guten_tag

June 1, 2025

1 Fine-Tuning GPT-2 with LoRA for Poetic Sci-Fi/Fantasy Story Generation

This notebook guides you through fine-tuning GPT-2 using Low-Rank Adaptation (LoRA) to generate poetic sci-fi/fantasy short stories. The process includes: - Dataset preparation from Project Gutenberg - Model fine-tuning with LoRA - Story generation based on user prompts

Note: This notebook is designed for execution in Google Colab.

1.1 Setup and Installation

[10]:
```
# Install necessary packages
!pip install -q transformers datasets peft trl bitsandbytes accelerate
!pip install -q beautifulsoup4 requests gutenbergpy
```

1.2 Dataset Preparation

[11]:
```
import os
import re
import json
import requests
from bs4 import BeautifulSoup
from pathlib import Path
from tqdm import tqdm
from gutenbergpy.textget import get_text_by_id
from gutenbergpy.gutenbergcache import GutenbergCache


# Define genres and corresponding Project Gutenberg bookshelf URLs
bookshelves = {
    'science_fiction': 'https://www.gutenberg.org/ebooks/bookshelf/41',
    'poetry': 'https://www.gutenberg.org/ebooks/bookshelf/60',
}

# Function to extract book IDs from a Project Gutenberg bookshelf
def get_book_ids_from_bookshelf(url, limit=10):
    response = requests.get(url)
    soup = BeautifulSoup(response.text, 'html.parser')
    book_links = soup.select('li.booklink a.link')
    book_ids = []

    for link in book_links:
        href = link.get('href')
        if href.startswith('/ebooks/'):
            book_id = href.split('/')[-1]
            if book_id.isdigit():
                book_ids.append(int(book_id))
                if len(book_ids) == limit:
                    break
    return book_ids

# Function to download book texts given their IDs
def download_books(book_ids, output_folder):
    from gutenbergpy.textget import get_text_by_id
    from gutenbergpy.gutenbergcache import GutenbergCache

    os.makedirs(output_folder, exist_ok=True)
    print("Loading Gutenberg metadata cache...")
    cache = GutenbergCache.get_cache()
    for book_id in book_ids:
        output_path = os.path.join(output_folder, f"{book_id}.txt")
        if os.path.exists(output_path) and os.path.getsize(output_path) > 0:
            print(f"Book {book_id} already exists at {output_path}, skipping␣
 ↪download.")
            continue
        print(f"Downloading book ID {book_id}...")
        try:
            text_bytes = get_text_by_id(book_id)
            text_str = text_bytes.decode('utf-8', errors='ignore')
            with open(output_path, 'w', encoding='utf-8') as f:
                f.write(text_str)
            print(f"Saved book {book_id} to {output_path}")
        except Exception as e:
            print(f"Error downloading book {book_id}: {e}")

# Utility function to download books by genre
def download_books_to_dataset(bookshelf_url, genre, limit=10,␣
 ↪base_folder="gutenberg_dataset"):
    output_folder = os.path.join(base_folder, genre)
    book_ids = get_book_ids_from_bookshelf(bookshelf_url, limit=limit)
    download_books(book_ids, output_folder=output_folder)

# Download books for each genre
for genre, url in bookshelves.items():
    download_books_to_dataset(url, genre=genre, limit=10)
```

```
Loading Gutenberg metadata cache…
Book 145 already exists at gutenberg_dataset/science_fiction/145.txt, skipping
download.
Book 2160 already exists at gutenberg_dataset/science_fiction/2160.txt, skipping
download.
Book 1259 already exists at gutenberg_dataset/science_fiction/1259.txt, skipping
download.
Book 4085 already exists at gutenberg_dataset/science_fiction/4085.txt, skipping
download.
Book 98 already exists at gutenberg_dataset/science_fiction/98.txt, skipping
download.
Book 2600 already exists at gutenberg_dataset/science_fiction/2600.txt, skipping
download.
Book 135 already exists at gutenberg_dataset/science_fiction/135.txt, skipping
download.
Book 120 already exists at gutenberg_dataset/science_fiction/120.txt, skipping
download.
Book 1837 already exists at gutenberg_dataset/science_fiction/1837.txt, skipping
download.
Book 73 already exists at gutenberg_dataset/science_fiction/73.txt, skipping
download.
Loading Gutenberg metadata cache…
Book 16328 already exists at gutenberg_dataset/poetry/16328.txt, skipping
download.
Book 1322 already exists at gutenberg_dataset/poetry/1322.txt, skipping
download.
Book 228 already exists at gutenberg_dataset/poetry/228.txt, skipping download.
Book 2490 already exists at gutenberg_dataset/poetry/2490.txt, skipping
download.
Book 14568 already exists at gutenberg_dataset/poetry/14568.txt, skipping
download.
Book 9622 already exists at gutenberg_dataset/poetry/9622.txt, skipping
download.
Book 3333 already exists at gutenberg_dataset/poetry/3333.txt, skipping
download.
Book 1321 already exists at gutenberg_dataset/poetry/1321.txt, skipping
download.
Book 20 already exists at gutenberg_dataset/poetry/20.txt, skipping download.
Book 847 already exists at gutenberg_dataset/poetry/847.txt, skipping download.
```

