MDS5210 Final Project - Large Language Models - A Case Study from GPT-2

Yuze Shi*
223040109@link.cuhk.edu.cn
Jiyuan Liu†
223040112@link.cuhk.edu.cn
Mengqi Li‡
223040115@link.cuhk.edu.cn
Yizhi Hong§
223040116@link.cuhk.edu.cn

December 26, 2023

Abstract

This report delves into the fine-tuning of the GPT-2 model for enhanced dialogue capabilities, utilizing the "HH-RLHF" Dataset with a focus on optimizing the training and validation processes. The study involves experimenting with different training algorithms, loss computation, and backpropagation techniques. A key component of the research includes evaluating the fine-tuned model's dialogue ability using diverse prompts generated by ChatGPT-4, spanning topics from science to social issues.

Significant findings reveal variations in the model's performance across different training steps, with indications of overfitting beyond a certain point. Further explorations into the impact of various optimizers (AdamW, SGD, etc.) and parameter-efficient fine-tuning methods like LoRA (Low-Rank Adaptation) were conducted. These experiments provided insights into the trade-offs between model efficiency and effectiveness. Additionally, the report touches upon the challenges and outcomes of using specialized datasets and advanced techniques like Gradient Accumulation and Direct Preference Optimization, highlighting the complexities and limitations encountered in fine-tuning large language models.

1 SFT

1.1 Training and Loss

In this section, our work is mainly on building the complete structure of `fit` funciton in `SFTTrainer` class in `trainers.py` file. We are using `EYLSFTStaticDataset` which consists of 84576 training samples and 3451 test examples to do the fine-tune on GPT-2.

For one single training step, as described in Algorithm 1, the training procedure involves a forward pass, loss computation, and a backward pass for optimization.

In the model evaluation section, we simply recognize test data as validation data. as described in Algorithm 2. Since we just use test data as the judge criteria of the model, data snooping doesn't occur in our model.

Due to the limitations of GPU memory, we set the batch size for processing the dataset to 1. We fine-tuned the pre-trained GPT-2 medium model using masked multi-head self-attention, the AdamW optimizer, and the default configurations. During training, we calculated the training and test errors using 200 samples from the training and test data, respectively. Initially, we trained the model for 40,000 iterations. We found that the model still has the trend in convergence when done with the training, and then we did the same 40,000 iterations on the trained checkpoint model. The output is shown in Figure 1. It is important to note that 'step 0' refers to the original, unmodified pre-trained GPT-2 model, and we performed validation 40 times during the whole training.

*Yuze Shi worked on data visualization and analysis.
†Jiyuan Liu contributed to understanding template code and article drafting.
‡Mengqi Li was responsible for programming and code on server deployment.
§Yizhi Hong was involved in article writing and review.

Algorithm 1 Training Step

Get a batch of training data $(\mathbf{x}_b, \mathbf{y}_b)$ from the train dataloader
Move $\mathbf{x}_b$ and $\mathbf{y}_b$ to the specified device
{Forward Pass}
Compute logits $\mathbf{L}$ using the model on $\mathbf{x}_b$
Reshape logits $\mathbf{L}$ from shape (B, T, C) to $(B \times T, C)$
Reshape targets $\mathbf{y}_b$ from shape (B, T) to $(B \times T)$
Compute loss using cross-entropy between $\mathbf{L}$ and $\mathbf{y}_b$
{Backward Pass and Optimization}
Zero the gradients of the optimizer
Compute the gradients by backpropagation
Update the model parameters using the optimizer

Algorithm 2 Estimate Loss for Training and Validation Data

Initialize an empty dictionary *out* to store losses
Set the model to evaluation mode
for each split in [train, val] **do**
  Initialize an array *losses* to store losses for each iteration
  for k from 1 to *numbers of evaluation iterations* **do**
    Get a batch (X, Y) from the train dataloader if split is 'train', else from the test dataloader
    (X, Y) are same as $\mathbf{x}_b$ and $\mathbf{y}_b$ in Algorithm 1
    Store the loss in *losses*
  end for
  Store the mean of *losses* in *out* with key as *split*
end for
Set the model to training mode
return *out*

Moreover, it took us total $424.51 + 423.98 = 848.49$ minutes ≈ 14 hours on P100 to finish. Subsequently, we attempted to increase the learning rate to 10^{-2} and 10^{-3}, but both adjustments led to non-convergence in the first few steps.

