Assignment 4: Fine-tuning Language Models

Due date: Friday, Nov 14th at 12:59PM EST

Academic Honesty: Please see the course syllabus for information about collaboration in this course. While you may discuss the assignment with other students, **all work you submit must be your own!**

Overview In this assignment, you will train two types of language models, encoder-only model (BERT) and an encoder-decoder model. The main goal is learning how to fine-tune language models for a specific task, and understanding the challenges involved in it.

In Part 1, you will fine-tune an encoder-only model for sentiment classification dataset, focusing on transformation of data to understand generalization. In Part 2, you will train an encoder-decoder model to translate natural language instruction for flight booking into SQL query. Here, you will study conditional generation: generating a variable length, structured output (SQL query) given a variable length input.

Starter Repo You will require GPUs for this HW. Please first go through the file `README.md` to set up the environment required for the project.

For part 1 of the homework, you were provided with `main.py` and `utils.py` as starter code to fine-tune an encoder-only model on a sentiment classification dataset.

For part 2 of the homework, the data can be found under the `data/` directory. The database you will be evaluating queries on is `flight_database.db`, with the `flight_database.schema` file containing the database schema. The `ents` (entities) section in the schema lists the 25 tables in the database, such as "airline", "restriction", "flight", etc. The schema also gives information about the columns in the table, such as their type and whether they're indexed.

The text-to-SQL data, on the other hand, is split into training, development and held-out test sets. The files with the `.nl` extension contain the natural language instructions, while the files with the `.sql` extension contain the corresponding, ground-truth SQL queries. The starter code contains various utility functions for evaluation (under `utils.py`) to abstract away the details of interacting with the SQL database. A skeleton of training code for T5 is provided in `train_t5.py`.

Submissions

Submission is done on Gradescope.

Written: A `.pdf` file must be submitted. We recommend using the released `.tex` file.

*** Important *** In this assignment, in addition to answers to written questions, the written PDF file should include (1) a link to your github repository (which contains your code for Part I and Part II) and (2) a link to Google drive, which contains your model checkpoint used to generate outputs for Q7. If you do extra credit assignment, you also need to provide a link to that model checkpoint as well.

Programming: You will submit multiple output files from your model for this assignment.

- Q1: `out_original.txt`
- Q2: `out_transformed.txt`
- Q3: `out_augmented_original.txt` and `out_augmented_transformed.txt`.

- Q7: Models' SQL query outputs and the database records associated with the SQL queries. `t5_ft_experiment_test.pkl` and `t5_ft_experiment_test.sql`
- Extra Credit (Optional): the models' SQL query outputs and the database records associated with the SQL queries. `t5_ft_experiment_ec_test.pkl` and `t5_ft_experiment_ec_test.sql`

Part 1. Fine-tuning BERT model for sentiment classification (50pts)

In this part, we will explore training BERT model for simple classification task. Our main goal here is exploring the role of **data** in classification results, simulating out-of-distribution data.

Q1. Fine-tuning BERT model

You will first write code to fine-tune a BERT model on the IMDB dataset for sentiment analysis.

1. (10 points, coding) We have provided a template for running your code to fine-tune BERT, but it contains some missing parts. Complete the missing parts in `do_train` function in `main.py`, which mainly includes running the training loop. You are not required to modify any hyperparameters.

   Initial Testing: Before proceeding, test your training loop on a small subset of the data to verify its correctness. Use the following command: `python3 main.py --train --eval --debug_train`. Upon successful execution, you should achieve an accuracy of more than 88% on this small subset. (Estimated time: `7 min` train + `1 min` evaluation).

   Full Training and Evaluation: Once you have confirmed that your training loop is functioning as expected, fine-tune BERT on the full training data and evaluate its performance on the test data using the command: `python3 main.py --train --eval`. An accuracy of more than 91% on the test data is required to earn full points. (Estimated time: `40 min` train + `5 min` evaluation).

   Submission: Please submit the output file `out_original.txt` generated from the full training and evaluation step.

