment, I trained two models with attention, after Luong et al. (2015). One with, and one without dropout. Trusting BLEU and Zaremba et al. (2014), I chose to use a ratio of 0.2. Once again, I will compare these models to the original baseline and the two which I trained myself. See figure 5.

The attentional models are the first to best the baseline where BLEU is concerned, scoring 24.243 and 25.089 at their peaks, a good 4 points higher than the best baseline. In terms of perplexity, it seems that my decision to use dropout payed off. The perplexity of the attentional model without dropout trails off into the sky, reaching the highest score I've found with a perplexity of 60.1225 at epoch 10. For the model *with* dropout, however, the perplexity takes a dive after the second epoch, and stays low throughout, finishing on 28.5150, the lowest recorded value for 10 epochs of training.

I was sad to see that my attentional model wasn't perfect, and in fact was still rather bad. However, looking over the sentences of the attentional model, I found that it more often than the baseline generated some of the expected words, for instance:

source	丘 の 上 に 建 っ て い る 家 は とても 古 い 。
reference	the house which stands on the hill is very old .
baseline	it is well in is the the the . . _EOS
attention+dropout	house hill very on hill on hill the . _EOS

And then there is the occasional gem, which captures the spirit, if not the exact phrasing, of the source:

source	ボブ は 読 み きれ な い ほど たくさん の 本 を 持 っ て い る 。
reference	bob has too many books to read .
baseline	bob is a a a a world . _EOS
attention+dropout	bob have books books books books books . _EOS

9. (10 points) Explain what you observe with attention. (Figures may appear in an appendix.)

There is a lot of variation. For instance, in figure 6 we see that the NN had a very strong attentional link between "hotel" and ホテル (correct) and a swath of attention for 山 while generating "mountain mountain . _EOS" (sort of correct). To a lesser extend, in figures 7 and 8, we see that it got the links between 何 and "what", and 時 and "time" right. This is good to see.

Even smaller victories are the swaths of attention associating the question particle か with "?"—yet, here we also see an association with 今日, the word for today. And this is only the beginning. In figure 7, we see a relatively strong link between し and "you". However, し is part of the *verb*. Here we may once again blame the tokeniser.

Furthermore, a lot of the time, the attention is more-or-less uniform. This can be seen best in figure 7.

10. Extension

As you may have noticed, I've been griping about the quality of the Japanese tokeniser used in tokenising the corpus for this assignment. In order to test whether this actually had an effect on the quality of the learned translation model, I re-tokenised the Japanese corpus using MeCab (Kudo, 2005), and trained an attentional model on the result. See figure 5.

The plot for this experiment is labeled 'MeCab', and has been trained with attention and dropout (with a ratio of 0.2) on the re-tokenised corpus. I only ran the regular and MeCab attentional experiments once, and therefore I cannot *really* draw any conclusions on the basis of these numbers. However, it does not seem like the tokeniser was as big of a factor as I thought. The highest BLEU value for the MeCab experiment is almost identical to the highest of the regular attentional experiment, and their perplexity are comparable as well.

I will have to be content in the knowledge that this tokeniser leaves adjective like 忙しい intact, and splits verbs like 行きます more sensibly, even if that doesn't seem to be helpful.

References

3A Corporation (1998). *Minna no Nihongo I: Translation & Grammatical Notes*. Minna no Nihongo. 3A Corporation, Tokyo.

Hinds, J. (1986). *Japanese*, volume 4 of *Croom Helm Descriptive Grammars*. Croom Helm, Routledge, London.

Kudo, T. (2005). MeCab: Yet another part-of-speech and morphological analyzer. Published on http://taku910.github.io/mecab/.

Luong, M.-T., Pham, H., and Manning, C. D. (2015). Effective approaches to attention-based neural machine translation. In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 1412–1421, Lisbon, Portugal. Association for Computational Linguistics.

Zaremba, W., Sutskever, I., and Vinyals, O. (2014). Recurrent neural network regularization. *CoRR*, abs/1409.2329.

