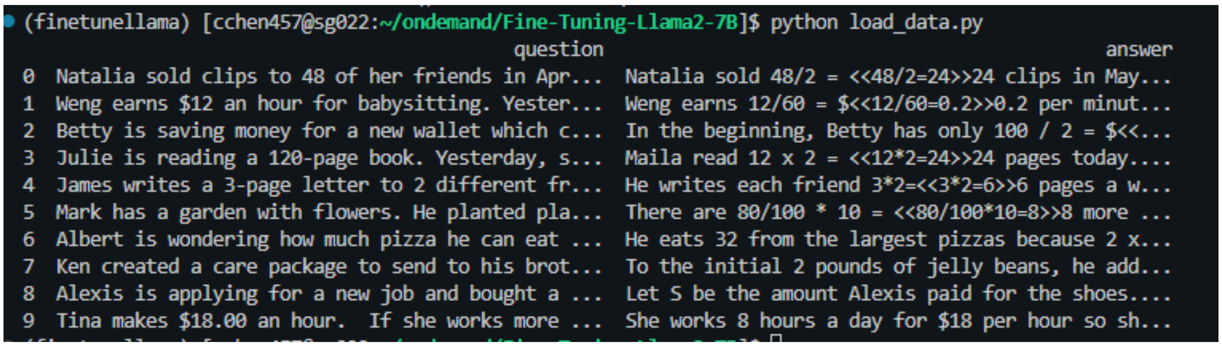
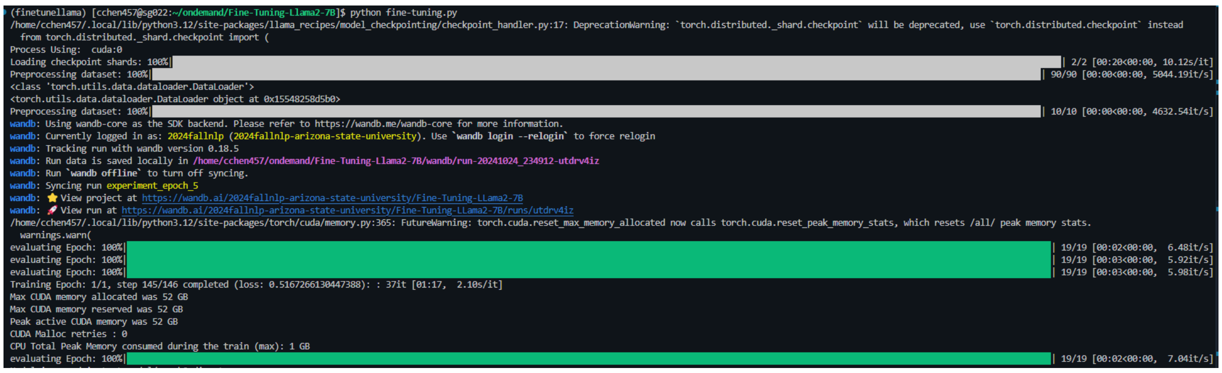
CSE 576
TOPICS IN NATURAL LANGUAGE PROCESSING
HOMEWORK- 4

1. **Screenshot of a sample data point from the training set.**



```
(finetunellama) [cchen457@sg022:~/ondemand/Fine-Tuning-Llama2-7B]$ python load_data.py
                                           question                                             answer
0  Natalia sold clips to 48 of her friends in Apr...  Natalia sold 48/2 = <<48/2=24>>24 clips in May...
1  Weng earns $12 an hour for babysitting. Yester...  Weng earns 12/60 = $<<12/60=0.2>>0.2 per minut...
2  Betty is saving money for a new wallet which c...  In the beginning, Betty has only 100 / 2 = $<<...
3  Julie is reading a 120-page book. Yesterday, s...  Maila read 12 x 2 = <<12*2=24>>24 pages today....
4  James writes a 3-page letter to 2 different fr...  He writes each friend 3*2=<<3*2=6>>6 pages a w...
5  Mark has a garden with flowers. He planted pla...  There are 80/100 * 10 = <<80/100*10=8>>8 more ...
6  Albert is wondering how much pizza he can eat ...  He eats 32 from the largest pizzas because 2 x...
7  Ken created a care package to send to his brot...  To the initial 2 pounds of jelly beans, he add...
8  Alexis is applying for a new job and bought a ...  Let S be the amount Alexis paid for the shoes....
9  Tina makes $18.00 an hour.  If she works more ...  She works 8 hours a day for $18 per hour so sh...
```

