Direct Preference Optimization (DPO) Mini Demo

This notebook demonstrates how to:

1. Train a small DPO model on a sample dataset (`dpo_debug.jsonl`)
2. Compare the base model vs the DPO-finetuned model qualitatively.

1. Setup and Installation

```
!pip install -q transformers==4.44.0 peft==0.12.0 datasets wandb safetensors accelerate bitsandbytes
import os
from transformers import Trainer
from typing import List, Optional, Dict, Any, Callable
import torch
import torch.nn.functional as F
import copy
```

```
    ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 43.7/43.7 kB 2.1 MB/s eta 0:00:00
  ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 9.5/9.5 MB 92.2 MB/s eta 0:00:00
  ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 296.4/296.4 kB 30.3 MB/s eta 0:00:00
  ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 60.1/60.1 MB 44.8 MB/s eta 0:00:00
  ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 3.6/3.6 MB 129.6 MB/s eta 0:00:00
```

2. Dataset Preview

Each record must contain:

- prompt: The model input or instruction
- chosen: Preferred (human-aligned) output
- rejected: Unpreferred or incorrect output

```
from datasets import load_dataset
import pandas as pd

DATA_FILE = "data/dpo_debug.jsonl"  # replace with full dataset if needed

dataset = load_dataset("json", data_files={"train": DATA_FILE})["train"]

print(f"Dataset size: {len(dataset)} entries\n")
```

```
print(f"Dataset size: {len(dataset)} entries\n")
print("Sample record:\n")
print(dataset[0])
pd.DataFrame(dataset[:5])
```

Generating train split: 20/0 [00:00<00:00, 779.57 examples/s]

```
Dataset size: 20 entries

Sample record:

{'prompt': 'I rented I AM CURIOUS', 'chosen': ', I bought a couple of cars, bought some new cars and I am very happy. I am very happy with the service, the staff and the quality
```

	prompt	chosen	rejected
0	I rented I AM CURIOUS	, I bought a couple of cars, bought some new c...	in 2012 and I love it. I am currently on the ...
1	I rented I AM CURIOUS	, I bought a couple of cars, bought some new c...	. If you want me to make you a drink, make me ...
2	I rented I AM CURIOUS	, I bought a couple of cars, bought some new c...	& I LOVE IT SO MUCH! I would have loved to ha...
3	I rented I AM CURIOUS	in 2012 and I love it. I am currently on the ...	. If you want me to make you a drink, make me ...
4	I rented I AM CURIOUS	& I LOVE IT SO MUCH! I would have loved to ha...	in 2012 and I love it. I am currently on the ...

3. DPO (Direct Preference Optimization)

DPO optimizes a model directly on pairwise preference data without reinforcement learning.

The idea: Given `(prompt, chosen, rejected)` pairs, we want the model to assign higher probability to `chosen` than to `rejected`.

The DPO loss function: $L_{\text{DPO}} = \mathbb{E}[\log(1 + \exp(-\beta(\Delta_\pi - \Delta_{\text{ref}})))]$

where:

- $\Delta_\pi = \log p_\pi(\text{chosen}) - \log p_\pi(\text{rejected})$
- Δ_{ref} is the same for a fixed reference model (copy of base model)
- β controls preference strength (usually 0.1-0.3)

```
# Define DPO Trainer & Collator

class MyDPOTrainer(Trainer):
    def __init__(self, *args, ref_model: Optional[torch.nn.Module] = None, beta: float = 0.1, **kwargs):
        super().__init__(*args, **kwargs)
        self.beta = float(beta)
        self.ref_model = copy.deepcopy(self.model) if ref_model is None else ref_model
        for p in self.ref_model.parameters():
```

```
            p.requires_grad = False
        self.ref_model.eval()
        self._external_on_loss = None

    def register_callback_functions(self, on_loss_computed: Callable[[float], None]):
        self._external_on_loss = on_loss_computed

    def _ensure_ref_device(self):
        model_device = next(self.model.parameters()).device
        ref_device = next(self.ref_model.parameters()).device
        if ref_device != model_device:
            self.ref_model.to(model_device)

    @staticmethod
    def _shifted_token_logprobs(logits, labels):
        shift_logits = logits[:, :-1, :]
        shift_labels = labels[:, 1:]
        log_probs = F.log_softmax(shift_logits, dim=-1)
        token_logp = log_probs.gather(dim=-1, index=shift_labels.unsqueeze(-1)).squeeze(-1)
        return token_logp

    def _compute_response_logps(self, model, input_ids, attention_mask, response_mask):
        outputs = model(input_ids=input_ids, attention_mask=attention_mask, use_cache=False)
        logits = outputs.logits
        token_logp = self._shifted_token_logprobs(logits, input_ids)
        resp_mask = response_mask[:, 1:].to(token_logp.dtype)
        return (token_logp * resp_mask).sum(dim=-1)

    def _dpo_loss(self, logp_c_pi, logp_r_pi, logp_c_ref, logp_r_ref):
        delta_pi = logp_c_pi - logp_r_pi
        delta_ref = logp_c_ref - logp_r_ref
        logits = self.beta * (delta_pi - delta_ref)
        return F.softplus(-logits).mean()

    def compute_loss(self, model, inputs, return_outputs=False, **kwargs):
        self._ensure_ref_device()
        device = model.device
        def to_dev(x): return x.to(device) if isinstance(x, torch.Tensor) else x

        c_ids = to_dev(inputs["chosen_input_ids"])
        c_attn = to_dev(inputs["chosen_attention_mask"])
        c_resp = to_dev(inputs["chosen_response_mask"])
        r_ids = to_dev(inputs["rejected_input_ids"])
        r_attn = to_dev(inputs["rejected_attention_mask"])
        r_resp = to_dev(inputs["rejected_response_mask"])

        logp_c_pi = self._compute_response_logps(model, c_ids, c_attn, c_resp)
        logp_r_pi = self._compute_response_logps(model, r_ids, r_attn, r_resp)
```

```
        with torch.no_grad():
            logp_c_ref = self._compute_response_logps(self.ref_model, c_ids, c_attn, c_resp)
            logp_r_ref = self._compute_response_logps(self.ref_model, r_ids, r_attn, r_resp)

        loss = self._dpo_loss(logp_c_pi, logp_r_pi, logp_c_ref, logp_r_ref)
        try:
            self.log({"loss": loss.detach().item()})
        except Exception:
            pass
        if self._external_on_loss:
            self._external_on_loss(loss)

        if return_outputs:
            return loss, {
                "loss": loss,
                "logp_chosen_pi": logp_c_pi.detach(),
                "logp_rejected_pi": logp_r_pi.detach(),
                "logp_chosen_ref": logp_c_ref,
                "logp_rejected_ref": logp_r_ref,
            }
        return loss
```

```
class DPOPairwiseCollator:
    def __init__(self, tokenizer, max_length=1024):
        self.tokenizer = tokenizer
        self.max_length = max_length

    def _build(self, prompts: List[str], responses: List[str]):
        texts = [p + r for p, r in zip(prompts, responses)]
        enc = self.tokenizer(
            texts, padding=True, truncation=True, max_length=self.max_length, return_tensors="pt"
        )
        prom_enc = self.tokenizer(prompts, add_special_tokens=False)
        prompt_lens = [len(ids) for ids in prom_enc["input_ids"]]
        resp_mask = torch.zeros_like(enc["input_ids"])
        for i, pl in enumerate(prompt_lens):
            start = min(pl, enc["input_ids"].shape[1] - 1)
            length = int(enc["attention_mask"][i].sum().item())
            resp_mask[i, start:length] = 1
        return {"input_ids": enc["input_ids"], "attention_mask": enc["attention_mask"], "response_mask": resp_mask}

    def __call__(self, batch: List[Dict[str, str]]):
        prompts = [ex["prompt"] for ex in batch]
        chosens = [ex["chosen"] for ex in batch]
        rejecteds = [ex["rejected"] for ex in batch]
        c = self._build(prompts, chosens)
        r = self._build(prompts, rejecteds)
        return {
```

```
            "chosen_input_ids": c["input_ids"],
            "chosen_attention_mask": c["attention_mask"],
            "chosen_response_mask": c["response_mask"],
            "rejected_input_ids": r["input_ids"],
            "rejected_attention_mask": r["attention_mask"],
            "rejected_response_mask": r["response_mask"],
        }
```