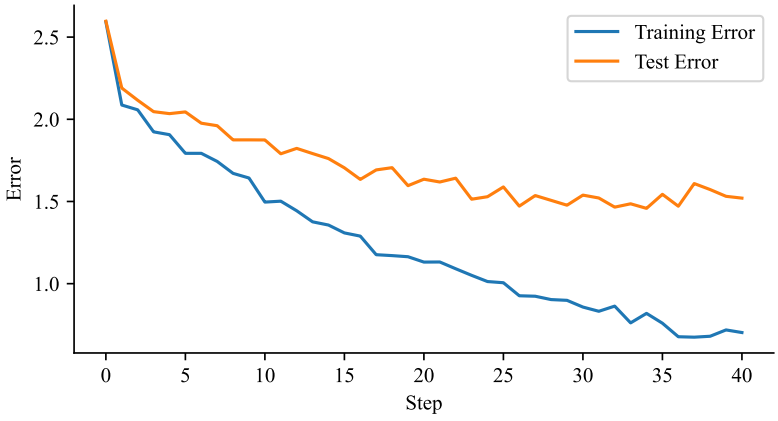

Figure 1: Training and Test Loss Over Steps

1.2 Evaluate the Dialogue Ability

To judge the dialogue ability of the model, we use "OpenAssistant/reward-model-deberta-v3-large-v2". And for the prompts, without loss of generality, we asked ChatGPT-4 [1] to generate 40 samples in a few general categories. Here are some examples.

- Science and Technology:
  - "How does the human immune system fight infections?"

- "How does a black hole form?"
- "Explain the process of photosynthesis in plants."

- History and Culture:
  - "What were the main causes of World War II?"
  - "How can technology be used to improve education?"
  - "Discuss the cultural significance of the Great Wall of China."

- Economics and Politics:
  - "How can businesses implement more sustainable practices?"
  - "Discuss the economic effects of global tourism."

- Psychology and Social Issues:
  - "What are the benefits of meditation for mental health?"
  - "Discuss the impact of social media on teenage self-esteem."

We calculate the score by averaging the 40 dialogue reward scores from prompts and their corresponding answers generated by fine-tuned GPT-2. The results can be found in Figure 2. The average score is negative, indicating that the model might not perform well overall. However, it is worth mentioning that our fine-tuning data is limited. We can only provide the model with limited knowledge on certain topics, making it an expert in just a few fields. For instance, the prompt "How does blockchain technology work?" earned a high score of 2.29. In contrast, the prompt "Explain the principles of sustainable architecture." received a poor score of -3.56.

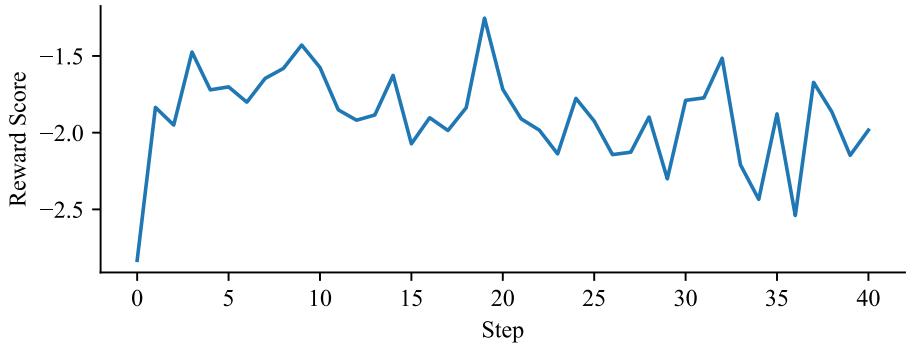

Figure 2: Reward Scores Over Steps

1.3 Findings

In Figure 1, it is evident that the model begins to overfit after approximately step 30. Furthermore, Figure 2 indicates that the model's dialogue ability fluctuates over the steps and shows a downward trend after step 20. Combining the insights from these two figures, we have selected the model trained up to step 20 as our final version for conducting interactive chats.