1.3 Data Cleaning and JSONL Conversion

[12]:
```python
# Define input directories and output file
INPUT_DIRS = {
    "science_fiction": Path("gutenberg_dataset/science_fiction"),
    "poetry": Path("gutenberg_dataset/poetry"),
}
```

```
OUTPUT_FILE = Path("gutenberg_dataset.jsonl")

# Regex patterns to remove Project Gutenberg headers/footers
HEADER_PATTERN = re.compile(
    r"\*{3}\s*START OF THIS PROJECT GUTENBERG EBOOK.*?\*{3}", re.IGNORECASE |␣
 ↪re.DOTALL
)
FOOTER_PATTERN = re.compile(
    r"\*{3}\s*END OF THIS PROJECT GUTENBERG EBOOK.*", re.IGNORECASE | re.DOTALL
)

# Function to clean Gutenberg text
def clean_gutenberg_text(text: str) -> str:
    text = HEADER_PATTERN.sub("", text)
    text = FOOTER_PATTERN.sub("", text)
    text = text.strip()
    return text

# Function to process and write data to JSONL
def process_and_write_jsonl(input_dirs: dict, output_path: Path):
    if output_path.exists() and output_path.stat().st_size > 0:
        print(f"{output_path} already exists and is non-empty, skipping␣
 ↪processing.")
        return
    with output_path.open("w", encoding="utf-8") as out_file:
        for source_label, folder in input_dirs.items():
            txt_files = list(folder.rglob("*.txt"))
            for txt_path in tqdm(txt_files, desc=f"Processing {source_label}"):
                try:
                    raw = txt_path.read_text(encoding="utf-8", errors="ignore")
                    clean = clean_gutenberg_text(raw)
                    if not clean:
                        continue
                    record = {
                        "source": source_label,
                        "filename": txt_path.name,
                        "text": clean,
                    }
                    out_file.write(json.dumps(record, ensure_ascii=False) +␣
 ↪"\n")
                except Exception as e:
                    print(f"Error processing {txt_path}: {e}")

# Process and write the dataset
os.makedirs(OUTPUT_FILE.parent, exist_ok=True)
process_and_write_jsonl(INPUT_DIRS, OUTPUT_FILE)
print(f"Dataset written to {OUTPUT_FILE}")
```

```
gutenberg_dataset.jsonl already exists and is non-empty, skipping processing.
Dataset written to gutenberg_dataset.jsonl
```

