CSCE-489, Programming Assignment #3 Text Generation
Due: Sunday, April 27, 2025 by 11:59pm

The goal of this homework is to deploy and use pre-trained language models to perform text generation and more. Specifically, we will adopt the GPT-2 model introduced in the following paper.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. "*Language models are unsupervised multitask learners.*" OpenAI blog 1, no. 8 (2019): 9.

We will use the implementation and pre-trained model from the `transformer` library by Hugging-Face. This project requires you to figure out the usage of "tokenizers" and "models" by reading documents and examples. We will need to get familiar with the Hugging-Face APIs. Below are some useful resources for the documents and examples.

- The transformer API documents: `https://huggingface.co/docs/transformers/index`.
- Descriptions and examples of the GPT-2 model: `https://huggingface.co/gpt2`.

The starter code consists of three files. You will implement your code for all tasks below in the `model.py` file, and run your code with `generation.py` and `sentiment.py` for text generation and sentiment analysis tasks below, respectively. You will need to install the following libraries to let the starter code work:

- transformer: `https://huggingface.co/docs/transformers/installation`.
- datasets: `https://huggingface.co/docs/datasets/installation`.
- PyTorch or TensorFlow: Follow instructions on individual websites. Modify Line 14 in `model.py` to match the specific framework no matter which you use.

1 Text Generation Basics

(2 Points!) Task 1: Generation - Your first task is to implement the `self.generate_text()` method under the `TransformerModel()` class. Given the first few words of a sentence or paragraph (dubbed "**prompt**"), the `self.generate_text()` method completes sentence or paragraph by generating the following words. The implemented method will further be used in some following tasks. In your report, please include three examples you generated from different prompts here and discuss the quality of the generated sentences in terms of grammatical correctness and semantic cohesiveness (**1 point**).

(2 Points!) Task 2: Better generations - The next step is to tune your implemented generation method by modifying parameters in the `transformer` generation APIs, *e.g.*, length, searching techniques, and temperature. A full list of parameters can be found at

`https://huggingface.co/docs/transformers/main_classes/text_generation`.

Test your tuned generation method with 3–5 examples (generated from different prompts) and briefly discuss them. You can write your example in the `prompt.txt` file. In your report, please also include three examples you generated here and discuss the quality of the generated sentences in terms of grammatical correctness and semantic cohesiveness (**1 point**).

(3 Points!) Task 3: Evaluation - Quantifying how good are the text generation results is a key to evaluating the performance of a pre-trained language model. This task is for you to understand and implement one of the most common evaluation metrics, **perplexity score**, for text (sequence) generation. Read the definition and examples here:

`https://huggingface.co/docs/transformers/perplexity`

and implement the `self.evaluate_ppl()` method under the `TransformerModel()`. Given the model and the test dataset, compute and return the perplexity score. In your report, report your reproduced perplexity score on the WikiText dataset, briefly explain what is perplexity in your own words (**0.5 point**), and answer what range its value can be within (**0.5 point**).

2 Sentiment Analysis as Text Generation

Next, we will explore further the potential of pre-trained language models. In particular, we will see how a language model for text generation can perform sentiment analysis of given documents. We will continue to work on the IMDB dataset we used in the previous project.

(3 Points!) Task 1: Few-shot predictions - A well-trained language model can perform sentiment analysis by generating the word 'positive' or 'negative' given only a few examples, as long as the prediction is formulated well as a generation task. Concretely, to formulate the sentiment analysis as text generation, we need to define a template to construct the prompt. Below is an example of the constructed prompt.

Review: I like this movie
Sentiment: Positive
###
Review: I regret watching it
Sentiment: Negative
###
Review: The scene is impressive
Sentiment: Positive
###
Review: It feels boring
Sentiment: Negative
###
Review: The movie can be better in its presentation
Sentiment:

The language model will then take the above prompt and generate the word "positive" or "negative" consecutively following "Sentiment:". Here the first four reviews come from the training set and the last review is for the model to predict. When appropriate examples and templates are chosen, the model can perform good predictions even without further training or finetuning.

In this task, you need to try different templates and examples for the model to make good predictions.

1. Modify the example template in the `self.get_template()` method under the `TransformerModel()` class.
2. Select up to four (in total, pos and neg) examples numbered [000-899] from the IMDB dataset you worked with in PA2, and put them in corresponding folders under directory `fewshot_examples`. You can **trim the review text** and keep only a few important sentences by editing example files if needed. Submit the folder together with your code.
3. Debug your code with `--debug` mode. The code will run on 10 test examples and print incorrect examples. Test your chosen template and examples and repeat steps 1–2 to refine the results. Sometimes the model predicts something other than "positive" or "negative". In this case, trying some other template may help.
4. Once done, remove `--debug` and run the testing with all 200 examples. Report your results in both debugging and non-debugging modes.

[Extra credits] (2 Points!) Task 2: Visualization - Transformers are attention-based models. Visualizing the attention scores can be one good approach to understanding how a transformer works on a specific example. In this extra task, you will utilize the "output attention" feature when calling the GPT-2 model by setting `output_attentions=True`. Implement the `self.visualize_attention()` method under `TransformerModel()` to return the weights (importance) of each input token. Specifically,

1. You can choose the attention scores from any layer, any head, any query token, and normalize the score if necessary to obtain the most reasonable visualizations.
2. Also make sure the length of the returned prompt matches the length of weights. Hint: the length of weights is always equal to the number of tokens in the sequence.

