GPT2 Finetuning

```
from google.colab import drive
drive.mount('/content/drive')
```

```
     Mounted at /content/drive
```

```
#final_0_1 = pd.read_csv('./QAData0_1.csv')
#final_0_2 = pd.read_csv('./QAData0_2.csv')
#final = pd.concat([final_0_1, final_0_2],ignore_index=True)
import pandas as pd

df1 = pd.read_csv('./QAData1.csv')
# df2 = pd.read_csv('./QAData2.csv')
# df3 = pd.read_csv('./QAData3.csv')
# df4 = pd.read_csv('./QAData4.csv')
# df5 = pd.read_csv('./QAData5.csv')
# df6 = pd.read_csv('./QAData6.csv')
# df7 = pd.read_csv('./QAData7.csv')

final = pd.concat([df1])

all_questions = final['questions'].tolist()
all_answers = final['answers'].tolist()
```

```
print(len(all_questions))
print(len(all_answers))
for i in range(5):
  print("Q:", all_questions[i])
  print("A:", all_answers[i])
  print()
```

```
     39801
     39801
     Q:  You're back here in this place where you've won before.  Indiscernible  w
     A:  I love the golf course but the fans are what makes it awesome.  To be abl

     Q:  I caught a glimpse of you looking at the wall of achievements.  I can't i
     A:  Yeah it's an honor.  A lot of names up there that I would've loved to pla

     Q:  In terms of making your comeback now can you give us an update on how you
     A:  I'm just taking it one day at a time really.  It's been a struggle.  I'm

     Q:  How tough is it to fight through that when you're trying to find the cons
     A:  Yeah it's pretty tough.  It's pretty tough to find.  I don't know man.  I

     Q:  Does it give you a different perspective having to fight through it this
```

```
    A:  Oh yeah.  It gives me a lot bigger perspective especially my kids and stu
```

Chat QA with GPT-2

```
!pip install transformers
```

```
from transformers import GPT2Tokenizer, GPT2LMHeadModel
```

```
import torch
device = "cuda" if torch.cuda.is_available() else "cpu"

tokenizer = GPT2Tokenizer.from_pretrained("gpt2")
tokenizer.add_special_tokens({"pad_token": "<pad>",
                                "bos_token": "<startofstring>",
                                "eos_token": "<endofstring>"})
tokenizer.add_tokens(["<ans>:"])

model = GPT2LMHeadModel.from_pretrained("gpt2")
model.resize_token_embeddings(len(tokenizer))

model = model.to(device)
```

```
from torch.utils.data import Dataset

class QAData(Dataset):
    def __init__(self, tokenizer, q, a):
        self.X = []
        for i in range(len(q)):
          self.X.append("<startofstring> "+ str(q[i]) +" <ans>: "+str(a[i])+" <en

        print(self.X[0])

        self.X_encoded = tokenizer(self.X, max_length=128, truncation=True, paddi
        self.input_ids = self.X_encoded['input_ids']
        self.attention_mask = self.X_encoded['attention_mask']

    def __len__(self):
        return len(self.X)

    def __getitem__(self, idx):
        return (self.input_ids[idx], self.attention_mask[idx])

ds = QAData(tokenizer, all_questions, all_answers)
```

```
    <startofstring>  You're back here in this place where you've won before.  Ind
```

```
from torch.utils.data import DataLoader
QAData =  DataLoader(ds, batch_size=16)
```

