TRL

Transformer Reinforcement Learning

ref : ***https://github.com/huggingface/trl***

A comprehensive library to post-train foundation models

license Apache-2.0 website online release v0.12.2

Overview

TRL is a cutting-edge library designed for post-training foundation models using advanced techniques like Supervised Fine-Tuning (SFT), Proximal Policy Optimization (PPO), and Direct Preference Optimization (DPO). Built on top of the 🤗 Transformers ecosystem, TRL supports a variety of model architectures and modalities, and can be scaled-up across various hardware setups.

Highlights

- **Efficient and scalable**:
  - Leverages 🤗 Accelerate to scale from single GPU to multi-node clusters using methods like DDP and DeepSpeed.
  - Full integration with `PEFT` enables training on large models with modest hardware via quantization and LoRA/QLoRA.
  - Integrates Unsloth for accelerating training using optimized kernels.
- **Command Line Interface (CLI)**: A simple interface lets you fine-tune and interact with models without needing to write code.
- **Trainers**: Various fine-tuning methods are easily accessible via trainers like `SFTTrainer`, `DPOTrainer`, `RewardTrainer`, `ORPOTrainer` and more.
- **AutoModels**: Use pre-defined model classes like `AutoModelForCausalLMWithValueHead` to simplify reinforcement learning (RL) with LLMs.

Installation

Python Package

Install the library using `pip`:

```
pip install trl
```

From source

If you want to use the latest features before an official release, you can install TRL from source:

```
pip install git+https://github.com/huggingface/trl.git
```

Repository

If you want to use the examples you can clone the repository with the following command:

```
git clone https://github.com/huggingface/trl.git
```

Command Line Interface (CLI)

You can use the TRL Command Line Interface (CLI) to quickly get started with Supervised Fine-tuning (SFT) and Direct Preference Optimization (DPO), or vibe check your model with the chat CLI:

SFT:

```
trl sft --model_name_or_path Qwen/Qwen2.5-0.5B \
    --dataset_name trl-lib/Capybara \
    --output_dir Qwen2.5-0.5B-SFT
```

DPO:

```
trl dpo --model_name_or_path Qwen/Qwen2.5-0.5B-Instruct \
    --dataset_name argilla/Capybara-Preferences \
    --output_dir Qwen2.5-0.5B-DPO
```

Chat:

```
trl chat --model_name_or_path Qwen/Qwen2.5-0.5B-Instruct
```

Read more about CLI in the relevant documentation section or use `--help` for more details.

How to use

For more flexibility and control over training, TRL provides dedicated trainer classes to post-train language models or PEFT adapters on a custom dataset. Each trainer in TRL is a light wrapper around the 🤗 Transformers trainer and natively supports distributed training methods like DDP, DeepSpeed ZeRO, and FSDP.

`SFTTrainer`

Here is a basic example of how to use the `SFTTrainer`:

```
from trl import SFTConfig, SFTTrainer
from datasets import load_dataset

dataset = load_dataset("trl-lib/Capybara", split="train")

training_args = SFTConfig(output_dir="Qwen/Qwen2.5-0.5B-SFT")
trainer = SFTTrainer(
    args=training_args,
    model="Qwen/Qwen2.5-0.5B",
    train_dataset=dataset,
)
trainer.train()
```

`RewardTrainer`

Here is a basic example of how to use the `RewardTrainer`:

```
from trl import RewardConfig, RewardTrainer
from datasets import load_dataset
from transformers import AutoModelForSequenceClassification, AutoTokenizer

tokenizer = AutoTokenizer.from_pretrained("Qwen/Qwen2.5-0.5B-Instruct")
model = AutoModelForSequenceClassification.from_pretrained(
    "Qwen/Qwen2.5-0.5B-Instruct", num_labels=1
)
model.config.pad_token_id = tokenizer.pad_token_id

dataset = load_dataset("trl-lib/ultrafeedback_binarized", split="train")

training_args = RewardConfig(output_dir="Qwen2.5-0.5B-Reward", per_device_train_batch_size=2)
trainer = RewardTrainer(
    args=training_args,
    model=model,
    processing_class=tokenizer,
    train_dataset=dataset,
```

```
)
trainer.train()
```

RLOOTrainer

`RLOOTrainer` implements a REINFORCE-style optimization for RLHF that is more performant and memory-efficient than PPO. Here is a basic example of how to use the `RLOOTrainer`:

```
from trl import RLOOConfig, RLOOTrainer, apply_chat_template
from datasets import load_dataset
from transformers import (
    AutoModelForCausalLM,
    AutoModelForSequenceClassification,
    AutoTokenizer,
)

tokenizer = AutoTokenizer.from_pretrained("Qwen/Qwen2.5-0.5B-Instruct")
reward_model = AutoModelForSequenceClassification.from_pretrained(
    "Qwen/Qwen2.5-0.5B-Instruct", num_labels=1
)
ref_policy = AutoModelForCausalLM.from_pretrained("Qwen/Qwen2.5-0.5B-Instruct")
policy = AutoModelForCausalLM.from_pretrained("Qwen/Qwen2.5-0.5B-Instruct")

dataset = load_dataset("trl-lib/ultrafeedback-prompt")
dataset = dataset.map(apply_chat_template, fn_kwargs={"tokenizer": tokenizer})
dataset = dataset.map(lambda x: tokenizer(x["prompt"]), remove_columns="prompt")

training_args = RLOOConfig(output_dir="Qwen2.5-0.5B-RL")
trainer = RLOOTrainer(
    config=training_args,
    processing_class=tokenizer,
    policy=policy,
    ref_policy=ref_policy,
    reward_model=reward_model,
    train_dataset=dataset["train"],
    eval_dataset=dataset["test"],
)
trainer.train()
```

