HW3: Fine-tuning Language Models

CSCI 662: Fall 2023

out: Oct 23, 2023
due: Nov 17, 2023

Acknowledgement: This homework is inspired by He He's CSCI-GA 2590 homework 3, which was in turn inspired by NL-Augmenter - https://arxiv.org/abs/2112.02721.

Overview

In this assignment, you will be fine-tuning a language model for a specific downstream task and understand the challenges involved in a practical setting. You will fine-tune a BERT-base model for sentiment analysis using the IMDB dataset and then explore the challenges of out-of-distribution test sets.

One of the main assumptions made in supervised learning is *i.i.d.* (independent and identically distributed) test distribution. In other words, the test data is drawn from the same distribution as the training data. However, this assumption does not hold well in practice, especially in the real world. For example, the training data may have been collected with text that is well-formed and clean, but the resulting system from this data is deployed into the real world where these assumptions may not hold. This means that there are certain features that are specific to the dataset that may not work for other examples of the same task.

Therefore, the next main objective of this homework is to create transformations of the dataset to create out-of-distribution (OOD) data and evaluate your fine-tuned model on it. The aim is to construct transformations that are 'reasonable' (e.g. something that you can actually expect at test time) but not very trivial.

As with the previous homework, the main goal is to conduct a scientific study and write a report that clearly communicates what you have done, why you have done it, and what is interesting about it. Your report should be at least two pages long and not exceed four pages, excluding references.

Setup

You will require GPUs for this HW. Depending on the platform that provides GPU access, you may want to use a Jupyter notebook or just regular Python scripts as we have done in the previous two homework. More guidance is provided in the `README.md` that is included in the starter code. Follow the instructions to set up your environment (e.g. Google Colab or a machine with GPUs).

Requirements

Fine-tuning

1. First, you will fine-tune a BERT-base language model on the sentiment analysis datset based on movie reviews on IMDB.[1]

2. We have provided you with both a Juypter notebook template and Python scripts for fine-tuning BERT, but it contains some missing parts. Choose one of the two folders (`notebook`, `python_script`) to work with. There is no need to do both.

3. Complete the missing parts in the `do_train` function in `main.py`, which is mostly the training loop. You are not required to modify any hyperparameters.

4. Run the jupyter notebook with `train: true` and `eval: true` or run `python main.py --train --eval`[2], 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 87% will earn you full points for code correctness.

Transformations

1. In this part, you will design transformations to apply to the evaluation dataset which will serve as an out-of-distribution evaluation for your model. These transformations should be designed such that the new transformed example has the **same label** as the original example. In other words, a human would assign the same label to the original and transformed example. For example, a transformation that maintains the same label as "Titanic is the best movie I've ever seen!" is "Titanic's the best film I have ever seen."

2. Design a transformation and explain what it does. You should also explain why it is a reasonable transformation. Whether a transformation is reasonable is subjective, but use your best judgement. We will be lenient with what is reasonable apart from extreme cases such as using gibberish (e.g. 'The soup was hot' becomes 'The unnjk rxasqwer hot.').

3. Implement your proposed transformations. At a minimum, implement synonym replacement and synthetic typos. We have provided the rough guidelines for these implementations, so you can use those or use your own. We also provide a template for a toy transformation named `example_transform`, which you should follow to fill in the function `custom_transform()` in `utils.py`. To debug and see a few examples, run `python main.py --eval_transformed --debug_transformation`.

4. Any drop of up to four accuracy points will give you half credit for code correction in this part. A drop of more than 4 will get you full points.

Data Augmentation

1. One simple way to improve performance on the transformed set is to apply a similar transformation to the training data and train your model on the combination of the original data and the transformed training data. This is known as data augmentation. Augment the training data with 5000 randomly transformed examples to create the new augmented training dataset.

2. Fill in `create_augmented_dataloader` in `main.py` to complete the code.

3. Run `python main.py --train_augmented --eval_transformed` to train a model on this augmented data and submit the generated output file.

[1] https://huggingface.co/datasets/imdb

[2] In all future instructions, you can apply the same command line parameters for the Jupyter notebook by editing the `configs.yml` file.

4. Evaluate the trained model on the original test data using `python main.py --eval --model_dir out_augmented` and on the transformed test data using `python main.py --eval_transformed --model_dir out_augmented` and report the accuracies.

5. Does the model performance improve on the transformed test set after data augmentation? How does the model performance compare on the original test data after data augmentation? Explain the result with intuitive reasons.