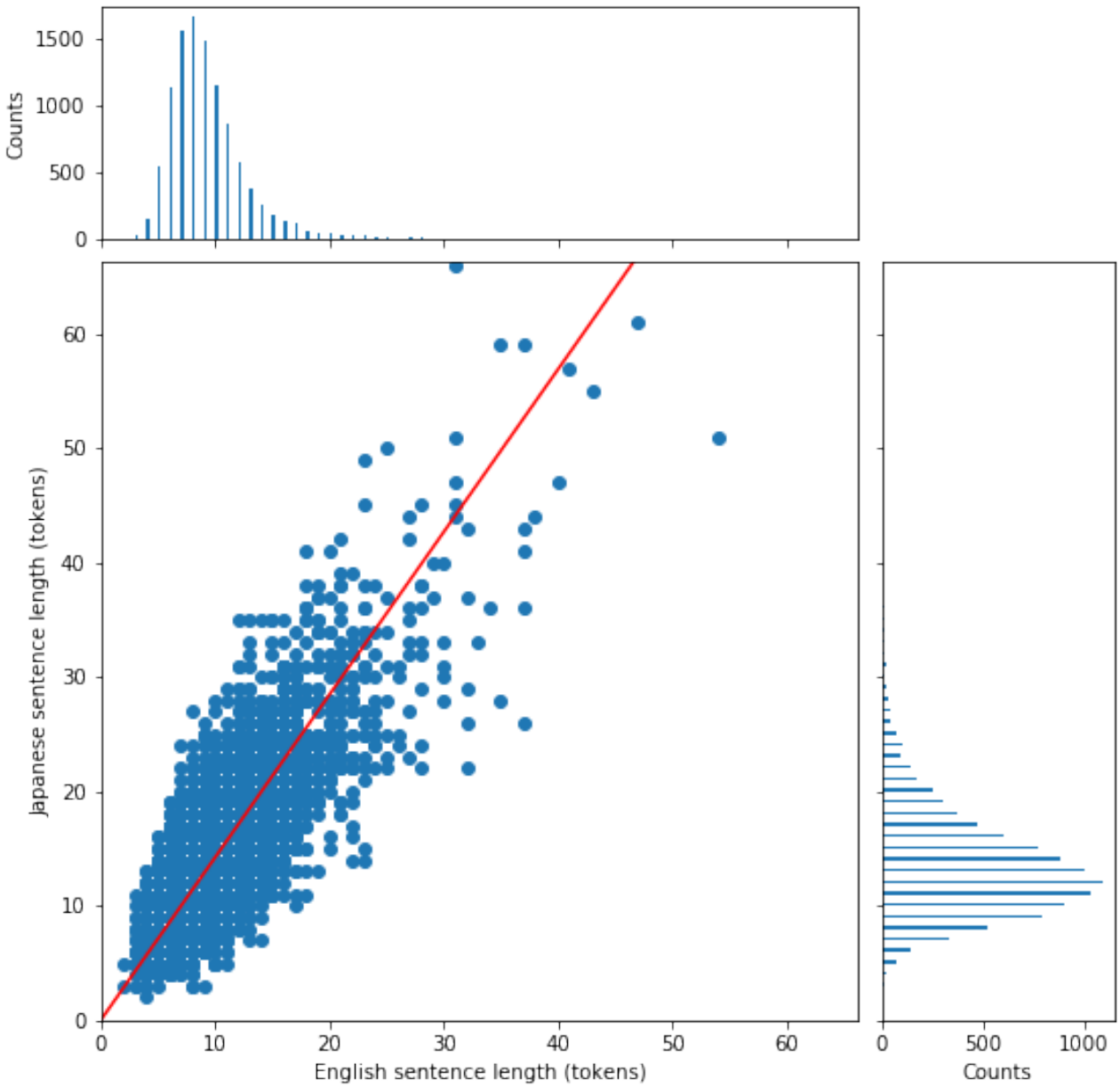


Figure 1: Distribution of and correlation between Japanese and English sentence lengths (tokens)

