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

HW3 - Fine-tuning Language Models

Name Chenxin Gu
NYU ID cg3423

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

   See coding

## 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.

For this assignment, I choose to introduce typos into sentences. This is a reasonable transformation because it's common that users mis-spelled when typing on keyboard. In order to response to this question, I introduced typos in original training dataset by setting a random key for substituting an original alphabet with one of its neighbors randomly on the keyboard, using the dictionary named "neighbors" in my code. The function custom_transform returns the transformed text with typos.

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.

   See coding

## 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)

   See coding

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`.

   - Accuracy for original test data: 0.929
   - Accuracy for transformed data: 0.87384
   - Accuracy for original test data after augmentation: 0.92664
   - Accuracy for transformed data after augmentation: 0.89332

   1) The model's performance increased on the transformed test data after applying data augmentation, accuracy increasing from 0.87 to 0.89.

   2) Augmentation does not affect the accuracy on the original test data very much. The original accuracy is 0.929 and after augmentation it becomes 0.927.

   By augmenting original training dataset by adding 5000 transformed examples, we also include typos for the model to learn before directly observing them in the transformed test dataset. Data augmentation makes the model more robust to similar typos in the transformed test dataset. However since the original dataset has high diversity naturally and might not include such much typos, the distribution of typos will potentially become unnatural thus the effect of data augmentation is not obvious in this case (in some cases might even negatively impact accuracy).

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

Within our data augmentation algorithm, we only introduced typos and it's a very limited improvement. The distribution we introduced might deviate too much compared to the natural distribution, which might cause the model to fit our hypothetical typo cases, but is not as robust on the real dataset.

There can be other cases in OOD problems, like synonyms and negative examples, as mentioned before. Only introducing typos cannot solve all OOD problems.