6. Elaborate on one limitation of the data augmentation approach used here for improving performance on OOD test sets.

Coding Recommendations

- For development purposes, start small and see that the training loop works as expected (i.e. loss drops, you can overfit to the training set, etc.). Training the model on the full dataset will take some time (more than 1 to 2 hours depending on the GPU you have), so only run training on the full dataset once you are confident that your set up is correct.

- If you use Google Colab, you may reach a free-account usage limit if you do not spend the time given with your GPU wisely. Disconnect your notebook from a GPU if you're not actively using it as an idle connection may cause you to reach your limit sooner. If you reach your limit, you can switch to a different Google account or you can purchase Colab pro.

- You are more than welcome to use any packages out there to help you train your model and augment the dataset. More time should be spent on trying out interesting transformation or modeling techniques to see if you can further boost performance on the original test set and your transformed test set.

Your Report

Your report should at a *minimum*:

- Describe the motivations for the transformations that you have applied and how you have implemented them.

- Analyze and discuss the performance difference for the original test set and the transformed test set.

- Describe technical details: any pre/post-processing steps, compute, training strategy, learning rate, etc.

Use the ACL style files : https://github.com/acl-org/acl-style-files

Your report should be at least two pages long, including references, and not more than four pages long, not including references (i.e. you can have up to four pages of text if you need to). Just like a conference paper or journal article it should contain an abstract, introduction, experimental results, and conclusion sections (as well as other sections as deemed necessary). Unlike a conference paper/journal article, a complete related works section is not obligatory (but you may include it if it is relevant to what you do).

Grading

Grading will be roughly broken down as follows:

- about 50% – Does your report read like a research paper? Did you clearly communicate your description of what you implemented, how you implemented it, what your experiments were, and what conclusions you drew from them? This includes appropriate use of graphics and tables where warranted that clearly explain your point. This also includes well written explanations that tell a compelling story. Grammar and syntax are a small part of this (maybe 5% of the grade, so 10% of this section) but much more

important is the narrative you tell. Also a part of this is that you clearly acknowledged your sources and influences with appropriate bibliography and, where relevant, cited influencing prior work.

- about 20% – Is your code correct? Did you implement what was asked for, and did you do it correctly?
- about 20% – Is your code well-written, documented, and robust? Will it run from a different directory than the one you ran it in? Does it rely on hard-codes? Is it commented and structured such that we can read it and understand what you are doing?
- about 10% – Did you go the extra mile? Did you push beyond what was asked for in the assignment, trying (well-justified) new models, features, or approaches? Did you use motivation (and document appropriately) from another researcher trying the same problem or from an unrelated but transferable other paper?

'Extra Mile' ideas

This is not meant to be comprehensive and you do not have to do any of the things here (nor should you do all of them). But an 'extra mile' component is 10% of your grade (**not extra credit!**).

- Implement augmentations beyond synonym replacement and synthetic typos.
- Try other model architectures for this task. Explain why you get better or lower performance.
- Try training with additional data from other sentiment analysis datasets.
- Dig into the ACL archives and find ideas for other architectures, augmentations, or approaches; try them out, analyze performance.

Rules

- This is an individual assignment. You may not work in teams or collaborate with other students. You must be the sole author of 100% of the code you turn in.
- Depending on need and class interest, we may collaborate *in class* or *publicly on Piazza* if you get stuck; this kind of collaboration is okay.
- You may not look for coded solutions on the web, or use code you find online or anywhere else. You can and are encouraged to read material beyond what you have been given in class (see above) but should not copy code.
- Generative language, code, and vision models (e.g. ChatGPT, Llama 2, Midjourney, Github Copilot, etc.; if you are unsure, ask and don't assume!!) can be used (to aid in report writing/coding, not to actually do the classification tasks) with the following caveats:
  - You must declare your use of the tools in your submitted artifact. If you don't declare the tool usage but you did use these tools, we will consider that plagiarism
  - For code and image generation, you must indicate the prompt used and output generated
  - For text generation you must provide either a link to the chat session you used to help write the content or an equivalent readout of the inputs you provided and outputs received from the system. You will lose credit if "the AI" is doing the work rather than you.
- Failure to follow the above rules is considered a violation of academic integrity, and is grounds for failure of the assignment, or in serious cases failure of the course.

- We use plagiarism detection software to identify similarities between student assignments, and between student assignments and known solutions on the web. Any attempt to fool plagiarism detection, for example the modification of code to reduce its similarity to the source, will result in an automatic failing grade for the course.
- If you have questions about what is and isn't allowed, post them to Piazza!