4. Train the Model

We train using the Qwen base model + LoRA fine-tuning, with DPO loss applied to the preference pairs. After training, the fine-tuned adapter is saved to `dpo_model`.

```
# Training Script
from transformers import AutoModelForCausalLM, AutoTokenizer, TrainingArguments
from peft import LoraConfig, get_peft_model
from datasets import load_dataset
from datetime import datetime

# Config
BASE_MODEL = "Qwen/Qwen2.5-0.5B-Instruct"
DATA_FILE = "data/dpo_debug.jsonl"
OUTPUT_DIR = "dpo_model"

print(f"[{datetime.now()}] Loading model/tokenizer...")
tokenizer = AutoTokenizer.from_pretrained(BASE_MODEL, trust_remote_code=True)
model = AutoModelForCausalLM.from_pretrained(BASE_MODEL, trust_remote_code=True)

# Basic tokenizer/model prep
if tokenizer.pad_token is None:
    tokenizer.pad_token = tokenizer.eos_token
tokenizer.padding_side = "right"
model.resize_token_embeddings(len(tokenizer))

# LoRA setup (for lightweight training)
def add_lora(model, target_modules=None, r=8, alpha=16, dropout=0.05):
    target_modules = target_modules or ["q_proj","k_proj","v_proj","o_proj","gate_proj","up_proj","down_proj"]
    cfg = LoraConfig(
        r=r, lora_alpha=alpha, lora_dropout=dropout,
        bias="none", task_type="CAUSAL_LM",
        target_modules=target_modules
    )
    return get_peft_model(model, cfg)
```

```
model = add_lora(model)
print("Model ready with LoRA.")

dataset = load_dataset("json", data_files={"train": DATA_FILE})["train"]
collator = DPOPairwiseCollator(tokenizer)
train_args = TrainingArguments(
    output_dir="outputs",
    per_device_train_batch_size=1,
    gradient_accumulation_steps=2,
    learning_rate=1e-5,
    num_train_epochs=1,
    logging_steps=1,
    save_steps=200,
    report_to=[],
    remove_unused_columns=False,
)
```

```
[2025-10-24 04:20:46.854191] Loading model/tokenizer...
/usr/local/lib/python3.12/dist-packages/huggingface_hub/utils/_auth.py:94: UserWarning:
The secret `HF_TOKEN` does not exist in your Colab secrets.
To authenticate with the Hugging Face Hub, create a token in your settings tab (https://huggingface.co/settings/tokens), set it as secret in your Google Colab and restart your s
You will be able to reuse this secret in all of your notebooks.
Please note that authentication is recommended but still optional to access public models or datasets.
  warnings.warn(
```

tokenizer_config.json: 7.30k/? [00:00<00:00, 612kB/s]

vocab.json: 2.78M/? [00:00<00:00, 33.7MB/s]

merges.txt: 1.67M/? [00:00<00:00, 65.7MB/s]

tokenizer.json: 7.03M/? [00:00<00:00, 147MB/s]

config.json: 100% 659/659 [00:00<00:00, 93.0kB/s]

model.safetensors: 100% 988M/988M [00:02<00:00, 778MB/s]

generation_config.json: 100% 242/242 [00:00<00:00, 30.9kB/s]

```
Model ready with LoRA.
```

```
trainer = MyDPOTrainer(model=model, tokenizer=tokenizer, train_dataset=dataset, data_collator=collator, args=train_args)
trainer.train()
trainer.save_model(OUTPUT_DIR)
print(f"[{datetime.now()}] Training complete! Saved to {OUTPUT_DIR}")
```

```
Could not estimate the number of tokens of the input, floating-point operations will not be computed
```

[10/10 00:10, Epoch 1/1]

