

NLU+ Coursework 2

February 14, 2019

Question 1: Understanding the Baseline Model

Our explanatory comments:

- (A) If `self.bidirectional` is set to `True`, we use a bidirectional LSTM that processes the input sequence both forwards and backwards. In this case, the output size also doubles to twice the hidden size. To get the output back in the form we need, `combine_directions` pulls the LSTM hidden and cell state outputs out into its even and odd-indexed parts (which correspond to the forward and backwards directions) and concatenates them back together.



In an LSTM, the cell states serve as memory, encoding things like plurality and gender information that can be changed using the input and forget gates. The hidden state is calculated using the cell state and passes this information on to the next time step¹. The state of these at the final time step are respectively outputted by `nn.LSTM` as `final_hidden_states` and `final_cell_states`.

- (B) The attention context vector is the product of the calculated attention weights and the output of the encoder. The attention scores are masked because the `AttentionLayer` has a fixed width, but the sentences have variable length. The fixed sentence length is required for batching since this puts multiple sentences into the same matrix for processing. The excess scores are then masked away before the context vector is calculated.



Review detailed attention code again!!!

```
if src_mask is not None:
    src_mask = src_mask.unsqueeze(dim=1)
    attn_scores.masked_fill_(src_mask, float('-inf'))
    attn_weights = F.softmax(attn_scores, dim=-1)
    attn_context = torch.bmm(attn_weights, encoder_out).squeeze(dim=1)
    context_plus_hidden = torch.cat([tgt_input, attn_context], dim=1)
    attn_out = torch.tanh(self.context_plus_hidden_projection(context_plus_hidden))
```

- (C) The attention scores are calculated by multiplying the encoder's hidden state by a `projection matrix and transpose` multiplying by the decoder's hidden state [2]. The `projection matrix` allows for each `position` in the input sentence to be given a `weight` for each position in the output sentence, which yields an `alignment vector` once the softmax is taken.
- (D) The decoder is initialized by either retrieving an incremental state from cache or by generating previous states. `cached_state` is `None` if there is no previous incremental state for this module instance available, usually only



¹Source: <http://colah.github.io/posts/2015-08-Understanding-LSTMs/>

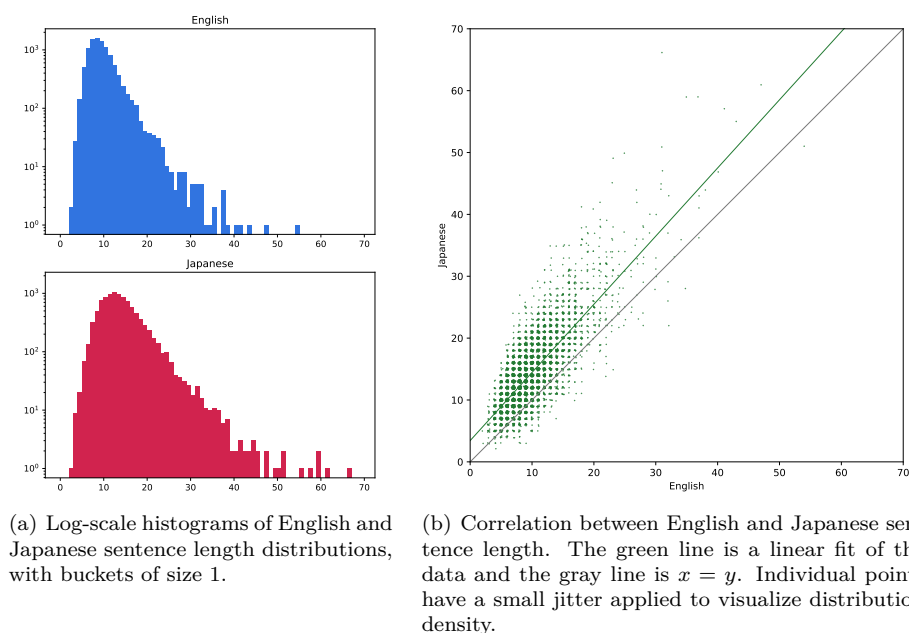




Figure 1: Plots of sentence lengths in English and Japanese training data, for Question 2.1.

true on the very first forward pass of the decoder. The role of `input_f` is to create a new matrix of zeros that shares the shape of the target embeddings, so it can be filled with the decoder's predictions.

- (E) The attention layer is integrated as nearly the last operation in the decoder's forward pass. It uses the encoder's output and the previous LSTM hidden state to compute both the attention weights and the context vector, then it concatenates and puts them under a trained projection again to form the layer outputs. Dropout is added during training to reduce overfitting, a sensible step for a jointly-trained encoder-decoder using relatively small amounts of training data.
- (F) These seven lines implement a single basic iteration of training. First a minibatch is sent through a forward pass of the model. Then, the loss is evaluated (cross-entropy in this case), backpropagated, and the gradient norm is clipped to avoid exploding gradients. Lastly, an optimization step is run (Adam in this case) and accumulated gradients are zeroed for the next iteration.