Submit the "explained.html" file and include a screenshot in your report (see Figure 3 for example).

[Extra credits] (3 Points!) Task 3: Fine-tuning - The GPT-2 model can be further improved by the fine-tuning with task-specific examples. Use the same template and examples in Task 1 to construct training and test samples. At this time, different samples are considered as individual prompts instead of being joined together. Implement the `self.finetune()` method under `TransformerModel()` to fine-tune parameters in `self.model` and perform prediction (code provided in `sentiment.py`) on test samples. Report your results after fine-tuning, compare them with the results in Task 1, and discuss the results.

Python Starter Code

You are encouraged to use Python for the programming assignments in this class, but if you prefer, you can use Java too. However, simple starter code will be provided in some of the programming assignments (including this one) and only in Python.

In the same directory where the provided code and data are, you can run the code:

```
python generation.py [--max M] [--num N]
# to generate sentence(s) given prompt.txt
# ‘M’ is the max number of generated new words, default to 10
# ‘N’ is the number of sentence you generate, default to 1

python generation.py --ppl
# to compute perplexity score

python sentiment.py [--debug] [--length L]
# to perform few-shot sentiment prediction
# enable debugging mode with --debug to run on 10 examples
# ‘L’ is the max length for each document, default to 100

(optional) python sentiment.py -x --debug [--length L]
# to perform few-shot sentiment prediction with explanations

(optional) python sentiment.py --tune [--length L]
# to perform fine-tuning and prediction
```

The functions (methods) you need to work on is mentioned in the task descriptions.

GRADING CRITERIA

Your program will be graded all based on mainly the correctness of your implementations (please refer back to the first section for how we will weight each task), and the validation results on the test examples (positive and negative reviews from [900–999]). We understand that you may produce slightly different results if you have implemented some details in a different way. Therefore, please describe enough details in your written report. In addition, we will test your code on a separate test set to make sure it runs properly. If we found any experimental result has been hard coded, you won't get any credit for that part.

ELECTRONIC SUBMISSION INSTRUCTIONS
(a.k.a. "What to turn in and how to tun in")

You need to submit 2 things:

1. The source code files and example folders for your program. Be sure to include all files that we will need to compile and run your program!

2. A report file that includes the following information:
    - how to compile and run your code
    - results and analysis
    - any known bugs, problems, or limitations of your program

    REMINDER: your program **must** compile and run. We will not grade programs that cannot be run.

How to turn in:

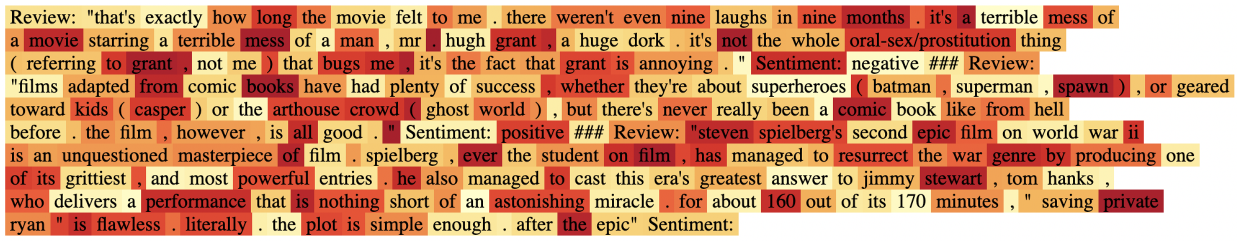
1. please submit your report as a pdf file, and include everything else in a single .zip package, and submit them via canvas.

```
Task 1: Sentence generation
===============

Text generation is the task of generating text with the
goal of with an output placeholder.

Text generation is the task of generating text with the
goal of with another output placeholder.
```

Figure 1: Example Output for Generation

```
negative file no. 997 got: Review: ”the kids in the hall
are an acquired taste ... Sentiment: positive

negative file no. 998 got: Review: ”there was a time when
john carpenter was a ... Sentiment: positive

negative file no. 996 got: Review: ”john boorman’s ” zardoz ”
is a goofy cinematic ... Sentiment: positive

negative file no. 995 got: Review: ”if anything , ” stigmata ”
should be taken ... Sentiment: positive

negative file no. 999 got: Review: ”two party guys bob their
heads to haddaway’s dance ... Sentiment: positive

Fewshot accuracy: 0.50
```

Figure 2: Example Output for Sentiment Analyzer



Review: "that's exactly how long the movie felt to me . there weren't even nine laughs in nine months . it's a terrible mess of
a movie starring a terrible mess of a man , mr . hugh grant , a huge dork . it's not the whole oral-sex/prostitution thing
(referring to grant , not me) that bugs me , it's the fact that grant is annoying . " Sentiment: negative ### Review:
"films adapted from comic books have had plenty of success , whether they're about superheroes (batman , superman , spawn) , or geared
toward kids (casper) or the arthouse crowd (ghost world) , but there's never really been a comic book like from hell
before . the film , however , is all good . " Sentiment: positive ### Review: "steven spielberg's second epic film on world war ii
is an unquestioned masterpiece of film . spielberg , ever the student on film , has managed to resurrect the war genre by producing one
of its grittiest , and most powerful entries . he also managed to cast this era's greatest answer to jimmy stewart , tom hanks ,
who delivers a performance that is nothing short of an astonishing miracle . for about 160 out of its 170 minutes , " saving private
ryan " is flawless . literally . the plot is simple enough . after the epic" Sentiment:

Figure 3: Example of explanation results.