Training the model

```
import torch
from transformers import GPT2LMHeadModel, GPT2Tokenizer
from torch.utils.data import Dataset, DataLoader
from torch.optim import Adam
from tqdm import tqdm

# Device configuration
device = "cuda" if torch.cuda.is_available() else "cpu"

# Initialize the model and tokenizer
tokenizer = GPT2Tokenizer.from_pretrained("gpt2")
model = GPT2LMHeadModel.from_pretrained("gpt2")
model = model.to(device)

# Add your special tokens and resize token embeddings
# (assuming you have custom tokens like in your previous setup)
tokenizer.add_special_tokens({"pad_token": "<pad>",
                              "bos_token": "<startofstring>",
                              "eos_token": "<endofstring>",
                              "additional_special_tokens": ["<ans>:"]})
model.resize_token_embeddings(len(tokenizer))

# Define the optimizer
optim = Adam(model.parameters(), lr=1e-3)

# Define the Dataset and DataLoader (ensure QAData class and dataset are defined)
# Assuming all_questions and all_answers are defined
# ds = QAData(tokenizer, all_questions, all_answers)
QAData = DataLoader(ds, batch_size=16)

# Define the training loop
def train(QADataLoader, model, optim):
    model.train()
    epochs = 1
    for epoch in tqdm(range(epochs)):
        for X, a in QADataLoader:
            X = X.to(device)
            a = a.to(device)
            optim.zero_grad()
            loss = model(X, attention_mask=a, labels=X).loss
            loss.backward()
            optim.step()
        # Save the model after each epoch
        # Save the model after each epoch
        torch.save(model.state_dict(), f"model_state_epoch_{epoch}.pt")

    # Save the tokenizer after training is complete
    tokenizer.save_pretrained('trained_tokenizer')

# Train the model
print("training .... ")
train(QAData, model, optim)
```

Loading, tuning, and validating the model

```
# Function to load the model
def load_model(model_path, tokenizer_path):
    # Load the trained tokenizer
    tokenizer = GPT2Tokenizer.from_pretrained(tokenizer_path)
    # Initialize the model
    model = GPT2LMHeadModel.from_pretrained("gpt2",
                                             pad_token_id=tokenizer.eos_token_id)
    # Update the token embeddings
    model.resize_token_embeddings(len(tokenizer))
    # Load the saved weights
    model.load_state_dict(torch.load(model_path))
    model.to(device)
    return model, tokenizer

def infer(prompt, model, tokenizer, tuned):
    model.eval()
    if tuned:
      prompt = f"<startofstring> {prompt} <ans>: "
    encoded_input = tokenizer(prompt, return_tensors='pt')
    input_ids = encoded_input['input_ids'].to(device)
    attention_mask = encoded_input['attention_mask'].to(device)

    max_length = 150
    temperature = 1.2
    top_k = 50
    top_p = 0.9
    repetition_penalty = 1.7

    output = model.generate(
        input_ids,
        attention_mask=attention_mask,
        max_length=max_length,
        temperature=temperature,
        top_k=top_k,
        top_p=top_p,
        repetition_penalty=repetition_penalty
    )

    generated_text = tokenizer.decode(output[0], skip_special_tokens=True)
    return generated_text



# Load the model and tokenizer
model, tokenizer = load_model("model_state_epoch_0.pt", 'trained_tokenizer')

# Define the sports broadcaster role prompt
#role_prompt = "I am a sports broadcaster that commentates on sorts games and pro

# Run the chatbot with custom prompting
print("Sports QA Bot: ")
while True:
    user_input = input("You: ")
    if user_input.lower() == "quit":
        break
    #response = infer(user_input, model, tokenizer, role_prompt)
    response = infer(user_input, model, tokenizer, True)

    print(f"Sports Bot: {response}")
```

```
Special tokens have been added in the vocabulary, make sure the associated wo
Sports QA Bot:
You: What?
Sports Bot:  What?   I think it's a little bit of an adjustment. It was just
You: quit
```

```
fine_tuned_model, fine_tuned_tokenizer = load_model("model_state_epoch_0.pt", 'tr
# Load the default pre-trained model and tokenizer
default_model = GPT2LMHeadModel.from_pretrained("gpt2")
default_tokenizer = GPT2Tokenizer.from_pretrained("gpt2")
default_model.to(device)
```

Comparing the general model to the fine-tuned model

```
prompt = "What?"

# Generate text with the fine-tuned model
print("Text from Fine-tuned Model:")
print(infer(prompt, fine_tuned_model, fine_tuned_tokenizer, True))

# Generate text with the default pre-trained model
print("\nText from Default Pre-trained Model:")
print(infer(prompt, default_model, default_tokenizer, False))
```

```
Text from Fine-tuned Model:
```

```
Text from Fine-tuned Model:
Setting `pad_token_id` to `eos_token_id`:50256 for open-end generation.
 What?   I think it's a little bit of an adjustment. It was just the first ti

Text from Default Pre-trained Model:
What?
The first thing I noticed was that the "I'm a fan of your work" message on my
```

```
import locale
locale.getpreferredencoding = lambda: "UTF-8"
```

```
!pip install requests
```

