Project 2: Building Large Language Models

CS324 (Winter 2022)

Language models are trained on raw text and therefore lack certain properties (e.g., controllability, ability to perform certain tasks, generalizability to various domains). In this project, you will take an existing language model (GPT-2) and try to instill certain properties by continued pre-training on a custom dataset.

1. In part 1, you will try to instill GPT-2 with *length controllability*, the ability to generate a sentence of a certain length.
2. In part 2, you will try to instill GPT-2 with the focal property that you proposed in Project 1.

In both cases, you will perform analysis to see if continued pre-training was effective.

You should think of part 1 as a warmup to get you familiar with the code, but you should focus most of your effort on part 2 and treat it like a final project.

In this project, you will use CodaLab Worksheets, a platform for running reproducible experiments, which will importantly give you access to GPUs (Google Cloud in our case, but the nice thing is that the CodaLab user can be agnostic to the underlying compute). **Please follow the instructions in this setup guide carefully**:
https://docs.google.com/document/d/1rWgWyYJc36Vow2eU8oZ0pfVU7SzFRg3Eb1dB1XHZzvE/edit?usp=sharing

1 Part 1: length controllability

In this part, you will try to instill GPT-2 [1] with length controllability, the ability to specify how many words a generated sentence should have. Other forms of controllable generation have been explored in previous works [2, 3].

Continued pre-training and dataset. We instill length controllability into GPT-2, we use *continued pre-training* [4, 5], where we start with the already-trained GPT-2 model and continue training on a dataset that demonstrates the desired property.

We first create a custom dataset, OpenWebText-WordLength, as follows. Define the following transformation, which takes a sentence and annotates it with the number of words:

$$\texttt{[sentence]} \quad \Rightarrow \quad \texttt{<len> [\# words] <text> [sentence]} \tag{1}$$

where `<len>` and `<text>` are special tokens, and `[# words]` is the number of words in `[sentence]`.[1] For example:

$$\text{Original: } \texttt{Alice went to the market.} \tag{2}$$

$$\text{New: } \texttt{<len> 6 <text> Alice went to the market.} \tag{3}$$

To construct OpenWebText-WordLength, we randomly sample a 1/6 of the sentences from OpenWebText and apply the above transformation.[2]

You will be given a trained GPT-2 and asked to perform continued pre-training on OpenWebText-WordLength to produce a model which we will call GPT-2-len.

[1] Words are defined by the NLTK word tokenizer — punctuation counts as words.
[2] Recall that GPT-2 was trained on WebText, which is a similar but closed version of OpenWebText.

Using GPT-2-len for length controllability. To generate a sentence with a certain number of words, we can now prompt GPT-2-len with the given number of words. For example, if we want 6 words, then we use the prompt:

```
<len> 6 <text>
```
(4)

The model will produce a completion conditioned on this prompt, which hopefully respects the provided length directive. Note that there is no guarantee that the resulting text will have 6 words—how close you get will be something you will explore!

Note: The HuggingFace generation code produces completions that includes the prompt. You'll need to be careful to remove the prompt and remove any text after further `<len>` tags that the model outputs after finishing the current sentence.

Metric for length controllability. To evaluate how well the model has mastered length controllability, we define an error metric. We first generate a target vector $y \in \mathbb{N}^n$ of n integer lengths. We use the model to generate sentences according to each target length, and we let the predicted lengths of each sentence to be $\hat{y} \in \mathbb{N}^n$. The word length for each word should be calculated by using NLTK's `word_tokenize` function to split a sentence into words. Then define the metric to be the mean absolute error of the generated lengths versus the desired lengths:

$$\text{Err}(y, \hat{y}) = \frac{1}{n} \sum_{i=1}^{n} |y_i - \hat{y}_i|. \tag{5}$$

Deliverables. The staff has set up a pre-processed version of the OpenWebText-WordLength dataset and the code for training a 12 layer GPT-2 model (`gpt2` in the HuggingFace model hub, roughly half the size of `gpt2-medium`). The code is built on HuggingFace Transformers. CodaLab command templates are in `scripts`; you'll need to run `scripts/cl-train.sh` and `scripts/cl-evaluate.sh`.

1. Methodology: Select training hyperparameters (such as the learning rate and batch size) so that the model learns the length controllability property within 50000 training steps. This may take about 24 hours (the staff is working to improve this time). You can run for a shorter number of steps, but as a ballpark, your final average evaluation error metric should be less than 1.5. We found this to be achievable within 10000 steps. Longer training should decrease this error further. In your writeup, describe your process for deciding on the best hyperparameters (e.g., which hyperparameters you tried and how you decided on best hyperparameters). Report the training and validation perplexity of your language model.

