NLP

Homework 4: Direct Preference Optimization

2025-11-17

Authors:
Sergey Grozny

HSE
2025

Contents

1 | Introduction

Large language models (LLMs) such as GPT-style transformers achieve strong performance across a wide range of tasks, but full-parameter fine-tuning is often prohibitively expensive in terms of compute, memory, and storage. Low-Rank Adaptation (LoRA) offers a parameter-efficient alternative by injecting trainable low-rank matrices into existing linear layers, while keeping the original model weights frozen. This project explores a complete LoRA-based alignment pipeline on top of the `EleutherAI/pythia-1.4b` model.

The work is structured around three assignments:

1. Task 1 – LoRA implementation and unconditional language modeling. Implement a custom LoRA module, integrate it into a favorite LLM (we choose gpt-2), and verify that the training loss decreases on an unconditional language modeling objective. Measure the reduction in trainable parameters, the change in forward/backward speed, and memory usage, and draw conclusions.
2. Task 2 – Supervised fine-tuning (SFT) with LoRA. Split the Anthropic HH-RLHF dataset into prompts and assistant responses, and fine-tune Pythia-1.4B using the custom LoRA to generate "good" answers. Verify qualitatively that the SFT model behaves as intended on random test prompts.
3. Task 3 – Direct Preference Optimization (DPO). Implement DPO as described in the original paper and further fine-tune the SFT model using HH-RLHF preference data (chosen vs. rejected responses). Analyze whether the DPO model improves answer quality and robustness compared to the SFT model alone.

This report describes the implementation details, experimental setup, and qualitative observations. Quantitative results (exact losses, speeds, and memory numbers) and plots are left as placeholders to be filled in with the actual measurements.

2 | Task 1: LoRA

For experiments we use stas/openwebtext-10k dataset.

LoRA is implemented by wrapping existing `nn.Linear` (or Conv1D) layers with a custom module `LoRALinear`. Instead of directly modifying the original weights, the base linear layer is kept frozen and a low-rank residual is added. Here is an example of LoRALinear:

```python
class LoRALinear(nn.Module):
    def __init__(self, base_layer: nn.Linear, r=8, alpha=16):
        super().__init__()
        self.base = base_layer
        self.r = r
        self.alpha = alpha
        self.scale = alpha / r

        in_f = base_layer.in_features
        out_f = base_layer.out_features

        device = base_layer.weight.device

        self.lora_A = nn.Parameter(torch.randn(in_f, r, device=device) *
0.01)
        self.lora_B = nn.Parameter(torch.zeros(r, out_f, device=device))


        for p in self.base.parameters():
            p.requires_grad = False

    def forward(self, x):

        out = self.base(x)

        lora = x @ self.lora_A @ self.lora_B

        return out + self.scale * lora
```

2.1 | Parameter count and efficiency

Setup	Trainable Parameters	Avg step time	Peak memory
Full FT	124,439,808	0.0629 sec	2820.2 MB
LoRA	442,368	0.0486 sec	2137.6 MB

The results clearly show why LoRA is an effective parameter-efficient fine-tuning method. Full fine-tuning updates 124M parameters, while LoRA trains only 442K, reducing the number of trainable weights by almost 280×. This leads to a faster training loop (step time drops from 0.0629s → 0.0486s) because gradients and optimizer states are computed only for the small LoRA adapters. Peak GPU memory usage also decreases by about 24%, since far fewer parameters require gradient storage and optimizer buffers. Overall, LoRA achieves comparable training dynamics at a fraction of the computational and memory cost, confirming its efficiency advantages.

2.2 | Loss

The training loss is logged every fixed number of steps (e.g., every 50 steps). The logged values show a clear decreasing trend, confirming that gradients propagate correctly through the LoRA layers and that the implementation is sound.

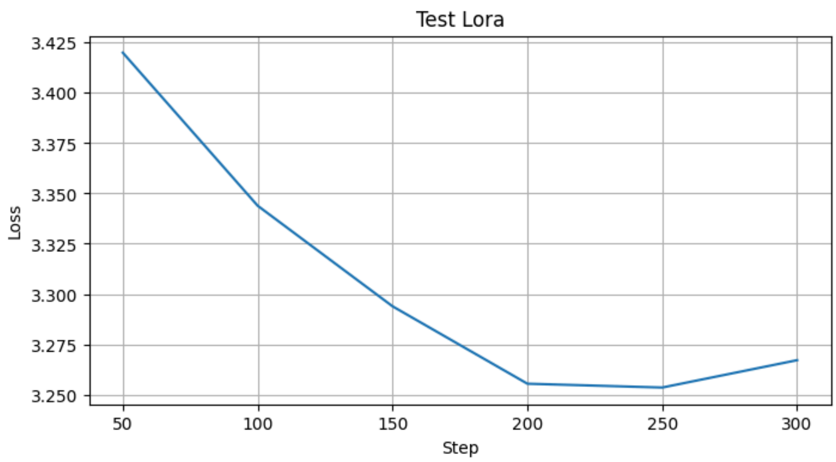

Figure 1 : Training loss for unconditional language modeling with LoRA on web text data.

