Assignment 3: Training a Transformer Language Model and using it for Classification

Deadline: Monday October 24, 2022 at 9:00pm
Late Penalty: There is a penalty-free grace period of one hour past the deadline. Any work that is submitted between 1 hour and 24 hours past the deadline will receive a 20% grade deduction. No other late work is accepted.

Welcome to the third assignment of ECE 1786! The goal of this assignment is to explore the use of Transformers as language models, and as classifiers. You will be using code that has already been written, and so a big part of the work will be understanding the code, and how it links to the material in the lectures. This assignment must be done individually. The specific learning objectives in this assignment are:

1. Understand the language modeling objective.
2. Become familiar with a PyTorch code implementation of a Transformer model.
3. Pre-train the model using the language modeling objective (predicting the next word of a sequence), with a small corpus, then a larger one.
4. Observe the output probabilities of the pre-trained model on a few examples.
5. Fine-tune the pre-trained transformer model for sentiment classification.
6. Become familiar with the Huggingface model hub, and how to fine-tune large pre-existing models with the Huggingface trainer class.

What To Submit

You should hand in the following files to this assignment on Quercus:

- A PDF file `assignment3.pdf` containing answers to the written questions in this assignment. You should number your answer to each question in the form `Question X.Y.Z`, where X is the section of this assignment, Y is the subsection, and Z is the numbered element in the question. **You should include the specific written question itself** and then provide your answer.
- Code files as specified in some individual questions.

1 Karpathy's minGPT [12 pts]

We will make use of Andrej Karpathy's code repository of the GPT-style Transformer code, which is a simplified PyTorch implementation of GPT-2 [1] made more readable, but less powerful, as Karpathy illustrates in Figure 1. A revised version of this code is provided in the assignment zip file which builds and trains a full language model. Below we give an overview of the code that is provided. This, together with the recent lectures, will help you answer the questions relating to the code below.

Figure 1: Karpathy's View of His Code vs. Others

1.1 Structure of Provided minGPT Code

In the zip file associated with this assignment, you will find a notebook `LM.ipynb` which imports several python files from the folder `mingpt`. Below we explain the role of each of this notebook and python code files. You may choose to convert your code into only python files, or perhaps one very large notebook.

- `LM.ipynb` This notebook sets up the data used for training, sets the key hyperparameters and performs the training. After storing the trained model in a file, it runs the 'generate' function from models.py to predict the words that come after a few starter words.
- `mingpt/bpe.py` This file contains the tokenizer that is used by minGPT, and most Transformers, in an approach called 'byte-pair encoding.' This method is described in the Jurafsky [2] textbook, Section 2.4.3. You will not need to look too deeply at this code, except to note that the vocabulary is actually downloaded the first time you use this code, and you can review in your home folder `~/.cache/mingpt`, where `~` is your home folder. One surprising thing I found was that some words are repeated, with a special character in front in the repetition.
- `mingpt/model.py` This file contains the transformer model definition, built up carefully on a set of sub-modules, include GELU, CausalSelfAttention, Block, and finally GPT. It also contains the `generate` function which will predict a sequence of words following one or more seed words.
- `mingpt/trainer.py` This contains the code to train the model. It also has a method for setting up the default hyperparameters of the training/network.
- `mingpt/utils.py` This file has some utility functions mostly related to setting up the configuration parameters/hyperparameters.

1.2 Questions

Read through all of the code to get a feel for it. This section helps you take in and learn the code by asking you to find parts of code, or explain it, or to relate the code to the material in the lectures of this course.

1. Which class does `LanguageModelingDataset` inherit from? (1 point)
2. What does the function `lm_collate_fn` do? Explain the structure of the data that results when it is called. (2 points)

3. Looking the notebook block [6], (with comment "Print out an example of the data") what does this tell you about the relationship between the input (`X`) and output (`Y`) that is sent the model for training? (1 point)

4. Given one such `X,Y` pair, how many different training examples does it produce? (1 point)

5. In the generate function in `model.py` what is the *default* method for how the generated word is chosen? (1 point)

6. What are the two kinds of heads that `model.py` can put on to the transformer model? Show (reproduce) all the lines of code that implement this functionality and indicate which method(s) they come from. (2 points)

7. How are the word embeddings initialized prior to training? (1 point)

8. What is the name of the object that contains the positional embeddings? (1 point)

9. How are the positional embeddings initialized prior to training? (1 point)

10. Which module and method implement the skip connections in the transformer block? Give the line(s) of code that implement this code. (1 point)

2 Training and using a language model on a Small Corpus [12 pts]

After your review of the code, you're ready to train the model on the file `smallsimplecorpus.txt` from Assignment 1. The code as given in the notebook will train the model using that file as input.

