IFT6289-H23 Assignment 2: Neural Machine Translation with LSTM-based Seq2Seq Model

February 22, 2023

- *Due Date:* ***March 10, 2023 (23:59 pm, EST timezone)***
- *Assignment 2 should be completed by individuals, which is worth* ***15%*** *of your grade.*
- *Maximum* ***2*** *late days. Late submissions would result in a lower grade.*
- *Please Submit your code(.zip) and report (PDF) on Studium.*

Instruction

In this assignment, you are going to implement the Seq2Seq with Attention model for Neural Machine Translation (NMT), published by Luong et al., 2015[1]. Then, you will test your model and analyze the results on the IWSLT 2014 Germen-English dataset (Ranzato et al., 2015).

Note: Although the dataset we use is rather small, training an NMT model would still require lots of computation, so we would recommend you to train your model on a GPU or TPU. If you cannot access a machine with GPU or TPU, we recommend you to use Google Colab to run this assignment, since it provides free GPU resources. It could take as long as 2 to 3 hours to train your model on a GPU, so please plan you time in advance.

Part 1: Implementing the Luong et al. Seq2Seq model

In this part, you will implement the Seq2Seq with Attention model proposed by Luong et al., 2015 based on **the notations and equations from the original paper** and **the comments provided in the codes**. The comments we provided contain detailed explanations of each step you should make.

In the class, we learnt that Luong et al.'s model used uni-directional LSTM for the encoder, but we would use a bi-directional LSTM instead to let you have a better understanding of how LSTM-based models work. Note that writing explainable comments for your code not only good behavior for coding but also will be easy for grading.

1. (1pt) Implement the `evaluate_ppl()` function in `nmt.py`.
   In general, perplexity is a measurement of how well a probability distribution or probability model predicts a sample. In the context of NLP, perplexity is one common way to evaluate language models, which is the normalized inverse probability of the target sentences:

$$PP(W) = P(w_1 w_2 ... w_N)^{-\frac{1}{N}}$$

[1]Minh-Thang Luong, Hieu Pham, and Christopher D Manning. 2015. Effective Approaches to Attention-based Neural Machine Translation.

Hint: we can simply evaluate the perplexity using the loss from the model.

2. (1pt) Implement the `input_transpose()` function in `utils.py`.
   In order to apply tensor operations, we have to ensure that the sentence lengths in a given mini-batch are the same. In this function, we need to identify the longest sentence in a mini-batch, and pad other sentences with the special `pad_token` to the same length. You'll also need to transpose the sentences for the output (which means exchange the dimensions of the output list of lists).

3. Implement the `__init__()` function in `nmt.py` file.

   (a) Step 1: Initialize the source and target embedding (`self.src_embed` and `self.src_embed`). These are for the different vocabularies for the source and target languages.

   (b) Step 2: Initialize the neural network layers for your Seq2Seq model. This includes the LSTMs for the encoder and decoder, the linear layers for attention computation or decoder state initialization, the readout layer for the decoder's output, and the dropout layer.

4. (1pt) Implement the `encode()` function in `nmt.py`.
   The encoder function contains three steps:

   (a) Step 1: Construct the input to the encoder using the source embedding.

   (b) Step 2: Apply bi-LSTM for the sequence, and get the encodings for all time steps, as well as the final state and final cell.

   (c) Step 3: Compute the initial state and initial cell for the decoder, using the final state and final cell from the encoder.

5. (1pt) Implement the `decode()` function in `nmt.py`.
   The decoder function contains four steps:

   (a) Step 1: Apply the attention projection layers to make sure the representations from encoder and decoder are the same.

   (b) Step 2: Construct the input to the decoder using the target embedding.

   (c) Step 3: Iterate over time. In each time step, compute the attention vector and the state & cell for the next time step.

   (d) Step 4: Stack the output hidden vectors of each time step from the decoder into a single tensor.

6. (1pt) Implement the `step()` function in `nmt.py`.
   This function performs a single time step in the decoder. It involves the attention computation.