Furthermore, we can compare the results generated from the fine-tuned model with the original pre-trained model based on the same prompt.

Prompt: Summarize how artificial intelligence could change jobs in technology and healthcare by 2043.

Fine-tuned GPT-2: One way of looking at this is as a prediction. Some researchers are imagining that, within the next few decades, the use of advanced AI systems will help significantly improve people's lives. People's standard of living will probably be much more robust, and artificial intelligence systems might help to centralize information, enable new kinds of economic and social interaction, and greatly enhance people's personal relationships. People's standard of living may also more precisely align with their

Original GPT-2: 上短粒持明: 下 粒持巴: 説川不多中。

Weirdly, the original pretrained GPT-2 will output Chinese, and it appears to be a nonsensical string of characters. It is likely resulting from a generation error when using the model with a prompt that was mismatched with its training data. This kind of issue is not uncommon when using deep learning models, especially when the content

being processed by the model is outside the scope or type of its training data. It's common that the content fine-tuned model is not complete due to the limited block size of the training(we used 1024).

Prompt: What is love?

Fine-tuned GPT-2: Love is an emotional experience that is formed when there is an experience of attraction, such as a love affair, or a romantic relationship. When we experience a particular feeling we call a "love affair", we usually begin to experience a pleasant feeling of joy, peace, or contentment. This is called "emotional love" or "emotional rest". Emotional love often occurs when there is no underlying emotional reaction, such as when we

Original GPT-2: It's a two person act ... the first person can be man or woman; the second person can be woman, man or man. It's an act of love.

From the prompts we interact with GPT-2, it's evident that the fine-tuned model is more informative on those simple prompts.

2 Further Exploration

2.1 Impact of Optimizer - Different Optimizers

We trained the model using four different optimizers Adam, SGD with Nesterov's acceleration, SGD with momentum and SGD, and compared the results with each other including the model with AdamW which we already trained. And we can get the result in Figure and GPU information in Table 1. The learning rate is set to 10^{-4}, and for optimizers utilizing momentum, the parameter is set to 0.9. Here we just iterate the training 20,000 times. Another thing to mention is that, in this section, we used the gradient accumulation technique.

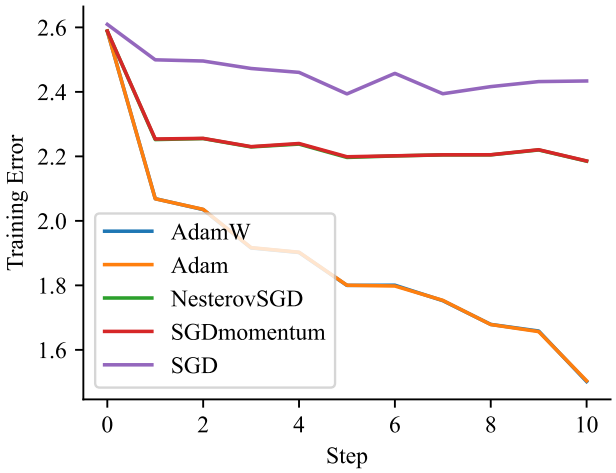

Figure 3: Reward Scores Over Steps (Training)