Q2. Data Transformations

The main objective in this part is creating transformations of the dataset as out-of-distribution (OOD) data, and evaluate your fine-tuned model on this transformed dataset. The aim will be to construct transformations which are "reasonable" (e.g. something we can actually expect at test time) but not very trivial.

In this part, you will design transformations of the evaluation dataset which will serve as out-of-distribution (OOD) evaluation for models. In most cases, the new transformed example should have the *same label* as original example — a human would assign the same label to original and transformed example. e.g. For an original movie review "Titanic is the best movie I have ever seen.", a transformation which maintains the label is "Titanic is the best film I have ever seen.".

1. (10 points, written) Design a transformation of the dataset, and explain the details of what it does. You should describe it clearly, such that someone who reads the description can reimplement it. You should also include why that is a "reasonable" transformation i.e. something which could potentially be an input at test time from a user. If you like, you can use LLMs to perform data transformation.

   Examples & Guidelines Suitable transformations might include:

   - Randomly replacing words in a sentence with their synonyms.
   - Introducing typos into sentences.

An example of an **unreasonable** transformation is converting coherent words into random gibberish, such as changing "The soup was hot." to "The unnjk rxasqwer hot.". We've provided code guidelines for two transformations (synonym replacement and typo introduction). You can choose to utilize these or feel free to design a unique transformation of your own.

Evaluation Criteria Determining whether a transformation is "reasonable" can be subjective. However, please exercise your best judgment. While we will be fairly flexible with the definition of "reasonable", extreme transformations like the gibberish example mentioned will not be accepted.

2. (15 points, coding) Implement the transformation that you designed in the previous part.

   Setup: We've provided an example transformation function named `example_transform`. Your task is to use this as a reference and complete the function `custom_transform` found in the `utils.py` file.

   Debugging: To ensure that your transformation is working as expected and to see a few examples of the transformed text, use the following command: `python3 main.py --eval_transformed --debug_transformation`

   Evaluation To assess the performance of the trained model on your transformed test set, execute: `python3 main.py --eval_transformed` (Estimated time: `5 min` evaluation).

   Your grade will be determined based on the decrease in performance on the transformed test set in comparison to the original test set:

   - A decrease in accuracy of up to 4 points will grant you partial credit (8 out of the total 15 points).
   - A decrease in accuracy of more than 4 points will award you the full 15 points.

   Submission: Please submit the output file `out_transformed.txt` generated from the evaluation step.

Q3. Data Augmentation

In this part, you will develop a simple way to improve performance on the transformed test set, according to the transformation you have defined.

One simple way to potentially improve performance on the transformed test set is to apply a similar transformation to the training data, and train your model on a combination of the original data and transformed training data. This method is usually known as data augmentation. In this part, you will augment the original training data with 5,000 random transformed examples, to create a new augmented training dataset. Fill in `create_augmented_dataloader` in `main.py` to complete the code.

Training & Evaluation Train a model using this augmented data by executing the following command: `python3 main.py --train_augmented --eval_transformed` (Estimated time: `50 min` train + `5 min` eval)

1. (15 points, written) In this task, you will evaluate the performance of the above trained model. Your objective is to assess how the data augmentation affects the model's ability to handle both original and transformed test data.

   Evaluation on Original Test Data Execute the following command to evaluate the model's performance on the original test data: `python3 main.py --eval --model_dir out_augmented` (Estimated time: `5 min` eval)

Evaluation on Transformed Test Data Use the command below to assess how the model performs on the transformed test data: `python3 main.py --eval_transformed --model_dir out_augmented` (Estimated time: `5 min` eval)

Report & Analysis

- Report the accuracy values for both the original and transformed test data evaluations.
- Analyze and discuss the following: (1) Did the model's performance on the transformed test data improve after applying data augmentation? (2) How did data augmentation affect the model's performance on the original test data? Did it enhance or diminish its accuracy?
- Offer an intuitive explanation for the observed results, considering the impact of data augmentation on model training.
- Explain one limitation of the data augmentation approach used here to improve performance on out-of-distribution (OOD) test sets.