1. Run the code up to the line `trainer.run()` and make sure it functions. Report the value of the loss. (1 point)

2. Run the two code snippets following the training that calls the `generate` function. What is the output for each? Why does the the latter parts of the generation not make sense? (2 point)

3. Modify the `generate` function so that it outputs the probability of each generated word. Show the output along with these probabilities for the two examples, and then one of your own choosing. (1 point)

4. Modify the `generate` function, again, so that it outputs, along with each word, the words that were the 6-most probable (the 6 highest probabilities) at each word output. Show the result in a table that gives all six words, along with their probabilities, in each column of the table. The number of columns in the table is the total number of generated words. For the first two words generated, explain if the probabilities in the table make sense, give the input corpus. (5 points)

5. Submit your code for `generate` in the file `A3_3.2.py`. (3 points)

3 Training on the Bigger Corpus and Fine-Tuning Classification [8 points]

In this section you will train the same model on a larger corpus, and then take the trained model and turn it into a classifier. You will then fine-tune the classifier to do sentiment analysis.

1. Modify the notebook to train on the larger corpus, by un-commenting the other two dataset line as given in the comment at the end of the block [4] of `LM.ipynb`. This dataset needs significantly more training, and as suggested in the notebook, set the number of iterations (`train_config.max_iters`) to be 100,000 and the batch size to 16. Run the training; it will take significantly longer. If you're using google colab, you should be able to make the training run faster by switching the runtime type to be GPU. This code will automatically use the GPU.

   If you are short on time, you can instead just load a model that has already been trained from the saved model file here. Report which of these two methods you used - trained yourself, or loaded the saved model.

2. Check that this model can generate words by seeding the generate function with a few examples different from the ones given. Report the examples you used and the generation results, and comment on the quality of the sentences. If you trained your own model, be sure to save it for re-use in the fine-tuning process below. [2 points]

3. Next, the goal is to convert this trained model into a classifier, and fine-tune it on another dataset to perform a new task. The classifier will be trained to determine the *sentiment* of a sequence of words predicting whether the sequence is positive sentiment (label 1) or negative (label 0). We will use the Stanford Sentiment Treebank dataset (`sst2`), which is part of the General Language Understanding Dataset (referred to as glue) provided by Huggingface.

   You'll need to install the `datasets` library from Huggingface using: `pip install datasets`. The command `datasets.load_dataset("glue", "sst2")` loads the Stanford Sentiment Treebank (binary classification) dataset from the Huggingface server. Use the first 1200 samples from the *train* split, and then use the train_test_split method in scikit-learn to split them into train and validation sets. Be sure to stratify the splits.

   Report the training and validation curves for the fine-tuning, and the accuracy achieved on the validation dataset. Submit your new code - the new notebook and all the `.py` files in a zip file named `A3_3_2.zip`. [6 points]

   Important: If you are able to write the code given to perform this classification task using your own knowledge, just go ahead and do that. Below we give some general guidance on what to do, but do not take this as a specification of what must be done. (Here I'm trying to avoid the questions on Piazza 'do we have to do it this way?' The answer to that question is no, as there are many ways to get this task done. With that, here is the guidance:

   - Create a new `Dataset` object (not to be confused with the very-similar-looking lower case Huggingface library `dataset`) similar to the `LanguageModelingDataset` in the provided code. This object should download the dataset as follows:

     ```
     import datasets
     sst = datasets.load_dataset('glue', 'sst2')
     ```

     You will need to create a training and validation split as indicated above.

- You will need to make use of the code in the given model that uses a classifier head rather than a language modelling head to the model. As you can see in the code, the two heads are both `nn.Linear` layers with the same input dimensions, but the output dimension of the LM head equals the vocabulary size while the output dimension of the classifier head equals the number of classes (in this assignment, 2). In the forward procedure of `model.py`, the code that you were given for the pre-training uses the language modeling head. The classifier head should be used during fine-tuning.

  Note that in the data input shapes are different in the pre-training versus the fine-tuning, as they are quite different tasks. During the language modeling pre-training, each token gives rise to a corresponding label at the output - the word to predict. However, for the classification fine-tuning each input sequence corresponds to only one label at the output. To make this work, you will need to write a new `collate_fn` that is used as part of a new trainer object (from class Trainer).

  The `trainer.py` script will need to be modified to allow language modeling training and fine-tuning training as well.

4 Fine-Tuning Using Huggingface Models and Library [5 pts]

The fine-tuning work of in the previous section involved some complex code that needed to be customized. Fortunately, there exists a library of models and code that make this process much easier.

1. Read (and watch) the tutorial from Huggingface on how to use their model hub and their datasets, here. This tutorial shows you how to both download a pre-trained model and to fine-tune it, using Huggingface's AutoModelForSequenceClassification class. (It also shows how to do a more detailed-level training with lower-level pytorch).

2. Modify that tutorial to implement a fine-tuned sentiment classifier using the same dataset that was used in Section 3. Rather than the one used in the tutorial, use the smallest version of a pre-trained GPT2 you can find.

   Report the classification accuracy on the validation set. Comment on the performance of this model: is it better than the model you fine-tuned in the previous section? Submit your code either as a notebook named `A3_4_2.ipynb` or a python file with a similar name. [5 points]

References

[1] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

[2] Dan Jurafsky and James H. Martin. Speech and language processing : an introduction to natural language processing, computational linguistics, and speech recognition, 3rd edition draft. *https://web.stanford.edu/ jurafsky/slp3/*, 2022.