2. **Screenshot of the commands used to run training script on Sol, including the beginning of the training outputs**



```
(finetunellama) [cchen457@sg022:~/ondemand/Fine-Tuning-Llama2-7B]$ python fine-tuning.py
/home/cchen457/.local/lib/python3.12/site-packages/llama_recipes/model_checkpointing/checkpoint_handler.py:17: DeprecationWarning: `torch.distributed._shard.checkpoint` will be deprecated, use `torch.distributed.checkpoint` instead
  from torch.distributed._shard.checkpoint import (
Process Using:  cuda:0
Loading checkpoint shards: 100%|                                                                  | 2/2 [00:20<00:00, 10.12s/it]
Preprocessing dataset: 100%|                                                                  | 90/90 [00:00<00:00, 5044.19it/s]
<class 'torch.utils.data.dataloader.DataLoader'>
<torch.utils.data.dataloader.DataLoader object at 0x15548258d5b0>
Preprocessing dataset: 100%|                                                                  | 10/10 [00:00<00:00, 4632.54it/s]
wandb: Using wandb-core as the SDK backend. Please refer to https://wandb.me/wandb-core for more information.
wandb: Currently logged in as: 2024fallnlp (2024fallnlp-arizona-state-university). Use `wandb login --relogin` to force relogin
wandb: Tracking run with wandb version 0.18.5
wandb: Run data is saved locally in /home/cchen457/ondemand/Fine-Tuning-Llama2-7B/wandb/run-20241024_234912-utdrv4iz
wandb: Run `wandb offline` to turn off syncing.
wandb: Syncing run experiment_epoch_5
wandb: ⭐ View project at https://wandb.ai/2024fallnlp-arizona-state-university/Fine-Tuning-LLama2-7B
wandb: 🚀 View run at https://wandb.ai/2024fallnlp-arizona-state-university/Fine-Tuning-LLama2-7B/runs/utdrv4iz
/home/cchen457/.local/lib/python3.12/site-packages/torch/cuda/memory.py:365: FutureWarning: torch.cuda.reset_max_memory_allocated now calls torch.cuda.reset_peak_memory_stats, which resets /all/ peak memory stats.
  warnings.warn(
evaluating Epoch: 100%|                                                                  | 19/19 [00:02<00:00,  6.48it/s]
evaluating Epoch: 100%|                                                                  | 19/19 [00:03<00:00,  5.92it/s]
evaluating Epoch: 100%|                                                                  | 19/19 [00:03<00:00,  5.98it/s]
Training Epoch: 1/1, step 145/146 completed (loss: 0.5167266130447388): : 37it [01:17,  2.10s/it]
Max CUDA memory allocated was 52 GB
Max CUDA memory reserved was 52 GB
Peak active CUDA memory was 52 GB
CUDA Malloc retries : 0
CPU Total Peak Memory consumed during the train (max): 1 GB
evaluating Epoch: 100%|                                                                  | 19/19 [00:02<00:00,  7.04it/s]
```

3. **Explanation of the training script, where each member should explain a portion(a functionality in the training process) and include their name next to their explanation.**

- Chao-Shiang, Chen

Collect training and evaluation metrics

In this training process, we use the perplexity of a probability distribution to evaluate our model's performance. Perplexity is an intrinsic measure, meaning that it doesn't use external datasets. It shows how certain a model is about the predictions it makes. The higher the perplexity, the poorer the model performs. Conversely, the perplexity will be closer to 1 when the model predicts the test cases perfectly.

As the perplexity equation shows below,

$$b^{-\frac{1}{N}\sum_{i=1}^{N}\log_b q(x_i)}$$

means that we can apply pytorch exponential function `torch.exp(1/N * )` to calculate perplexity.

$$H = -\sum p(x)\log p(x)$$

* H equals to cross-entropy loss.

Hence, we calculate the mean of the total loss in every epoch and put the value inside the exponential function to get the perplexity of every epoch.

After collecting all metrics, such as perplexity, training, and evaluating loss, we pack all of the params and use weight and bias API format to pass data and get visualized train/loss (eval/loss) and train/perplexity (eval/perplexity) curves from weight and bias platform.

