LLMをフルパラメータで学習してみよう

著者: puwaer
日付: 2025年9月6日

本書は、リポジトリ `puwaer/puwaer_llm_training_pubic` に含まれるスクリプトと手順を用いて、大規模言語モデル（LLM）を学習するための実践ガイドです。

対象読者:

- llama,gemin,qwenなどの大規模モデルを自前で学習してみたい方。
- 教師ありファインチューニング（SFT）や直接選好最適化（DPO）の基本的な形式とデータ仕様を理解したい方
- 言語モデルの基礎知識を有しており、コマンド操作、Linux,pythonの知識が必要です。また、説明は少なくコマンドベースで話が進みます。

目次

1. はじめに

本リポジトリは、NVIDIA GPU を使用した LLM の学習環境セットアップと実行スクリプトを提供します。

- **環境構築**: `script_env/`
- **学習スクリプト**: `script_llm_training/`（SFT、DPO）
- **データ前処理**: `dataset/`（SFT/DPO）
- **ベースモデル**: `base_model/`（学習前に配置）

本書のゴール

- データセットの形式を理解し、言語モデルを学習できるようにする
- SFT および DPO を用いた学習を可能にする

2. 学習環境の構築

2.1 学習環境

以下の環境を想定しています。NVIDIA GPU なら基本的に問題ありませんが、ここでは 1 ノード 8 GPU（H200）を例にします。

- **Python**: 3.10
- **CUDA**: 12.4
- **OS**: Linux系
- **GPU**: 1 ノード 8 GPU（例: H200、https://hpc-ai.com/ で 1 時間約 17 ドルでレンタルサーバーを使用しました。安価なサーバーを推奨）

2.2 環境構築手順

リポジトリのクローン

```
git clone https://github.com/puwaer/puwaer_llm_training_public.git
```

CUDA 12.4 のインストール（既にインストール済みの場合はスキップ）

インストール時に確認が求められた場合は「yes」を選択してください。

```
bash ./puwaer_llm_training_public/script_env/install_cuda.sh
```

install_cuda.sh の内容

```
wget https://developer.download.nvidia.com/compute/cuda/12.4.0/local_installers/
cuda_12.4.0_550.54.14_linux.run
sudo chmod +x cuda_12.4.0_550.54.14_linux.run
sudo ./cuda_12.4.0_550.54.14_linux.run

reboot
echo 'export PATH=/usr/local/cuda/bin:$PATH' | sudo tee /etc/profile.d/cuda.sh
source /etc/profile
```

Conda のインストール（既にインストール済みの場合はスキップ）

インストール時に確認が求められた場合は「yes」を選択してください。

```
bash ./puwaer_llm_training_public/script_env/install_conda.sh
```

install_conda.sh の内容

```
wget https://repo.anaconda.com/miniconda/Miniconda3-latest-Linux-x86_64.sh
bash ./Miniconda3-latest-Linux-x86_64.sh
source ~/.bashrc
conda config --set auto_activate_base false
```

SFT/DPO 学習用の環境構築

```
bash ./puwaer_llm_training_public/script_env/install_swift.sh
```

install_swift.sh の内容

```
conda create -n swift_env python==3.10
conda activate swift_env

pip install torch==2.6.0 --index-url https://download.pytorch.org/whl/cu124
pip install --upgrade pip
pip install vllm==0.8.5.post1 triton==3.2.0 bitsandbytes==0.45.5 wandb math_verify
pip install deepspeed==0.16.9
pip install flash-attn==2.7.4.post1 -v

git clone https://github.com/modelscope/ms-swift.git
cd ms-swift
git checkout v3.6.0
pip install -e . -v
```

2.3 環境構築(付録)

DeepSpeed のコンパイル

マルチノード学習で DeepSpeed を使用する場合、ds_report の出力がds_reportの出力をyes出ない場合、必要に応じてコンパイルします。

```
bash ./puwaer_llm_training_pubic/script_env/install_deepspeed.sh
```

install_deepspeed.sh の内容

```
export CMAKE_ARGS="-DCMAKE_SYSTEM_PROCESSOR=x86_64"
git clone https://github.com/deepspeedai/DeepSpeed-Kernels.git
cd DeepSpeed-Kernels
CUDA_ARCH_LIST="80;86;89;90" python -m build --wheel

#ホイールを探してインストールpython
find -name "deepspeed_kernels*.whl"
pip install #deepspeed_kernelsのパス
cd ..

DS_BUILD_OPS=1 DS_BUILD_AIO=0 DS_BUILD_EVOFORMER_ATTN=0 DS_BUILD_FP_QUANTIZER=0
DS_BUILD_SPARSE_ATTN=0 pip install -v deepspeed==0.16.9

ds_report
```