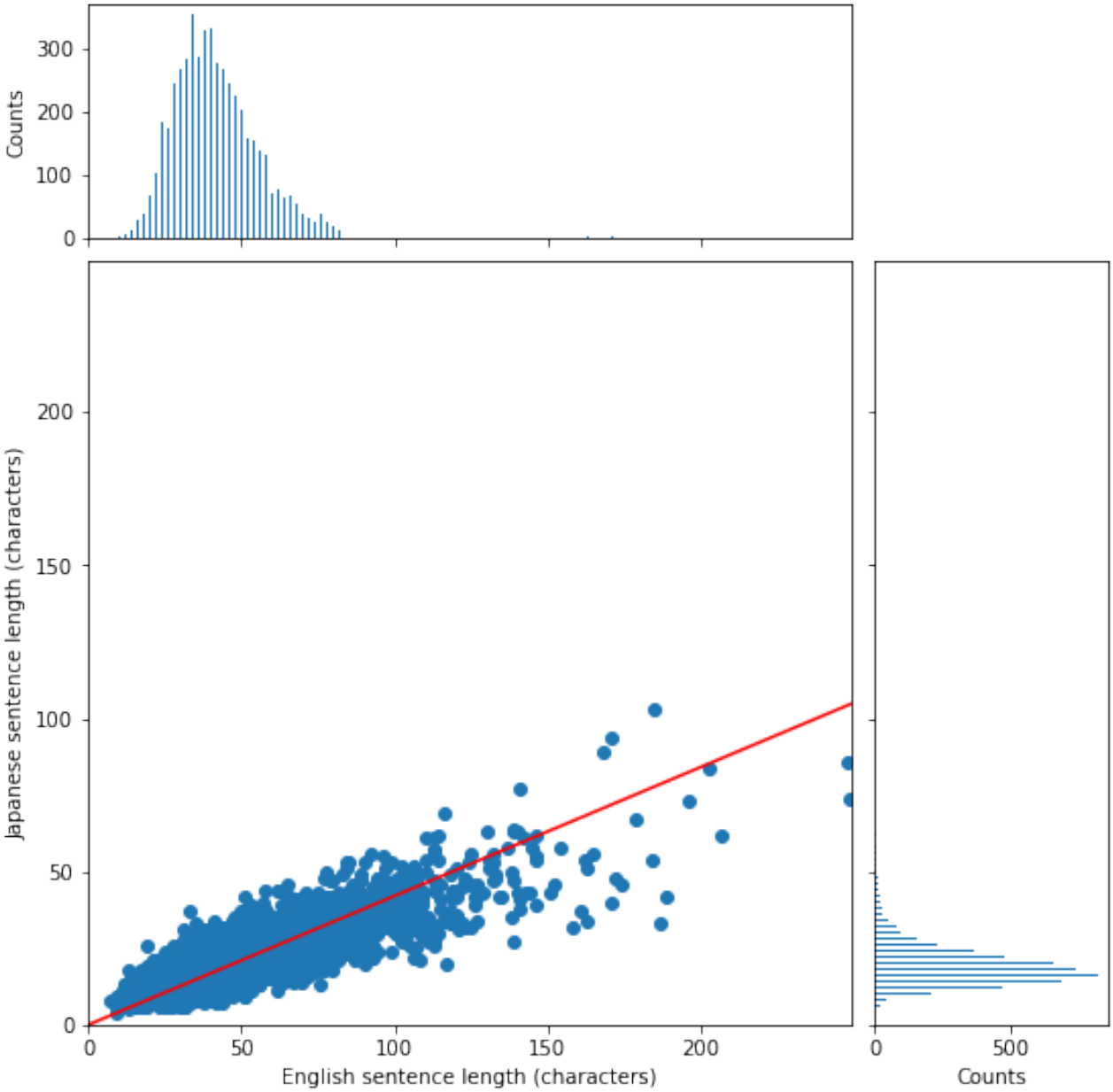


Figure 2: Distribution of and correlation between Japanese and English sentence lengths (tokens)