Below is the API used to create a visualization of the loss curve and perplexity.

```
`if wandb_run:
    wandb_run.log({
            'eval/perplexity': eval_ppl,
            'eval/loss': eval_epoch_loss,
        }, commit=False)`
```

- ## Ankitha Dongerkerry Pai

  The training data was loaded and prepared for model training using PyTorch's DataLoader. We shuffled the data to reduce bias and tokenized to convert text inputs into model-compatible formats. Each batch was prepared and fed into the model iteratively during training. Proper data handling ensured stability and consistency in the learning process.

  Hyperparameters were carefully selected and tuned to optimize model performance within memory constraints. A learning rate of 0.0001 and a batch size of 1 were used due to OutOfMemoryErrors with larger batches. Additionally, stochastic gradient descent was chosen for stability in smaller batches, and the number of epochs was set to 20 to observe convergence trends. These settings aimed to balance efficient training with achievable memory use.

- ## Rakshita Madhavan

  Evaluation metrics were key to understanding model performance. Perplexity, calculated at each epoch, provided insights into the model's prediction certainty. Post-processing steps convert model outputs into interpretable forms. Evaluation metrics were logged in Weights & Biases to track progress over time, enabling easy identification of areas for model improvement.

  Model checkpointing was implemented to save progress after each epoch, storing checkpoints if validation loss improved. This ensured that the best-performing model could be recovered even in case of training interruptions. Checkpoints were stored with their validation metrics, allowing us to select the model with the lowest validation loss for final evaluation.

- ## Bala Sujith Potineni

  The training process for the Llama causal LLM consists of several key steps organized into a comprehensive training loop. It begins with initialization, setting up the GradScaler for optional mixed-precision training, defining paths for saving model checkpoints, and initiating variables for tracking metrics and memory usage. The training loop includes early stopping conditions, a detailed per-epoch training loop that measures memory usage, handles batch processing, and performs forward passes with loss calculation and optional mixed precision. Critical steps include gradient accumulation and optimization, where backpropagation is performed, gradients are updated, and gradients are zeroed for the next iteration. Profiling and logging capture step-wise performance and training metrics, which are optionally logged to Weights & Biases. Validation occurs at specified intervals, assessing performance against a validation set and saving metrics. At the end of each epoch, the model might save checkpoints and update the progress bar. The process concludes by calculating average training and validation metrics across all epochs, saving the best models based on validation loss, and returning a dictionary of results, ensuring a detailed record of training performance and outcomes.

GradScaler

GradScaler is a utility in PyTorch, specifically designed for automatic mixed precision (AMP) training, which involves using both 32-bit and 16-bit floating-point arithmetic to speed up training and reduce memory usage without significantly impacting the accuracy of the model.

$$scaler \ = \ GradScaler()$$

Gradient Accumulation Steps

Gradient accumulation is a technique used to effectively train models with high computational requirements on hardware with limited memory capacity. It involves splitting the computation of gradients over multiple forward and backward passes. gradient_accumulation_steps specifies the number of forward passes to perform before actually performing a backward pass (updating model weights). This allows the use of larger batch sizes than what could be normally processed at once by accumulating gradients over several smaller batches. By doing this, models that require larger batch sizes for stable training can still be trained on hardware with lower memory, albeit with slower computational speed due to the multiple smaller steps.

