

BAI SandboxでのLLM Finetuning

Slurm環境でのTRL SFTTrainer実行例

ソフトバンク株式会社

AI戦略室 AI&データサイエンス統括部 AI基盤開発部

廣淵 亮太

概要：Slurm環境においてLLMのFinetuningを行う実行例の紹介

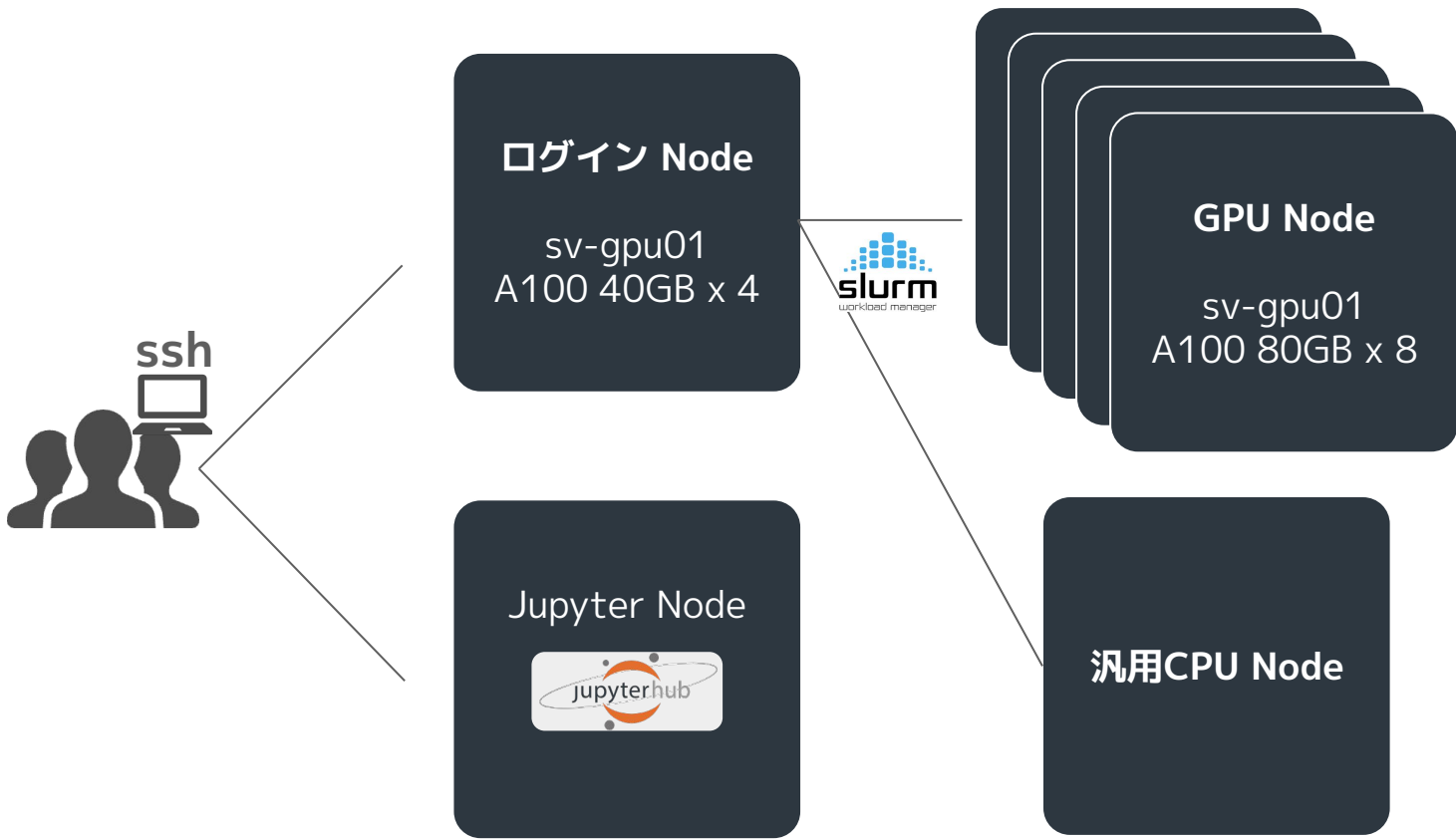
- Beyond AI SandboxとSlurm概要
- LLMのFinetuning
 - 要素技術概要
 - TRLのSFTTrainerベースのコードの実行例
 - vLLMでの推論例



Sandboxの使用例を紹介しますが、ベストプラクティスというわけではありません。
Sandbox運用チームの方や、その他Slurmなど詳しい方 ぜひコメントください

