DS-GA-1011: Natural Language Processing with Representation Learning, Fall 2024

HW3 - Fine-tuning Language Models

Aditeya Baral
N19186654

Please write down any collaborators, AI tools (ChatGPT, Copliot, codex, etc.), and external resources you used for this assignment here.
Collaborators:
AI tools:
Resources:

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 draws inspiration from "NL-Augmenter", which can be accessed at `https://arxiv.org/abs/2112.02721`. Also acknowledgement to Nitish Joshi for his invaluable contributions to the initial version of the problems presented in this assignment.

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.

Due Data: This homework is due on October 30, 2024, at 12 pm Eastern Time.

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.

   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` eval)

   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` eval)

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

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.

   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.

   Judgment 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.

In this transformation, we propose 3 different approaches to creating synthetic examples from original training examples and attempt to simulate common linguistic errors found in text.

(a) **Random Lowercasing**: We randomly lowercase each word in a sentence with a sampling probability of $p = 0.5$. This helps to mimic situations where words might not have been capitalised correctly and helps the model learn similar and more informative feature representations for the word ”titanic” as well as ”Titanic” when they both occur in the context of the word ”movie”. This is especially realistic in scenarios where users may type quickly or use devices without automatic capitalisation. By introducing this, the model is exposed to both correctly and incorrectly capitalized words, helping it generalize better and not depend on capitalisation cues. This trains the model to understand that the case of a word does not necessarily change its meaning and enhances the model’s ability to ignore the case when it’s irrelevant leading to better generalization.

(b) **Synonym Replacement**: We randomly choose a noun (with lengths > 3 characters) with a sampling probability of $p = 0.5$ and replace them with their synonyms from WordNet. This increases the vocabulary the model can learn more informative representations about and adds linguistic and lexical variation to the training dataset while retaining the semantic integrity of the transformed and original example. Real-world inputs often contain words with similar meanings and synonym replacement exposes the model to alternative phrasings, preventing it from overfitting to specific terms and improves flexibility in handling synonymous words making the model more adaptable to unseen vocabulary. It thus helps the model tackle synonyms such as the words ”best” and ”greatest” when they occur in similar contexts and allows it to be more robust vocabulary variation.

(c) **Random Typo Insertion**: We randomly sample words with a sampling probability of $p = 0.5$ and further randomly replace up to half of the word’s constituent characters with those adjacent to it on the QWERTY keyboard. This is achieved by first storing a map of all keys mapped to its adjacent keys and then randomly picking one of the substituting characters if the sampled substitution probability is greater than $p = 0.75$. This transformation mimics the common typing errors made while entering text from a QWERTY keyboard and thus a realistic simulation of user errors, especially those made on physical or virtual keyboards where mistyping adjacent keys is common. It aids the model in tackling typographic errors and understanding when two similar words such as "Titanic" and "Totanic" refer to the same concept. Unlike random substitutions that produce gibberish, this approach introduces targeted errors to create slight deviations that resemble natural user input and focuses on variations it would likely see at test time, increasing its tolerance to noisy input.

Justification of Reasonability of Transformation Techniques:

(a) The above three approaches simulate common erroneous and non-erroneous behaviour of users and various linguistic variations and imperfections in text entered by a random user: missing or extra capitalization, differences in synonyms and random typos in spelling.

(b) Each transformation preserves the meaning of the original text. Lowercasing and synonym replacements retain the semantic structure and a controlled typo introduction keeps the words *mostly* recognizable. This ensures that the model isn't trained on data that is extremely different from the input distribution.

(c) These changes are realistic and reflective of real-world scenarios, representing what a user might input during test time. Introducing these transformations in a controlled way makes the model more robust to authentic and noisy input data, improving its real-world applicability and user experience.

2. (6 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` eval)

   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 (3 out of the total 6 points).
   - A decrease in accuracy of more than 4 points will award you the full 6 points.

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

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 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)

2. (4 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.