Submission Please submit the following output files obtained after running model evaluation on both data: `out_augmented_original.txt` and `out_augmented_transformed.txt`.

Part 2. Fine-tuning T5 for Text-to-SQL (50pts).

Now you will be working with the small variant of the T5 encoder-decoder architecture (Raffel *et al.*, 2019)[1]. The encoder will ingest natural language queries (i.e., the input sequence) while the decoder will predict the corresponding SQL queries (i.e., the output sequence). You will finetune the **pretrained T5 architecture**. The grading for this part of the homework will consists of **written part (25pt)** as well as **performance (25pt)** of your trained model.

We provide you a training loop code skeleton where you will only need to implement model initialization (using the relevant Huggingface transformers implementation[2]), the evaluation loop and data processing.

Simply implementing data processing with the existing T5 tokenizer and varying simple hyperparameters should lead to good baseline results. You may, however, choose to experiment with data processing or architecture details to push performance higher. During finetuning, for instance, a common design choice is to freeze part of the model parameters (i.e., only finetune a subset of the parameters).

In the assignment report, we ask you about your strategies for data processing, tokenization and architecture details (such as freezing parameters). If you experiment with these dimensions and find them to improve performance, we will expect ablation experiments supporting your findings in the results section.

Task Evaluation Details As different SQL queries can correctly respond to the same natural language instruction, evaluation is commonly performed by executing the SQL queries directly on the database. The evaluation code in the starter repository will provide you with three separate metrics: Record F1, Record Exact Match and SQL Query Exact Match (EM).

Record F1 is the metric that we will use in the leaderboard and computes the F1 score between database records produced by model-generated and ground-truth SQL queries. Record EM, on the other hand, is a stricter metric checking whether the produced records match exactly, similar to an accuracy score. Finally, SQL Query EM will tell you whether the SQL queries themselves match exactly. As the latter two can help with debugging and error analysis, we will ask you to also report them on the development set component of the results section.

To verify the correctness of your output format, you may use the `evaluate.py` script. For instance, to evaluate submission files on the dev set, run:

```
python evaluate.py
  --predicted_sql results/t5_ft_dev.sql
  --predicted_records records/t5_ft_dev.pkl
  --development_sql data/dev.sql
  --development_records records/ground_truth_dev.pkl
```

Q4. Data statistics and processing (5pt, written)

Describe the data statistics before and after any pre-processing respectively by filling the following two tables Table 1 and Table 2. Use the T5 tokenizer to report the statistics. The gray text in each row is there to guide you and should be removed in your submitted report. Depending on your pre-processing, some numbers may be identical across tables.

[1] https://arxiv.org/pdf/1910.10683

[2] https://huggingface.co/google-t5/t5-small

Statistics Name	Train	Dev
Number of examples	XXX	XXX
Mean sentence length	XXX	XXX
Mean SQL query length	XXX	XXX
Vocabulary size (natural language)	XXX	XXX
Vocabulary size (SQL)	XXX	XXX

Table 1: Data statistics before any pre-processing. You need to at least provide the statistics listed above, and can add new entries.

Statistics Name	Train	Dev
Model name		
Statistics Name	XXX	XXX

Table 2: Data statistics after pre-processing. You need to at least provide the statistics listed in Table 1 (except for the number of examples), and can add new entries.

Q5. T5 Fine-tuning (10pts, written)

Use Table 3 to describe your data processing steps (if any) and the implementation details, for the fine-tuned T5 model. The gray text in each row is there to guide you and should be removed in the submitted report. Be clear enough that one can replicate your approach in PyTorch using only your descriptions. You'll find these tables in latex template we provide.

Design choice	Description
Data processing	Describe the data processing steps you undertook, if any.
Tokenization	Describe how you did the tokenization for the inputs to the encoder and decoder. If you use anything else than the default T5 tokenizer, specify what you use, and why you choose to use it.
Architecture	Describe the components in the T5 architecture that you chose to fine-tune. Did you fine-tune the entire model, specific layers?
Hyperparameters	List the key hyperparameters that you used, including the learning rate, batch size, and stopping criteria.