DPOTrainer

`DPOTrainer` implements the popular Direct Preference Optimization (DPO) algorithm that was used to post-train Llama 3 and many other models. Here is a basic example of how to use the `DPOTrainer`:

```
from datasets import load_dataset
from transformers import AutoModelForCausalLM, AutoTokenizer
from trl import DPOConfig, DPOTrainer

model = AutoModelForCausalLM.from_pretrained("Qwen/Qwen2.5-0.5B-Instruct")
tokenizer = AutoTokenizer.from_pretrained("Qwen/Qwen2.5-0.5B-Instruct")
dataset = load_dataset("trl-lib/ultrafeedback_binarized", split="train")
training_args = DPOConfig(output_dir="Qwen2.5-0.5B-DPO")
trainer = DPOTrainer(model=model, args=training_args, train_dataset=dataset, processing_class=tokenizer)
trainer.train()
```

Development

If you want to contribute to `trl` or customize it to your needs make sure to read the contribution guide and make sure you make a dev install:

```
git clone https://github.com/huggingface/trl.git
cd trl/
pip install -e .[dev]
```

Citation

```
@misc{vonwerra2022trl,
  author = {Leandro von Werra and Younes Belkada and Lewis Tunstall and Edward Beeching and Tristan Thrush and Nathan Lambert ar
  title = {TRL: Transformer Reinforcement Learning},
```

RLHF pipeline for the creation of StackLLaMa: a Stack exchange llama-7b model.

There were three main steps to the training process:

1. Supervised fine-tuning of the base llama-7b model to create llama-7b-se:
   - `torchrun --nnodes 1 --nproc_per_node 8 examples/research_projects/stack_llama/scripts/supervised_finetuning.py --model_path=<LLAMA_MODEL_PATH> --streaming --learning_rate 1e-5 --max_steps 5000 --output_dir ./llama-se`
2. Reward modeling using dialog pairs from the SE dataset using the llama-7b-se to create llama-7b-se-rm:
   - `torchrun --nnodes 1 --nproc_per_node 8 examples/research_projects/stack_llama/scripts/reward_modeling.py --model_name=<LLAMA_SE_MODEL>`
3. RL fine-tuning of llama-7b-se with the llama-7b-se-rm reward model:
   - `accelerate launch --multi_gpu --num_machines 1 --num_processes 8 examples/research_projects/stack_llama/scripts/rl_training.py --log_with=wandb --model_name=<LLAMA_SE_MODEL> --reward_model_name=<LLAMA_SE_RM_MODEL> --adafactor=False --tokenizer_name=<LLAMA_TOKENIZER> --save_freq=100 --output_max_length=128 --batch_size=8 --gradient_accumulation_steps=8 --batched_gen=True --ppo_epochs=4 --seed=0 --learning_rate=1.4e-5 --early_stopping=True --output_dir=llama-se-rl-finetune-128-8-8-1.4e-5_adam`

LoRA layers were using at all stages to reduce memory requirements. At each stage the peft adapter layers were merged with the base model, using:

```
python examples/research_projects/stack_llama/scripts/merge_peft_adapter.py --adapter_mo
```

Note that this script requires `peft>=0.3.0`.

For access to the base llama-7b model, please see Meta's release and request form.

ref :
https://github.com/huggingface/trl/blob/main/examples/research_projects/stack_llama/scripts/README.md

DPO pipeline for the creation of StackLlaMa 2: a Stack exchange llama-v2-7b model

Prerequisites

Install all the dependencies in the `requirements.txt` :

```
$ pip install -U -r requirements.txt
```

Since we will use `accelerate` for training, make sure to run:

```
$ accelerate config
```

Training

There were two main steps to the DPO training process:

1. Supervised fine-tuning of the base llama-v2-7b model to create llama-v2-7b-se:
   ```
   accelerate launch examples/research_projects/stack_llama_2/scripts/sft_llama2.py \
       --output_dir="./sft" \
       --max_steps=500 \
       --logging_steps=10 \
       --save_steps=10 \
       --per_device_train_batch_size=4 \
       --per_device_eval_batch_size=1 \
       --gradient_accumulation_steps=2 \
       --gradient_checkpointing=False \
       --group_by_length=False \
       --learning_rate=1e-4 \
       --lr_scheduler_type="cosine" \
       --warmup_steps=100 \
       --weight_decay=0.05 \
       --optim="paged_adamw_32bit" \
       --bf16=True \
       --remove_unused_columns=False \
       --run_name="sft_llama2" \
       --report_to="wandb"
   ```
2. Run the DPO trainer using the model saved by the previous step:
   ```
   accelerate launch examples/research_projects/stack_llama_2/scripts/dpo_llama2.py \
       --model_name_or_path="sft/final_checkpoint" \
       --output_dir="dpo"
   ```

Merging the adaptors

To merge the adaptors into the base model we can use the `merge_peft_adapter.py` helper script that comes with TRL:

```
python examples/research_projects/stack_llama/scripts/merge_peft_adapter.py --base_model_
```

which will also push the model to your HuggingFace hub account.

Running the model

We can load the DPO-trained LoRA adaptors which were saved by the DPO training step and load them via:

```python
from peft import AutoPeftModelForCausalLM


model = AutoPeftModelForCausalLM.from_pretrained(
    "dpo/final_checkpoint",
    low_cpu_mem_usage=True,
    torch_dtype=torch.float16,
    load_in_4bit=True,
)

model.generate(...)
```

ref:
https://github.com/huggingface/trl/blob/main/examples/research_projects/stack_llama_2/scripts/README.md