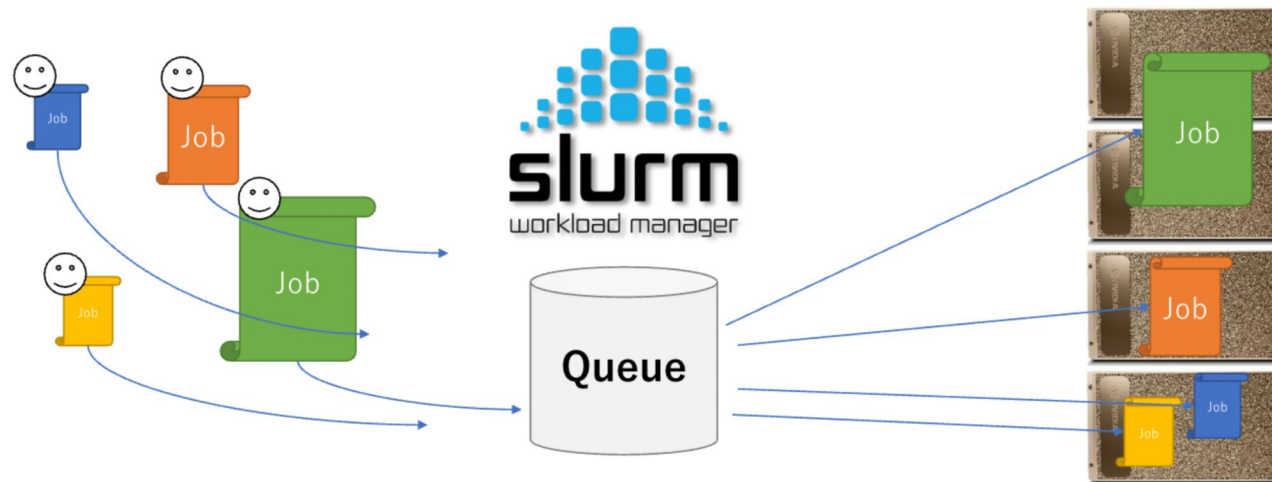
Beyond AI Sandbox概要

Beyond AI Sandbox概要



※実際には踏み台などあり

Slurm: HPC環境向けのジョブスケジューリングとリソース管理システム



- 情報取得系
 - **sinfo**: ノード状況の確認
 - **squeue**: jobキューの表示
- 実行・停止系
 - **srun**: 対話的なプログラムの実行など(jobの追加)
 - **sbatch**: バッチジョブの投入 (基本こちら)
 - **scancel**: 指定したjobを停止

流れ

1. batch実行ファイル(.sh)の作成
2. sbatchでジョブの投入
3. squeueで投入されたジョブの確認
4. logの確認
5. 完了

sinfo

ノードの状態を確認する

```
ryota.hirobuchi@sv-gpu01:~$ sinfo
PARTITION  AVAIL  TIMELIMIT  NODES  STATE NODELIST
sv-dgx      up 4-04:00:00      1   mix sv-dgx01
sv-dgx      up 4-04:00:00      1  alloc sv-dgx05
sv-dgx      up 4-04:00:00      3   idle sv-dgx[02-04]
test*       up   1:00:00      1   mix sv-dgx01
test*       up   1:00:00      1  alloc sv-dgx05
test*       up   1:00:00      3   idle sv-dgx[02-04]
cpuonly     up 4-04:00:00      1   idle sv-cpu01
sb_cotofure up 4-04:00:00      1   mix sv-dgx01
sb_cotofure up 4-04:00:00      1  alloc sv-dgx05
sb_cotofure up 4-04:00:00      3   idle sv-dgx[02-04]
```

squeue

```
ryota.hirobuchi@sv-dgx01:~/vllm_infer$ squeue
      JOBID PARTITION    NAME    USER  ST       TIME  NODES NODELIST(REASON)
      81170 sb_cotofu [REDACTED]  R 2-04:40:11      1 sv-dgx01
      81201  sv-dgx infer_ll ryota.hi  R        0:43      1 sv-dgx01
      81176  sv-dgx [REDACTED]  R 1-02:07:18      1 sv-dgx05
```


SBATCHで推論を投げてみる

```
#!/bin/bash
#SBATCH --job-name=test-slurm    # name
#SBATCH -p sv-dgx                # partition
#SBTACH --mem=10G                # mem
#SBATCH --time 1:00:00           # maximum execution time (HH:MM:SS)
#SBATCH --output=job_%x-%j_stdout.out  # output file name
#SBATCH --error=job_%x-%j_stdout.out   # stdout file name

echo "hello world"
sleep 30
echo "goodbye world"
```

