Homework 10

11-967

Due date: 04/06/2025 11:59 PM EDT

In this homework, you will explore strategies for making language models more efficient both in their training and inference phases. Large language models require significant computational resources due to their high memory consumption, large parameter counts, and the extensive compute needed for both training and serving. Efficiently training these models involves optimizing memory usage and using distributed training setups, which allow us to leverage multiple GPUs to process larger models or batches without running out of memory. Here you will first deal with long sequences. ***Note: please save your trained models. You might need them in later assignments.***

For this homework, if you are using AWS, please use the following setup, which is the setting we tested on:

- Instance: g5.2xlarge
- Image: ami-0aada1758622f91bb
- Disk space: 100 GB

There is a bit of coding in this homework, but there is no code submission on Gradescope. You only need to upload your PDF.

Problem 1: Retrofitting to Larger Sequence Length

[Question 1.1] (***Coding, 5 points***) You are going to pre-train a 160m parameter language model with sequences of length 512. Your starting point is a randomly-initialized model based on the Pythia architecture. Follow the README file to download it to a local folder. We provide two validation sets for you to compute perplexity: one with sequences of length 512, and another with 2048. Recalling what you learned from homework 1, set the missing hyperparameters under `configs/512_eager_wikitext.json`. Then, you should be able to run `"scripts/launch_single_gpu.sh"` to train a model. The README file contains instructions on how to launch the script.

DELIVERABLES FOR Q1.1

A - Present the following values:

- Validation perplexity on 512-len set;
- Validation perplexity on 2048-len set;
- Training time;
- GPU VRAM usage.

B - Justify the difference, if any, between the perplexity on the two sets.

[Question 1.2] (***Coding, 5 points***) The previous configuration, as the name suggests, leverages an eager attention implementation - just like the one you implemented in homework 1. Recently, more efficient attention backends have been proposed, such as Flash Attention. First, implement the missing functionality of `forward()` regarding flash attention under the `models/GPTNeoX-160m/modeling_custom.py` file (you will only see this folder after running the initialization script following README. Then, in the config file, change `attention_type` to `flash_attention_2` and re-train the model with the same hyperparameters you used for the previous model.

DELIVERABLES FOR Q1.2

A- To implement the missing functionality, familiarize yourself with the `_flash_attention_forward` function that is already imported. Then, write unit tests **by yourself** to ensure that the eager attention and flash attention are consistent. You should think of what kinds of unit tests are useful.
B - Present the following values:

- Validation perplexity on 512-len set;
- Validation perplexity on 2048-len set;
- Training time;
- GPU VRAM usage.

C - Compare the results to the ones you achieved on the previous question. Support your comments by explaining how speedups and memory reductions are achieved by Flash Attention.

[Question 1.3] (***Coding, 5 points***) Another technique that allows saving memory is gradient checkpointing. Train a new model, creating a config with your choice of hyperparameters, and setting the following:

- `model_to_train`: your previous model from Q1.2
- `attention_type`: `flash_attention_2`;
- `gradient_checkpointing`: true;
- `seq_len`: 2048.

DELIVERABLES FOR Q1.3

A- Before training: What is the maximum batch size you can achieve with `seq_len`=2048, but without flash attention and gradient checkpointing? And with both turned on?
B- Before training: Read the linked information about gradient checkpointing. In two sentences, describe how it works and the trade-off it entails.
C - After training, present the following values:

- Validation perplexity on 512-len set;
- Validation perplexity on 2048-len set;
- Training time;
- GPU VRAM usage.

D- Did the valid perplexity on the 2048 set increase or decrease, when compared to Q1.1/Q1.2? Why?
E- Train a similar model, but start from scratch (i.e., from the downloaded `models/GPTNeoX-160m`) rather than your previous model. Comment on the obtained perplexities.