Assignment #2

Natural Language Processing, 2024 Fall

November 6, 2024

Assignment Guidelines

- The assignment is due on **11:59 PM, December 5**. We have a strict late policy: penalties increase to 10%, 20%, 50%, 100% for each day, where #days are rounded up.
- The assignment is a coding assignment. You should **submit a zip file, which contains your code and a report**.
- The full score is 20 points. For each subquestion, we will give partial score to partially correct answer. Double-check your submission to prevent typo and compilation error.
- You are free to discuss the problems with your classmates or search online, however, you must acknowledge the collaborators or online source at the end of the homework.
- You are not allowed to use LLMs to assist you working on this assignment.

Part 1 Pretrain Transformer(7 points)

You'll train a Transformer to perform a task that requires accessing world knowledge—knowledge not provided in the task's training data (especially if generalization beyond the training set is needed). Initially, you'll observe that the Transformer largely fails at the task. Then, you'll learn to pretrain the Transformer on Wikipedia text, which contains relevant world knowledge. Fine-tuning the Transformer on this knowledge-intensive task afterward allows it to leverage the knowledge gained during pretraining. This approach enables the model to perform significantly above chance on a held-out development set.

(a) **Review NameDataset (0 points):** Our goal is to predict birthplace based on name, using the `NameDataset` class in `src/dataset.py`. Run `python src/dataset.py namedata` to inspect sample data. No code or submission required.

(b) **Construct your model (1 points):** Construct a decoder-only model. You may want to complete several coding blocks in `model.py` and `attention.py`. Note that you can try different model architectures, which may help you in the later steps.

(c) **Implement Finetuning (without pretraining) (1 points):** Modify `run.py` for finetuning a Transformer on `NameDataset`. After training, you should report your prediction score on the dev set. You can refer `run_finetune.sh` and `run_evaluate.sh` for launching details.

(d) **Pretrain (1 points):** Implement `getitem()` in `CharCorruptionDataset` for a span corruption task, as per instructions in `dataset.py`. Debug with the command `python src/dataset.py charcorruption`. Then you should run `run_pretrain.sh` to get a pretrained model. Your pretrain budget is less than an hour(in cpu).

(e) **Finetune at pretrained model, and Make Predictions (1 points):** finetune at the pretrained model, and also report your prediction score on dev set. We expect an accuracy above 10%.

(f) **Ablation study at whatever your want(2 points)!** You are required to conduct an ablation study on one (or more) component of the neural network training process. You may choose any part of the process, such as model architecture variations, pretraining setups, optimization techniques, or regularization methods. However, you must clearly justify your choice of components, explaining why these parts are important to investigate. For each component, describe your experiment setup, your expectations, and how the results affect model performance. It's normal that you cannot observe clearly difference between two different setups. We will buy it as long as you can state the motivation and explain the outcome clearly.

Note: You cannot just conduct trivial component replacement, like change Adam optimizer into SGD, because there is no solid motivation behind this to makes you do so. So pick your ablation study smart, and **don't let the reviewer know you are padding the experiments**! We also provide a potential topic list for you to refer.

- Post-norm and Pre-norm in transformer blocks.
- Different position embedding
- Scaling law for pretrained model, how to choose learning rate, weight decay, or batch size
- Inference setup: top-p, top-k or temperature.

Part 2: 3D Parallelism (7 points)

In this assignment, you will work with a simplified Transformer training script. Your task is to develop three kinds of parallel strategies to improve training efficiency: **Data Parallelism**, **Tensor Parallelism**, and **Pipeline Parallelism**.

We will cover these concepts in detail during the TA lecture on **November 17th**. If you wish to start early, you can refer to Song Han's lecture for foundational knowledge.

Tasks

1. **Try PyTorch Distributed Framework (1 point)**

   - **Objective**: Set up and familiarize yourself with PyTorch's distributed framework.
   - **Instructions**:
     - Install the necessary packages for PyTorch distributed training.
     - Run a simple script to ensure that the distributed environment is correctly configured.
     - Verify that multiple processes can communicate and synchronize using PyTorch's `torch.distributed` package.

2. **Run the Baseline Script**

   - **Objective**: Understand the baseline training process.
   - **Instructions**:
     - Run the provided `run.py` script without any modifications.
     - Observe the training behavior, resource utilization, and performance metrics.
     - This step is crucial to establish a performance baseline and sanity check for your further modification.
     - The training loss will be outputed into `loss_log_DP1_PP1_TP1_stage0.txt`.
   - **How we accept your code** All your modification must be indifferent to you user, which means they will have same training loss. But in practice, because associative law does not hold when add and multiply operation under finite precision, we cannot produce baseline's result every time. So we consider as long as 99% of loss have less than 1% relative error, then we accept they have the same outcome.