7. (1pt) Implement the `dot_prod_attention()` function in `nmt.py` file.
   This function computes the dot attention in Luong et al.

   (a) Step 1: Compute the attention scores.

   (b) Step 2: Apply softmax to obtain the normalized attention scores.

   (c) Step 3: Obtain the context vector using the normalized attention scores and the source encoding.

After you finish the coding part, you can train your model on the IWSLT 2014 German-English dataset.

Part 2: Analyzing Seq2Seq and NMT models

1. The attention mechanism has become one of the most crucial part of modern deep learning models, and the attention mechanism was first proposed to be part of Seq2Seq models for NMT tasks in Bahdanau et al.

(a) (1pt) In this class, we have seen two different types of attention mechanisms. The first is the ones we see in Seq2Seq models like the one we just implemented; the other is the self-attention mechanism in Transformers. Recall that we have the terms "query", "key" and "values" for self-attention. What are the query, key and value for the *dot* and *general* alternatives in Luong's attention?

(b) (1pt) The computation of the attention score can be regarded as similarity computation between the query and key. Based on your answer of the previous subquestion, what kind of assumption do *dot* and *general* attentions make about the encoder's and decoder's hidden vector space? Does Bahdanau's attention (the *concat* alternative) make a stronger or weaker assumption compared to the other alternatives?

2. For NMT models, BLEU score is the most commonly used automatic evaluation metric. It is usually calculated across the entire test set, but now we consider BLEU defined for a single example. Suppose we have a source sentence $\mathbf{s}$, a set of k reference translations $\mathbf{r}_1, \ldots, \mathbf{r}_k$, and a candidate translation $\mathbf{c}$. First, we compute the *modified n-gram precision* p_n of $\mathbf{c}$, for each of $n = 1, 2, 3, 4$:

$$p_n = \frac{\sum_{\text{ngram} \in \mathbf{c}} \min\left(\max_{i=1,\ldots,k} \text{Count}_{\mathbf{r}_i}(\text{ngram}), \text{Count}_{\mathbf{c}}(\text{ngram})\right)}{\sum_{\text{ngram} \in \mathbf{c}} \text{Count}_{\mathbf{c}}(\text{ngram})} \tag{1}$$

Here, for each of the n-grams that appear in the candidate translation $\mathbf{c}$, we count the maximum number of times it appears in any one reference translation, capped by the number of times it appears in $\mathbf{c}$, which is in the numerator. We divide this by the number of n-grams in $\mathbf{c}$, which is in the denominator.

Next, we compute the *brevity penalty* BP. Let $len(c)$ be the length of $\mathbf{c}$ and let $len(r)$ be the length of the reference translation that is closest to $len(c)$ (in the case of two equally-close reference translation lengths, choose $len(r)$ as the shorter one).

$$BP = \begin{cases} 1, & \text{if} \quad len(c) \geq len(r), \\ \exp\left(1 - \frac{len(r)}{len(c)}\right), & \text{otherwise} \end{cases} \tag{2}$$

Lastly, the BLEU score for candidate $\mathbf{c}$ w.r.t. $\mathbf{r}_1, \ldots, \mathbf{r}_k$ is:

$$BLEU = BP \times \exp\left(\sum_{n=1}^{4} \lambda_n \log p_n\right) \tag{3}$$

where $\lambda_1, \lambda_2, \lambda_3, \lambda_4$ are weights that sum to 1. The log here is natural log.

Suppose we have four reference texts for a single instance:

- Ref1: *the gunman was shot to death by the policeman*
- Ref2: *policeman killed the gunman*
- Ref3: *the gunman was shot dead by the policeman*
- Ref4: *the gunman was shot to death by the policeman*

And the output translation from the model is: *gunman is shot dead by the policeman*

In the following questions, we assume that $\lambda_1 = \lambda_2 = \lambda_3 = \lambda_4 = 0.25$.

(a) (2pt) Please compute the BLEU score for the output translation above. For this question, please show your computation steps, including the results of $p_1, p_2, p_3, p_4, len(c), len(r)$ and BP. In this question, please scale your result BLEU score to use the **0 to 1** scale.

