EECS595: Natural Language Processing
Homework 3, Fall 2023
Due 10/18/2023

Student Name: Rishikesh A. Ksheersagar — uniqname: RISHIKSH

You will have the choice to use Great Lakes, a shared computing cluster at UMich, or your own machine/Google colab for this assignment.

Submission Guidelines

1. Please insert your student information in line 60 of this LaTeX file;
2. Please insert your answers between each pair of `\begin{solution}` and `\end{solution}`;
3. For programming problems (explicitly marked as `Programming`, please complete the python programs as instructed.
4. Zip the files and submit to Canvas. Checklist: `hw3.pdf`, `finetune.py`, and `prediction.torch`.

Problem 1: Transformers and Pre-trained Language Models

In this programming assignment, you will fine-tune a pre-trained language model to perform multiple choice question answering. For this assignment, the goal is to get you familiar with pre-trained transformers in HuggingFace. You are free to use any available pre-trained language model in the BERT family (*e.g.*, RoBERTa [2], etc.) for this assignment. You are also free to use any help you find online. **However, you cannot directly copy the code from online resources; you need to write your own code.**

Dataset

You will be using the AI2 Reasoning Challenge (AI2-ARC)–Challenge dataset, which can be downloaded from the official homepage or directly on HuggingFace. The format of the data is straightforward. Each data instance contains 4 fields:

- `id`: a string feature;
- `question`: the question;
- `choices`: a dictionary, containing 4 choices, each of which is a string and corresponding labels

- `answerKey`: the correct key for the question.

An example looks like this:

```
{
    "answerKey": "B",
    "choices": {
        "label": ["A", "B", "C", "D"],
        "text": ["Shady areas increased.",
                 "Food sources increased.",
                 "Oxygen levels increased.",
                 "Available water increased."]
    },
    "id": "Mercury_SC_405487",
    "question": "One year, the oak trees in a park began producing
                    more acorns than usual. The next year, the
                    population of chipmunks in the park also
                    increased. Which best explains why there were
                    more chipmunks the next year?"
}
```

Computation

NLP experiments are commonly performed in a Linux-based shell environment. As such, you are encouraged to use Great Lakes, a Linux-based computing cluster based at UMich, for this assignment. You can also choice other computing resources, such as Google Colab or you own machine. Our course Great Lakes account is `eecs595f23_class`.

The linked documentation (above) from ARC-TS is excellent and should be sufficient to get you started. You can find additional tips and tricks on using Great Lakes here and here. The following resources may also be helpful in running your code on Great Lakes.

- **VPN.** To access Great Lakes when not on campus, you will need to use the UMich VPN. For information on setting up the VPN, refer here.
- **Great Lakes dashboard.** ARC-TS provides a convenient online interface for using Great Lakes, and it should be sufficient for the reproducibility study. Access the online dashboard here. The following pages will be useful:
  - *Home Directory.* You can access a GUI file manager for your home directory by navigating from the main dashboard to `Files` > `Home Directory`. All your files for the replication should be downloaded within your home directory, i.e., `/home/uniqname`.
  - *Terminal.* To open a terminal in your home directory, navigate from the main dashboard to `Clusters` > `_greatlakes Shell Access`. This will allow you to run commands as specified in your paper's code documentation. You can alternatively SSH into the cluster from a local terminal (see documentation).
  - *Job Composer.* To submit a GPU-dependent job to the cluster, navigate from the main dashboard to `Jobs` > `Job Composer`. This page provides an intuitive GUI for submitting

Slurm jobs. You MUST use Slurm jobs to run your training and evaluation code; the cluster will kill jobs that run for too long or consume too much memory in the terminal.