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

(a) Accuracy Report:
   - **Original Test Data**: 0.93208
   - **Transformed Test Data**: 0.87368

(b) Yes, the model's performance on the transformed test dataset **significantly improved** after applying the data augmentations. It **increased from** 0.84736 **to** 0.87368 ($\approx 3.11\%$ improvement).
   - This shows that the data augmentation technique was effective in helping the model handle various test set examples which it could not previously correctly classify, thus also pointing towards the fact that there were noisy and slightly altered inputs related to the training set in the test set.
   - During training, the data augmentation introduced the aforementioned noise and gave the model a chance to learn how to recognize and correctly interpret slightly altered text, helping it **generalize better to transformed or "noisy" test data** that includes similar modifications. This encouraged the model to recognize that meaning can remain consistent even when specific words differ slightly, building a kind of semantic flexibility that benefits both noisy and clean data.
   - By introducing small but realistic text variations, the model learned to handle a **broader range of input styles**, making it more **robust** to noisy data. This is evident in its improved performance on the transformed test data, as the model encountered similar variations during training. The 3.11% gain suggests that the augmentation closed a performance gap that would have otherwise caused the model to struggle with real-world, imperfect data.
   - However, the lack of further improvement is mostly because the augmentation only focused on surface-level changes and not deeper and more complex semantic and grammatical structure-level changes to the dataset. The augmentations **don't expose the model to entirely new vocabulary that might be specific to a different domain, nor does it introduce a different sentence structure** or contextual shifts.

(c) The model's performance on the original test dataset **marginally improved** as well. It increased from **increased from** 0.9254 **to** 0.93208 ($\approx 0.72\%$ improvement).
   - This indicates that the model retained its ability to perform well on the original training dataset comprising standard and clean data. It shows that data augmentation **didn't compromise the model's ability to perform on clean data**.

- The slight improvement suggests that the augmented training data may have helped the model become more flexible to noise such as random capitalisation, typos and other changes without reducing its accuracy on clean text. It allowed the model to observe varied examples of words and phrases without completely changing the context.
- This variety helped the model learn a more informative representation of words that still apply in clean data such as handling the range of synonyms for a given context. Additionally, the minor lowercase transformations helped the model become less sensitive to capitalisation, which can be an unnecessary restriction on model accuracy if relied on too strictly.
- For more significant improvements, the augmentation would need to incorporate a **broader range of linguistic and contextual variability**, allowing the model to generalize better to different forms of text.

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

(a) A limitation of the data augmentation approach employed above is that it only introduces **lexical and surface-level variations** of the original training examples.

(b) Transformations such as lowercasing, synonym replacement and typo introduction help introduce noise into the training data, but do not add further dimensions to the training dataset since they are ultimately *minor* variations of the original examples. Thus, they do not cover a full range of complex examples that differ **semantically** from existing training data and thus do not address deeper conceptual shifts and distributions of data in OOD scenarios in the real-world.

(c) The data augmentation approach simply aids the model to be robust to minor variations of the same sentence, ensuring there is consistency in the predicted label across various linguistic differences. It ensures that the **outcome remains constant regardless of the words in the sentence**. This allows all sentences with the words ”Titanic” and ”good” to preserve a positive sentiment, however if the topic is changed, it will under-perform.

(d) True OOD data can come from entirely different genres or topics (e.g., news articles, social media, etc). Surface-level augmentations don’t replicate these topic shifts, so the model will exhibit poor performance on texts that diverge in genre or content distribution. It **doesn’t develop the broad adaptability needed for cross-genre tasks**. Surface-level augmentations don’t prepare the model for these deeper shifts, limiting its generalisation ability across domains. This means the model only learns to deal with superficial variations within the same domain, rather than being exposed to fundamentally different sentence structures or concepts it might see in OOD settings.

(e) Further, since the changes are only at a word-level, the augmentation approach does not factor in various linguistic structures like jargons, phrases, unfamiliar terms, special domain-specific words and nuanced meanings beyond surface-level changes. OOD data often introduces specialised vocabulary or technical terms. Synonym replacement doesn’t expose the model to this kind of varied yet specialised vocabulary. Additionally, the concepts in OOD data can differ significantly from the data on which the model was trained. For instance, scientific texts often have longer, more complex sentence structures and frequent use of abbreviations. Since the augmentation approach does not alter the sentence structure, it also **does not affect the grammatical flow and syntactical structure of the sentence**, and thus will not be as effective if both the words as well as the grammatical structure of the input sentence is altered.

(f) Real-world data comprises various complex text variations including but not limited to surface-level changes. Without introducing new contexts and concepts, changing the underlying semantics, expanding the domain coverage, and altering sentence structures, the model cannot handle OOD test sets very well. While the augmentation approach used here helps the model handle noisy data and minor typographic errors, it does not provide the **deeper exposure needed to handle significant contextual, syntactic, and conceptual shifts** characteristic of true OOD data.

(g) For OOD generalization, the model would benefit from augmentations that introduce more substantial linguistic variation—such as transformations that simulate domain shifts, alter sentence structure, or introduce complex vocabulary—preparing it for the full range of variability found in diverse, real-world data.