Partition

Slurmでのリソース管理の単位
運用者に指定されたパーティションを使用する

メモリ指定

#SBATCH --mem=xxxG を極力指定しましょう

```
ryota.hirobuchi@sv-gpu01:~/slurm_examples$ ls
echo.slurm
ryota.hirobuchi@sv-gpu01:~/slurm_examples$ sbatch echo.slurm
Submitted batch job 81353
ryota.hirobuchi@sv-gpu01:~/slurm_examples$ squeue
      JOBID PARTITION    NAME    USER  ST       TIME  NODES NODELIST(REASON)
      81353   sv-dgx  test-slu ryota.hi  R        0:01        1 sv-dgx02
      81322   sv-dgx  [REDACTED]  R    4:48:26        1 sv-dgx01
ryota.hirobuchi@sv-gpu01:~/slurm_examples$ cat job_test-slurm-81353_stdout.out
hello world
ryota.hirobuchi@sv-gpu01:~/slurm_examples$ squeue
      JOBID PARTITION    NAME    USER  ST       TIME  NODES NODELIST(REASON)
      81322   sv-dgx  [REDACTED]  R    4:49:07        1 sv-dgx01
ryota.hirobuchi@sv-gpu01:~/slurm_examples$ cat job_test-slurm-81353_stdout.out
hello world
goodbye world
```

srunで対話環境に入る方法

srun -p sv-dgx \ #パーティション指定
 --nodes=1 \ #ノード数指定
 --gres=gpu:2 \ # GPU数指定
 --mem=10G \ # メモリ指定
 --time 1:00:00 \
 --pty bash # interactiveな環境に入る

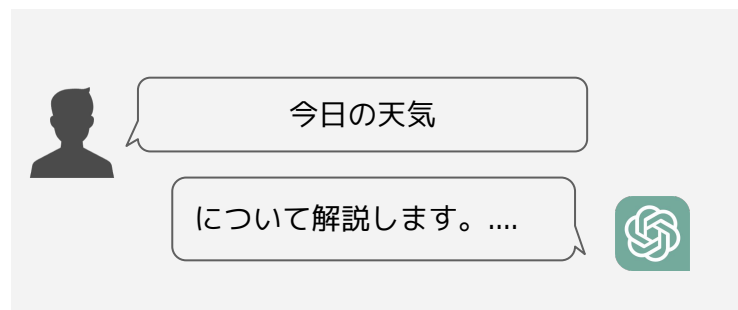
つけっぱなし防止のためTimeはつけておいた方が無難

```
ryota.hirobuchi@sv-gpu01:~$ srun -p sv-dgx --nodes=1 \  
> --ntasks-per-node=1 \  
> --cpus-per-task=3 \  
> --gres=gpu:2 \  
> --mem=10G \  
> --time 1:00:00 \  
> --pty bash  
ryota.hirobuchi@sv-dgx01:~$ nvidia-smi  
Wed Apr 17 19:49:28 2024
```

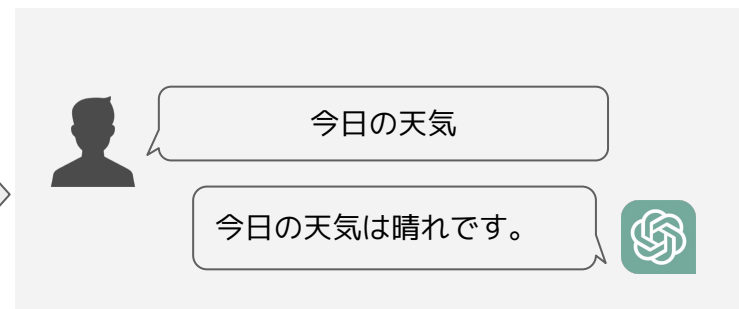
NVIDIA-SMI 450.80.02 Driver Version: 450.80.02 CUDA Version: 12.3									
GPU	Name	Persistence-M	Bus-Id	Disp.A	Volatile Uncorr. ECC				
Fan	Temp	Perf	Pwr:Usage/Cap	Memory-Usage	GPU-Util	Compute M.			
						MIG M.			
0	A100 Graphics D...	On	00000000:07:00.0	Off	0				
N/A	38C	P0	208W / 400W	6850MiB / 81252MiB	51%	Default			

基本的にはdebugなど短時間の作業のみに利用

LLMをFinetuningしてみる



Pretrained Model



Chat Model

Instruct Tuning

InstructTuning: 特定の指示やタスクに対する応答を学習するプロセス

Point

- レスポンス部分のみ学習
- ChatTemplateを設定（必要に応じて特殊トークンも追加）

label: <|im_start|>user\n 今日の天気は<|im_end|>\n<|im_start|>assistant\n 今日の天気は晴れです<|im_end|>