2. To show how well the model learns length controllability throughout training, load the model checkpoints saved throughout training and compute the error metric for each saved checkpoint. We provide 5 sets of test prompts for you to use for this evaluation in `wordlength_eval_data/` — record the mean and standard deviation of the metric over the 5 sets. We also provide some code for evaluation (`src/evaluate_wordlength_model.py`), which outputs results over the 5 evaluation sets in a file called `results.json`.

3. Experimental results: Make a plot (with the mean and standard deviations shown in the plot) where the x-axis is training steps (include 0 training steps in the plot) and the y-axis is the error metric for length controllability. Discuss conclusions from this plot in the writeup.

4. Error analysis: Perform an error analysis on the generated sentences. Include 5-10 example generated sentences. For what sentence lengths does length controllability work or not work?

5. Supporting materials: Submit code for all the experiments in this section, as well a link to your CodaLab worksheet. Make sure to annotate your CodaLab worksheet with textual descriptions (it should not be just a list of bundles).

2 Part 2: Focal property

We will follow part 1, except with your focal property rather than length controllability. You will create a new custom dataset (analogous to OpenWebText-WordLength for length controllability) and perform continued pre-training on that. By default, we only expect that a change to the dataset is necessary. However, you should feel free to also change the way that the model is trained, or even what model you start with; treat this like a final project so feel free to customize and make this part your own! If you do decide to do something very far from the general flow of Part 1, please make a post on Ed to the staff for approval.

We provide a raw version of OpenWebText and our code for processing it into OpenWebText-WordLength in case it is helpful.

Deliverables.

1. Overview: Provide an overview of the focal property of interest, and define the evaluation metrics for the focal property as done in Project 1.
2. Dataset: Describe the custom dataset that you used for continued pre-training, including (i) high-level motivation (why did you choose this one), (ii) basic statistics of the dataset, (iii) concrete examples from the dataset, and (iv) a thorough description of how this dataset was created.
3. Experimental details: how did you train the model and how you chose hyperparameters (similar to part 1)?
4. Results: After training the model, how did your model compare to the vanilla GPT-2 model on your evaluation metrics? Discuss any conclusions from the results, and provide example generated text for qualitative analysis.
5. Supporting materials: Submit the code for all your experiments in this section, as well as a link to your CodaLab worksheet. Make sure to annotate your CodaLab worksheet with textual descriptions (it should not be just a list of bundles).
6. At the end of the write-up, please include an authorship statement detailing the contributions of the members of the group.

3 Tips for training and development

1. During development, you should run on small datasets to keep the iteration cycle time low. You should track metrics such as the train loss and the validation loss, and make sure that 1) the train loss is going down, and 2) that the validation loosely tracks the train loss. Inspecting some generations from trained models can also be helpful to see if you're going in the right direction.
2. Take a look at GPT-2's default hyperparameters to know how the initialized weights were obtained. This information is available on HuggingFace: https://huggingface.co/docs/transformers/model_doc/gpt2
3. A general strategy for increasing the batch size is via *data parallelism*, which parallelizes across GPUs by having one copy of the model on each GPU and allowing each GPU to see a different minibatch of data. The gradient updates across GPUs are combined before they are applied to the model copies. In HuggingFace, we can specify the *per-device* batch size and freely scale up the number of GPUs. The tradeoff here is that increasing the number of GPUs will deplete your compute quota more quickly.
4. Another strategy for increasing the batch size, but trades off for sequential computation time, is gradient accumulation. In gradient accumulation, we run multiple minibatches one-by-one through the model and compute the average of the gradients across the minibatches. The effective batch size is then multiplied by the number of accumulation steps, but now every step of training will be slower by a factor of the number of accumulation steps. Gradient accumulation can also be combined with data parallelism.

References

[1] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. *OpenAI Blog*, 1(8), 2019.

[2] Nitish Shirish Keskar, Bryan McCann, Lav R. Varshney, Caiming Xiong, and Richard Socher. CTRL: A conditional transformer language model for controllable generation. *CoRR*, abs/1909.05858, 2019.

[3] Armen Aghajanyan, Dmytro Okhonko, Mike Lewis, Mandar Joshi, Hu Xu, Gargi Ghosh, and Luke Zettlemoyer. HTLM: hyper-text pre-training and prompting of language models. *CoRR*, abs/2107.06955, 2021.

[4] Suchin Gururangan, Ana Marasović, Swabha Swayamdipta, Kyle Lo, Iz Beltagy, Doug Downey, and Noah A Smith. Don't stop pretraining: adapt language models to domains and tasks. *arXiv preprint arXiv:2004.10964*, 2020.

[5] Colorado J. Reed, Xiangyu Yue, Ani Nrusimha, Sayna Ebrahimi, Vivek Vijaykumar, Richard Mao, Bo Li, Shanghang Zhang, Devin Guillory, Sean Metzger, Kurt Keutzer, and Trevor Darrell. Self-supervised pretraining improves self-supervised pretraining. *arXiv*, 2021.