1.4 Model Fine-Tuning with LoRA

[13]:
```
import torch
from datasets import load_dataset, Dataset
from transformers import AutoModelForCausalLM, AutoTokenizer,␣
 ↪BitsAndBytesConfig, TrainingArguments
from peft import get_peft_model, LoraConfig, prepare_model_for_kbit_training
from trl import SFTTrainer, SFTConfig

# Load the dataset
print("Loading dataset...")
with open(OUTPUT_FILE, 'r', encoding='utf-8') as f:
    data = [json.loads(line) for line in f if line.strip()]
dataset = Dataset.from_list(data)
print(f"Dataset loaded with {len(dataset)} records.")

# Load tokenizer and model
MODEL_NAME = "gpt2"
tokenizer = AutoTokenizer.from_pretrained(MODEL_NAME)
tokenizer.pad_token = tokenizer.eos_token

# Load model with 8-bit precision
bnb_config = BitsAndBytesConfig(load_in_8bit=True)
model = AutoModelForCausalLM.from_pretrained(MODEL_NAME,␣
 ↪quantization_config=bnb_config, device_map="auto")
model = prepare_model_for_kbit_training(model)

# Configure LoRA
lora_config = LoraConfig(
    r=8,
    lora_alpha=32,
    target_modules=["c_attn", "c_proj"],
    lora_dropout=0.05,
    bias="none",
    task_type="CAUSAL_LM"
)
model = get_peft_model(model, lora_config)

# Tokenization function
def tokenize_function(examples):
    return tokenizer(examples["text"], truncation=True, padding="max_length",␣
 ↪max_length=1024)

# Tokenize the dataset
```

```python
tokenized_dataset = dataset.map(tokenize_function, batched=True,␣
↪remove_columns=["text", "filename", "source"])

# Create training arguments (SFTConfig wraps TrainingArguments internally)
training_args = SFTConfig(
    output_dir="./poetic_sci_fi_model",
    per_device_train_batch_size=4,
    num_train_epochs=3,
    logging_steps=50,
    save_strategy="epoch",
    fp16=True,
    push_to_hub=False,
    report_to="none",
    overwrite_output_dir=True
)

# Initialize trainer (tokenizer no longer passed here)
trainer = SFTTrainer(
    model=model,
    args=training_args,
    train_dataset=tokenized_dataset,
)

# Train the model
print("Starting training...")
trainer.train()

# Save the model
model_path = "./poetic_sci_fi_model"
trainer.save_model(model_path)
tokenizer.save_pretrained(model_path)
print(f"Model saved to {model_path}")
```

```
Loading dataset…
Dataset loaded with 20 records.

Map:   0%|          | 0/20 [00:00<?, ? examples/s]

Truncating train dataset:   0%|          | 0/20 [00:00<?, ? examples/s]

No label_names provided for model class `PeftModelForCausalLM`. Since
`PeftModel` hides base models input arguments, if label_names is not given,
label_names can't be set automatically within `Trainer`. Note that empty
label_names list will be used instead.

Starting training…

/usr/local/lib/python3.11/dist-packages/torch/_dynamo/eval_frame.py:745:
UserWarning: torch.utils.checkpoint: the use_reentrant parameter should be
passed explicitly. In version 2.5 we will raise an exception if use_reentrant is
```

```
not passed. use_reentrant=False is recommended, but if you need to preserve the
current default behavior, you can pass use_reentrant=True. Refer to docs for
more details on the differences between the two variants.
  return fn(*args, **kwargs)
/usr/local/lib/python3.11/dist-packages/bitsandbytes/autograd/_functions.py:185:
UserWarning: MatMul8bitLt: inputs will be cast from torch.float32 to float16
during quantization
  warnings.warn(f"MatMul8bitLt: inputs will be cast from {A.dtype} to float16
during quantization")
```

```
<IPython.core.display.HTML object>
```

```
/usr/local/lib/python3.11/dist-packages/torch/_dynamo/eval_frame.py:745:
UserWarning: torch.utils.checkpoint: the use_reentrant parameter should be
passed explicitly. In version 2.5 we will raise an exception if use_reentrant is
not passed. use_reentrant=False is recommended, but if you need to preserve the
current default behavior, you can pass use_reentrant=True. Refer to docs for
more details on the differences between the two variants.
  return fn(*args, **kwargs)
/usr/local/lib/python3.11/dist-packages/bitsandbytes/autograd/_functions.py:185:
UserWarning: MatMul8bitLt: inputs will be cast from torch.float32 to float16
during quantization
  warnings.warn(f"MatMul8bitLt: inputs will be cast from {A.dtype} to float16
during quantization")
/usr/local/lib/python3.11/dist-packages/torch/_dynamo/eval_frame.py:745:
UserWarning: torch.utils.checkpoint: the use_reentrant parameter should be
passed explicitly. In version 2.5 we will raise an exception if use_reentrant is
not passed. use_reentrant=False is recommended, but if you need to preserve the
current default behavior, you can pass use_reentrant=True. Refer to docs for
more details on the differences between the two variants.
  return fn(*args, **kwargs)
/usr/local/lib/python3.11/dist-packages/bitsandbytes/autograd/_functions.py:185:
UserWarning: MatMul8bitLt: inputs will be cast from torch.float32 to float16
during quantization
  warnings.warn(f"MatMul8bitLt: inputs will be cast from {A.dtype} to float16
during quantization")
```

```
Model saved to ./poetic_sci_fi_model
```

1.5 Story Generation

[14]:
```python
from transformers import pipeline

# Load the fine-tuned model
tokenizer = AutoTokenizer.from_pretrained(model_path)
model = AutoModelForCausalLM.from_pretrained(model_path)
generator = pipeline("text-generation", model=model, tokenizer=tokenizer)

# Define a prompt
```

```python
prompt = "In the twilight of the cosmos, a lone traveler discovers a hidden␣
  ↪world where dreams manifest into reality."

# Generate a story
output = generator(
    prompt,
    max_new_tokens=1000,
    do_sample=True,
    temperature=0.95,
    top_k=50,
    top_p=0.92,
    repetition_penalty=1.1,
    num_return_sequences=1,
    eos_token_id=tokenizer.eos_token_id
)

print("\nGenerated Poetic Sci-Fi Story:\n")
print(output[0]["generated_text"])
```

```
/usr/local/lib/python3.11/dist-packages/peft/tuners/lora/layer.py:1768:
UserWarning: fan_in_fan_out is set to False but the target module is `Conv1D`.
Setting fan_in_fan_out to True.
  warnings.warn(
Device set to use cuda:0


Generated Poetic Sci-Fi Story:

In the twilight of the cosmos, a lone traveler discovers a hidden world where
dreams manifest into reality. But when he's in need of help to find his brother
who is also possessed by an unknown spirit, Captain Vastra goes from trying to
make sense of what was just described through visions into believing it isn't
real and ultimately falling off at last as time runs out for him! And yet this
man has already been sent back down with untold vengeance on Earth that fateful
night…
```