SI 630 -- Homework 5: Generation and Detection via Transformers

Nominally Due: Wednesday April 15, 5:30pm

Introduction

Pretrained language models like BERT and ELMo have revolutionized many areas of NLP. These models have been trained on massive volumes of text and their parameters reflect a basic understanding of the structure and meaning of many kinds of text. The killer application for such models is *fine-tuning*, where the parameters are modified to reflect a particular type of text (e.g., social media) and to use them to perform some specific task (e.g., classification). As a result, with just a limited amount of data, these models are capable of generalizing to a much wider set of instances; for example, a BERT-based classifier trained on a few thousand classifiers might perform substantially better than a logistic regression classifier trained on the same data. Homework 5 will get you experience with these kinds of models.

In the long long ago of Lecture 2, we learned about language models that could generate text and briefly played around with the Talk to Transformer website,[1] which can generate stories from the start of one. Wouldn't it be cool to try building and using that kind of technology yourself? In this assignment, you'll finally get face to face with some of the latest NLP deep learning libraries to do two tasks (1) generate the start of a short story and (2) detect whether a short story you've been given was generated by a machine or not. As a part of this assignment, you'll work with the HuggingFace transformers library which has implementations of many of the latest-and-greatest NLP models and, *conveniently*, has included example scripts for how to effectively fine-tune pretrained models to do both of these tasks.

The overarching goal of Homework 5 is to generate as realistic of a story start as you can from the generation model. However, you're also tasked with detecting these machine-generated stories, which creates a dilemma--you want your generation system to fool your classification system (making it think the stories are human), yet you also want the classification system to do well.[2]

Unlike previous homeworks, **Homework 5 is a two-person homework**. The intent is not that it's difficult (it's not) but that by dividing up the work for generation and classification, you can try

[1] https://talktotransformer.com/

[2] We'll talk about this concept much more in class as a part of a special kind of network called a Generative Adversarial Network (GAN) but you are *not* implementing that here.

testing the generation output on the classifier to see if it can fool it. Most of the homework effort will consist of familiarizing yourself with how to run these models and will likely not involve writing much code (unless you want to go wild on creating cool generators/classifiers, which I fully support).

Homework 5 has the following (very practical) learning goals:

1. Learn how to use modern deep learning libraries like `transformers` to fine-tune pretrained models like BERT (for classification) and GPT (for generation)
2. Learn how to run your code on a GPU (as provided by Great Lakes)
3. Try your hand at tuning hyperparameters to make better models

In summary, we want to help you get familiar with the basics of these models in a way that gives you skills you might use on a job, a course project, or in a research setting (e.g., fine-tuning a BERT classifier for your particular domain).

Part 1: Generating The Start of a Story

For Part 1, you'll fine-tune one of the transformers from the HuggingFace library[3] to generate text. I personally recommend the `openai-gpt` model, as it's small enough to reliably fit on the GPUs given to you on the Great Lakes clusters.[4] The library provides a sample script in the examples directory called run_language_model.py which you can/should use to do your training.

We have provided example stories from various sources for you to use in three files in the data/generation/ directory: train.txt, dev.txt, test.txt. You should use the train.txt to fine-tune your model, choosing hyperparameters that converge to a low loss. To figure out the right hyperparameters, you should use the dev.txt to select the best hyperparameters that give the lowest perplexity (reminder: perplexity is the metric we discussed back in Lecture 2 that measure how surprised the model is by the words that come next; a lower perplexity indicates the language model expects to be generating this kind of word, which is what we want).

What to do:

- Train a model using train.txt and save it to some output directory. You can/should train multiple models and evaluate them using dev.txt to pick the model with the lowest perplexity on the dev.txt set.
- Report the following
    a. The loss on train.txt for your best model and its perplexity (on train.txt)
    b. the perplexity on dev.txt
    c. once you have finalized the model, the perplexity on test.txt

[3] https://github.com/huggingface/transformers
[4] for those with more powerful GPUs, maybe try `gpt2`

- Using your final model and random seed 9001,[5] Generate story starts of length 32 for the following 10 prompts:
    a. My
    b. The
    c. One
    d. When
    e. If
    f. Our
    g. First
    h. Natural
    i. We
    j. Because

Implementation notes:

- You're welcome to add data to train.txt should you want to go off and collect more. Go wild with stories from different domains, styles, authors, etc. The world is your textual oyster.
- One of the big slow-downs in fine-tuning is converting the text into features, which takes 5-10 minutes. However, if you're using the same model over and over, the code will save these featurized data to a pickle and save lots of time.
- The block_size parameter specifies the maximum length of any sequence. With a smaller block size, you can fit more instances in a batch (potentially giving better convergence) but with a larger sequence, you get more text for the model to learn from. It's a trade-off you'll need to explore.

Part 2: Training a classifier to recognize machine-generated text

Part 2 will develop a classifier to recognize machine-generated text. You'll once again use the transformers library to train a model. I personally recommend the `distilbert-base-cased` model, as it's small enough to reliably fit on the GPUs given to you on the Great Lakes clusters. This particular model is a pared-down version of BERT with fewer parameters, however you're welcomed to try out different models!

The huggingface library doesn't provide a script to train this off the shelf. However, there is a fantastic example notebook that will walk you through all the steps to train a classifier here: https://colab.research.google.com/drive/1pTuQhug6Dhl9XalKB0zUGf4FldYFlpcX In particular, your task will involve two main steps:

1. Pretrain your BERT model to recognize the language of short stories

[5] Technical justification.

2. Train a classifier on the encoded to text

For #1, you should use the training data from the data/generation/ directory to fine-tune your model. For #2, you'll use the three provided files in the data/classification/ : train.json.txt, dev.json.txt, and test.json.txt.

You will need to write some code to load the data yourself and assign the classification label of each sentence (check out 3.3 in the example notebook for where to get started). This is the majority of the code you'll need to write for this homework (aside from code that is copied from this notebook).

Important note: Be sure to use a sequence length of 32 when training your classifier, as we're going to set up our prediction task to only look at the first 32 words of a story to decide whether it was written by humans or machines.

What to do:

- Fine-tune your BERT model (or the DistilBERT model) and save that to a directory
- Using your fine-tuned model, train the same model for classification using the train.txt in data/classification and save it to some output directory. You can/should train multiple models and evaluate them using dev.txt to pick the model with the best classification accuracy.
- Report the following
    a. The accuracy on train.txt for your best model
    b. the accuracy on dev.txt
    c. once you have finalized the classification model,
        - the accuracy on test.txt
        - the accuracy on the 10 sentences you have generated from Part 2

What to submit

One person should submit the following items to canvas **for both people**:

1. A group report (one per team) with
    a. Answers to all the questions (all the models' performances)
    b. The 10 generated sentences
    c. Your classification model's performance on those 10 sentences
2. All the code for your generation and classification systems