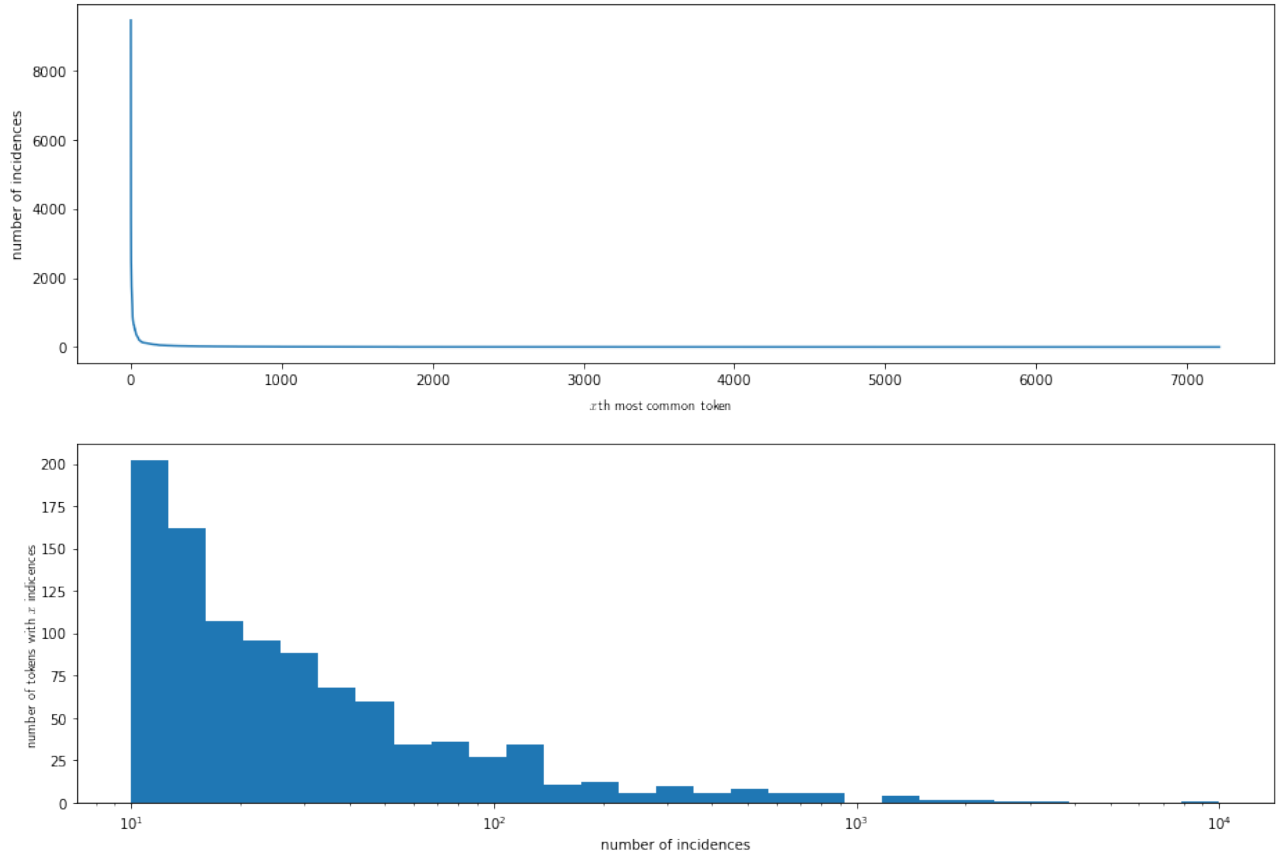


Figure 3: Counts for tokens and token types (English).

