Homework 2

11-967

Due date: 1/31/2025 11:59 PM EST

In this homework, you will experiment with training the Transformer architecture you implemented in HW1.
*Note: there are **two** Gradescope submission pages, one for your code and one for your `report.pdf`. Please submit your code to **Assignment** as instructed, and your `report.pdf` to **Assignment (PDF)**.*

Problem 1: Training the Transformer

Now that you have implemented the transformer, it is time to train the model! For ease of implementation and testing, we will provide the tokenized input for you. We will train on a subset of the C4 corpus[1], which is itself a cleaned subset of the Common Crawl web corpus[2]. This dataset is downloaded automatically as a part of the training script, and there is no need for you to access it manually.

An outline of the code is provided for you in `src/lm/train.py`. Keep in mind the various hyperparameters that are relevant for training: batch size, learning rate (and its scheduler), gradient accumulation, etc. These hyperparameters are read from a configuration file. We have provided sample configuration files for you for adjusting the hyperparameters. There are five functions that you are expected to implement:

- `train` - the main training loop.
- `random_batch_sampler` - a data sampling function used in training that yields randomly shuffled batches of the data.
- `sequential_batch_sampler` - a data sampling function used in validation that yields a sequential pass through the data.
- `cosine_lr_schedule` - learning rate scheduler with Cosine annealing (see question 1.2).
- `compute_language_modeling_loss` - the loss function used for training and evaluating the model.

[Question 1.1] (***Coding, 20 points***) Complete `train.py`, implementing the above functions. All the unit tests are provided in the test script `test_train.py`. Your implementation will be awarded points for each of the five unit tests that pass.

Note: There is a special test case `test_train_sanity_check()`, which trains and evaluates your model on a fundamentally unlearnable sequence (random integers sampled from $\{0, \ldots, 7\}$). A test loss below $\log(8) \approx 2.08$ on this task suggests that your model is cheating by looking at the token it is trying to predict. This testcase itself does not carry any points, but failing it would preclude you from scoring points for Q1.3, Q1.4 and Q1.5.

DELIVERABLES FOR Q1.1

Upload your code to Gradescope as instructed in the starter code README and make sure all test cases pass.

[Question 1.2] (***Written, 10 points***) Cosine annealing with warmup is a commonly used dynamic learning rate schedule in training neural nets. It has two phases, in the warmup phase where $t \in [0, a)$, the learning rate increases linearly from 0 to $\text{lr}_{\max}$. In the annealing phase where $t \in [a, b)$, the learning rate decays from $\text{lr}_{\max}$ to $\text{lr}_{\min}$ following a half-cosine curve. When $t \geq b$, the learning rate stays at $\text{lr}_{\min}$. Figure 1 shows an example of this schedule.

Using the symbols provided, write down **two** expressions, one for the learning rate during warmup ($t \in [0, a)$) and one for the learning rate during annealing ($t \in [a, b)$).

[1] https://huggingface.co/datasets/allenai/c4

[2] https://commoncrawl.org/

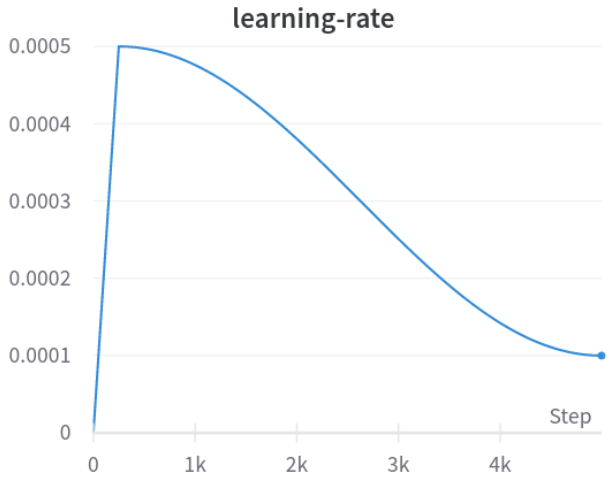

Figure 1: Example Cosine Schedule

DELIVERABLES FOR Q1.2

In your report, answer the following:

A. What is the mathematical expression for cosine annealing?

B. Why would one want to use cosine annealing? What are some advantages of cosine annealing over a constant learning rate? Answer in at most two sentences.

[Question 1.3] (***Written, 9 points***) What is the validation loss after training with the configuration provided in `GPT-tiny.yaml`? For approximately how many training steps should the model be trained to achieve optimal performance? Report the training loss curve (use a screenshot from weights & biases: `https://wandb.ai/site`).

DELIVERABLES FOR Q1.3

In your report, answer the following:

A. Write the validation loss for `GPT-tiny`.

B. Write the number of training steps needed for `GPT-tiny` to saturate in training loss.

C. Plot the training curve for `GPT-tiny`.

[Question 1.4] (***Written, 20 points***) Next, you will perform hyperparameter tuning. When optimizing neural networks, a common way to measure the total computation needed is with floating-point operations (FLOPs). E.g., a single multiply-add of floats is counted as a FLOP. Assume that you have a compute budget of 1e+15 FLOPs for training. Experiment with your model and training hyperparameters to find the best configuration when training with at most 1e+15 FLOPs (the code for computing the number of FLOPs used to train a model is provided in `train.py`). Note that the FLOP limit is intentionally low: it should not take more than 10 minutes per hyperparameter test.

As long as you stay within the FLOP budget, you are free to modify any of the following hyperparameters, all of which affect the number of FLOPs used: model hyperparameters such as `n_embd`, `n_head`, `n_positions` and `n_layer`; and training hyperparameters such as `batch_size`, `seq_len`, `grad_accumulation_steps`, and `num_training_steps`. You will use perplexity of a withheld validation set to evaluate which configuration is best.

DELIVERABLES FOR Q1.4

Report results for **at most five** hyperparameter configurations. For each, describe what hyperparameter values were modified and the resulting validation set perplexity (PPL). For your best performing setting, you should provide the final configuration (YAML file) for this setting. Your report should answer the following questions:

A. In one paragraph, describe your experimental procedure.

B. In one paragraph, describe your experimental results.

C. Paste your final YAML configuration into your report.

[Question 1.5] (***Coding, 20 points***) Using the model configuration reported above, train your final model. For training your final model, you may train 100x more FLOPs (i.e. up to 1e+17 FLOPS), but do not change your model hyperparameters. Your final model should be able to achieve a PPL under 50 on the validation set. Please include your final model checkpoint `model.pt` in your submission. We will verify your model with an offline correctness test, which will measure the perplexity on a test dataset (not provided). Submissions with PPL under 50 will get full points, and submissions with PPL under 75 will get half points. **Note: Gradescope allows a maximum upload size of 100MB. Please ensure that your model checkpoint does not exceed this size.**

DELIVERABLES FOR Q1.5

Report your final validation loss and PPL in your report and upload the model checkpoint as instructed.