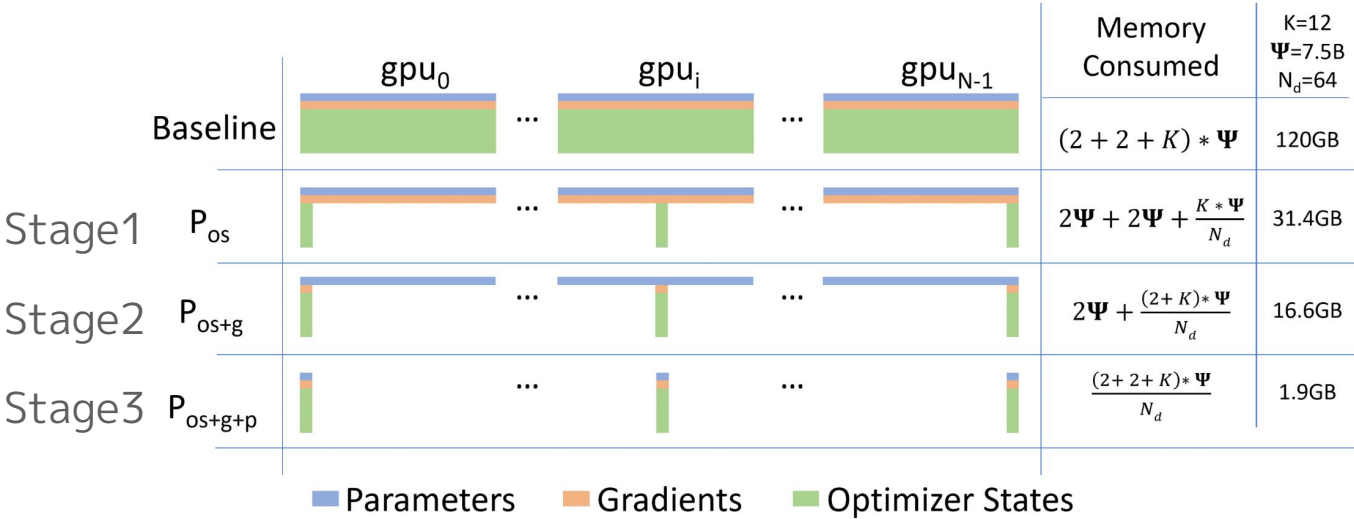
input: <|im_start|>user\n 今日の天気は<|im_end|>\n<|im_start|>assistant\n 今日の天気は晴れです<|im_end|>

chatTemplate



分散並列学習の技術: ZeRO

ZeRO:分散並列学習におけるGPUメモリ使用量を削減する手法
Tensor並列などの手法に比べ基本的にモデルの書き換え不要なのメリット



ZeRO: Memory Optimizations Toward Training Trillion Parameter Models

参考: <https://huggingface.co/docs/transformers/v4.17.0/en/parallelism>

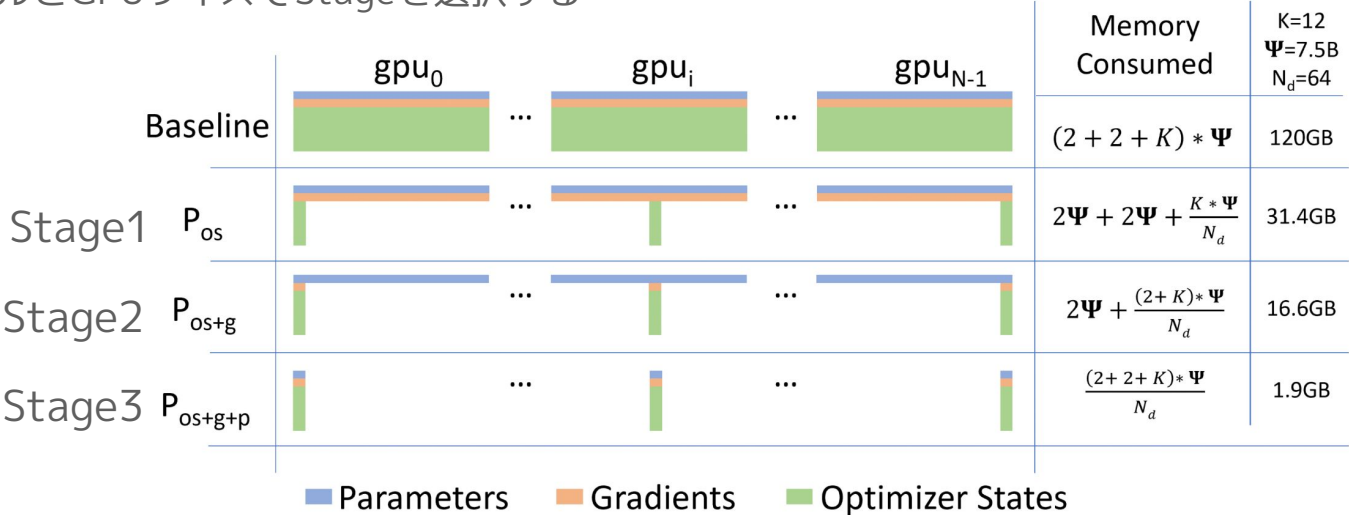
ZeRO Stage2? Stage3?

