AI Task 1: Fine-Tuning GPT-2 for Text Generation

Introduction

This guide provides a step-by-step explanation of how to fine-tune GPT-2, a transformer model developed by OpenAI, to generate coherent and contextually relevant text. You will learn how to fine-tune GPT-2 on a custom dataset using Google Colab.

Prerequisites

Before starting, ensure you have:

- A Google Colab account
- Basic understanding of Python and deep learning frameworks
- Familiarity with Hugging Face's Transformers library
- A prepared text dataset for training

Step-by-Step Process

Step 1: Setting Up the Environment

- Open Google Colab.
- Select Runtime > Change runtime type and choose GPU for faster training.
- Install necessary dependencies:

```
!pip install -q gradio
!pip install -q transformers datasets torch
```

```
!pip install datasets
```

Show hidden output

Load and Prepare Your Custom Dataset

Upload your text dataset (CSV format) to Google Colab.

Read the CSV file into a DataFrame and preprocess the text

```
import pandas as pd

# Load CSV dataset
file_path = "/content/Conversation[1].csv"

# Read CSV file into a DataFrame
df = pd.read_csv(file_path)

# Check the first few rows to verify
print(df.head())
```

```
   Unnamed: 0                           question  \
0           0               hi, how are you doing?
1           1        i'm fine. how about yourself?
2           2  i'm pretty good. thanks for asking.
3           3    no problem. so how have you been?
4           4     i've been great. what about you?

                                  answer
0            i'm fine. how about yourself?
1      i'm pretty good. thanks for asking.
2        no problem. so how have you been?
3         i've been great. what about you?
4  i've been good. i'm in school right now.
```

Merging Question and Answer into Conversation Pairs

- Merging question and answer columns to create conversation pairs.
- Keeping only the text column for training by dropping unnecessary columns.
- Ensuring the dataset is properly formatted for fine-tuning GPT-2.
- Preparing the data for tokenization and model training.
- Optimizing the dataset to improve model performance and relevance.

```
# Merge question and answer to form conversation pairs
df["text"] = df["question"] + " " + df["answer"]

# Drop unnecessary columns
df = df[["text"]]
```

- Converting the DataFrame into a Hugging Face Dataset for efficient processing.

```
from datasets import Dataset

# Convert DataFrame to Hugging Face Dataset
```

```
dataset = Dataset.from_pandas(df)
```

Tokenization of Text Data Using GPT-2 Tokenizer

- **Import:** Load the AutoTokenizer from the transformers library.
- **Load Tokenizer:** Initialize the GPT-2 tokenizer.
- **Set Padding Token:** Assign the padding token to the end-of-sequence token.
- **Tokenization Function:**
  - Define a function to tokenize input text. Enable truncation and set padding to maximum length (512 tokens).
  - Create labels from input IDs for causal language modeling.
- **Tokenize Dataset:** Apply the tokenization function to the dataset in batches.

```
from transformers import AutoTokenizer

# Load GPT-2 tokenizer
tokenizer = AutoTokenizer.from_pretrained("gpt2")
tokenizer.pad_token = tokenizer.eos_token

def tokenize_function(examples):
    tokens = tokenizer(examples["text"], truncation=True, padding="max_length", max_length=512)
    tokens["labels"] = tokens["input_ids"].copy()  # GPT-2 requires labels for CLM
    return tokens

# Tokenize dataset with labels
tokenized_dataset = dataset.map(tokenize_function, batched=True)
```

```
Map: 100% 3725/3725 [00:02<00:00, 1670.99 examples/s]
```

Saving the GPT-2 Tokenizer

Save the tokenizer to a specified directory (e.g., ./gpt2-finetuned).

```
from transformers import AutoTokenizer

# Load the GPT-2 tokenizer
tokenizer = AutoTokenizer.from_pretrained("gpt2")

# Save it to the same directory as the model
```

```
tokenizer.save_pretrained("./gpt2-finetuned")
```

```
('./gpt2-finetuned/tokenizer_config.json',
 './gpt2-finetuned/special_tokens_map.json',
 './gpt2-finetuned/vocab.json',
 './gpt2-finetuned/merges.txt',
 './gpt2-finetuned/added_tokens.json',
 './gpt2-finetuned/tokenizer.json')
```

Loading a Fine-Tuned GPT-2 Model and Tokenizer

- **Import:** Load AutoModelForCausalLM and AutoTokenizer from the transformers library.
- **Specify Model Path:** Define the path where the fine-tuned model and tokenizer are saved (e.g., ./gpt2-finetuned).
- **Load Model:** Initialize the fine-tuned GPT-2 model from the specified path.
- **Load Tokenizer:** Initialize the tokenizer from the same path.
- **Confirmation:** Print a success message indicating that the model and tokenizer have been loaded successfully.

```
from transformers import AutoModelForCausalLM, AutoTokenizer

model_path = "./gpt2-finetuned"

# Load the fine-tuned model and tokenizer
model = AutoModelForCausalLM.from_pretrained(model_path)
tokenizer = AutoTokenizer.from_pretrained(model_path)

print("Model and tokenizer loaded successfully!")
```

```
Model and tokenizer loaded successfully!
```

Saving the Fine-Tuned Model

Save Model: Use the save_model method of the trainer to save the fine-tuned model to a specified directory (e.g., ./gpt2-finetuned)

```
trainer.save_model("./gpt2-finetuned")
```

Generating Responses with a Fine-Tuned GPT-2 Model

- **Import**: Load AutoModelForCausalLM and AutoTokenizer from the transformers library.
- **Load Model and Tokenizer:**
  - Specify the path to the fine-tuned model (e.g., ./gpt2-finetuned).