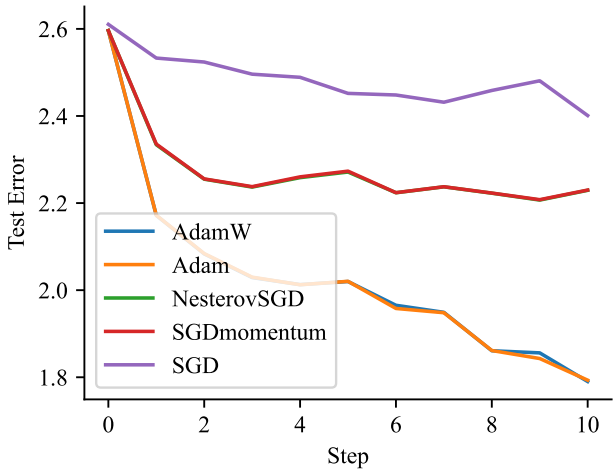

Figure 4: Reward Scores Over Steps (Testing)

Table 1: Comparison of Optimizers on GPU P100

	AdamW	Adam	SGD-Nesterov	SGD-Momentum	SGD
Allocated GPU Memory (GB)	13.52	13.52	11.23	11.23	8.41
Training Time (minutes)	200.0	188.5	183.5	181.8	170.9

In the Figure 3 and 4 comparing different optimizers over training and testing steps, AdamW and Adam show similar performance trends, just like the identical plot. They both converge rapidly initially and tend to converge even at step 10. SGD with Nesterov's acceleration and SGD with momentum display close performance. Their training and testing performance sharply improves but then slightly fluctuates and plateaus. Plain SGD has the slowest initial convergence rate in both training and testing, but it shows steady improvement over time.

The graphs indicate that while all optimizers improve over time, those with momentum and adaptive learning rates (like AdamW and Adam) tend to converge faster and reach lower error rates than those without (like plain SGD).

For exploration on GPU memory consumption, based on the principles[2] discussed:

1. AdamW and Adam: Both require similar memory usage (13.52 GB) because they maintain 32-bit floating-point (fp32) copies of the model parameters, gradients, and additional states like the moving averages and variances of the gradients. This requirement is due to the Adam optimizer's structure, where precise calculations for parameter updates are essential.

2. SGD with Momentum and SGD with Nesterov's Acceleration: These optimizers show a reduced memory usage (11.23 GB) compared to Adam. The reduction arises because they do not need to store variance estimates of the gradients, which are a part of the Adam algorithm. They only require storage for the model parameters and gradients, along with the momentum term, which is less memory-intensive than the variance term in Adam.

3. Plain SGD: Exhibits the lowest memory usage (8.41 GB), which aligns with its simplicity. Plain SGD does not require additional memory for storing momentum terms or variance estimates of the gradients. It only needs to store the model parameters and the gradients, hence the significantly lower memory requirement.

In conclusion, the memory requirements vary based on the additional states each optimizer needs to store, with Adam variants being the most memory-intensive due to their complex structure and additional state requirements.

2.2 Parameter Efficient Fine-tuning - LoRA with Different Ranks

LoRA[3] is a technique that inserts low-rank matrices into pre-existing weights of a large model, allowing for efficient adaptation with minimal additional parameters. It is applied to the linear layers of the model, offering a balance between model performance and efficiency. After just changing the rank of LoRA and training 20,000 iterations in 1 batch size, we get the GPU usage result shown in Tabel 2. The advantage of LoRA is clear it reduces the usage of GPU memory greatly, and the smaller the rank, the lower the usage memory will be. However, the disadvantage is that the model using LoRA is underfitted.

Table 2: Comparison of Different LoRA ranks on GPU P100

	Rank 1	Rank 2	Rank 8	Rank 16	Original
GPU Memory Percentage (%)	50.47	50.52	50.74	51.29	74.06
GPU Memory Allocated (GB)	8.0245	8.0323	8.0675	8.1554	11.7745

Compared with the full-parameter-tuned model, we get the Figures 5, 6 and 7. All LoRA with different ranks behave similarly in training, they firstly sharply down, and then converge steadily. But for testing, the larger the rank, the more punctuated. In comparison, the full-parameter-tuned model converges faster and more.