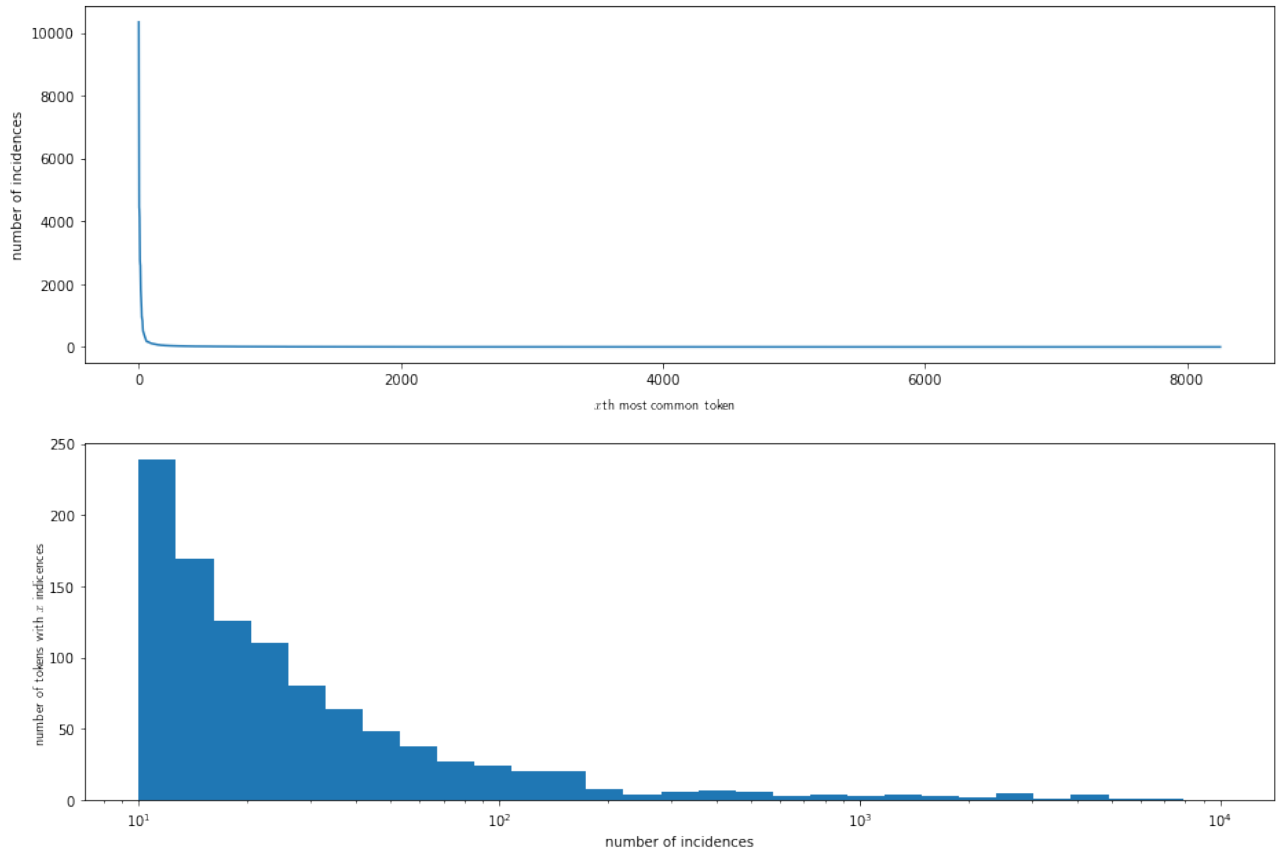


Figure 4: Counts for tokens and token types (Japanese).