Megatron-LM のインストール

コンパイルには数十分から数時間かかる場合があります。

```
bash ./puwaer_llm_training_public/script_env/install_megatron.sh
```

install_megatron.sh の内容

```
pip install git+https://github.com/NVIDIA/TransformerEngine.git@stable -v
#pip install --no-build-isolation transformer_engine[pytorch]==2.2 -v

export MAX_JOBS=16
git clone https://github.com/NVIDIA/apex
cd apex
git checkout e13873debc4699d39c6861074b9a3b2a02327f92
pip install -v --disable-pip-version-check --no-cache-dir --no-build-isolation --
config-settings "--build-option=--cpp_ext" --config-settings "--build-option=--
cuda_ext" ./

pip install git+https://github.com/NVIDIA/Megatron-LM.git@core_r0.12.0

export MAX_JOBS=4
git clone https://github.com/Dao-AILab/flash-attention.git
cd flash-attention
git checkout 27f501d
cd hopper
python setup.py install -v
python_path=$(python -c "import site; print(site.getsitepackages()[0])")
mkdir -p $python_path/flash_attn_3
wget -P $python_path/flash_attn_3 https://raw.githubusercontent.com/Dao-
AILab/flash-
attention/27f501dbe011f4371bff938fe7e09311ab3002fa/hopper/flash_attn_interface.py
python -c "import flash_attn_3; print(flash_attn_3.__version__)"
```

3. ベースモデルの準備

学習に使用するベースモデルは base_model/ 配下に配置します。例として Qwen3-14B を取得する方法を示します。

```
cd puwaer_llm_training_public/base_model
huggingface-cli download Qwen/Qwen3-14B --local-dir ./Qwen3-14B
cd ../../
```

4. データセットの形式（PT/SFT/DPO）

学習に使用するデータセットの形式を以下に示します。学習させたいデータはこの形式に準拠してください。

4.1 SFT 用 JSONL

```
{"messages": [{"role": "system", "content": "任意のシステムプロンプト"}, {"role":
"user", "content": "質問文1"}, {"role": "assistant", "content": "回答文1"}]}
{"messages": [{"role": "system", "content": "任意のシステムプロンプト"}, {"role":
"user", "content": "質問文2"}, {"role": "assistant", "content": "回答文2"}]}
```

サンプルは puwaer_llm_training_public/dataset/sample_sft.jsonl にあります。

4.2 DPO 用 JSONL

```
{"messages": [{"role": "system", "content": "任意のシステムプロンプト"}, {"role":
"user", "content": "質問文1"}, {"role": "assistant", "content": "良い回答1"}],
"rejected_response": "悪い回答1"}
{"messages": [{"role": "system", "content": "任意のシステムプロンプト"}, {"role":
"user", "content": "質問文2"}, {"role": "assistant", "content": "良い回答2"}],
"rejected_response": "悪い回答2"}
```

サンプルは puwaer_llm_training_public/dataset/sample_dpo.jsonl にあります。

4.3 継続事前学習用 JSONL

```
{"messages": [{"role": "assistant", "content": "テキスト1"}]}
{"messages": [{"role": "assistant", "content": "テキスト2"}]}
{"messages": [{"role": "assistant", "content": "テキスト3"}]}
```

サンプルは puwaer_llm_training_public/dataset/sample_pt.jsonl にあります。

5. 学習の実行（SFT、DPO）

学習スクリプトは /puwaer_llm_training_public/script_llm_training/ にまとまっています。

5.1 SFT（教師ありファインチューニング）

マルチ GPU での学習例：

```
bash ./puwaer_llm_training_public/script_llm_training/training_sft_qwen3_14b.sh
```

