Assignment 2: Build your own LM

Release date: 02/10 (Tue)
Due date: 02/24 (Tue) 05:59PM
Latest possible due date with late days: 02/27 (Fri) 05:59PM

Overview: This assignment consists of 4 parts including tokenizer, transformers, and trainers. The starting code for the assignment can be found in this repository - https://github.com/zinengtang/cs288-sp26-a2. You have 10 max tries to submit to gradescope.

Part 1: Train a tokenizer model on TinyStories (30 points)

In Part 1, you will implement and train a BPE tokenizer on TinyStories. In this part, you will implement Byte Pair Encoding (BPE) from scratch. Start by implementing the core BPE algorithm in `tokenizer.py`, which should handle vocabulary building through iterative merging of the most frequent token pairs. Your `train_bpe.py` should load the TinyStories dataset, run the BPE training procedure, and save the learned vocabulary and merge rules. Make sure your tokenizer can both encode text into token IDs and decode token IDs back to text. Submit the following files on gradescope:

Deliverables:

- tokenier.py, train_bpe.py
  - Problem (tokenizer.py): BPE Tokenizer
  - Problem (train_bpe.py): BPE Training on TinyStories

Part 2: Implement an end-to-end transformer model (50 points)

In Part 2, you will implement all common components of transformers. For RoPE, ensure your positional encodings are applied correctly to queries and keys. Your causal attention must properly mask future tokens. Test each component individually before assembling the full model. Submit the following files on gradescope:

Deliverables:

- model.py
  - Problem (linear): Linear module
  - Problem (embedding): the embedding module
  - Problem (rmsnorm): Root Mean Square Layer Normalization
  - Problem (SiLU): SiLU activation function. https://arxiv.org/abs/1702.03118
  - Problem (SwiGLU): SwiGLU feedforward function. https://arxiv.org/pdf/2002.05202
  - Problem (rope): RoPE https://arxiv.org/abs/2104.09864
  - Problem (softmax): Softmax
  - Problem (scaled_dot_product_attention): Scaled dot-product attention

  - Problem (multihead_self_attention): Causal multi-head self-attention
  - Problem (multihead_self_attention_with_rope): MultiHeadSelfAttentionWithRoPE
  - Problem (transformer_block): the Transformer block
  - Problem (transformer_lm): the Transformer LM
  - Problem (count_flops_per_token): FLOPs estimation
  - Problem (estimate_memory_bytes): Memory estimation

Part 3: Implement training utilities (20 points)

In Part 3, you will implement trainers including loss function, optimizer, and scheduler. The learning rate scheduler should implement linear warmup followed by cosine decay. Gradient clipping should clip by global norm to prevent exploding gradients. These utilities can be used in Part 4 for actual training. Submit the following files on gradescope:

Deliverables:

- nn_utils.py
  - Problem (softmax): Numerically stable softmax (for training)
  - Problem (cross_entropy): Cross entropy
  - Problem (gradient_clipping): Gradient clipping
  - Problem (perplexity): Token perplexity
  - Problem (token_accuracy): Implement token accuracy metric

Part 4: Pre-train + Fine-tune + Prompting mini-study (bonus 20 points)

In Part 4, you will use your own written transformer/utils/tokenizer. This bonus section ties everything together. First, pre-train your transformer on TinyStories using your implemented components. Then, fine-tune on the multiple-choice QA task by adding a classification head that pools transformer outputs to predict the correct answer choice. For prompting 4B), design prompt templates that frame the QA task as text completion and compare performance against the fine-tuned approach. Your final predictions should beat our provided baseline to earn the bonus. We will provide a baseline dev metric for comparison. If your final model beats the two baselines (strictly better than the fine-tune baseline and prompting baseline metric), you earn a +20% bonus in total.
Note:

1. You are free to use other datasets
2. We provided a baseline script but you are free to use your own script or external tools or other hyperparameters like model size. The minimum requirement would be to use your own written tokenizer and transformer model and train them from scratch.

4A) Tiny pre-training + finetuning (12 points)
Pre-train your Transformer LM on TinyStories (provided split/shard). Generate a few samples with greedy + top-k (top-k can be provided starter code, or optional).

Finetune on a Multiple-choice QA dataset with the format (context, question, choices) → correct choice predict choice via pooled representation.
Deliverables:

- finetuned_predictions.json (predicted QA labels)

External sources: you are free to write your own code from scratch while not adding significantly pre-packaged tools like huggingface trainer, llama-factory, etc.

4b) Prompting (8 points)
Use your finetuned model in 4A) and experiment with your own prompts. Tips: few-shot prompts can potentially improve performance.
Deliverables:

- prompting_predictions.json (predicted QA labels)
    - Relative performance should outperform 4A)

Grading Logistics: Our autograder will check all your implementation components (as shown in deliverables) and end-to-end model implementation for correctness with fixed inputs. There are about 50% hidden tests when submitted to gradscope for autograding. Part4 will be graded with your submitted predictions and we will briefly check your code for reproduction. The rule is for 4a) your score will scale from 30% to 50% accuracy, which means you get 0 for 30% or below and 12 points for 50% or above. For prompting, your method should outperform your 4a) by 2% to get a full score and scales down to 0%.

Additional Details:

Third-party Packages: This assignment emphasizes building from scratch when possible. The two models allow different third-party packages. Please follow these directions carefully. There is a heavy penalty for using packages beyond these specified below.

Development Environment and Third-party Tools: All allowed third-party tools are specified in the `requirements.txt` file in the assignment repository. The goal of the assignment is to gain experience with specific methods, and therefore using third-party tools and frameworks beyond these specified is not allowed. Please follow these directions carefully. We will penalize heavily for using packages beyond these specified below. Please consult the course generative AI policy with regard to using such tools.

You may only `import` packages that are specified in the `requirements.txt` file or that come with Python's Standard Library. The version of Python allowed for use is 3.10.x. Do not use older or newer version. We strongly recommend working within a fresh virtual environment for each assignment.

Acknowledgement

The assignment is taken and adapted from Tatsunori Hashimoto and Percy Liang's Stanford CS 336.