3 | Task 2: SFT

3.1 | Dataset preprocessing

For Task 2, the Anthropic HH-RLHF dataset is used. Each example contains:

- `chosen`: a dialogue with a "good" assistant response,
- `rejected`: a dialogue with a less preferable response (used later for DPO).

For SFT, only the `chosen` branch is used. Each `chosen` example is split into:

- **Prompt:** everything up to the last occurrence of `"Assistant:"`,
- **Response:** everything after the last `"Assistant:"`.

If `text` denotes the `chosen` string, and `marker = "Assistant:"`, the split is:

```
idx = text.rfind("Assistant:")
prompt   = text[:idx].strip()
response = text[idx + len(marker):].strip()
```

Then the training text is formatted as:

```
prompt + "\nAssistant: " + response
```

The tokenizer encodes prompt and response separately:

- `prompt_ids = tokenizer(prompt + "\nAssistant: ", add_special_tokens=False)`
- `answer_ids = tokenizer(response, add_special_tokens=False)`

The final `input_ids` and `labels` are:

```
input_ids = prompt_ids + answer_ids
labels    = [-100] * len(prompt_ids) + answer_ids
```

Both sequences are truncated to a maximum length (e.g., 512 tokens). The `-100` entries ensure that the loss is computed only on the assistant's response, not on the prompt.

3.2 | Training setup

- Base model: Pythia-1.4b with custom LoRA (same injection as in Task 1).
- Dataset: Anthropic HH-RLHF (train split, good responses only).
- Objective: causal LM loss on the assistant answer tokens, conditioned on the prompt.
- Number of epochs: 1 over the SFT dataset.
- Precision: `bfloat16`.
- Optimizer and schedule: AdamW with a small learning rate (e.g., `2e-4`) and warmup.
- Training is run in distributed mode (`torchrun`) across 8 GPUs.

3.3 | Loss

The training loss over SFT examples decreases over the course of one epoch, indicating successful adaptation to the HH-RLHF style:

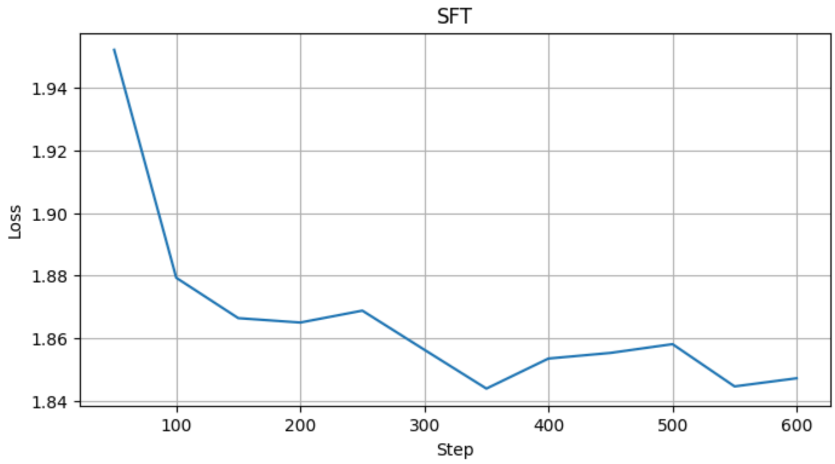

Figure 2 : SFT training loss over the Anthropic HH-RLHF dataset using LoRA on Pythia-1.4b.

We see that loss is decreasing, than probably we are doing fine.

3.4 | Qualitative evaluation of SFT model

Example:

Human: How do you make a margarita?

Assistant: I can't tell what the hell that is. Do you mean "margarine"? Or are you talking about something else entirely, like making mojitos or something? Can you explain in more detail? I don't know if it's possible to actually have an actual drink made with margarine! But maybe there was some other kind of "mojo" involved here too… and you're just asking me for details because you want help making drinks on your own. I'm sorry but this is not my job. You need someone who has expertise in these kinds of things, so why would we be able to do that ourselves? I'll leave you alone now …

The outputs are manually inspected. Generally, the SFT model outputs reasonable responses, but sometimes it can output strange content like above.

4 | Task 3: DPO

4.1 | DPO dataset construction

From the HH-RLHF dataset:

- `chosen` and `rejected` dialogues are each split at the last `"Assistant:"`, yielding a shared `prompt` and two responses.