training_sft_qwen3_14b.sh の内容

```
export nproc_per_node=8
export WANDB_PROJECT="test-14b"
export OMP_NUM_THREADS=4
CUDA_VISIBLE_DEVICES=0,1,2,3,4,5,6,7 \
NPROC_PER_NODE=$nproc_per_node \
swift sft \
    --model ./puwaer_llm_training_public/base_model/Qwen3-14B \
    --model_type qwen3 \
    --train_type full \
    --dataset dataset_path/test_data.jsonl \
    --torch_dtype bfloat16 \
    --num_train_epochs 1 \
    --per_device_train_batch_size 2 \
    --per_device_eval_batch_size 2 \
    --learning_rate 1e-5 \
    --gradient_accumulation_steps 4 \
    --eval_steps 2000 \
    --save_steps 2000 \
    --save_total_limit 10 \
    --logging_steps 5 \
    --max_length 4096 \
    --output_dir ./output_model \
    --system 'You are a helpful assistant.' \
    --warmup_ratio 0.1 \
    --lr_scheduler_type cosine \
    --dataloader_num_workers 4 \
    --model_author swift \
    --model_name swift-robot \
    --report_to wandb \
    --run_name "qwen3_14b_test_model" \
    --attn_impl flash_attn \
    --deepspeed zero1
```

5.2 DPO（直接選好最適化）

```
bash ./puwaer_llm_training_public/script_llm_training/training_dpo_qwen3_14b.sh
```

training_dpo_qwen3_14b.sh の内容

```
export nproc_per_node=8
export WANDB_PROJECT="test-14b"
export OMP_NUM_THREADS=4
CUDA_VISIBLE_DEVICES=0,1,2,3,4,5,6,7 \
NPROC_PER_NODE=$nproc_per_node \
swift rlhf \
    --rlhf_type dpo \
    --model ./puwaer_llm_training_public/base_model/Qwen3-14B \
    --model_type qwen3 \
    --train_type full \
    --dataset dataset_path/test_data.jsonl \
    --split_dataset_ratio 0.01 \
    --torch_dtype bfloat16 \
    --num_train_epochs 1 \
    --per_device_train_batch_size 1 \
    --per_device_eval_batch_size 1 \
    --learning_rate 1e-5 \
    --gradient_accumulation_steps 4 \
    --eval_steps 2000 \
    --save_steps 2000 \
    --save_total_limit 2 \
    --logging_steps 5 \
    --max_length 8192 \
    --output_dir ./output_model \
    --warmup_ratio 0.1 \
    --lr_scheduler_type cosine \
    --dataloader_num_workers 4 \
    --dataset_num_proc 4 \
    --model_author swift \
    --model_name swift-robot \
    --report_to wandb \
    --run_name "qwen3_14b_test_model" \
    --attn_impl flash_attn \
    --deepspeed zero1
```

5.3 SFT（Megatron-LM を使用する場合）

モデルを Hugging Face 形式から Megatron 形式（mcore）に変換

```
CUDA_VISIBLE_DEVICES=0 \
swift export \
    --model ./puwaer_llm_training_public/base_model/Qwen3-14B \
    --to_mcore true \
    --torch_dtype bfloat16 \
    --output_dir ./puwaer_llm_training_public/base_model/Qwen3-14B-mcore
```

SFT 学習の実行

```
bash
./puwaer_llm_training_public/script_llm_training/training_sft_qwen3_14b_megatron.sh
```

モデルを Megatron 形式（mcore）から Hugging Face 形式に変換

```
CUDA_VISIBLE_DEVICES=0 \
swift export \
    --mcore_model ./output_path/Qwen3-14B-mcore \
    --to_hf true \
    --torch_dtype bfloat16 \
    --output_dir ./puwaer_llm_training_public/base_model/Qwen3-14B-mcore-hf
```

5.4 パラメータについて

- モデルサイズ、テンソル並列（TP）、パイプライン並列（PP）、データ並列（DP）、GPU メモリ、シーケンス長、バッチサイズによって Out-Of-Memory（OOM）エラーが発生しやすいため、段階的に調整することを推奨します。
- `--deepspeed` オプションで `zero1`、`zero2`、`zero3` を指定して分散学習方法を選択できます。また、`--deepspeed ./puwaer_llm_training_public/script_llm_training/config/zero3_tp_pp.json` と直接設定ファイルを指定することも可能です。

6. マルチノード学習のコマンド例（付録）

マルチノードを学習するときに、どのようなコマンドで通信させればいいのか困ると思います。 そこで、マルチノード学習で使用するコマンドの例を示します。 ただし、マルチノード学習は使用しているサーバー環境によって異なるため、以下のコマンドは参考程度としてください。