Stage 2 とStage3ではトレードオフがあり、使い分けが必要

Stage2: 全てのGPUにモデルを載せる必要あり

Stage3: メモリ効率◎ GPU間の通信が必要▲

⇒モデルとGPUサイズでStageを選択する



ZeRO: Memory Optimizations Toward Training Trillion Parameter Models

参考: <https://huggingface.co/docs/transformers/v4.17.0/en/parallelism>

DeepSpeed

大規模モデルの分散・並列学習を実現するライブラリ

- Microsoft製
- **アルゴリズムZeROを利用可能**
- 大幅にコードを変えることなく実行可能

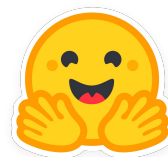


DeepSpeed

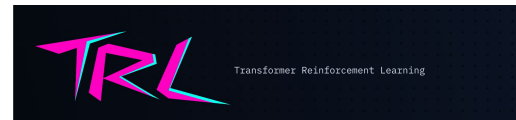
Transformers

GPTをはじめとするTransformerモデルを内包するライブラリ

- Huggingface製
- 世の中のモデルは大抵この形式に変換され公開される
- **DeepSpeedに対応**
 - 色々使用方法の記事があるが[公式ドキュメント](#)をみる



- **TRL**
 - huggingface製
 - 薄めのライブラリ
- **axolotl**
 - yamlで設定したらパッと回せるフレームワーク的な
- **llama factory**
 - axolotlに近い。webUIがある
- **OpenRLHF**
 - transformer.Trainerを使用せず独自実装
- **torch tune**
 - pytorch純正LLM学習ライブラリ



OpenRLHF

LLM-jp SFT (Supervised Fine-Tuning)

<https://github.com/llm-jp/llm-jp-sft>

- llm-jpにより作成されたSFTサンプルコード
- TRLのSFTTrainerをつかったシンプルな実装例
- 利用モデル：llm-jp-13B

 TRLのSFTははまりどころが割とあるので[公式ドキュメント](#)を一読する

BAI Sandboxではホームディレクトリが他のNodeでも共有されている

⇒ □グインnodeで環境構築すれば基本的に問題なく動く

※singularity がインストール済みなので、きっちり管理する場合はコンテナの方がいいかもしれない

例 (python 3.10@llm-jp-sft rootディレクトリ)

```
python -m venv .venv
source .venv/bin/activate
export LD_LIBRARY_PATH=/usr/local/cuda/compat:${LD_LIBRARY_PATH}
```

```
pip install vllm==0.4.0
pip install -r requirements.in
pip uninstall wandb
```

#hfではwandbがインストールされていると勝手に使おうとする。

CUDAバージョン

cuda11.4がデフォルト

CUDA12.3に切り替えは以下のように環境変数を設定

```
export LD_LIBRARY_PATH=/usr/local/cuda/compat:${LD_LIBRARY_PATH}
```

train.py

```
107 def main() -> None:
108     parser = HfArgumentParser((TrainingArguments, SFTTrainingArguments))
109     training_args, sft_training_args = parser.parse_args_into_dataclasses()
110
111     tokenizer_name_or_path: str = (
112         sft_training_args.tokenizer_name_or_path or sft_training_args.model_name_or_path
113     )
114     logger.info(f"Loading tokenizer from {tokenizer_name_or_path}")
115     tokenizer = AutoTokenizer.from_pretrained(
116         tokenizer_name_or_path,
117         use_fast=sft_training_args.use_fast,
118         additional_special_tokens=sft_training_args.additional_special_tokens,
119         trust_remote_code=True,
120     )
121
122     logger.info("Loading data")
123
124     train_dataset = load_datasets(sft_training_args.data_files)
125     if sft_training_args.eval_data_files:
126         eval_dataset = load_datasets(sft_training_args.eval_data_files)
127         training_args.do_eval = True
128     else:
129         eval_dataset = None
```

Tokenizer ロード

Dataset ロード

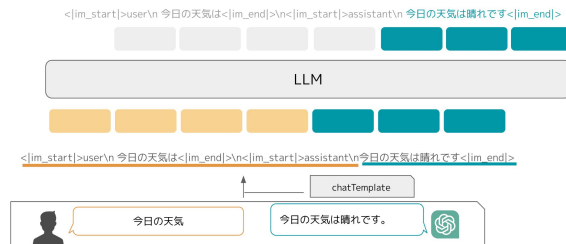
sample data

```
{
  "text": "以下は、タスクを説明する指示です。要求を適切に満たす応答を書きなさい。\\n\\n### 指示:\\n日本で一番高い山は？\\n\\n### 応答:\\n富士山"
},
{
  "text": "以下は、タスクを説明する指示です。要求を適切に満たす応答を書きなさい。\\n\\n### 指示:\\n日本の初代内閣総理大臣は？\\n\\n### 応答:\\n伊藤博文"
},
{
  "text": "以下は、タスクを説明する指示です。要求を適切に満たす応答を書きなさい。\\n\\n### 指示:\\n日本の首都は？\\n\\n### 応答:\\n東京"
},
{
  "text": "以下は、タスクを説明する指示です。要求を適切に満たす応答を書きなさい。\\n\\n### 指示:\\n日本で一番北にある都道府県は？\\n\\n### 応答:\\n北海道"
}
```