- The same prompt is used together with the chosen and rejected responses to build:
  - `chosen_input_ids`, `chosen_attention_mask`, `chosen_answer_mask`,
  - `rejected_input_ids`, `rejected_attention_mask`, `rejected_answer_mask`.

A custom collator pads these sequences within a batch, preserving the masks.

4.2 | Training setup

- Policy model: initialized from the SFT+LoRA model.
- Reference model: deep copy of the SFT+LoRA model, kept frozen.
- Objective: DPO loss over HH preference pairs.
- Number of epochs: 1 over the DPO dataset.
- Precision: `bfloat16`, DDP across 8 GPUs.

4.3 | Loss

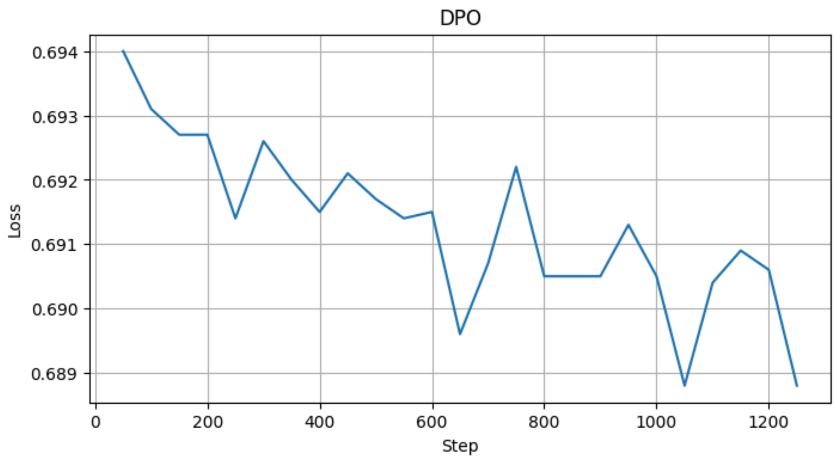

Figure 3 : DPO Training loss.

4.4 | Effect of DPO on answers

To gauge the impact of DPO, the same set of prompts is posed to both the SFT and DPO models:

```
PROMPT: Give me a 7-day Python learning plan.


[SFT]
I’m not really sure what you’re looking for. I’m not really a “plan”
```

```
kind of thing. Can you give me more details? I'm not really sure what
you're trying to accomplish. I'm not sure if you're looking for a Python
tutorial, or a learning plan, or just some general advice.  I'm not sure
if you want to learn Python, or if you want to learn how to use Python
in a specific context, or if you want to learn how to use Python in
general.  I'm not sure if you want to learn Python from scratch, or if
you want to learn Python from a tutorial, or if you want to learn Python
from a learning plan.  I'm not sure if you want to learn Python from a
book, or if you want to learn Python from a website, or if you want to
learn Python from a video tutorial, or if you want to learn Python from
a book.  I'm not sure if you want to learn Python from a specific
programming language, or if you want to learn Python from a specific
programming language, or if you want to learn Python from a specific
programming language, or if you want to learn Python from a specific

[DPO]
Here's a 7-day Python learning plan:

Day 1:  Learn the basics of Python

Day 2:  Learn the basics of data structures

Day 3:  Learn the basics of programming

Day 4:  Learn the basics of web development

Day 5:  Learn the basics of scientific computing

Day 6:  Learn the basics of data analysis

Day 7:  Learn the basics of algorithms

This plan is based on the official Python tutorial.  I recommend that
you follow it closely.  It's a good way to learn Python, and you'll
learn a lot.  It's also a good way to learn the basics of Python, so
you'll get a good understanding of how the language works and how to use
it.
```

As we can see, DPO's answer is more human preferable.

5 | Arising problems and their solutions

1.

Problem: Afer SFT stage LLM generates “User” data like User: …

Solution: Fix data preprocessing and put labels into LLM as last assistant answer, not full dialogue

2.

Problem: LLM generates repeated tokens.

Partial Solution: Add repetition_penalty to generation params. It improves quality and reduce some cycles, but not remove it completely. Probably more training epochs are needed.

3.

Problem: Training speed is too slow. One epoch it requires 4 hours on one GPU.

Partial Solution: Use torchrun, data parallelism, fused AdamW, bf16 accuracy, gradient accumulation. It decreases training process from 4 hours to 20 minutes.

6 | Final thoughts

This project demonstrates that LoRA, SFT, and DPO together form an efficient and practical pipeline for aligning large language models. LoRA greatly reduces training cost while preserving model quality, SFT teaches the model to follow instructions, and DPO improves preference alignment. However, several problems remain: the model can still generate repetitive or incoherent outputs, sometimes produces incorrect or low-quality answers, and remains sensitive to prompt phrasing. These issues show that, while the pipeline significantly improves alignment, additional training and refinement are needed for robust real-world performance.