- Load the fine-tuned model.
- Load the GPT-2 tokenizer.

- **Define Response Generation Function**:
  - Create a function generate_response that takes a prompt as input.
  - Tokenize the prompt and prepare it for the model.
  - Generate a response using the model with specified parameters (e.g., max_length=50).
  - Decode the generated output to convert it back to text, skipping special tokens.
- **Test the Function:**
  - Define a test prompt (e.g., "hi, how are you doing?").
  - Call the generate_response function with the prompt.
  - Print the generated response.

```
from transformers import AutoModelForCausalLM, AutoTokenizer

# Load the fine-tuned model and tokenizer
model_path = "./gpt2-finetuned"
model = AutoModelForCausalLM.from_pretrained(model_path)
tokenizer = AutoTokenizer.from_pretrained("gpt2")

# Generate a response
def generate_response(prompt):
    inputs = tokenizer(prompt, return_tensors="pt")
    output = model.generate(**inputs, max_length=50, pad_token_id=tokenizer.eos_token_id)
    return tokenizer.decode(output[0], skip_special_tokens=True)

# Test it
prompt = "hi, how are you doing?"
response = generate_response(prompt)
print(response)
```

```
hi, how are you doing? i'm doing okay.
```

Using the Text Generation Pipeline with a Fine-Tuned GPT-2 Model

Import

- Load the pipeline function from the transformers library

Load Model and Tokenizer:

- Specify the path to the fine-tuned model (e.g., ./gpt2-finetuned).
- Create a text generation pipeline using the specified model and tokenizer.

Test Input

- Define a prompt (e.g., "Hello, how are you?").
- Use the generator to produce text based on the prompt, specifying parameters like max_length=50 and num_return_sequences=1.

Print Generated Text:

- Access and print the generated text from the output.

```
from transformers import pipeline

# Load model and tokenizer
model_path = "./gpt2-finetuned"
generator = pipeline("text-generation", model=model_path, tokenizer=model_path)

# Test input
prompt = "Hello, how are you?"
output = generator(prompt, max_length=50, num_return_sequences=1)

# Print generated text
print(output[0]['generated_text'])
```

```
Device set to use cuda:0
Truncation was not explicitly activated but `max_length` is provided a specific value, please use `truncation=True` to explicitly truncate examples to max length. Defaulting to
Setting `pad_token_id` to `eos_token_id`:50256 for open-end generation.
Hello, how are you? i'm well and comfortable, so why are you in here?
```

Creating a Text Generation API with FastAPI

Import FastAPI: Load the FastAPI class from the fastapi library.

Import Pipeline: Load the pipeline function from the transformers library.

Initialize FastAPI App: Create an instance of the FastAPI application.

Load Model and Tokenizer:

- Specify the path to the fine-tuned model (e.g., ./gpt2-finetuned).
- Create a text generation pipeline using the specified model and tokenizer.
- *Define API Endpoint:**
- Create a POST endpoint at /generate/ that accepts a prompt as input.
- Use the generator to produce text based on the provided prompt, specifying parameters like max_length=50 and num_return_sequences=1.
- Return the generated text in a JSON format.

Run the Application:

- Use the command uvicorn filename:app --reload to run the FastAPI application, replacing filename with the name of your Python file.

```
from fastapi import FastAPI
from transformers import pipeline

app = FastAPI()

# Load model and tokenizer
model_path = "./gpt2-finetuned"
generator = pipeline("text-generation", model=model_path, tokenizer=model_path)

@app.post("/generate/")
async def generate_text(prompt: str):
    output = generator(prompt, max_length=50, num_return_sequences=1)
    return {"generated_text": output[0]['generated_text']}

# Run: uvicorn filename:app --reload
```

```
Device set to use cuda:0
```

```
import torch
from transformers import AutoModelForCausalLM, AutoTokenizer

# Set device to GPU
device = "cuda" if torch.cuda.is_available() else "cpu"

# Load model and tokenizer
model_path = "./gpt2-finetuned"
model = AutoModelForCausalLM.from_pretrained(model_path).to(device)
tokenizer = AutoTokenizer.from_pretrained("gpt2")

# Generate text
prompt = "Hello, how are you?"
inputs = tokenizer(prompt, return_tensors="pt").to(device)
outputs = model.generate(**inputs, max_length=50)

# Decode and print result
print(tokenizer.decode(outputs[0], skip_special_tokens=True))
```

```
Setting `pad_token_id` to `eos_token_id`:50256 for open-end generation.
Hello, how are you? i'm fine.
```

```
prompt = "How was your day?"
inputs = tokenizer(prompt, return_tensors="pt").to(device)

# Generate response with better sampling
```

```
outputs = model.generate(
    **inputs,
    max_length=50,
    temperature=0.7,  # Controls randomness (lower = predictable, higher = creative)
    top_p=0.9,         # Nucleus sampling (filters unlikely words)
    top_k=50           # Limits vocabulary size per step
)

print(tokenizer.decode(outputs[0], skip_special_tokens=True))
```

```
/usr/local/lib/python3.11/dist-packages/transformers/generation/configuration_utils.py:629: UserWarning: `do_sample` is set to `False`. However, `temperature` is set to `0.7`
  warnings.warn(
/usr/local/lib/python3.11/dist-packages/transformers/generation/configuration_utils.py:634: UserWarning: `do_sample` is set to `False`. However, `top_p` is set to `0.9` -- thi
  warnings.warn(
Setting `pad_token_id` to `eos_token_id`:50256 for open-end generation.
How was your day? i was busy.
```

