```python
# -*- coding: utf-8 -*-
"""Fine-tuning and Inference with Falcon-7B using TRL and QLoRa
"""

# !pip install -q -U bitsandbytes
# !pip install -q -U git+https://github.com/huggingface/transformers.git
# !pip install -q -U git+https://github.com/huggingface/peft.git
# !pip install -q -U git+https://github.com/huggingface/accelerate.git
# !pip install -q -U datasets
# !pip install -q -U trl
# !pip install -q -U einops

"""Import only what we need:"""

import torch, einops
from datasets import load_dataset
from peft import LoraConfig
from transformers import (
    AutoModelForCausalLM,
    AutoTokenizer,
    BitsAndBytesConfig,
    AutoTokenizer,
    TrainingArguments
)
from peft.tuners.lora import LoraLayer

from trl import SFTTrainer

"""Create function that will:


1.   Create and prepare the bitsandbytes configuration for QLoRa's quantization
2.   Download, load, and quantize on-the-fly Falcon-7b
3.   Create and prepare the LoRa configuration
4.   Load and configuration Falcon-7B's tokenizer


"""

def create_and_prepare_model():
    compute_dtype = getattr(torch, "float16")

    bnb_config = BitsAndBytesConfig(
        load_in_4bit=True,
        bnb_4bit_quant_type="nf4",
        bnb_4bit_compute_dtype=compute_dtype,
        bnb_4bit_use_double_quant=True,
    )

    model = AutoModelForCausalLM.from_pretrained(
        "tiiuae/falcon-7b", quantization_config=bnb_config, device_map="auto", trust_remote_code=True
    )

    peft_config = LoraConfig(
        lora_alpha=16,
        lora_dropout=0.1,
        r=64,
        bias="none",
        task_type="CAUSAL_LM",
        target_modules=[
            "query_key_value"
        ],
    )

    tokenizer = AutoTokenizer.from_pretrained("tiiuae/falcon-7b", trust_remote_code=True)
    tokenizer.pad_token = tokenizer.eos_token

    return model, peft_config, tokenizer

"""Setting up the training parameters:
*Note: I put 1 training epoch so that it can be trained in Colab within a few hours, but I recommend to set at least 5, ideally 10 epochs.*
"""

training_arguments = TrainingArguments(
    output_dir="./results",
    per_device_train_batch_size=1, # increase this value if you have more VRAM
    gradient_accumulation_steps=4,
    optim="paged_adamw_32bit", # This parameter activate QLoRa's pagination
    save_steps=100,
    logging_steps=100,
    learning_rate=2e-4,
    fp16=True,
    max_grad_norm=0.3,
    num_train_epochs=1,
    warmup_ratio=0.03,
    lr_scheduler_type="constant"
)

import json
with open('/content/final_input_output_pairs.json') as f:
  data = json.load(f)

train_data = []
```

```
for item in data["train"]["coherence"]:
    combined_text = f"{item['input']}\n\n{item['output']}"
    train_data.append({"text": combined_text})

# Create the Dataset object
train_dataset = Dataset.from_list(train_data)



"""Create and prepare the model and dataset:"""

model, peft_config, tokenizer = create_and_prepare_model()
model.config.use_cache = False # Gradient checkpointing is used by default but not compatible with caching
dataset = load_dataset("timdettmers/openassistant-guanaco", split="train")

print(dataset)

"""Train the model on the guanaco dataset:
*It should complete within 5 hours.*
"""

trainer = SFTTrainer(
    model=model,
    train_dataset=dataset,
    peft_config=peft_config,
    dataset_text_field="text",
    max_seq_length=512,
    tokenizer=tokenizer,
    args=training_arguments,
    packing=True,
)

trainer.train()
```