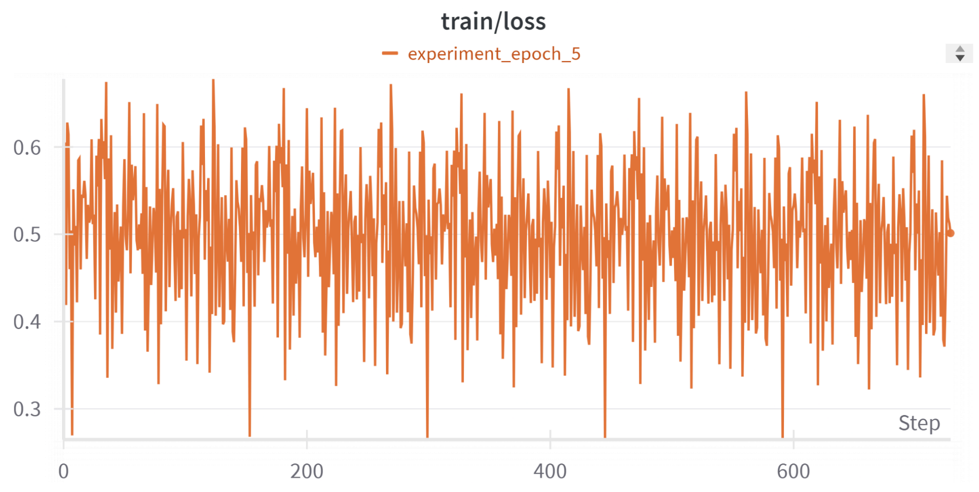
SGD Optimizer

The Stochastic Gradient Descent (SGD) optimizer is a simple yet highly effective algorithm used to minimize an objective function (typically loss) by iteratively updating the model parameters in the opposite direction of the gradient of the objective function with respect to the parameters. The core idea behind SGD involves taking the gradient of the objective function, which indicates the direction of the steepest increase, and moving in the opposite direction to find a minimum. SGD updates the parameters using a learning rate, which determines the size of the steps taken towards the minimum. While basic SGD updates parameters using only the gradient from a single batch, variants such as SGD with momentum and Nesterov accelerated gradient are commonly used to improve convergence rates by incorporating the direction and speed of the gradients over past steps.

In the following code snippet, the accumulated loss is divided by gradient_accumulation_steps to find exact mean loss at that step for that training batch. In case of backpropagation, it is performed only when the gradients are accumulated for one whole batch instead of at each step.

```
loss = loss / gradient_accumulation_steps

loss.backward()



                if (step + 1) % gradient_accumulation_steps == 0 or step == len(train_dataloader) - 1:

                    if train_config.gradient_clipping and train_config.gradient_clipping_threshold > 0.0:

                        torch.nn.utils.clip_grad_norm_(model.parameters(), train_config.gradient_clipping_threshold)

                    optimizer.step()

                    optimizer.zero_grad()
```

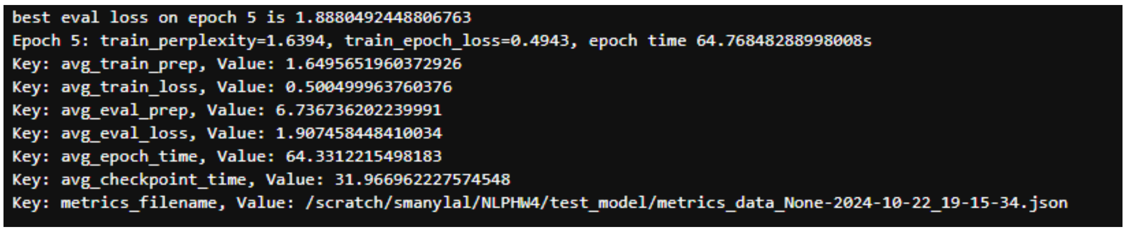
- Suraj Kumar Manylal



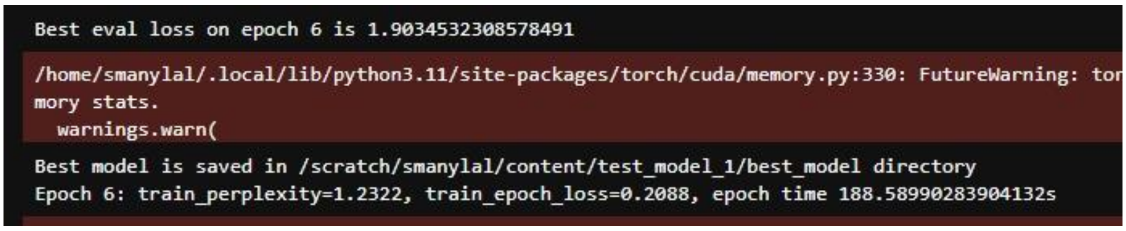
The batch size for training was set to 1 since any value greater than 1 would result in an OutOfMemoryError. Stochastic gradient descent was chosen to be the optimizing function rather than the Adam due to the same reason. The learning rate was set to 0.0001.

The small batch size and choice of learning rate could be possible reasons for the train\loss curve to not converge. Another reason could be that the model wasn't trained for long enough. However, the graph displayed a similar pattern despite being trained for 20 epochs as shown below.