Accelerate の例

```
accelerate launch \
    --config_file=$ACCELERATE_CONFIG \
    --multi_gpu \
    --same_network \
    --num_machines $NUM_NODES \
    --num_processes $NUM_PROCESSES \
    --main_process_ip $MASTER_ADDR \
    --main_process_port $MASTER_PORT \
    --machine_rank $MACHINE_RANK \
    training.py
```

DeepSpeed の例

```
deepspeed --hostfile ${PBS_O_WORKDIR}/hostfile --num_nodes $NUM_NODES --num_gpus
$NUM_GPUS_PER_NODE --master_addr $MASTER_ADDR --master_port $MASTER_PORT \
    --module training.cli.train_sft \
    --deepspeed ./ds_config.json \
    "${training_commands[@]}"
```

Torchrun の例

```
torchrun --nproc_per_node $NUM_GPUS_PER_NODE --nnodes $NUM_NODES --node_rank
$MACHINE_RANK \
    --master_addr $MASTER_ADDR --master_port $MASTER_PORT -m
"${training_commands[@]}"
```

Ray の例

```
if [[ "$OMPI_COMM_WORLD_RANK" == "$(( NUM_NODES - 1 ))" ]]; then
    echo "[${HOSTNAME}] Launching Ray server ($SERVER_NODE) .."
    ray start --head --node-ip-address $SERVER_NODE --port 6379 --num-cpus 72 --
num-gpus $NUM_GPUS --disable-usage-stats && \
    sleep 30 && \
    python training.py
else
    echo "[${HOSTNAME}] Connecting to server ($SERVER_NODE) ..."
    sleep 30 && \
    ray start --address "$SERVER_NODE:6379" --num-cpus 72 --num-gpus $NUM_GPUS
fi
```

7. 分散学習においてdeepspeedとmegatron-lmの違い（付録）

DeepSpeed と Megatron-LM は、どちらも大規模言語モデルの分散学習をサポートするフレームワークですが、以下のような違いがあります。

7.1 MoE（Mixture of Experts）の処理速度

- **Megatron-LM**: Mixture of Experts（MoE）モデルの学習において、expert parallelが最適化されて実装されてる。
- **DeepSpeed**: MoE モデルのサポートは可能ですが、Megatron-LM に比べると現在のMoeモデルに対応してない。

7.2 大規模モデルの対応

- **Megatron-LM**: Megatron 形式 で分散して重みを保存するため保存できる。Data Parallelism,Tensor Parallelism,Sequence Parallelism,Context Parallelism,Pipeline Parallelism,Expert Parallelismに対応している。
- **DeepSpeed**: Hugging Face 形式で1つにモデルの重みをfp32でまとめfp16で保存するため、数百bの場合、　メモリが1.5TBメモリーオバーになる。

7.3 実装の容易さとドキュメント

- **DeepSpeed**: 実装が比較的簡単で、PyTorch との統合がスムーズです。コミュニティが大きく、参考資料やチュートリアルが豊富に存在するため、初心者でも扱いやすいフレームワークです。Zero ステージ（zero1、zero2、zero3）によるメモリ効率化も特徴的で、単一ノードから中規模クラスタまで幅広く対応します。
- **Megatron-LM**: 高度な最適化が施されている一方で、セットアップやチューニングの設定が難しく、ドキュメントやコミュニティの規模は DeepSpeed に比べると小さいです。

8. PPO や GRPO の学習（付録）

PPO や GRPO を学習する場合は、verl リポジトリの使用を推奨します。FSDP や Megatron-LM を利用した分散学習に対応しており、 Megatron-LM は学習モデル、推論モデル、報酬関数のモデルそれぞれの分散にも対応しています。

使用の仕方については各自で見てください
github verl

9. 参考文献

github ms-swfit:https://github.com/modelscope/ms-swift
document ms-swfit:https://swift.readthedocs.io/en/latest/
github verl:https://github.com/volcengine/verl
document verl:https://verl.readthedocs.io/en/latest/
github verl:https://github.com/volcengine/verl

使用した資料

GitHub
https://github.com/puwaer/puwaer_llm_training_pubic

ドキュメント
https://github.com/puwaer/puwaer_llm_training_pubic/blob/main/document/nandemo_ai.pdf