DataCollatorForCompletionOnlyLM

SFTにおいてprompt部分の勾配を含まないようにmaskをつけるutilクラス
instruction_template, response templateで指定したtokenの間全てをmask

```
131 logger.info("Formatting prompts")
132 instruction_ids = tokenizer.encode("\n\n### 指示:\n", add_special_tokens=False)[1:]
133 response_ids = tokenizer.encode("\n\n### 応答:\n", add_special_tokens=False)[1:]
134 collator = DataCollatorForCompletionOnlyLM(
135     instruction_template=instruction_ids,
136     response_template=response_ids,
137     tokenizer=tokenizer,
138 )
```



⚠ 文字列について文章中と単体と異なるtokenizerになる場合がある

ex: llama2のtokenizer

"### Assistant:"

文中:[2277, 29937, 4007, 22137, 29901]

単体:[835, 4007, 22137, 29901]

```
145 model = AutoModelForCausalLM.from_pretrained(  
146     sft_training_args.model_name_or_path,  
147     trust_remote_code=True,  
148     **kwargs,  
149 )
```

Modelのload

HFのモデルパスかモデルディレクトリへのパスを指定

```
172 trainer = SFTTrainer(  
173     model,  
174     args=training_args,  
175     tokenizer=tokenizer,  
176     train_dataset=train_dataset,  
177     eval_dataset=eval_dataset,  
178     dataset_text_field="text",  
179     data_collator=collator,  
180     peft_config=peft_config,  
181     max_seq_length=sft_training_args.max_seq_length,  
182 )  
183  
184 logger.info("Training")  
185 trainer.train()  
186  
187 logger.info("Saving model")  
188 trainer.save_model()
```

SFTTrainer

Finetuningを実行するクラス
transformers.Trainerを拡張
train()で学習

- args: transformers.Trainerのargsを渡せる
 - [TrainingArgs詳細リンク](#)
- data_text_field: 入力データの列を指定
 - データセット形式によっては不要
 - 今回だと"text"

```
{ "text": "以下は、タスクを  
{ "text": "以下は、タスクを  
{ "text": "以下は、タスクを  
{ "text": "以下は、タスクを  
{ "text": "以下は、タスクを
```

run_train.slurm

```
#!/bin/bash
#SBATCH --job-name=test-sft      # name
#SBATCH -p sv-dgx
#SBATCH --nodes=1               # nodes
#SBATCH --ntasks-per-node=1     # crucial - only 1 task per dist per node!
#SBATCH --cpus-per-task=30      # number of cores per tasks
#SBATCH --gres=gpu:8             # number of gpu
#SBATCH --mem=300G
#SBATCH --time 24:00:00         # maximum execution time (HH:MM:SS)
#SBATCH --output=logs/%x-%j_log.out # output file name
#SBATCH --error=logs/%x-%j_log.out # output file name

export LD_LIBRARY_PATH=/usr/local/cuda/compat:${LD_LIBRARY_PATH} # set cuda 12.3
export BASE_MODEL="llm-jp/llm-jp-13b-v1.0"
echo "train ${BASE_MODEL}"

export GPUS_PER_NODE=8
export MASTER_ADDR=$(scontrol show hostnames $SLURM_JOB_NODELIST | head -n 1)
export MASTER_PORT=9901

srun --jobid $SLURM_JOBID bash -c 'python -m torch.distributed.run \
--nproc_per_node $GPUS_PER_NODE --nnodes $SLURM_NNODES --node_rank $SLURM_PROCID \
--master_addr $MASTER_ADDR --master_port $MASTER_PORT \
train.py \
--deepspeed ds_config/ds_config_stage2.json \
--data_files data/example.jsonl \
--model_name_or_path $BASE_MODEL \
--output_dir results/$BASE_MODEL \
--num_train_epochs 10 \
--per_device_train_batch_size 1 \
--gradient_accumulation_steps 32 \
--learning_rate 1e-5 \
--warmup_ratio 0.1 \
--lr_scheduler_type cosine \
--gradient_checkpointing \
--logging_steps 5 \
--save_strategy epoch \
--save_only_model'
```

