CSCI-GA 2590: Natural Language Processing

HW3 - Fine-tuning Language Models

Name
NYU ID

Collaborators:
By turning in this assignment, I agree by the honor code of the College of Arts and Science at New York University and declare that all of this is my own work.

Acknowledgement: This HW is inspired from NL-Augmenter - `https://arxiv.org/abs/2112.02721`.

Before you get started, please read the Submission section thoroughly.

Submission

Submission is done on Gradescope.

Written: You can either directly type your solution in the released `.tex` file, or write your solution using pen or stylus. A `.pdf` file must be submitted.

Programming: Questions marked with "coding" at the start of the question require a coding part. Each question contains details of which functions you need to modify. You should submit all `.py` files which you need to modify, along with the generated output files as mentioned in some questions.

Fine-tuning Language Models

The goal of this homework is to get familiar with how to fine-tune language models for a specific task, and understand the challenges involved in it. More specifically, we will first fine-tune a BERT-base model for sentiment analysis using the IMDB dataset.

Next, we will look at one of the assumptions commonly made in supervised learning — we often assume *i.i.d.* (independent and identically distributed) test distribution i.e. the test data is drawn from the same distribution as the training data (when we create random splits). But this assumption may not always hold in practice e.g. there could be certain features specific to that dataset which won't work for other examples of the same task. The main objective in this homework will be 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.

First go through the file `README.md` to set up the environment required for the class.

1. Fine-tuning

As mentioned above, you will first write code to fine-tune a BERT model on the IMDB dataset for sentiment analysis. This dataset is a large sentiment anlaysis dataset based on movie reviews on IMDB. You can find more details here - `https://huggingface.co/datasets/imdb`.

You will require GPUs for this HW.

1. (4 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.

   Run `python3 main.py --train --eval` which will fine-tune BERT and evaluate on the test data. You should submit the generated output file for this part. An accuracy of more than 91% on the test data will earn full points.

2. Transformations

In this part you will design transformations of the evaluation dataset which will serve as out-of-distribution (OOD) evaluation for models. These transformations should be designed so that in most cases, the new transformed example has 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. (3 points, written) Design a transformation of the dataset, and explain the details of what it does. 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.

   Some examples of transformations you could implement are — randomly replacing the words in a sentence with synonyms, or introducing typos in sentences. An example of a transformation which is not 'reasonable' is replacing words by random gibberish in the sentence (e.g. 'The soup was hot.' to 'The unnjk rxasqwer hot.'). We have provided rough guidelines in the code to implement two different transformations — synonym replacement and typos. You can either use those, or you are free to design your own transformation. In general, deciding whether a transformation is 'reasonable' can be subjective but you should use your best judgement in this case. We will be reasonably lenient with what is 'reasonable' apart from extreme transformations like the example above.

2. (6 points, coding) Implement the transformation that you designed in the previous part. We have provided a template for a toy transformation (named `example_transform`) which you should follow to fill in the function `custom_transform()` in the file `utils.py`. To debug and see a a few examples, run `python3 main.py --eval_transformed --debug_transformation` .

   To evaluate the trained model on the transformed test set, run `python3 main.py --eval_transformed`. You will be graded depending on how much the performance drops on this transformed test set compared to the original test set — any drop of up to 4 accuracy points would be awarded a fixed number of partial points (3 points out of 6 total). A drop of more than 4 accuracy points would be awarded full points.

3. Data Augmentation

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

1. (3 points, coding) 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 augmented the original training data with 5000 random transformed examples, to create a new augmented training dataset. Fill in `create_augmented_dataloader` in `main.py` to complete the code. Run `python3 main.py --train_augmented --eval_transformed` to train a model on this augmented data. Submit the generated output file.

2. (2 points, written) Evaluate the above trained model on the original test data (using `python3 main.py --eval --model_dir out_augmented`) and on the transformed test data (using `python3 main.py --eval_transformed --model_dir out_augmented`) and report the accuracies. Does the model performance improve on the transformed test data after data augmentation? Does the model performance improve or worsen on the original test data after data augmentation? Give an intuitive reason to explain the result.

3. (2 points, written) Explain one limitation of the data augmentation approach used here to improve performance on out-of-distribution (OOD) test sets.