Homework 5

11-967

Due date: 02/23/2025 11:59 PM EST

In this homework, you will investigate existing pre-training datasets and replicate a data pre-processing pipeline that converts raw web-crawled data into a cleaned dataset that is easy and efficient to train on. You should be able to briefly understand (a) attributes of large-scale pre-training and their formats of availability [*Problem 1*] and (b) different steps involved in cleaning, reducing, and diversifying pre-training data [*Problem 2.1-2.2*] and (c) converting your pre-training data into a standardized format for improved use in training and accessibility [*Problem 2.3*].

You will submit your code `mini_ccc.py` and `homework.py` to the coding assignment, and your `report.pdf` with your answers to the written assignment to Gradescope. Please be as concise in your responses as possible. Note: this homework requires about 3GB of disk space. You don't need a GPU for this.

Problem 1: Understanding Training Data

[Question 1.1] (***Writing, 5 points***) Pick two datasets from the following and answer the questions below for each of them.

- Dolma
- C4
- Project Gutenberg
- The Pile
- RedPajama V1
- The Stack

DELIVERABLES FOR Q1.1

For your two chosen datasets, answer the following questions. ***Please answer as concisely as possible (2-3 lines).***

A. When is the "knowledge cutoff" for your chosen dataset, that is, the latest creation date of any document in the dataset.

B. Where was the data sourced? Does the dataset consist primarily of a single type of content or does it have multiple sources?

C. Identify an open-sourced language model that was trained entirely or in part on this dataset.

D. What license was the dataset released under? Do all documents in the dataset have the same copyright protections (or lack thereof)? If yes, describe these protections *briefly*. If no, give an example of two portions of the dataset that may have different protections.

E. Name a task that a language model trained solely on this dataset might perform poorly at, even with finetuning. Give reasons to support your answer.

[Question 1.2] (***3 points***) Mitchell et al. (2019) proposed the idea of Model Cards, documentation accompanying every released machine learning model that details its performance and training characteristics. Though nearly all state-of-the-art large language models release Model Cards, the amount of information present in them can vary substantially. Locate the Model Cards for three language models, one that meets each of the following descrpiptions.

- A language model that is only accessible via an API; it does not have public weights.

- A language model that has publicly accessible weights, but the pre-training dataset is not publicly available.
- A language model that has publicly accessible weights and a publicly accessible pre-training dataset.

Then, answer the questions based on the Model Cards you located.

DELIVERABLES FOR Q1.2

Concisely answer the following questions.

A. What are the three models you have chosen?

B. Compare and contrast what the three Model Cards say about the data they are trained on and how it was processed.

C. Discuss a language model use case where you might opt for a language model trained on publicly accessible data, even if its performance is worse than a closed source model.

Problem 2: Training Data Preprocessing Pipeline

The Common Crawl is a public index of the internet. It has been used as a source of internet data for nearly all open-source language models. Language model trainers first however have to take the raw data released by Common Crawl and process it into training-ready datasets. For example, to train T5, Common Crawl was processed into the dataset C4, and to train OLMo, it was processed into the dataset Dolma. In this problem, you will implement a data processing pipeline for Common Crawl data. You will start with the raw files released by Common Crawl and end up with cleaned text in a dataset format that it ready to be trained on.

[Question 2.1] (*Coding, 6 points*) Common Crawl data is split across three file formats:

- The Web ARChive (WARC, for short) format is used to store the HTTP responses for urls included in the crawl.
- The WARC Encapsulated Text (WET, for short) format contains plaintext extracted from each webpage.
- The Web Archive Transformation (WAT, for short) format is used to store metadata around each request.

The CommonCrawl is broken up into thousands of *slices*, each of which contains a small fraction of the total data. We have provided you `download_data.sh`, which downloads the three file formats for a single shard of a 2018 Common Crawl dump. Your task is to start from the WARC file and implement a data processing pipeline that turns the raw HTML dump into formatted and cleaned text that is ready for model training. You are already given the `read_warc_file` function in `utils.py`. This function reads the warc file you downloaded and returns a dictionary containing a URL as the key and the raw HTML as its value. After reviewing this function, make the following modifications to `homework.py`.

1. Implement the `html_to_text` function. This function should take as input the raw HTML and output plaintext without any HTML tags. You are allowed to use any of the following packages. If you would like to use another package not listed here, please write a Piazza post with your request, explaining what your desired package does that is different than these ones.
   - html2text
   - nltk
   - BeautifulSoup
2. Modify the code in `__main__` in order to answer the questions below. Your code for this will not be graded.

DELIVERABLES FOR Q2.1

Answer the following questions. ***Please answer as concisely as possible (max 3 sentences).***

A. How many HTML pages are in this shard of the crawl? (Note: you might need to wait for a few minutes to get the answer)

B. How does your `html_to_text` handle the following HTML components: headers (e.g. `h1`, `h2`, etc.), embedded images (e.g. the `img` tag), and tables?

