Beam Search for Text Generation

In this assignment, you'll explore text generation and machine translation. The main programming task for this assignment is implementing the *beam search* decoding algorithm for text generation. You'll use beam search to generate text with GPT-2 (a precursor to ChatGPT that you use can on a CPU). You will also evaluate the effectiveness of the BLEU score method for machine translation evaluation.

Learning Goals

Once you complete this assignment, you should:

- Deepen your understanding of decoding methods for text generation
- Appreciate the effectiveness and limitations of automated methods for machine translation evaluation

This assignment is connected to the following overall learning goals of the course:

- Be familiar with NLP methods in three key areas: text classification, text generation, and language understanding
- Be able to effectively use python libraries that are part of the large ecosystem of tools for NLP
- Explore various NLP applications, including applications with positive societal impact

Implementation Details

For this assignment, you will implement the beam search algorithm and test your implementation using a GPT-2 model. Beam search is a particularly useful method for machine translation, but to ensure that you understand the input/output text to your model (and that I don't need to write examples in many different languages), you'll implement it for plain-old text generation.

You can use beam search to generate text with GPT-2 by writing less than 5 lines of python, but that will not help you to develop an understanding of how the algorithm works. That means that using `pipeline("text-generation")`, `model.generate`, or similar methods is not allowed in your `decoding.py` implementation.[1] If you aren't sure if a method is allowed, you can ask me; generally, you should find that you need to write very little huggingface/pytorch code, as you can rely on the provided `next_token_probs` function.

Here are a few important details:

1. You should stop generating text once you have generated `max_new_tokens` tokens. It is important to note that this number should be less than 1024 - the length of your prompt (GPT-2 can only handle 1024 tokens). **You should do this rather than waiting until an EOS token is generated.**
2. The textbook discusses a length normalization method that can be used with your scores. **You should not implement this** – just use the probability itself (or better yet, the log probability). In practice, all of your sequences will be the same length so this won't matter.

Resources

Starter Code

util.py

[1] I **strongly** encourage you to write your own code that uses these methods to test your implementation.

`next_token_probs` This function returns a dictionary where the keys are the most probable tokens given a prompt and the values are the probabilities (or log probabilities if `log=True`). Here's an example:

```
next_token_probs(tokenizer.encode("The sun is shining"),
                                        model, 2, log=True)
```

Returns

```
{319: -2.0777230262756348, 11: -2.183382511138916}
```

Where 319 is the token ID for `" on"` and 11 is the token ID for `","`.

decoding.py

`generate_greedy` This is my implementation of greedy decoding using the `next_token_probs` function. Make sure that you understand if fully before you try to implement beam search, which will be more complex. This demonstrates that you don't need to interact directly with the model - the function in the starter code will do that for you.

test_mini.py Much like the `test_mini.py` scripts provided with Homework 2 and Homework 3, the `test_mini.py` script tests your `beam_search` function on the examples that we saw in class.

Data & Pre-Trained Models

Data

Machine Translation The Machine Translation datasets (used for the report) are from the Tatoeba Challenge, e.g., English-Spanish.

Models

Text Generation The main generation model for this assignment is GPT-2.

Machine Translation You won't use any machine translation models directly, but the translations for the report are created with the models released by the University of Helsinki's NLP group.

Deliverables

Code

The main deliverable for this homework assignment is your `beam_search` function in `decoding.py`. While not required, I strongly recommend writing a helper function for the `add_to_beam` function that is shown in slp (See Figure 13.10). I also recommend writing a class to represent your states in your beam search.

Possible Extensions (OPTIONAL)

There is no leaderboard for this assignment (largely because the evaluation makes the most sense for translation, but it would be difficult to compare results across languages). However, there are lots of things you could do as extensions if you are interested!

More efficient huggingface code I intentionally sacrificed some efficiency in this code to give you a simple `next_token_probs` function that doesn't require you to interface with pytorch/huggingface internals. However, there are some clear inefficiencies here (like tokenizing and decoding tokens every single time the function is called). There is no batching, and we are not using the `past_key_values` which can speed up your search.

If you want to get more practice with huggingface and pytorch, I'd suggest trying to improve some of these inefficiencies.

Sampling You implemented beam search, but not any of the sampling approaches that we talked about. Try implementing top-k, top-p, and/or temperature sampling (this should not be written in the beam search function that is graded).

Machine translation If you'd like to experiment more with machine translation, I'd suggest experimenting with BLEU using different values of n. Check out the OPUS-MT models from the Helsinki NLP group that are linked in this assignment. For efficiency, I'd recommend using the huggingface generation functions rather than yours for this experimentation.

Report

In addition to your code, you must fill out the report in `report.md` (from the starter code). Note that the report focuses on machine translation, and asks you to do some human evaluation of machine translation. The translations are provided for the following languages:

- Arabic
- Chinese
- French
- German
- Hebrew
- Italian
- Portuguese
- Russian
- Spanish

If you are not comfortable evaluating text written in any of these languages, I encourage you to work with a partner who is or ask a friend for help! If there is another language that you'd like to see on this list, contact me ASAP and I can try to add it (it depends on a model and test set being available).

As a last resort, I've added an English option that uses paraphrases rather than translations. The paraphrases were generated by using *backtranslation*, specifically, translating from English to Arabic and then back to English.