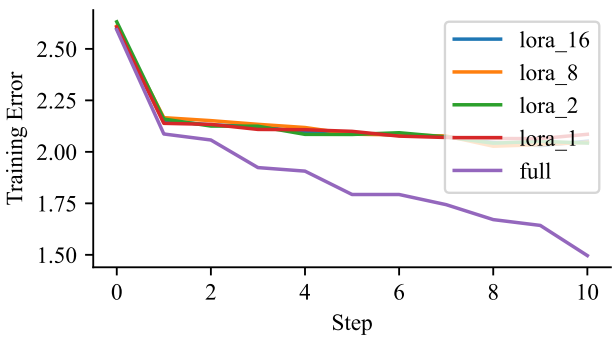

Figure 5: Reward Scores Over Steps (Training)

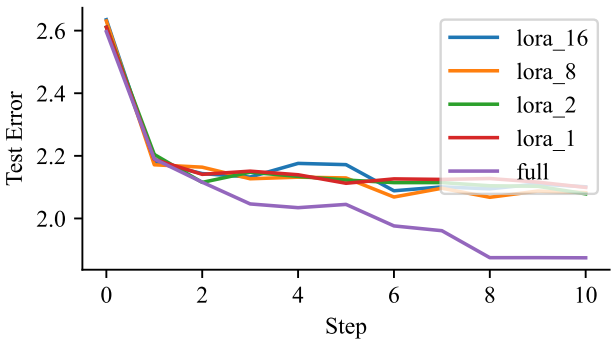

Figure 6: Reward Scores Over Steps (Testing)

Surprisingly, from Figure 7, even though models using LoRA do perform not well at training and testing errors, they get reward scores better than the full-parameter-tuned model just from 20,000 times iterations. We think the LoRA is just like the Dropout function in the neuro networks, it successfully prevents the model from overfitting.

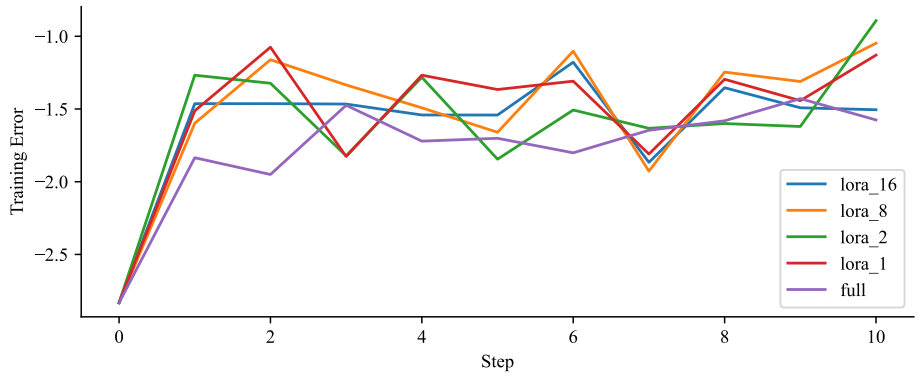

Figure 7: Reward Scores Over Steps

3 Other Attempts

3.1 Only Helpful Dataset

By only using `SFTDataset` as the training and test data, it has 21917 training samples and 1177 test samples. We can get results shown in Figure 8 and 9. We trained using batch size 1 and iterated 40,000 times. It seems that the model quickly overfits and the dialogue ability is not quite worse than the model fine-tuned on the whole reconstructed SFT dataset.