Question 2: Understanding the Data

1. See Figure 1 for plots. English and Japanese sentence lengths are **strongly positively correlated with a Pearson product-moment correlation coefficient of 0.768** (calculated using `numpy.corrcoef`). From this and the plots, we can infer that translating between the two languages is feasible: pairs of sentences are generally close to the same length (though Japanese sentences are a bit longer), which **reduces the amount of potential one-to-many and many-to-one translations, simplifying the task.** 
2. The English training set has 93,086 word tokens, including punctuation. The Japanese set has 136,899 tokens, including punctuation.
3. The English training set has 7,040 word types. The Japanese set has 5,558 types.
4. There are 3,331 unique word tokens in the English set that would be UNK'd. There are 2,673 such tokens in the Japanese set.
5. The English training set has fewer word tokens, but relatively more word types and unique word tokens. This means that English has a relatively high fraction of UNK's ($3,331/93,086 \approx 3.6\%$ vs $2,673/136,899 \approx 2.0\%$), so it may have trouble figuring out what to do with these. Since Japanese sentences are generally a bit longer, there will be many one-to-many translations from English to Japanese, which may make it difficult for the attention to figure out correct alignments. 

Question 3: Improving Decoding

1. Greedy decoding may be **problematic if higher-probability words are “hiding” behind lower-probability words**. For example, consider two possible translations: “the angry cyclops” (incorrect) and “the exasperated man” (correct). If we simply do a greedy decoding, the **former translation may be selected because “angry” is a more common word than “exasperated”** which still closely relates to the original sentence, and the fact that “man” is more likely than “cyclops” is never considered.
2. One simple way to implement beam search for NMT is the original as follows, as proposed by [3]. Going from left to right, keep B most-likely translations of the sequence up to the current time step (we call these our hypotheses). Then, at the current time step, add every possible word in the vocabulary to each of the B hypotheses. Of these new possible translations, discard all but the B best ones. Move on to the next time step and repeat.
3. Since the **probability of a sentence is determined by multiplying the probabilities of the words in the sentence, and all probabilities are ≤ 1** (but < 1

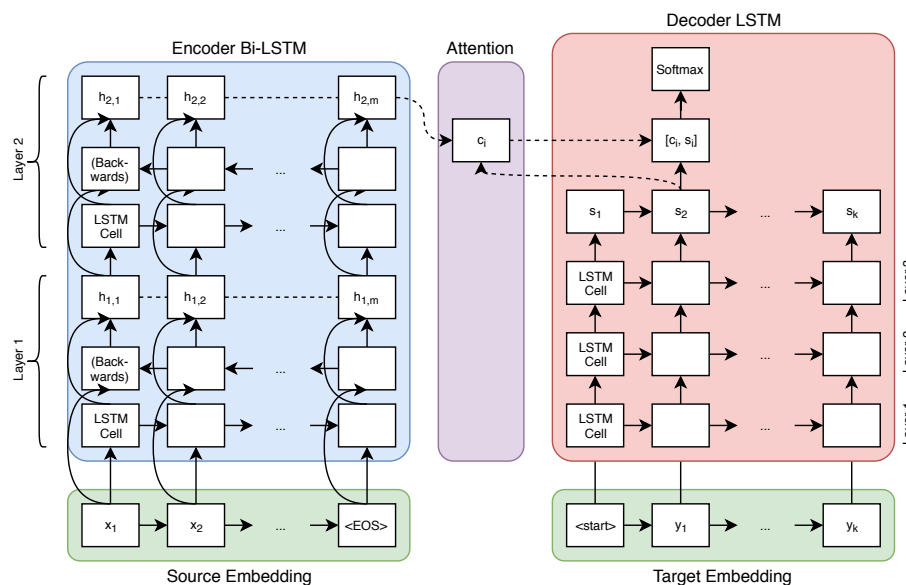


Figure 2: Diagram of the NMT system architecture with a 2-layer encoder (bi-directional LSTMs) and a 3-layer decoder (LSTMs). Figure adapted from [1].

in practice since the vocabulary has > 1 word), adding an additional word to a sentence and multiplying its probability by a number < 1 will necessarily decrease its probability. So, the decoder will favor short sentences. One problem that length normalization can introduce is that it adds another hyperparameter to the model which needs to be tuned and could result in the system producing either too short or too long translations if it is not tuned correctly.

Question 4: MOAR Layers!

1. I used the following command:

```
$ python train.py --encoder-num-layers 2 --decoder-num-
  layers 3 --save-dir checkpoints_moar_layers
```

2. See Figure 2. In the Encoder, the bottom two rows of each layer represent the forward and backward pass over the input embeddings, and the top row represents the resulting hidden states. Figure adapted from [1].
3. See Table 1 for the effect of going to a 2-layer encoder and 3-layer on dev-set perplexity, test BLEU score and training loss. Across the board, the new model does worse than the baseline, perhaps because there is not enough training data to successfully train this deeper model with more

Model	Q.	Loss (Train)	Perpl. (Dev)	BLEU (Test)
Pretrained Baseline		2.334	22.4	7.98
MOAR Layers	4.3	2.527	26.6	5.84
Lexical	6	1.745	20.0	10.90

Table 1: Performance comparison of the different architectures. The parenthesis in the header represents which part of the data that particular score was calculated on; **Q.** refers to the question corresponding to the model.

parameters. Validation (dev set) loss on the MOAR Layers model is 3.28, which is higher than the training loss, indicating that the model may be starting to **overfit** on the training set, as deeper models are in general more likely to do.