(b) (1pt) Suppose that we **only have Ref1 and Ref4 in our dataset**, and the model's output remains the same. Compare the BLEU score in this setting with the one for the subproblem 2a. What is your observation and what conclusions do you make?

(c) (1pt) Suppose that we have **another tokenizer** that splits the sentence into subwords instead of words. Under this tokenizer, the word *gunman* would be separated into two subwords *gun* and *###man*, and the word *policeman* would be separated into two subwords *police* and *###man*, and everything else remains the same. **Under this scenario, all the references and the output would be affected.** which means,

- Ref1: *the gun ###man was shot to death by the police ###man*
- Ref2: *police ###man killed the gun ###man*
- Ref3: *the gun ###man was shot dead by the police ###man*
- Ref4: *the gun ###man was shot to death by the police ###man*
- Output: *gun ###man is shot dead by the police ###man*

Compare the BLEU score in this setting with the BLEU score computed for the subproblem 2a. What is your observation and what conclusions do you make?

Part 3: Training your Seq2Seq model

In this part, you will analyze your results and indicate your findings. Please refer to `train.sh` file to find the commands to run your code. If you are using Linux, you can run all the

1. First, please run `vocab.py` to extract a vocabulary file from the training data using the "vocab generation" command given in `train.sh`.
2. Second, train the model using the "training" command given in `train.sh`. This will save the best performing model parameters in `work_dir`.
3. Third, run decoding on the test set using the "decoding" command given in `train.sh`. This will generate `decode.txt` in `work_dir`, which contains the decoded outputs of your model. You can manually compare your results with the ground-truth outputs in `data/test.de-en.en`.
4. (3pt) Finally, run the "bleu evaluation" command given in `train.sh`. This will call the official evaluation script `multi-bleu.perl` to compute the corpus-level BLEU score of the decoding results against the gold-standard. This perl script is a widely used BLEU evaluation script, which can be found in libraries like Fairseq or OpenNMT. However, note that the evaluation provided by `multi-bleu.perl` can be affected by the choice of tokenizers, so without using a proper detokenizer, it is only suitable for internal evaluation. To run this command, you will have to install perl first. Note that if you are using Windows, this command might not work with PowerShell, but you can run this command using command prompt, git bash or WSL. **Please provide a screenshot of the multi-bleu.perl's output in your report, and describe your results and observations.**
5. (1pt) (Brainstorming) Training our NMT model requires datasets containing language pairs or parallel data, which means that each input should be mapped to its corresponding translation in the dataset. However, not all languages are resource-rich, and there still exists a huge amount of languages that does not have abundant parallel data. What can be a possible solution to make full use of a small amount of parallel data? How would the large amount of unparalleled corpus of the language or even corpus from other languages benefit to our objective?

Part 4: Beam Search

1. (2pt) Given the output probabilities of the model for each step, which define a sequence of 5 words over a vocab of 4 words ('a', 'b', 'c', 'd') in Table 1, please write the 2 most likely sequences and their probabilities by using Beam Search algorithm.
2. (1pt) The sentences generated by Beam Search are usually very similar. Is there a way to enlarge the diversity of the sentences generated by Beam Search? (Brainstorming)

Table 1: Probabilities outputted by the model.

	‘a’	**‘b’**	**‘c’**	**‘d’**
Step 1	0.2	0.1	0.3	0.4
Step 2	0.4	0.3	0.1	0.2
Step 3	0.2	0.1	0.4	0.2
Step 4	0.3	0.4	0.2	0.1
Step 5	0.2	0.35	0.35	0.1

Writing your report

1. The template of the assignment 2 report is the same as that for the reading assignment, which can be downloaded from StudiUM or slack.
2. Please indicate in your report if you used any open-source codes or materials.

Submission Instructions

Submit your PDF report and code (.zip file) in on Studium. When submitting the code, please zip all files in your project, instead of zipping the project’s root folder.