4. Accuracy metric after evaluating model on the test set.

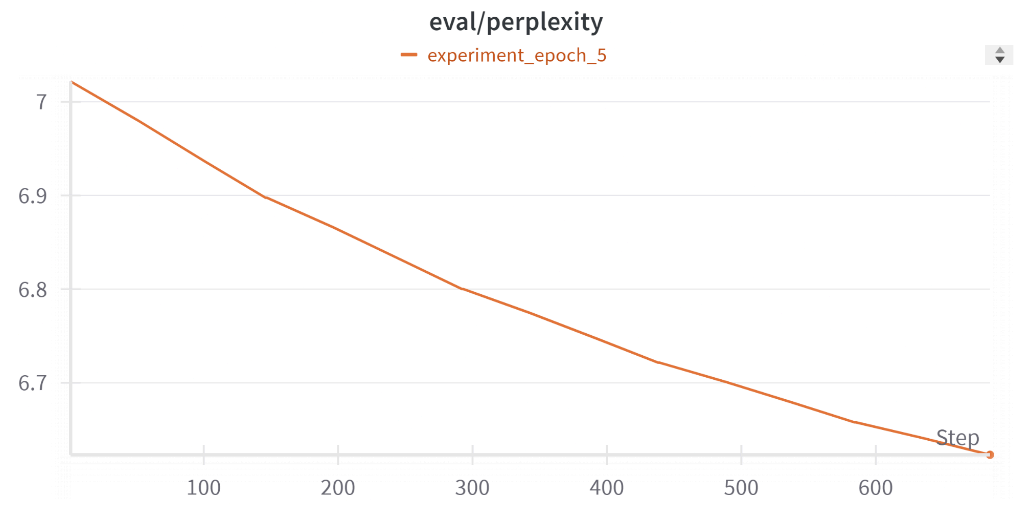


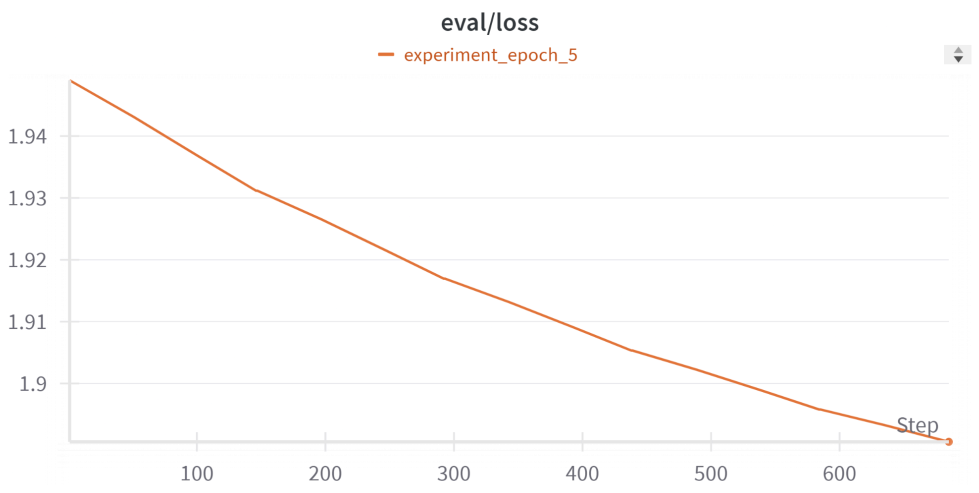
```
best eval loss on epoch 5 is 1.8880492448806763
Epoch 5: train_perplexity=1.6394, train_epoch_loss=0.4943, epoch time 64.76848288998008s
Key: avg_train_prep, Value: 1.6495651960372926
Key: avg_train_loss, Value: 0.500499963760376
Key: avg_eval_prep, Value: 6.736736202239991
Key: avg_eval_loss, Value: 1.907458448410034
Key: avg_epoch_time, Value: 64.3312215498183
Key: avg_checkpoint_time, Value: 31.966962227574548
Key: metrics_filename, Value: /scratch/smanylal/NLPHW4/test_model/metrics_data_None-2024-10-22_19-15-34.json
```



```
Best eval loss on epoch 6 is 1.9034532308578491
/home/smanylal/.local/lib/python3.11/site-packages/torch/cuda/memory.py:330: FutureWarning: tor
mory stats.
  warnings.warn(
Best model is saved in /scratch/smanylal/content/test_model_1/best_model directory
Epoch 6: train_perplexity=1.2322, train_epoch_loss=0.2088, epoch time 188.58990283904132s
```

5. Screenshots of Train Loss and Validation Loss curves from WandB(Weights&Biases) and a short explanation.





Contribution Sheet

Team Member and contribution:

1. Chao-Shiang, Chen - 20%
2. Ankitha Dongerkerry Pai - 20%
3. Rakshita Madhavan - 20%
4. Suraj Kumar Manylal - 20%
5. Bala Sujith Potineni - 20%