3. **Implement Data Parallelism (2 points)**

   (a) **Data Parallelism with `dp_size=2`, `stage=0` (1 point)**

   - **Objective**: Implement data parallelism across two devices, using `ZeroOptimizer` with **stage 0**
   - **Instructions**:
     - Entrypoint: `python run.py --world-size=2`
     - This step, you should observe the same cuda memory as the baseline code occupy.

   (b) **Data Parallelism with `dp_size=2`, `stage=1` (Optional,1 point)**

   - **Objective**: Implement `ZeroOptimizer` with **stage 1**
   - **Instructions**:
     - Entrypoint: `python run.py --world-size=2 --stage=1`
     - You will see gpu memory dropping at this step, and as you increase dp_size, memory per process will keep decreasing.

4. **Implement Pipeline Parallelism (2 points)**

   (a) **Vanilla Pipeline Parallelism (1 point)**

   - **Objective**: Implement a basic form of pipeline parallelism by dividing the model into sequential stages.
   - **Instructions**:
     - Split the model into different segments, each assigned to a separate device.
     - Adjust the training loop to pass data through the pipeline stages.
     - Handle synchronization and ensure correct forward and backward passes.

   (b) **GPipe Implementation (1 point)**

   - **Objective**: Implement GPipe-style pipeline parallelism using micro-batching.
   - **Instructions**:
     - Modify your pipeline to process micro-batches to improve device utilization.
     - Implement methods to split batches into micro-batches and aggregate results.
     - Address challenges like bubble overhead and synchronization delays.

   (c) **Advanced Implementations (Optional, +1 point each)**

   - **Objective**: Explore advanced pipeline parallelism techniques.
   - **Examples**:
     - Implement interleaved pipelines.
     - Use 1F1B (one-forward-one-backward) scheduling to reduce pipeline stalls.
     - Experiment with dynamic load balancing between pipeline stages.

5. **Implement Tensor Parallelism (2 point)**

   - **Objective**: Split model tensors across multiple devices to parallelize computations within layers.
   - **Instructions**:
     - Entrypoint: `python run.py --world-size=2 --tp-size=2`
     - You will observer gpu memory halved at this time.

6. **Combine Parallel Strategies (1 point)**

   - **Objective**: Integrate all your parallel strategy together

Submission Guidelines

- **Code**:
  - You may modify any part of original codes.
  - Submit all codes in the `part2` folder.

- **Report**:
  - Provide a brief report that includes:
    * Screenshots or logs that demonstrate successful execution.
    * GPU memory you saved by each parallel strategy.
    * Performance comparisons between different parallel strategies.

Notes

- The total points you can earn are capped at **7 points**, even if you complete additional advanced tasks.
- It is recommended to test your code on a system with at least 1 GPU to fully realize the benefits of parallelism.
- Utilize PyTorch's distributed features, such as `DistributedDataParallel`, and other relevant APIs.

Resources

- PyTorch Distributed Overview
- PyTorch DistributedDataParallel Tutorial
- GPipe Paper
- Song Han's Lecture Slides

Good luck, and we look forward to your innovative solutions!

Part 3 Small project(6 points)

For this task, you are required to choose a recent research paper in the field of NLP. You will reproduce the results from the paper (so we suggest you to choose a paper with public code), conduct a brief analysis of the approach, and suggest possible improvements.

Requirements

(a) **Paper Selection**: We have already provided a list of NLP paper at the first lecture, 19th page of lec1. You can also choose any other NLP paper.

(b) **Reproduce**: Replicate key results from the paper. **Reproduce is not a trivial work, no one run code perfectly on the first try**. So document errors you encounter and your resolving process, as this will be a great part in the evaluation.

(c) **Analysis**: Provide a concise analysis that covers the following:

- **Strengths**: What aspects of the paper's approach or results stand out?
- **Weaknesses**: What limitations or areas of improvement do you observe?

You need experimentation to justify your argument.

(d) **Proposed Improvements**: Suggest potential modifications or alternative methods that could enhance the results or address identified limitations.

Notes

- This small project is independent of the larger project and will be graded separately. You may select a paper related to your large project, but it is not required.
- Collaboration in terms of discussion among group members is encouraged, but coding and experimental work must be completed individually.