Mainpipe

The project contains two pieces of code:

The data acquisition code downloads four datasets from '*huggingface'* and stores them locally into *'json'* files. The datasets are within the following subsets:

- PubMed Abstracts
- Web crawl
- GitHub
- Wikipedia

They can be found in:

- https://huggingface.co/datasets/devanshamin/PubMedDiabetes-LLM-Predictions
- https://huggingface.co/datasets/kaifahmad/allenai-complex-web-QnA
- https://huggingface.co/datasets/ed001/ds-coder-instruct-v1
- https://huggingface.co/datasets/agentlans/wikipedia-paragraph-sft

The search was filtered to retrieve English texts in *'json'* format. Since the files are stored locally, the data acquisition code is separated from data preparation code. Note that the files in *'github'* (stored in Dataset-Samples directory) contain only the first 5 lines of the original files to keep it small enough to be uploadable to github.

The data preparation code contains 4 functions. I kept the functions close the pieces of codes calling them to make code inspection more convenient. The first two functions are:

- normalize(line): which normalizes a line of text. It uses a *'unicode'* normalization form (see https://unicode.org/reports/tr15/).
- prmpt_rspns_split(raw_dic_lst, prmpt_key, rspns_key, is_code): which takes a list of raw dictionaries and returns a dictionary list with normalized and tokenized 'prompt' and 'response' keys. It checks if the text is code, then keeps the source text.

The code uploads the pre-stored json files, makes lists of 2D tuples of 'prompt' and 'response' texts as the input and output of the LLM and aggregates all four lists to one text dataset.

The next part is a histogram, with logarithm vertical axis, which illustrates the distribution of sentences length.

Train, test and validation sets come next in which, 15% of dataset are allocated to validation set, 15% to test set and 70% to training set. Validation set is required for model tunning and overfitting tests.

Plan for upscaling:

The scale of the dataset can be adjusted by changing the vocabulary size of prompt and response parts of the training set. Since the bank of words in machine responses must be more comprehensive than the prompt (which are usually small sentences) then the vocabulary size for prompt and response are set to 30000 and 100000 respectively. The output_mode is set to 'tf-idf' (lines 118 and 126) till adapt() (lines 132 and 133) learn the document frequencies of each token in the input dataset. When scaling up the model the vocabulary size can be reduced. The sentence length also can be limited to a few words (e.g. 50) but in this case the output_mode needs to be set to 'int'. Uncomment lines 119, 120, 127 and 128.

The next part is vectorizing the sentences. Each model can use its own vectorizer by building a mapping table of each unique token to a unique integer. I used ‘TextVectorization’ in ‘tensorflow.keras.layers’ which is widely used by data engineers.

The next function, shape_ds(prmpt, rspns), formats the prompt response pairs into a dictionary with the source and target data for training. The source is made of encoding input (prompt) and the decoding input (response) of previous sentences, and the target is the response of the last sentence until the model can update the parameters.

And finally, make_ds(pairs, batch_size) function makes the dataset with a format defined in shape_ds function and map it to a tensor with a right shape for the model. For small datasets the batch size can be set to 64 or lower, but as the ration of dataset size to hardware capacity raises, it can take larger values.

In the last parts, the training and validation datasets are made, and the shape of the encoder and decoder are printed out for double checking.

11/09/2025

Kian Keshavarzian.