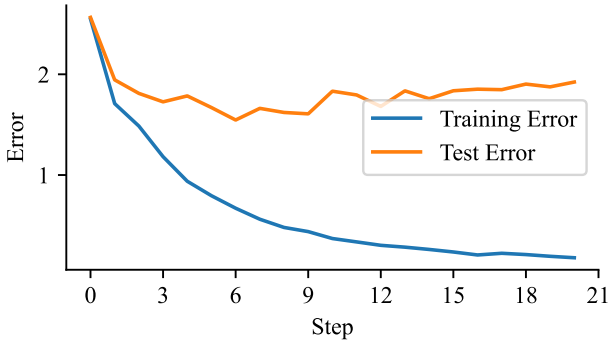

Figure 8: Helpful Dataset Errors Over Steps

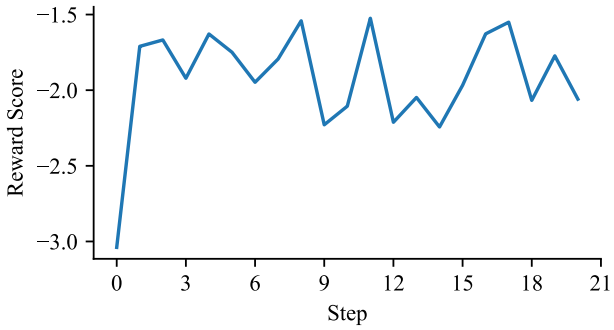

Figure 9: Helpful Dataset Reward Scores Over Steps

3.2 Gradient Accumulation

By using Gradient Accumulation, we can use a larger batch size. The optimizers section above uses Algorithm 3 with batch size 4. However, we found that the performance of that seems no different compared with the model trained with batch size 1. Maybe we should try larger ones.

3.3 DPO

Unfortunately, even though we tried kinds of techniques to use two GPU cards to make it happen, due to the limitations of the GPU memory, we finally failed to deploy the DPO shown in Algorithm 4. We tried to mount different models into different cuda, but the data can't communicate between each GPU when we are going to calculate the loss. We chose `RLHFDataset` as the training data, and `EYLSFTStaticDataset` as the test data. Two models, the policy model and reference model are created to do the training, and the policy model is used to do the validation.

References

[1] OpenAI. Chatgpt. https://chat.openai.com, 2023.

[2] Shrish. Breaking the memory barrier: how zero revolutionizes large model training. https://medium.com/@Shrishml/breaking-the-gmemory-barrier-how-zero-revolutionizes-large-language-model-training-8e00d2e2f3 2023.

[3] Edward J. Hu, Yelong Shen, and et al. Lora: Low-rank adaptation of large language models. *arXiv preprint arXiv:2106.09685*, 2021.

Appendix

A Some Algorithms

Algorithm 3 Gradient Accumulation

Set accumulation steps to a desired value, e.g., 2
Get a batch of training data $(\mathbf{x}, \mathbf{y})$ from the train dataloader
for each step in the batch size **do**
  Select a subset of data $(\mathbf{x}_b, \mathbf{y}_b)$ corresponding to the current step
  Move $\mathbf{x}_b$ and $\mathbf{y}_b$ to the specified device
  {Forward Pass}
  Compute logits $\mathbf{L}$ using the model on $\mathbf{x}_b$
  Reshape logits $\mathbf{L}$ from shape (B, T, C) to $(B \times T, C)$
  Reshape targets $\mathbf{y}_b$ from shape (B, T) to $(B \times T)$
  Compute the loss using cross-entropy between $\mathbf{L}$ and $\mathbf{y}_b$
  Accumulate the loss: loss = loss/accumulation steps
  {Backward Pass}
  Compute the gradients by backpropagation
  {Optimization Step}
  if step is a multiple of accumulation steps **then**
    Update the model parameters using the optimizer
    Zero the gradients of the optimizer
  end if
end for

Algorithm 4 DPO Training

Get a batch of input IDs and attention masks from the train dataloader
{Prepare Inputs}
Split the input IDs and attention masks into chosen and rejected sets
Move chosen and rejected inputs and masks to the specified device
{Compute Logits for Policy and Reference Models}
Compute logits for chosen and rejected inputs using the policy model
Compute logits for chosen and rejected inputs using the reference model without gradient calculation
{Combine Logits and Labels for Loss Calculation}
Concatenate logits from chosen and rejected inputs for both policy and reference models
Concatenate labels and masks from chosen and rejected inputs
{DPO Loss Calculation}
Compute log softmax of the policy and reference model logits
Select the log probabilities corresponding to the true labels
Calculate the difference in log probabilities between policy and reference models
Compute DPO loss as the negative log-sigmoid of the scaled logits difference, averaged over the batch
{Backward Pass and Optimization}
Zero the gradients of the optimizer
Compute the gradients by backpropagation
Update the policy model parameters using the optimizer