C. Inspect the WET file either by running `less data.wet` or by using the `read_wet_file` in `utils.py`. Based on your observations, discuss some ways the html-to-text conversion implemented by the Common Crawl team differs from what you implemented. Which version do you think makes for better language model training data?

[Question 2.2] (***Coding, 8 points***) For the remainder of this homework, we will refer to the web-pages that have been converted to plaintext as *documents*. It is typical to filter training datasets to only contain "high quality" text. Filtering can either be at the document level–removing entire documents which don't meet the quality bar—or it can be applied within each document—removing or altering substrings within the text. There are several ways of measuring quality, including using an automatic classifiers, heuristics such as document length, and selecting only from certain web domains. In this question, you should implement a method for filtering out individual lines and entire documents if they do not abide by simple conditions designed to judge their likelihood of being useful text. You can read more about the motivation of these conditions in Gopher and RedPajama. You will also be implementing a method to mask Personal Identifiable Information in order to limit regurgitation of personal information by language models trained on your data. You will continue modifying `homework.py` and implement the following methods: *Note: use `string.punctuation` for all punctuation checks.*

- `heuristic_quality_filter`: This method should return `True` (meaning a "high quality" text) iff the document

  - contains no words from the bad words list located here. For convenience, we've included this file in the homework starter code, as well as a function in `utils.py` to read it in. AND

  - contains punctuation. AND

  - contains non-whitespace characters. AND

  - at least 80% of characters in the document are one of: alphanumeric, punctuation, whitespace.

- `remove_pii`: This method should modify the input string to remove personal identifiable information.

  - Transform any occurrences of a US social security number of the form XXX-XX-XXXX, replacing all digits with the letter X.

  - Transform any 10-digit phone numbers (a 10-digit number preceded by +1) by replacing all digits with the letter X.

- `clean_text`: This method should remove low-quality portions of the text by performing the following operations. For the purpose of this question, let's define a paragraph as text separated from other text by a newline, that is: `paragraphs = text.split("\n")`

  - Remove any paragraphs that contain >100 alphanumeric characters with no whitespace between them.

  - Remove any paragraphs that do not contain punctuation.

DELIVERABLES FOR Q2.2

Answer the following questions. ***Please answer as concisely as possible (max 3 sentences).***

A. Follow the data cleaning and filtering order in the provided `__main__` function. How many documents were deleted in total for WARC? Assume there are 3 billion documents in the Common Crawl, and use this to estimate the percentage of the CommonCrawl which would be considered low-quality according to your classifier.

B. Identify two documents you perceive as low-quality which was not caught by your filter. Would it be possible to design heuristic-based filter for documents like these? Do you think it would be more or less effective to train a classifier to identify similarly problematic documents? Explain your reasoning.

C. How do the filtering and cleaning methods you have implemented impact the inclusion of non-English text in the final dataset?

D. You have implemented cleaning and filtering for webpages. Name a different data domain that would require designing domain-specific data cleaning and quality filters. Describe *one* cleaning and *one* filtering methods you might implement in your chosen domain.

E. The survey paper at this link describes a variety of methods for data filtering. You've already implemented some stages that they discuss in their overview. Identify and discuss a data cleaning stage that you've not implemented yet that is known to improve training efficiency by reducing data redundancy.

[Question 2.3] (***Coding, 6 points***) The HuggingFace Datasets library is a popular framework for distributing datasets that can easily be used for model training. In this question, you will convert the dataset you created into the HuggingFace Datasets format. You will find the documentation at this url useful.

You are given `mini_ccc.py`. Make the following modifications to it.

1. Implement the `_info` function. Your implementation should return a `DatasetsInfo` object with a `description` of the dataset and a list of `features` passed in as arguments.
2. Implement the `split_generators()` function. This function should download the .warc file using the `dl_manager` and label it as a "train" split for your data. Try and be as consistent with the intended nomenclature of splits as described in the huggingface documentation for Splits.
3. Implement the function `generate_example()` function. This function should read your `.warc` file and yield a tuple containing a url and its corresponding processed plaintext. Note that you **should use** the functions implemented in `homework.py` and `utils.py` for reading and processing the `.warc` file. The output of this function should be the cleaned documents which pass the quality filter.

DELIVERABLES FOR Q2.3

Answer the following questions. ***Please answer as concisely as possible (max 3 sentences).***

A. How long in seconds does your code take to process the entire dataset. Assume there are 3 billion documents in the 2018 CommonCrawl. Estimate how long it would take your code to process the entire CommonCrawl.

B. Describe how you could modify your code to make it significantly faster.

C. To actually train on your dataset, it needs to be tokenized and turned into batches. The size of a batch is the maximum sequence length for the Transformer model being used. One common way to batch data is by *packing* examples. When documents are longer than the max sequence length, they are broken up across multiple batches. When documents are shorter than the max sequence length, multiple are concatenated together into a single batch. Describe an advantage of using packing over the approach we discussed in class of padding too-short sequences with a no-op token.