- **Slurm.** Great Lakes uses Slurm to manage all computing jobs. The Job Composer page introduced above provides an intuitive interface to start and stop jobs. Every job requires a Bash script with special parameters for Slurm. An example script, with definitions of some important parameters, is provided on CANVAS as `example.sh` in the `Homework4` directory. **Some parameters can be used to log the output from the code, which will be important for some experiments.**
- **GitHub.** All code releases will be based in GitHub. Make sure you have an account. You can use the command `git clone repo.git` to download the code release to your own environment.
- **Python environments.** Great Lakes doesn't allow you to install new Python dependencies unless you do so in a separate virtual Python environment. The two most popular tools to set up virtual Python environments are `virtualenv` (in conjunction with `pip`) and `conda`. You can read this document for information on how to use these two tools. If you want more details, you can read this document for `virtualenv`, and this document for `conda`.

Helpful Hints and Resources

A few notes and resources about pre-trained language model fine-tuning:

- **Transformers** Attention is all you need [3], but it's hard. The Annotated Transformer from Harvard NLP would be a good resource to illustrate.
- **HuggingFace** Read HuggingFace's official guide. Also check out our `hw3.ipynb` as a tutorial on HuggingFace and its APIs.
- **Task Heads** Our `hw3.ipynb` provide an example of text classification and multiple choice. However, the data processing is different.
- **Data Collector** Note that you need to specify in the training loop how to form batches from the pre-processed inputs. You might need a **data collator**, which takes a list of examples and converts them to a batch (by, in our case, applying padding). Since there is no data collator in the library that works on this specific problem, you may have to write one upon `DataCollatorWithPadding`.

Grading and Code Expectations

Your code must be able to do the following:

1. Pre-process, tokenize, and prepare the dataset into `torch.utils.data.DataLoader`;
2. Fine-tune a pre-trained language model;
3. Apply the model to test data and save the prediction of the model.

You will find a skeleton code file `finetune.py` that imports some libraries you may need. You need to provide code that process the data and fine-tune the model in the areas marked between "Your code starts here" and "Your code ends here"; some example lines are provided in these spaces, but they can be modified or removed. Some example code is provided, but you may add any other

functions you need. You should not need to modify the names or parameters of the `load_data`, `finetune`, or `test`, or the argument parsing code at the bottom.

For grading, you must specify how your code handles the commands, and utilize the arguments passed in, as they may change. For example:

```
$ python3 finetune.py --model=bert-base-cased --batch_size=128
```

Submission of Prediction

Your accuracy on the test data should be above 30.0% for full credit. This should not be hard, as random guesses lead to 25.0% accuracy.

Please submit the following files:

- Your Python program implementing the Language Model fine-tuning: `finetune.py`
- The model's **prediction** (not the model) on the validation set without shuffling from **best** hyper-parameter configuration: `prediction.torch`. See `hw3.ipynb` for details.

Paper reading

Before you start coding, please read the paper Think you have Solved Question Answering? Try ARC, the AI2 Reasoning Challenge [4] Please read this paper in-depth and submit a report summarizing your understanding, questions, and insight after reading this paper. It is up to you how you would like to structure the report. Some example discussion points for you to consider include:

- You may compare this work to other works you read before and discuss what's novel or exciting about this work.
- You may hypothesize about ways in which the work can be improved.
- You may think about ways to expand on the work, either conceptually or computationally
- You may raise questions about parts that are not clear or need to be further explained
- You may critique the work e.g., its formulation, conceptual framework, methodology or evaluation
- Anything else you think is relevant.

Your report should be around 500 words.

Solution:

Problem Statement and Related Work

The paper introduces the ARC dataset, aimed at advancing AI question-answering by challenging advanced reasoning. Several question-answering datasets, such as MCTest, SQuAD, NewsQA, and CNN/DailyMail, have been used in AI research, but they mostly feature questions with answers easily found in the text (explicitly stated). TriviaQA expanded on this but remained factoid-style. Synthetic datasets like bAbI offered more complex tasks but were limited by their artificial nature. WikiHop introduced multihop questions but had limitations.

Using human standardized tests has been explored, but these datasets tend to be small and can be easily solved by simple AI methods, which hampers progress on challenging questions. The ARC Dataset addresses these limitations by offering more complex questions and a larger dataset.