Jobの設定

train.pyの実行

train.pyのオプション

run_train.slurm

```
#!/bin/bash
#SBATCH --job-name=test-sft      # name
#SBATCH -p sv-dgx
#SBATCH --nodes=1                # nodes
#SBATCH --ntasks-per-node=1      # crucial - only 1 task per dist per node!
#SBATCH --cpus-per-task=30       # number of cores per tasks
#SBATCH --gres=gpu:8             # number of gpu
#SBATCH --mem=300G
#SBATCH --time 24:00:00          # maximum execution time (HH:MM:SS)
#SBATCH --output=logs/%x-%j_log.out  # output file name
#SBATCH --error=logs/%x-%j_log.out  # output file name
```

slurmの設定

- リソース
- logの出力先
- timeout時間
- etc

node1台
GPU8台

run_train.slurm

```
...  
export GPUS_PER_NODE=8  
export MASTER_ADDR=$(scontrol show hostnames  
$SLURM_JOB_NODELIST | head -n 1)  
export MASTER_PORT=9901  
  
srun --jobid $SLURM_JOBID bash -c 'python -m torch.distributed.run \  
  --nproc_per_node $GPUS_PER_NODE \  
  --nnodes $SLURM_NNODES \  
  --node_rank $SLURM_PROCID \  
  --master_addr $MASTER_ADDR \  
  --master_port $MASTER_PORT \  
train.py \  
  --deepspeed ds_config/ds_config_stage2.json \  
...'
```

train.pyの実行

- torch.distributed.runで起動
 - Trainer/deepspeed/slurmの場合
- [Huggingfaceのドキュメント](#)を参照
- **deepspeed config**を指定

DeepSpeed config

DeepSpeedの各種設定 (基本ドキュメントからコピペ)
Stage2, Stage3の変更もここで行う

ds_config_stage2.json

```
{
  "train_batch_size": "auto",
  "train_micro_batch_size_per_gpu": "auto",
  "gradient_accumulation_steps": "auto",
  "gradient_clipping": "auto",
  "zero_allow_untested_optimizer": true,
  "fp16": {
    "enabled": "auto",
    "loss_scale": 0,
    "initial_scale_power": 16,
    "loss_scale_window": 1000,
    "hysteresis": 2,
    "min_loss_scale": 1
  },
  "zero_optimization": {
    "stage": 2,
    "allgather_partitions": true,
    "allgather_bucket_size": 5e8,
    "reduce_scatter": true,
    "reduce_bucket_size": 5e8,
    "overlap_comm": false,
    "contiguous_gradients": true
  }
}
```

ds_config_stage3.json

```
...
"zero_optimization": {
  "stage": 3,
  "reduce_bucket_size": "auto",
  "overlap_comm": true,
  "contiguous_gradients": true,
  "offload_optimizer": {
    "device": "none"
  },
  "offload_param": {
    "device": "none"
  },
  "sub_group_size": 1e9,
  "stage3_prefetch_bucket_size": "auto",
  "stage3_param_persistence_threshold": "auto",
  "stage3_max_live_parameters": 1e8,
  "stage3_max_reuse_distance": 1e8,
  "stage3_gather_16bit_weights_on_model_save": true
}
```

```
. . .  
train.py \  
--deepspeed ds_config/ds_config_stage2.json \  
--data_files data/example.jsonl \  
--model_name_or_path $BASE_MODEL \  
--output_dir results/$BASE_MODEL \  
--num_train_epochs 3 \  
--per_device_train_batch_size 1 \  
--gradient_accumulation_steps 32 \  
--learning_rate 1e-5 \  
--warmup_ratio 0.1 \  
--lr_scheduler_type cosine \  
--gradient_checkpointing \  
--logging_steps 5 \  
--save_strategy epoch \  
--save_only_model '
```

BatchSize

= per_device_train_batch_size
✕ gradient_accumulation_steps
✕ ノードあたりのGPU
✕ ノード数

save_only_model

checkpointでoptimizer状態などを保存しない
学習再開をしないならつけておいた方がよい

jobを投入し学習状況をlogで確認する

```
(.venv) ryota.hirobuchi@sv-gpu01:~/llm_consortium/llm-jp-sft$ sbatch scripts/run_train.slurm  
Submitted batch job 81403
```

```
▼ logs  
≡ test-sft-81393_log.out  
≡ test-sft-81395_log.out  
≡ test-sft-81396_log.out  
≡ test-sft-81398_log.out  
≡ test-sft-81402_log.out  
≡ test-sft-81403_log.out
```

Tips:ログは最新をtailで表示できるスクリプトなど作っておくと楽

```
ls -t1 logs/*.out 2>/dev/null | head -n 1 | xargs tail -f -n 1000
```

```
{'loss': 1.7723, 'grad_norm': 0.9445900321006775, 'learning_rate': 4.54545454545455e-06, 'epoch': 0.54}  
{'loss': 1.5806, 'grad_norm': 0.7404970526695251, 'learning_rate': 9.090909090909091e-06, 'epoch': 0.68}  
{'loss': 1.551, 'grad_norm': 0.6435043215751648, 'learning_rate': 9.958100506132127e-06, 'epoch': 0.81}  
29%|{'loss': 1.5401, 'grad_norm': 0.5988965630531311, 'learning_rate': 9.789086620939936e-06, 'epoch': 0.95}  
33%|  | 36/108 [21:02<42:25, 35.35s/it]/home/ryota.hirobuchi/llm_consortium/llm-jp-sft/.venv/lib/pyth
```