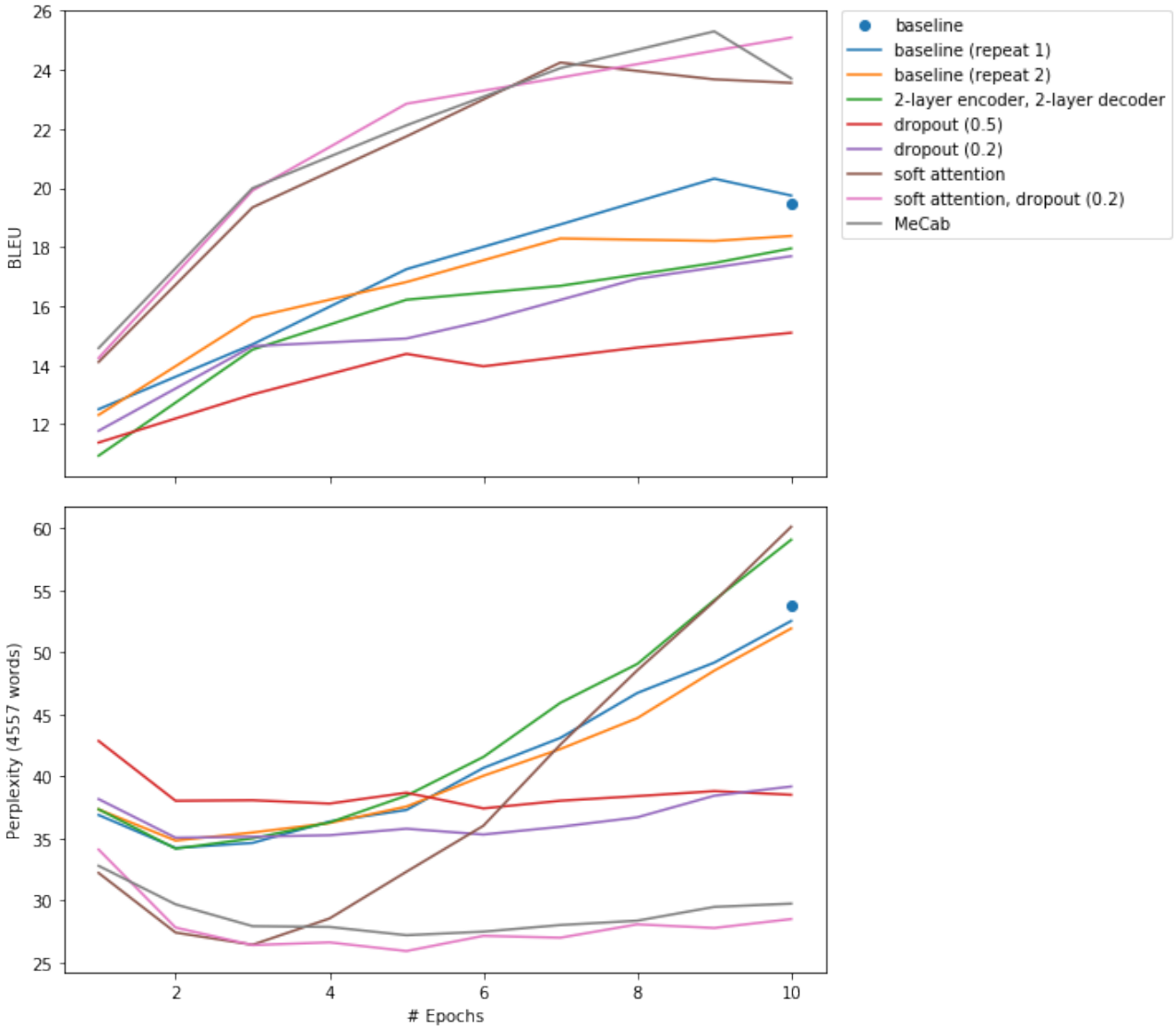
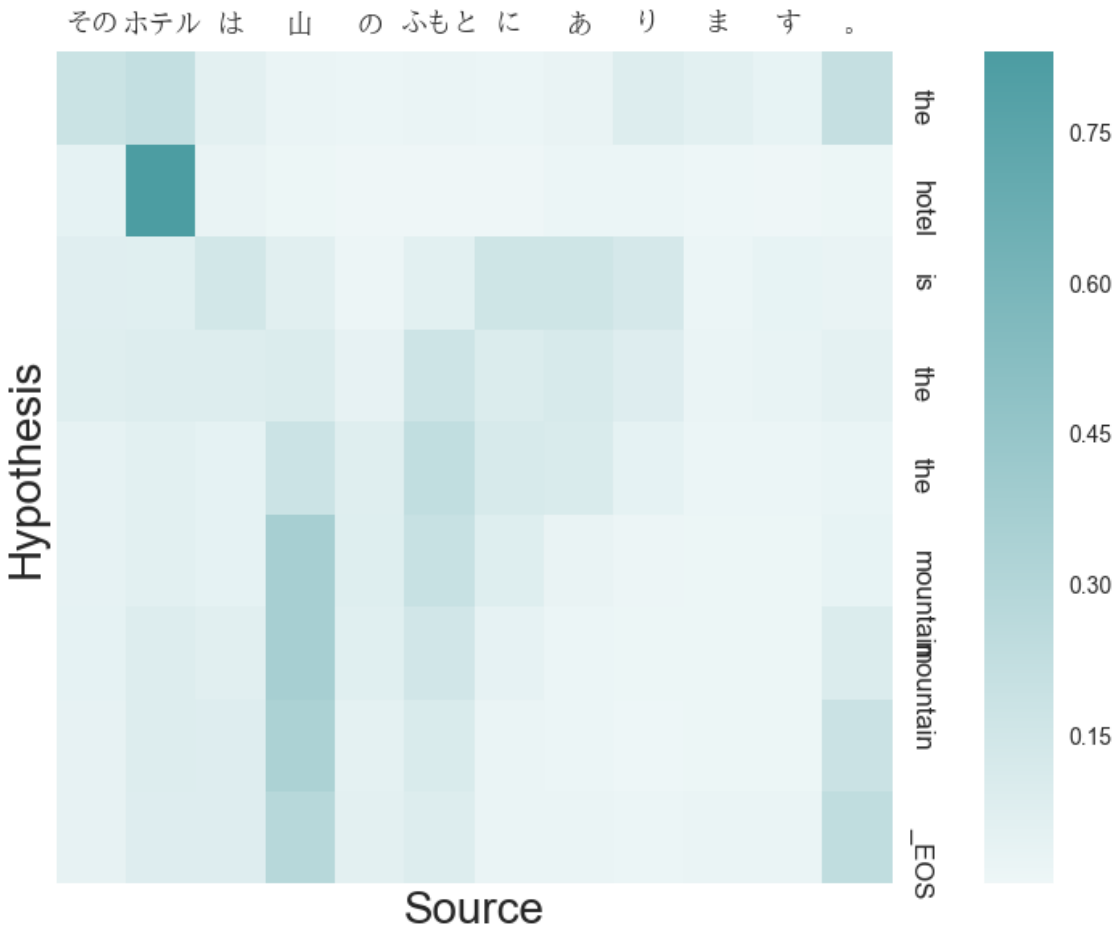