Table 3: Details of the best-performing T5 model configurations (fine-tuned).

Q6. Results (10pt, written)

Quantitative Results: Use Table 4 to report your test and development set results. Your test results should match with the results in the leaderboard. For the development set, you should also report results from experiments you conducted to arrive at your final configuration. The full model refers to the best model you described in previous section. For T5, if you experimented with different design choices, you can add rows specifying the variants and what you tried. The text in gray is only for example purpose, and should be removed and replaced with your own choices. You may add more rows if needed.

System	Query EM	F1 score
Dev Results		
T5 fine-tuned		
Full model	XX.XX	XX.XX
Test Results		
T5 fine-tuning	XX.XX	XX.XX

Table 4: Development and test results. Use this table to report quantitative results for both dev and test results.

Error Type	**Example Of Error**	**Error Description**	**Statistics**
Error name	Snippet from datapoint exemplifying error	Describe the error in natural language	Provide statistics in the form "COUNT/TOTAL" on the prevalence of the error. TOTAL is the number of relevant examples (e.g. number of queries, for query-level error), and COUNT is the number of examples that showed this error.

Table 5: Use this table for your qualitative analysis on the dev set.

Qualitative Error Analysis Conduct a detailed error analysis for each of the two models. Identify common error types and discuss possible reasons for these errors in Table 5.

You must identify at least three classes of errors for the queries, and use examples to illustrate them. It must be clear what model makes the errors you are analyzing. If you identified the same type of error for different models, you don't need to duplicate the descriptions, but you need to clearly specify an example for each of the model, indicate the statistics for each model, and specify to which model each statistics correspond to. You may add more rows to the table.

Q7. Performance Evaluation (code, 25pts)

Submission We will maintain a leaderboard for the test set, with submissions ranked according to the F1 score. As it can be resource intensive to run SQL queries, you will submit two files for each task: `t5_ft_experiment_test.pkl` the models' SQL query outputs; and `t5_ft_experiment_test.sql` the database records associated with the SQL queries.

Performance Evaluation Criteria To get a full credit for performance, your model should perform ≥ 65 F1 on **the test set**. While you can submit multiple times to Gradescope to get test set result, you should not be using this functionality to get the highest number. You should develop your model checking performance on the provided development set, and run on the test set only a few times (ideally once). We might deduct points if you run on test set excessively.

If your model performs below this threshold, you'll receive a partial credit, computed as below: $\frac{\text{your score}}{65} * 25$.

The `save_queries_and_records` function in `utils.py` will help you with this. Please refer to the `README.md` file in the code skeleton for more details on formatting.

Extra Credit 1: We will provide an extra credit (1% of the course total grade) to top 3 students.

Extra Credit 2 (Optional Assignment): Training T5 model from Scratch

In this optional task, you will be to train the exact same architecture **from scratch (i.e., from randomly initialized weights)**. This is more challenging than fine-tuning pretrained weights, and performance will be lower. Yet, you should be able to achieve decent results with existing T5 tokenizer and varying hyperparameters. But you could find experimenting with data processing leading to better outcomes. If you find that the tokenizer is ill-suited for SQL code, for instance, you could choose to design your own tokenizer for SQL and learn new embeddings and language modeling output heads for the decoder.[3] You should describe your new model in the report (i.e., make a new version of Table 2, 3, 4, 5 with your new model (which is trained from scratch) and clearly label them).

If you achieve $\geq$ 50 F1 with this model on the hidden test set, along with valid descriptions, we will award 1.5% of the course total grade as an extra credit.

Please submit `t5_ft_experiment_ec_test.pkl` and `t5_ft_experiment_ec_test.sql` for extra credit.

Acknowledgement: Part I was originally developed by He He for previous offerings of this course, developed based on "NL-Augmenter", which can be accessed at `https://arxiv.org/abs/2112.02721`. Part II was modified from assignment from LM-class.org, created by Omer Gul, Anne Wu, and Yoav Artzi.

[3] These are ideas to get you thinking, rather than specific suggestions to try. We did not fully experiment with these approaches.