Fine-Tuning a Chatbot Model Using OpenAI API

A Practical Approach to Building and Monitoring a Chatbot

Vaishnavi Patil - 20133

11-19-2024

Table of Contents

Introduction

Overview:

- This project focuses on fine-tuning OpenAI's GPT-3.5 model to create a factual chatbot.
- The process includes preparing data, uploading it, fine-tuning the model, and monitoring its performance.

Objective:

- Build a sarcastic chatbot with real-world factual responses, simulating a humorous conversational style.

Design - Identify and Understand the Problems

Problem:

- Standard chatbots often give straightforward responses but lack personality.
- **Goal**: Make the chatbot humorous and sarcastic while maintaining factual accuracy.

Solution:

- Fine-tune GPT-3.5 on a specially curated dataset containing sarcastic dialogues to add humor and factual correctness.

Design - Investigating Possible Solutions

Solution Options:

1. Use an existing sarcastic chatbot model.
2. Fine-tune an OpenAI model with custom data (chosen solution).

Reason for Choosing Fine-Tuning:

- Fine-tuning GPT-3.5 with custom data ensures the chatbot retains flexibility and can adapt to various conversational contexts.
- It also allows for personalized responses based on the trained dataset.

Design - Comparing Solutions

Solution 1: Pre-built chatbot

- **Pros:** Quick setup.
- **Cons**: Less customizable, may not align with specific needs.

Solution 2: Fine-tuning GPT-3.5

- **Pros**: Highly customizable, able to integrate new humor and factual accuracy.
- **Cons**: Requires custom dataset preparation and more time.

Selected Solution: Fine-tuning GPT-3.5 using a dataset of factual responses.

Implementation - How It Was Done

Step 1: Convert JSON Data to JSONL Format

- The dataset is initially in JSON format but needs to be in JSONL (JSON Lines) format for OpenAI's fine-tuning process.
- Python function `json_to_jsonl()` converts the data.

Step 2: Upload File to OpenAI

- Upload the prepared JSONL file using OpenAI's file upload API.

Step 3: Start Fine-Tuning

- Trigger the fine-tuning job via the OpenAI API, specifying the model (`gpt-3.5-turbo-1106`) and providing the file ID.

Implementation - File Preparation

JSONL File Format Example:

- {"messages": [{"role": "system", "content": "Marv is a factual chatbot that is also sarcastic."}, {"role": "user", "content": "What's the capital of France?"}, {"role": "assistant", "content": "Paris, as if everyone doesn't know that already."}]}

Dataset:

- The dataset consists of dialogues between an assistant and the user.
- Each entry includes system, user, and assistant roles for interaction.

Implementation - Monitoring and Saving Metrics

Monitoring the Fine-Tuning Job:

- The script checks the status of the fine-tuning job periodically.
- Once completed, metrics such as training loss and accuracy are extracted and saved to a CSV file.

Metrics Saved:

- `train_loss`
- `total_steps`
- `train_mean_token_accuracy`

Testing the Model

Testing the Fine-Tuned Model:

- Sample queries (e.g., "What's the capital of France?") are fed to the fine-tuned model.
- The model responds with answers, confirming it learned the desired behavior.

Evaluation:

- Responses are evaluated based on humor, accuracy, and relevance.
- Model's performance is monitored via metrics saved during the fine-tuning process.

Enhancement Ideas

Additional Improvements:

1. **Expand Dataset:** Add more varied sarcastic responses for a wider conversational scope.

2. **Fine-Tune with More Context:** Introduce longer dialogues or multi-turn conversations to increase the model's ability to handle more complex interactions.

3. **Use Reinforcement Learning:** Use user feedback to fine-tune the model even further, focusing on sarcasm and humor.

Future Enhancements:

- Introduce personalized sarcasm based on the user's previous interactions.

Conclusion

Summary:

- We have successfully fine-tuned GPT-3.5 to create a sarcastic chatbot that gives factual yet humorous responses.
- The project includes steps for data preparation, fine-tuning, monitoring, and testing.

Key Takeaways:

- Fine-tuning an existing model is a powerful way to create custom AI solutions.
- The chatbot offers potential applications in customer service, entertainment, and conversational AI.