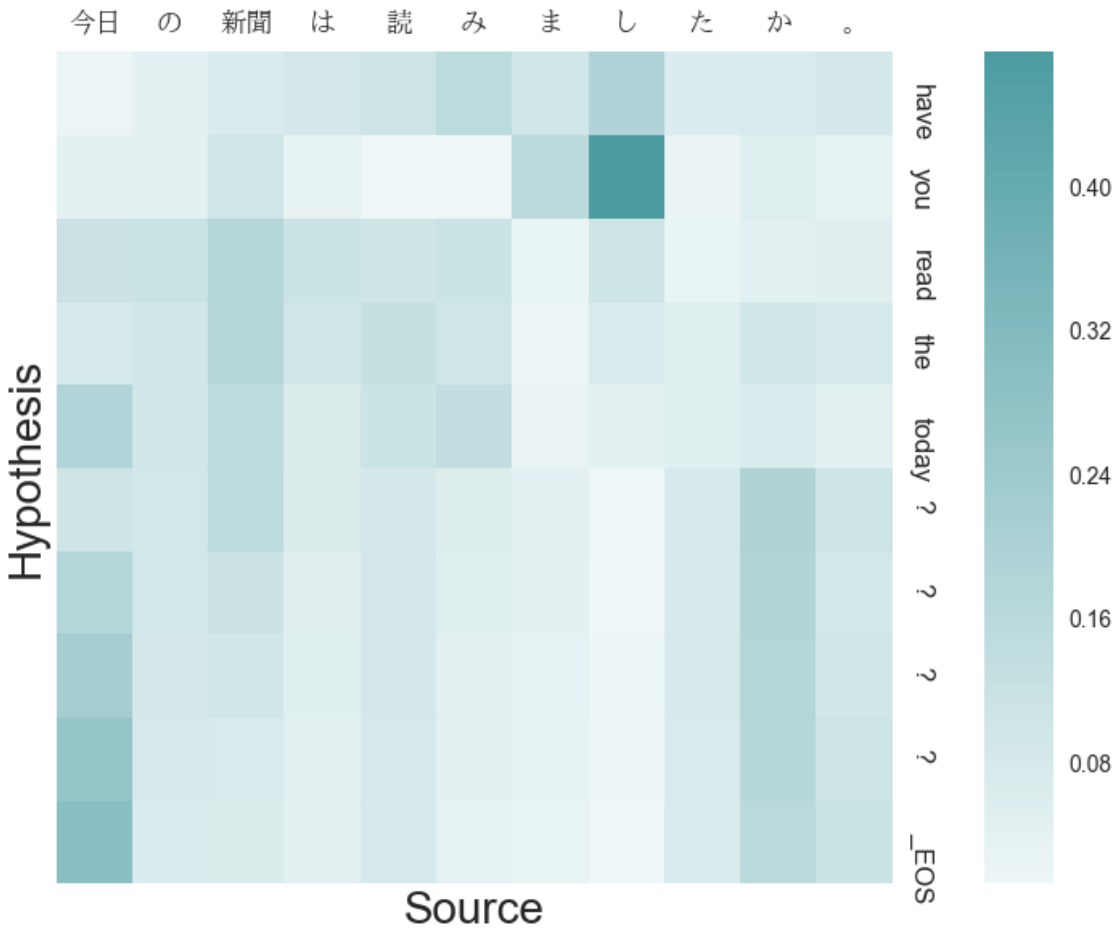