vLLM

PagedAttentionなどを用いて効率的にLLMの推論を行うライブラリ

```
from vllm import LLM, SamplingParams

sampling_params = SamplingParams(temperature=0.8, top_p=0.95, max_tokens=200)
model_path = "./results/llm-jp/llm-jp-13b-v1.0"
llm = LLM(
    model=model_path, tensor_parallel_size=1
) # 1 GPUに乗らない場合はtensor_parallel_sizeを増やす
tokenizer = llm.get_tokenizer()

while True:
    input_text = input("user:\n")
    prompt = f"以下は、タスクを説明する指示です。要求を適切に満たす応答を書きなさい。

    ### 指示:
    {input_text}

    ### 応答:
    ""

    prompt_ids = [
        tokenizer.encode(prompt)[:1]
    ] # llm-jp tokenizerはeosトークンを自動的に追加するため取り除く

    outputs = llm.generate(None, sampling_params, prompt_ids, use_tqdm=False)
    print("output:\n", outputs[0].outputs[0].text)
```

ChatTemplate用のトークンを追加する例

```
im_start_token = "<|im_start|>"
im_end_token = "<|im_end|>"
pad_token="</s>"
tokenizer.pad_token = pad_token
tokenizer.eos_token = im_end_token
tokenizer.add_special_tokens(
    {"additional_special_tokens": [im_start_token, im_end_token]}
)
# set chat format for tokenizer
model.resize_token_embeddings(
    len(tokenizer),
    pad_to_multiple_of=resize_to_multiple_of
    if resize_to_multiple_of is not None
    else None,
)
# Make sure to update the generation config to use the new eos & bos token
if getattr(model, "generation_config", None) is not None:
    model.generation_config.eos_token_id = tokenizer.eos_token_id
```

ChatTemplate例(ChatML)

```
<|im_start|>user
Hi there!<|im_end|>
<|im_start|>assistant
Nice to meet you!<|im_end|>
<|im_start|>user
Can I ask a question?<|im_end|>
```

eosとpad

TRLではeos!=padである必要がある



eos==padのモデルは結構ある

- **eos**: 文末トークン。生成終了のトリガー
- **pad**: attentionで無視される

特殊トークンの追加

ChatTemplate用のトークンを追加する例

```
im_start_token = "<|im_start|>"
im_end_token = "<|im_end|>"
pad_token="</s>"
tokenizer.pad_token = pad_token
tokenizer.eos_token = im_end_token
tokenizer.add_special_tokens(
    {"additional_special_tokens": [im_start_token, im_end_token]}
)
# set chat format for tokenizer
model.resize_token_embeddings(
    len(tokenizer),
    pad_to_multiple_of=resize_to_multiple_of
    if resize_to_multiple_of is not None
    else None,
)
# Make sure to update the generation config to use the new eos & bos token
if getattr(model, "generation_config", None) is not None:
    model.generation_config.eos_token_id = tokenizer.eos_token_id
```

ChatTemplate例(ChatML)

```
<|im_start|>user
Hi there!<|im_end|>
<|im_start|>assistant
Nice to meet you!<|im_end|>
<|im_start|>user
Can I ask a question?<|im_end|>
```

eos

TRL



eo

•

•

```
llama3
<|begin_of_text|><|start_header_id|>system<|end_header_id|>
{{ system_prompt }}<|eot_id|><|start_header_id|>user<|end_header_id|>
{{ user_msg_1 }}<|eot_id|><|start_header_id|>assistant<|end_header_id|>
{{ model_answer_1 }}<|eot_id|>
```

SFTモデルとChatTemplateはセットで管理することが望ましい

⇒ tokenizer.chat_template に設定して保存するのが (tokenizer_config.json に保存される)

```
messages = [  
    {"role": "user", "content": "Hi there!"},  
    {"role": "assistant", "content": "Nice to meet you!"},  
    {"role": "user", "content": "Can I ask a question?"}  
]
```

ChatTemplate

```
{% for message in messages %}  
{{ '<|im_start|>' + message['role'] + '\n' + message['content'] + '<|im_end|>' + '\n' }}  
{% endfor %}  
{% if add_generation_prompt %}  
{{ '<|im_start|>assistant\n' }}  
{% endif %}
```

```
tokenizer.apply_chat_template(messages, tokenize=False, add_generation_prompt=True)  
""<|im_start|>user  
Hi there!<|im_end|>  
<|im_start|>assistant  
Nice to meet you!<|im_end|>  
<|im_start|>user  
Can I ask a question?<|im_end|>  
<|im_start|>assistant  
""
```

⚠ SFTでうまくいかない原因は大抵特殊トークン(tokenizer)とTemplate

[Templates for Chat Models](#)

EOF