Humboldt-Universität zu Berlin
Institut für Informatik
Prof. Dr. Alan Akbik

Berlin, 31.05.2024
Lehrstuhl Maschinelles Lernen

Introduction to Natural Langauge Processing

Exercise 6

Submission: Hand your homework until 10.06.2024 (Monday), 23:59 via Github Classroom. Each exercise is worth a total of **10 points** and you need **80%** (of total points over all exercises) to be admitted to the final exam. Sometimes, there are optional tasks that will give you extra points. Exercises in Github Classroom should be completed and submitted individually. However, we encourage you to work on the problems in teams. Please note the information available in Moodle: `https://hu.berlin/nlp24-moodle`.

Github Classroom: `https://classroom.github.com/a/wSNnb2Q4`

Task 1 (Seq2Seq model) — 1 + 2 = 3 points

In this task, you will be working on an implementation of a sequence-to-sequence model. As you recall from the lecture, they are widely used in tasks where the input and output are sequences, potentially of different lengths. One prominent example of such a task is machine translation.
You will be provided with a file `sequence_to_sequence.py`, in which the model class `Seq2Seq()` is implemented. You will need to:

(a) Complete the class constructor. The names of the class objects are already given, so all you need to do is to correctly instantiate them with appropriate PyTorch modules. This will be important in Task 3.

(b) Implement the `translate()` method of the Seq2Seq model in `sequence_to_sequence.py`. This method allows for inference with a trained Seq2Seq model. With the source sentence:

- Compute the final hidden state of the encoder via `encode()` method
- Using the logic from the `decode()` method, pass this hidden state & a special token `<SEP>` to the decoder, return the decoder outputs and the updated hidden state, and compute the log probabilities for the next token
- Choose the next token based on the returned log probabilities

Continue generating the translation either until the model reaches `max_symbols()` or the token `<STOP>` is chosen as next token.

When choosing the next token, implement two options:

- Greedy decoding: the token with the highest probability is always chosen next
- Multinomial sampling: the token is sampled from the distribution over all tokens in the vocabulary (refer to Exercise 5 LSTM Code for hints)

Task 2 (BLEU Score) **3 + 1 + 1 = 5 points**

In this task, we will be implementing BLEU (Bilingual Evaluation Understudy) score calculation from scratch. BLEU score is a metric for evaluating the quality of text which has been machine-translated from one language to another. It compares the machine-translated text to one or more reference translations and provides a score between 0 and 1, where 1 indicates a perfect match with the reference translations. It is defined as:

$$\text{BLEU} = \underbrace{\min\Big(1, \exp\big(1 - \frac{\text{reference-length}}{\text{output-length}}\big)\Big)}_{\text{brevity penalty}} \underbrace{\Big(\prod_{i=1}^{n} precision_i\Big)^{1/n}}_{\text{n-gram overlap}}$$

You are provided with the file `bleu_score.py`, in which the function `corpus_bleu_score()` calculates the BLEU score for a set of translations (e.g. in a test dataset). You need to implement:

(a) The function `count_lengths_get_precision()`, in which you:
- Go through the list of candidate and reference translations.
- Calculate lengths of (tokenized, i.e. split on whitespace) candidate and reference translations. When considering the length of the reference translations in cases, where there are multiple reference translations, choose the translation that is nearest to the candidate one in terms of the length (amount of tokens).
- Calculate modified precision in a given candidate-reference(s) pair for each n-gram level up to `max_n` using `modified_precision()` function. This function returns the numerator and denominator for the final precision calculation.
- After iterating over the candidate-reference pairs, calculate the final precision score for each n-gram level. In cases when the denominator `p_n` is 0, the precision score for that n-gram level should be set to 0.

(b) The function `get_ngram_overlap()`, in which you calculate the geometric mean of precision scores.

(c) The function `brevity_penalty()` that penalizes generated translations that are too short compared to the reference length with an exponential decay.

Task 3 (Seq2Seq Evaluation) **2 points**

In this task, we will be evaluating already trained `Seq2Seq` models. You need to download the `state_dicts` of those models from here and put them into folder `models/`. You are also provided with a sample of translation pairs in `data/translations.txt`. You will need to evaluate sequence-to-sequence models using the `translate()` method from Task 1 and your BLEU score implementation from Task 2:

- Correctly instantiate the sequence-to-sequence models using the `Seq2Seq()` implementation from Task 1 and load the provided `state_dicts` into the models. You can find info on how to do so here.

  Note: pay attention to the dimensions of the initialized class objects!

- Load the sequences in the source (first column) and target (second column) languages.

- Generate the translations for the sequences in the source languages using your loaded models.
- Calculate the BLEU score for your generated translations.
- Report your results in the `results.txt` file.

Refer to the `eval.py` script for the evaluation setup. Ideally we want to see BLEU scores for each of the models provided using two translation decoding strategies (greedy and multinomial), i.e. at least 6 scores in total.