**CS-461, Fall 2025**
**Homework #2: Question Answering with Transformers**
**Due Date: Monday, November 10th @ 11:59PM**
**Total Points: 10.0**

**Description:** Question-answering with transformer-based architectures is the dominant paradigm in natural language processing. Among the various formulations of this task, multiple-choice question-answering is among the most tractable. In this assignment, you will perform question-answering on the OpenBookQA dataset (see https://arxiv.org/abs/1809.02789). OpenBookQA is a multiple-choice question-answering dataset of elementary school-level scientific questions. Questions are posed in natural language with four possible choices. Background knowledge in the form of a one-sentence "fact" is also provided.

This task's primary formulation requires you to identify the correct "fact" from among 1,326 facts. To focus on transformer architectures in this assignment, you are given the "fact" associated with each question. The training dataset consists of approximately 5,000 observations; the validation and test sets each contain 500 observations. Each observation includes a question stem, the associated fact, four labeled choices and a correct answer label. You may ignore all other information.

You will use transformer-based architectures for multiple-choice question-answering, including classification and generative approaches. Starter code is provided to read the JSON dataset files. You may need to install Huggingface Transformers (see https://huggingface.co/docs/transformers/installation) on your machine.

**Tasks:**

1. **Classification Approach (5 pts):** For this approach, you must use the bert-base-uncased checkpoint for the BertModel (see https://huggingface.co/docs/transformers/v4.26.1/en/model_doc/bert) and fine-tune the model on the training set using your own code. Implementations using BertForMultipleChoice or other specialized variants will not be accepted. The format of the text input to BERT and the classification methodology for this task are up to you. One approach is to leverage the [CLS] token, which can be implemented as:

    a. Encode each instance as `[CLS] <fact> <stem> <text of choice #1> [END]`

    ```
    [CLS] <fact> <stem> <text of choice #2> [END]
    [CLS] <fact> <stem> <text of choice #3> [END]
    [CLS] <fact> <stem> <text of choice #4> [END]
    ```

    b. Retrieve the context embedding at the top of the transformer stack for the `[CLS]` token and take the dot product with a $[d_{mod} \times 1]$ linear layer.
    c. Treat the resulting vector as logits, apply the softmax and compute a multiclass cross-entropy loss over the four choices.
    d. Fine-tune BERT the complete dataset with this encoding for multiple epochs.
    e. Use the trained model to perform inference on the validation and test sets.
    f. Report accuracy.

    Please describe your approach in sufficient detail so that someone familiar with transformers can replicate your results. Also, report your zero-shot and fine-tuned accuracies on the validation and test sets. Please discuss any limitations of your approach and possible solutions. Using the approach described, I achieved an accuracy of 55.9% on the test set.

2. **Generative Approach (5 pts):** For this approach, I provide a simple decoder-only transformer model pre-trained on the WikiText-103 dataset. You can run the model from a Linux command line with:

```
python starter.py -d_model 768 -n_layers 12 -heads 12 -seqlen 512 -
loadname pretrain -dir_name hw2
```

You will know if you loaded the pre-trained weights correctly if you get an perplexity of 34.6 on the Wikitext-103 test set.

You must design a prompt and output format to perform this task. The design will include a methodology for interpreting the generated output to evaluate model performance. Note: The generation may be a single letter to designate a multiple-choice answer. One approach is to:

a. Encode each instance as `[START] <fact> <stem> [A] <text of choice #1> [B] <text of choice #2> [C] <text of choice #3> [D] <text of choice #4> [ANSWER] <correct label A, B, C, or D>`
b. Fine-tune your model on the complete dataset with this encoding for multiple epochs. (Note: Using the loss on the last token only may boost performance.)
c. Use the trained model to generate a label following the [ANSWER] token on the validation and test.
d. Use exact match to calculate the accuracy of your approach.

Please describe your approach in sufficient detail such that someone familiar with transformers can replicate your results. Also, report your zero-shot and fine-tuned accuracies on the validation and test sets, and provide examples of your prompts and outputs. Please discuss any limitations of your approach and possible solutions. Discuss the performance of your generative approach compared to your classification approach. Using an exact-match performance metric based on A, B, C, or D encoding, I can achieve an accuracy of 36.2% with the provided pre-trained model.

**What to Submit:** Please submit a single .zip file for the group. The zip file should contain the following:

- A single .pdf for all written responses and results.
- Individual .py files (two files maximum) for your Classification and Generative approaches.

Only one group member needs to submit.