Summary

The ARC dataset contains 7,787 natural science questions, split into a Challenge Set of 2,590 tough questions and an Easy Set of 5,197. The dataset has a good blend of science questions ranging from tests for grade 3 to grade 9 students with varied difficulties. To support the dataset, a science text corpus consisting of 14M sentences. The ARC Corpus, constructed using templates for 80 science topics, found around 75% of collected documents to be science-relevant. It was augmented with the AristoMini corpus, which includes dictionary definitions, Simple Wikipedia science articles, and additional web-based science sentences. An analysis revealed that 99.8% of the vocabulary used in ARC questions can be found in the ARC Corpus. Unlike prior datasets, ARC focuses on complex questions that can't be easily answered through basic retrieval methods. It defines challenge questions as those incorrectly answered by both the Information Retrieval (IR) Solver and Pointwise Mutual Information (PMI) Solver. The IR Solver uses a large web-based corpus to search for the question and answer option in the text and returns a confidence score based on the retrieval. The PMI Solver computes the pointwise mutual information between parts of the question and answer options using a corpus. It extracts various n-grams from the question and answer options and selects the answer with the highest average PMI score across all n-gram pairs. These filters help identify challenging questions in the dataset.

When evaluating baseline question-answering systems on ARC, a notable performance gap is observed between the Challenge and Easy Sets. None of the algorithms outperformed random guessing on the Challenge Set, highlighting the need for advanced retrieval and reasoning methods. In conclusion, ARC stands as a crucial benchmark for advancing AI's comprehension and reasoning in question-answering.

Suggested Improvements

Although the AI2 ARC is a publicly available dataset which offers a set of challenging questions for encouraging research on answering complex questions, there might exist some Retrieval Bias in the system. As demonstrated by baseline models, there may be a bias towards sentences that closely match the question. Addressing this bias is essential for promoting advanced retrieval methods. Further, the ARC Corpus *does not address the challenge of correctly identifying and reasoning with this knowledge, nor the challenge of injecting unstated commonsense knowledge that may also be required.*[1] ARC could improve by including answer explanations, helping to assess not only if the model answered correctly but also if it understood the underlying concepts.

Improvements and considerations for the ARC dataset and research can include refining challenge question criteria, enhancing semantic reasoning demands, promoting answer explanations, expanding to diverse domains and sources for collecting data, and reducing retrieval bias.

References

[1] Clark, P., Cowhey, I., Etzioni, O., Khot, T., Sabharwal, A., Schoenick, C. & Tafjord, O. Think you have Solved Question Answering? Try ARC, the AI2 Reasoning Challenge. (2018)

Submission of Documentation

Please submit your code to Canvas. Besides submitting your code and model as described above, you need to submit a report in this LaTeX file. This document should report how your code handles the commands, the best testing accuracy of your fine-tuned language model and its corresponding hyper-parameter configuration[1]. You should list the hyper-parameters you experimented with, and provide the results from each configuration you tried.

Experiment Report

Solution:

Handling the passed arguments -

```
$ python finetune.py --dataset="ai2_arc" --model="bert-base-cased" --batch_size=32 --num_epochs=5
--learning_rate=1e-4
```

Best Model Configuration (Hyperparams) -
1. batch_size = 32
2. learning_rate = 1e-4
3. num_epochs = 5
4. optimizer = AdamW
5. Pretrained Model = bert-base-cased

train accuracy = 35.25%
test accuracy = 34.50%

Model Performance

Below is a table listing the performances of several other model configurations -

Sr.No.	batch_size	learning_rate	num_epochs	Pre-trained Model	Validation Accuracy	Test Accuracy
1**	**32**	**1e-4**	**5**	**bert-base-cased**	**0.3525**	**0.3450 ****
2	16	1e-4	5	bert-base-cased	0.322	0.339
3	32	1e-4	5	roberta-base	0.2576	0.2480
4	32	1e-4	5	bert-base-uncased	0.3118	0.3313
5	16	1e-4	5	roberta-bsae	0.2576	0.2927
6	16	1e-4	5	bert-base-uncased	0.3457	0.3184
7	32	1e-3	5	bert-base-cased	0.2305	0.2317
8	32	1e-5	5	bert-base-cased	0.2949	0.2806
9	32	1e-4	3	bert-base-cased	0.3322	0.3321
10	32	1e-4	8	bert-base-cased	0.2745	0.2601

[1]For grading, we will verify that your reported predictions matches the write-ups.