Figure 5: BLEU and perplexity scores per epoch.



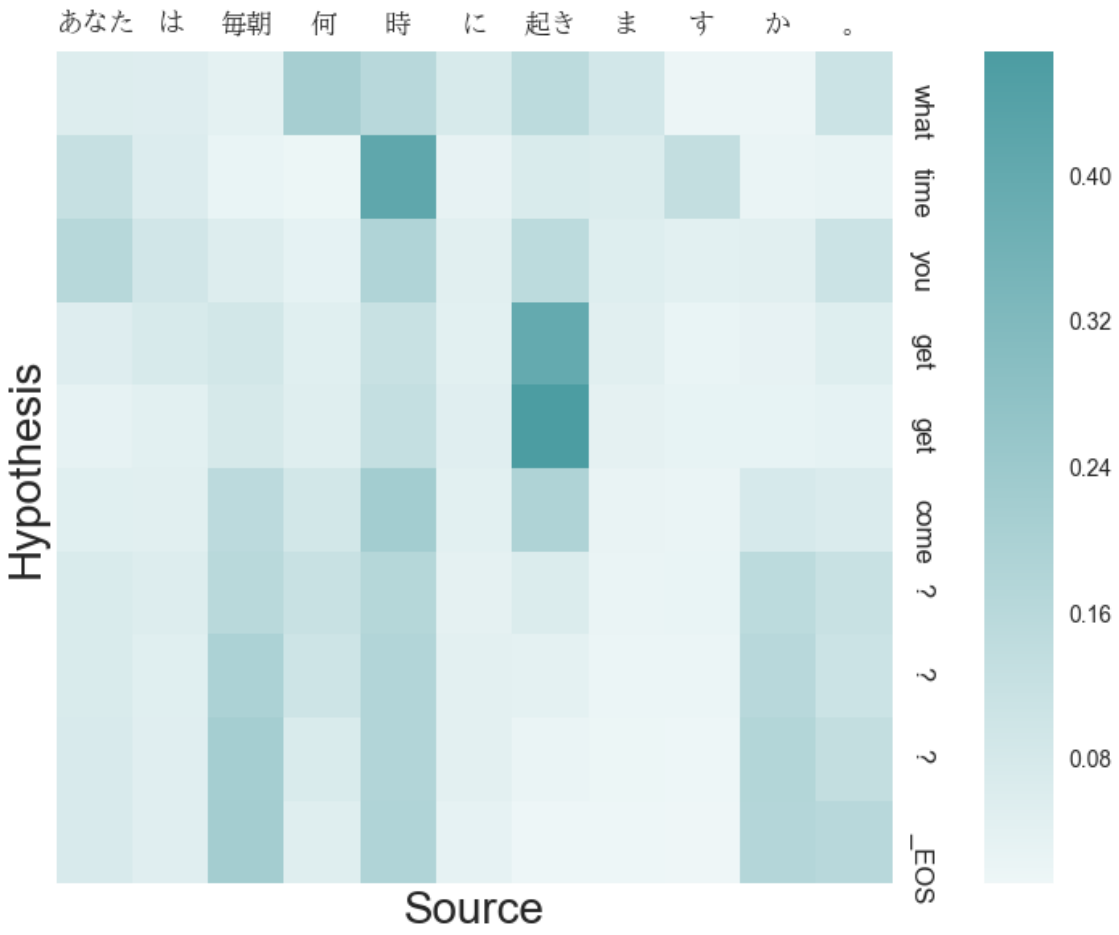
Reference: “The hotel is at the foot of a mountain.”

Figure 6: Attention plot (1)



Reference: “Have you read today’s paper?”

Figure 7: Attention plot (2)



Reference: “What time do you get up every morning?”

Figure 8: Attention plot (3)