Using GPT and the Fine-Tuned model for realistic text generation

```
import requests
import json

def query_openai(prompt, api_key):
    url = "https://api.openai.com/v1/chat/completions"
    headers = {
        "Content-Type": "application/json",
        "Authorization": f"Bearer {api_key}"
    }
    data = {
        "model": "gpt-4",
        "messages": [{"role": "user", "content": prompt}],
        "temperature": 0.1
    }

    response = requests.post(url, headers=headers, data=json.dumps(data))
    return response.json()

api_key = ""

prompt = ("You are a sports caster. Make up a fake play-by-play of a game. "
          "Describe the actions, the atmosphere, the crowd's reactions. "
          "Now, transition into an interview with a player. Say 'BEGIN INTERVIEW'
          "Generate two random interview questions, each enclosed with 'START QUE
          "Conclude the interview with 'END INTERVIEW'.")


response = query_openai(prompt, api_key)
generated_response = response['choices'][0]['message']['content']

# Function to extract questions from the response
def extract_tagged_questions(text):
    interview_start = text.find("BEGIN INTERVIEW")
    interview_end = text.find("END INTERVIEW", interview_start)
    interview_text = text[interview_start:interview_end] if interview_start != -1

    questions = []
    question_start = interview_text.find("START QUESTION")
    while question_start != -1:
        question_end = interview_text.find("END QUESTION", question_start)
        if question_end != -1:
            question_text = interview_text[question_start + len("START QUESTION")
            questions.append(question_text)
            question_start = interview_text.find("START QUESTION", question_end)
        else:
            break  # End if no 'END QUESTION' tag is found

    return questions

interview_questions = extract_tagged_questions(generated_response)
for idx, question in enumerate(interview_questions, 1):
    print(f"Question {idx}: {question}")
    print(infer(question, fine_tuned_model, fine_tuned_tokenizer, True))
```

```
Question 1: Johnson, you've had an incredible game tonight. Can you walk us tl
 Johnson, you've had an incredible game tonight. Can you walk us through your
Question 2: This season has been a rollercoaster for the Knights. How do you
 This season has been a rollercoaster for the Knights. How do you think this
```

Comparing to only GPT4 text

```
import requests
import json

def query_openai(prompt, api_key):
    url = "https://api.openai.com/v1/chat/completions"
    headers = {
        "Content-Type": "application/json",
        "Authorization": f"Bearer {api_key}"
    }
    data = {
        "model": "gpt-4",
        "messages": [{"role": "user", "content": prompt}],
        "temperature": 0.1
    }

    response = requests.post(url, headers=headers, data=json.dumps(data))
    return response.json()
```

```
api_key = ""

prompt = ("You are a sports caster. Make up a fake play-by-play of a game. "
          "Describe the actions, the atmosphere, the crowd's reactions. "
          "Now, transition into an interview with a player. Say 'BEGIN INTERVIEW'
          "Generate two random interview questions, each enclosed with 'START QUE
          "Conclude the interview with 'END INTERVIEW'.")


response = query_openai(prompt, api_key)
generated_response = response['choices'][0]['message']['content']

print(generated_response)
```

```
Ladies and gentlemen, we're here at the packed Madison Square Garden for the

The Titans have the ball. Johnson passes it to Rodriguez, who's been on fire

The buzzer sounds, and it's over! The New York Titans have won the championsh

Now, we're going to transition into an interview with the star of the night,

START QUESTION: Rodriguez, you've been phenomenal tonight. What was going thr

Rodriguez: Honestly, I was just focused on the basket. I knew I had to make i

START QUESTION: This is a huge win for the Titans. How does it feel to be a p

Rodriguez: It's an incredible feeling. We've worked so hard for this, and to

END INTERVIEW.

There you have it, folks. A night of unforgettable basketball, a historic win
```