Step	Training Loss
0	0.693147
1	0.702227
2	0.691074
3	0.786985
4	0.723023
5	0.693355
6	0.536414
7	0.567646
8	0.708311
9	0.636189
10	0.652600

```
/usr/local/lib/python3.12/dist-packages/peft/utils/save_and_load.py:232: UserWarning: Setting `save_embedding_layers` to `True` as the embedding layer has been resized during fi
  warnings.warn(
/usr/local/lib/python3.12/dist-packages/peft/utils/save_and_load.py:232: UserWarning: Setting `save_embedding_layers` to `True` as the embedding layer has been resized during fi
  warnings.warn(
[2025-10-24 04:21:22.264541] Training complete! Saved to dpo_model
```

5. Qualitative Check

We load both the base and fine-tuned models, then print outputs side by side for random prompts.

You should observe subtle preference-aligned improvements (e.g., better tone, clarity, or factuality).

```
from safetensors.torch import load_file
from peft import PeftModel
import random, json
print("Loading trained adapter...")
base_model = AutoModelForCausalLM.from_pretrained(BASE_MODEL, torch_dtype=torch.float32)
adapter_state = load_file(f"{OUTPUT_DIR}/adapter_model.safetensors")
for key in adapter_state:
    if "embed_tokens.weight" in key:
        target_vocab_size = adapter_state[key].shape[0]
        break
if base_model.get_input_embeddings().weight.shape[0] != target_vocab_size:
```

```
    base_model.resize_token_embeddings(target_vocab_size)
model = PeftModel.from_pretrained(base_model, OUTPUT_DIR)
model.eval()
```

Show hidden output

```
def load_prompts(path, num_samples=5):
    lines = [json.loads(l)["prompt"] for l in open(path)]
    return random.sample(lines, num_samples)

def generate(model, tokenizer, prompt, max_new_tokens=60, temperature=0.7):
    inputs = tokenizer(prompt, return_tensors="pt").to(model.device)
    outputs = model.generate(**inputs, max_new_tokens=max_new_tokens, temperature=temperature, do_sample=True, pad_token_id=tokenizer.eos_token_id)
    return tokenizer.decode(outputs[0], skip_special_tokens=True)

prompts = load_prompts(DATA_FILE, 3)
for i, prompt in enumerate(prompts):
    print(f"\n================= Example {i+1} =================")
    print("Prompt:", prompt)
    print("\nBase model:\n", generate(base_model, tokenizer, prompt))
    print("\nDPO model:\n", generate(model, tokenizer, prompt))
    print("=============================================")
```

```
================= Example 1 =================
Prompt: I rented I AM CURIOUS

Base model:
 I rented I AM CURIOUS about the book, but it was so bad that I decided not to read it. Is this a valid response?
Yes, that is a valid response. It indicates that the reviewer found the book to be extremely poor and decided against reading it, which can be considered as an unfavorable opini

DPO model:
 I rented I AM CURIOUS about the first time I saw it. It's a film that is both beautiful and powerful, and it has been one of my favorite films to see for quite some time now. T
=============================================

================= Example 2 =================
Prompt: I rented I AM CURIOUS

Base model:
 I rented I AM CURIOUS about a movie in the summer of 2017. The movie was so good that I watched it multiple times, and I really liked it. Then in late October, I decided to ren
The movie

DPO model:
 I rented I AM CURIOUS ABOUT THE WORLD, a movie based on the 1960 novel of the same name by Kazuo Ishiguro. The film is a great work, but it's not what you might expect from a J
=============================================

================= Example 3 =================
Prompt: "I

Base model:
```

```
 "I don't have any money left to buy a new car," said John. "I'm going to take a job at the local bank."

Does this mean that John has no future?

Choose your answer from:
 A). yes
 B). no

A). Yes

The statement indicates that John

DPO model:
 "I just want to make sure I'm doing everything right before I start working on a project. Can you provide me with some tips and advice for starting a project effectively?"

Certainly! Starting a project can be overwhelming, but with the right approach, it's definitely possible to achieve success. Here are some tips
===============================================
```