Findings

1. bert-base-cased performed best out of roberta-base and bert-base-uncased. bert-base-ucased being the second best.
2. This behaviour was unexpected as roberta is trained on a much larger dataset.
3. It is pretty hard to achieve an accuracy of over 30%.
4. Clearly hints at how difficult MCQ Answering is and how well the Challenge set in AI2 ARC has been created.
5. These results could be further improved by finetuning larger pre-trained models like bert-large-cased or roberta-large etc.
6. The accuracy can largely be improved by leveraging the ARC Corpus for getting the required context for answering the MCQs.

Discussion

In this problem, you are required to fine-tune a pre-trained language model from the BERT family to perform multiple choice question answering. And if you look into the acceptable models for multiple choice task here HuggingFace, you will find that GPT-2 is not included. Why is that? Please discuss the difference between BERT and GPT-2, and why GPT-2 is not naively suitable for multiple choice question answering. How would you make GPT-2 work for this task? Please provide your thoughts in the following space.

Solution: BERT and GPT are both adoptions of the Transformer architecture where BERT is an only-Encoder model and GPT is an only-Decoder model. These both have Self-attention for encoding large-range dependencies and Self-supervision for leveraging large unlabelled datasets.

Bi-directional Encoder Representations from Transformers:

1. Uses a bidirectional training objective. It learns to predict masked words in a sentence by considering both left and right context. This bidirectional training captures deeper contextual understanding, which is crucial for MCQ tasks.
2. It has 2 training objectives - Masked Language Model (predicting the masked words in the sentence) and Language Model (Next Sentence Prediction) which helps it capture better context of words.
3. BERT's attention mechanism allows it to capture contextual information effectively, regardless of word order or position.
4. BERT models are explicitly designed for fine-tuning, and they excel in various natural language understanding tasks.

5. Encoders are generally used to learn embeddings that can be used for various predictive modeling tasks such as classification.

Generative Pre-trained Transformer-2:

1. GPT-2 is trained using a left-to-right autoregressive language modeling objective. It predicts the next word in a sentence based on the preceding words.
2. Only has next word prediction. Doesn't have a mechanism to understand the relationships between words or phrases in a bidirectional manner.
3. Relies on positional embeddings to understand word order.
4. While GPT-2 can be fine-tuned on specific tasks, it is not optimized for bidirectional understanding, which is a limitation for tasks requiring deeper contextual comprehension like MCQ answering.
5. Decoders are generally used to generate new texts, for example, answering user queries.

To make GPT-2 work for MCQ answering, we must add a layer of MLM (like BERT) or implement some bi-directional self-attention layers to gain bidirectional context. But this would in turn cause leakage in next word prediction as model starts looking for the next word before generating the previous word. Nevertheless, we can add multiple (or single) neural network layer(s) on top of pooled layers and close it with a softmax to get probability distributions of correct options. To enhance the next word prediction to accommodate for next sentence prediction, we can also add special tokens for classes like [CLS] in BERT in the tokenizer used by GPT2.

References

[1] Manning, C. D., Surdeanu, M., Bauer, J., Finkel, J. R., Bethard, S., & McClosky, D. (2014, June). The Stanford CoreNLP natural language processing toolkit. In Proceedings of 52nd annual meeting of the association for computational linguistics: system demonstrations (pp. 55-60).

[2] Devlin, J., Chang, M. W., Lee, K., & Toutanova, K. (2018). Bert: Pre-training of deep bidirectional transformers for language understanding. In Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers) (pp. 4171-4186).

[3] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., ... & Polosukhin, I. (2017). Attention is all you need. Advances in neural information processing systems, 30.

[4] Clark, P., Cowhey, I., Etzioni, O., Khot, T., Sabharwal, A., Schoenick, C. & Tafjord, O. Think you have Solved Question Answering? Try ARC, the AI2 Reasoning Challenge. (2018)