Pushing a Fine-Tuned Model and Tokenizer to the Hugging Face Hub

```
from huggingface_hub import notebook_login
notebook_login()

model.push_to_hub("pravallika6167/gpt2-finetuned-chatbot")
tokenizer.push_to_hub("pravallika6167/gpt2-finetuned-chatbot")
```

Copy a token from your Hugging Face tokens page and paste it below.

Immediately click login after copying your token or it might be stored in plain text in this notebook file.

Token: ••••••••

Add token as git credential?

Login

Pro Tip: If you don't already have one, you can create a dedicated 'notebooks' token with 'write' access, that you can then easily reuse for all notebooks.

model.safetensors: 100% 498M/498M [00:16<00:00, 54.0MB/s]

README.md: 100% 5.17k/5.17k [00:00<00:00, 221kB/s]

```
CommitInfo(commit_url='https://huggingface.co/pravallika6167/gpt2-finetuned-chatbot/commit/e6c72b1ff04aa8d7bb9beb82851c86f41f7552a1', commit_message='Upload tokenizer',
commit_description='', oid='e6c72b1ff04aa8d7bb9beb82851c86f41f7552a1', pr_url=None, repo_url=RepoUrl('https://huggingface.co/pravallika6167/gpt2-finetuned-chatbot',
```

```python
from transformers import AutoModelForCausalLM, AutoTokenizer

model_name = "pravallika6167/gpt2-finetuned-chatbot"

# Load model and tokenizer from Hugging Face
tokenizer = AutoTokenizer.from_pretrained(model_name)
model = AutoModelForCausalLM.from_pretrained(model_name)

# Generate response
input_text = "Hello, how are you?"
inputs = tokenizer(input_text, return_tensors="pt")

output = model.generate(**inputs, max_length=50)
response = tokenizer.decode(output[0], skip_special_tokens=True)

print("Chatbot:", response)
```

```
tokenizer_config.json: 100%  475/475 [00:00<00:00, 29.3kB/s]
vocab.json: 100%  798k/798k [00:00<00:00, 10.4MB/s]
merges.txt: 100%  456k/456k [00:00<00:00, 13.0MB/s]
tokenizer.json: 100%  3.56M/3.56M [00:00<00:00, 10.5MB/s]
special_tokens_map.json: 100%  99.0/99.0 [00:00<00:00, 2.33kB/s]
config.json: 100%  924/924 [00:00<00:00, 26.0kB/s]
model.safetensors: 100%  498M/498M [00:11<00:00, 42.8MB/s]
generation_config.json: 100%  124/124 [00:00<00:00, 5.30kB/s]
```

```
from transformers import pipeline

chatbot = pipeline("text-generation", model=model_name)
response = chatbot("Hi, what's up?", max_length=50)
print(response[0]["generated_text"])
```

```
Device set to use cuda:0
Truncation was not explicitly activated but `max_length` is provided a specific value, please use `truncation=True` to explicitly truncate examples to max length. Defaulting to
Setting `pad_token_id` to `eos_token_id`:50256 for open-end generation.
```

Creating a Chat Interface with Gradio

In this example, we built a simple chat interface using Gradio to interact with a fine-tuned GPT-2 model. The chat function takes a user message as input, tokenizes it, generates a response using the model, and then decodes the output back into text. By utilizing Gradio's Interface, we easily set up a web-based interface where users can input text and receive generated responses in real-time. This approach allows for quick prototyping and sharing of machine learning models, making it accessible for users to interact with AI models without needing extensive programming knowledge.

```
import gradio as gr

def chat(message):
    inputs = tokenizer(message, return_tensors="pt")
    output = model.generate(**inputs, max_length=50)
    return tokenizer.decode(output[0], skip_special_tokens=True)

gr.Interface(fn=chat, inputs="text", outputs="text").launch()
```

message

how are you

output

how are you doing? i'm doing well.

Clear Submit Flag