Question 6: Effects on Model Quality

See Table 1 for the lexical model’s affect on the NMT system’s performance, which is better than the baseline across the board. Training loss is lower (1.745 compared to 2.334), which **indicates** that the model is better able to fit the training data. **Perplexity** on the dev set is also lower (20.0 compared to 22.4), indicating that the lexical model has **more predictive power**. Finally, the **BLEU** score is much higher (10.90 compared to 7.98), indicating **better translation quality**. All of these may be explained by the lexical model doing its intended job: improving translations of rare words.

Question 7: Effects on Attention

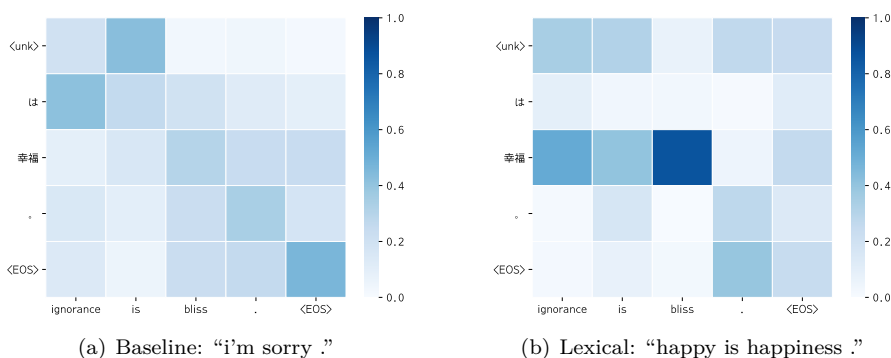


Figure 3: Attention of baseline and lexical models when translating “無知 は 幸福。” with target translation “ignorance is bliss.” Word-for-word translations from Google Translate are 無知 → *ignorance*, は → *is*, and 幸福 → *happiness*.

In general, attention in the lexical model appears more extreme: whereas attention is usually relatively evenly distributed between different source words in the baseline model, the lexical model puts more emphasis on single words when translating. This is in line with expectations from how the lexical model should work: we add the source embeddings weighted by the attention to the predictive distribution, so we can expect the attention activation on those specific source words to be higher.

This can be seen, for example, in Figure 3. In this example, the lexical model puts much higher attention on 幸福 for target word *bliss* than the baseline does. Because of this, it captures the word *happiness* in its translation, which is the top translation for 幸福 according to Google Translate. So, the attention looks reasonable.

Question 8

Because of the relatively small dataset on which both the baseline and lexical models were trained, neither produces coherent translations of long sentences. Therefore, I focus on shorter sentences in this analysis.

1. The types of sentences that will benefit most from lexical information are those where the target sentence has relatively uncommon words that can easily be swapped out for other, more common words, in a way that the sentence will stay coherent² but that its meaning will change.
2. I have collected several sentences which provide evidence for this hypothesis in Table . I also counted unique words across the different translations to English: the reference (test set) has 1,306; the baseline has 491; and the lexical model has 716.
3. This evidence supports my hypothesis. First, the unique word counts give a rough indication that the baseline has a smaller vocabulary, favoring more common words over less common ones—the lexical model has about 46% more unique words, indicating that it is less likely to fall back on common words.

²(or, at least, the level of coherent that such a small model is capable of producing)

ID	Reference	Baseline Model	Lexical Model
3	he is a german by origin .	he is a man of great ability .	he is a germany person .
14	i stopped smoking .	i let my smoking at once .	i stopped smoking .
50	he earns a good salary .	he has a good news .	he has a good salary .

Table 2: Several examples of reference translations compared to the baseline and lexical model’s translations. *ID* indicates line number in `test.en`.

The sentences in Table also support my hypothesis: in sentence 14, the **baseline** swaps *stopped* (occurs 9 times in the training data) for *let* (137 occurrences); this **produces a semi-coherent sentence** since *let* is still a verb. Similarly, for sentence 3 the baseline swaps “german” (11 occurrences) for “great” (70 occurrences)³; and for sentence 50 it swaps *salary* (5 occurrences) for *news* (43 occurrences). These examples illustrate that the **baseline favors more common, coherent-sounding words while the lexical model is able to overcome this and favor correct translations.**

References

- [1] Denny Britz, Anna Goldie, Minh-Thang Luong, and Quoc Le. Massive exploration of neural machine translation architectures. *arXiv preprint arXiv:1703.03906*, 2017.
- [2] Minh-Thang Luong, Hieu Pham, and Christopher D. Manning. Effective approaches to attention-based neural machine translation. *CoRR*, abs/1508.04025, 2015.
- [3] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pages 3104–3112, 2014.

³The baseline’s translation here is actually 100% memorized; the exact phrase it produces exists in the training data.