References

Technical References:

1. OpenAI API Documentation: https://beta.openai.com/docs/
2. Fine-Tuning GPT-3: https://beta.openai.com/docs/guides/fine-tuning
3. Python JSON and JSONL Handling: https://docs.python.org/3/library/json.html
4. Python CSV Handling: https://docs.python.org/3/library/csv.html

Appendix

Fine_Tune.py

Fine_Tune.py > json_to_jsonl

```python
import openai, csv, os
import time, json
from dotenv import load_dotenv, find_dotenv

# Load environment variables for OpenAI API
load_dotenv(find_dotenv())
openai.api_key = os.getenv("OPENAI_API_KEY")

# Function to convert JSON to JSONL format
def json_to_jsonl(input_json_path, output_jsonl_path):
    """
    Converts a JSON file to JSONL format by writing each object in the JSON array to a new line in the output JSONL file.
    """
    try:
        # Open and load the JSON file
        with open(input_json_path, 'r') as json_file:
            data = json.load(json_file)

        # Write the data to the jsonl file, each item on a new line
        with open(output_jsonl_path, 'w') as jsonl_file:
            for item in data:
                jsonl_file.write(json.dumps(item) + '\n')

        print(f"Conversion successful! {input_json_path} has been converted to {output_jsonl_path}")
    except Exception as e:
        print(f"Error during conversion: {e}")

# Example usage of the conversion function
input_json_path = "data.json"  # Path to your input JSON file
output_jsonl_path = "data_prepared.jsonl"  # Path to save the output JSONL file
json_to_jsonl(input_json_path, output_jsonl_path)

# Function to upload the training file to OpenAI
def upload_file(file_path):
    """
    Uploads the training file to OpenAI's server for fine-tuning.
    Returns the file ID if the upload is successful, else returns None.
    """
    try:
        response = openai.files.create(file=open(file_path, "rb"), purpose="fine-tune")
        print(f"File uploaded: {response.id}")
        return response.id
    except Exception as e:
        print(f"Error uploading file: {e}")
        return None

# Function to start the fine-tuning job
def start_fine_tuning(training_file_id, model="gpt-3.5-turbo-1106"):
    """
    Starts a fine-tuning job with the specified training file ID and model.
    Returns the job ID if the job starts successfully, else returns None.
    """
    try:
        response = openai.fine_tuning.jobs.create(
            training_file=training_file_id,
            model=model,
            suffix="Vaishnavi Model"  # Adding the suffix directly in the fine-tuning job
        )
        print(f"Fine-tuning job started: {response.id}")
        return response.id
    except Exception as e:
        print(f"Error starting fine-tuning: {e}")
        return None

# Function to monitor the fine-tuning job and save metrics to a CSV file
def monitor_and_save(job_id, output_csv):
```

Fine_Tune.py

Fine_Tune.py > json_to_jsonl

```python
# Function to monitor the fine-tuning job and save metrics to a CSV file
def monitor_and_save(job_id, output_csv):
    """
    Monitors the fine-tuning job's status. Once the job is completed,
    it saves the metrics (like loss and accuracy) to a CSV file.
    """
    try:
        while True:
            job_status = openai.fine_tuning.jobs.retrieve(job_id)
            # Check the job's status
            if job_status.status == 'succeeded':
                print("Fine-tuning completed successfully!")
                events = openai.fine_tuning.jobs.list_events(job_id)
                save_metrics_to_csv(events, output_csv)  # Save metrics to CSV
                break
            elif job_status.status == 'failed':
                print("Fine-tuning failed!")
                break
            else:
                print(f"Fine-tuning in progress... (status: {job_status.status})")
                time.sleep(60)  # Wait for 60 seconds before checking again
    except Exception as e:
        print(f"Error monitoring job: {e}")

# Function to save metrics to a CSV file
def save_metrics_to_csv(events, output_csv):
    """
    Saves the fine-tuning job's metrics (like training loss, sequence accuracy, token accuracy) to a CSV file.
    """
    try:
        with open(output_csv, mode='w', newline='') as file:
            writer = csv.writer(file)
            # Write CSV header
            writer.writerow(["step", "train_loss", "total_steps", "train_mean_token_accuracy"])

            # Iterate through the events and extract metrics
            for event in events:
                if event.type == 'metrics':  # Only process metric events
                    metrics = event.data  # Extract metric data
                    writer.writerow([
                        metrics.get('step', 'N/A'),
                        metrics.get('train_loss', 'N/A'),
                        metrics.get('total_steps', 'N/A'),
                        metrics.get('train_mean_token_accuracy', 'N/A')
                    ])
        print(f"Metrics successfully saved to {output_csv}")
    except Exception as e:
        print(f"Error saving metrics to CSV: {e}")

# Main function to drive the fine-tuning process
def main():
    file_path = "data_prepared.jsonl"  # Path to your dataset (converted to JSONL format)
    output_csv = "fine_tuning_metrics.csv"  # Output CSV file for metrics

    # Step 1: Upload the training file
    training_file_id = upload_file(file_path)
    if not training_file_id:
        return  # Exit if the file upload fails

    # Step 2: Start the fine-tuning job
    job_id = start_fine_tuning(training_file_id)
    if not job_id:
        return  # Exit if the fine-tuning job fails to start

    # Step 3: Monitor the fine-tuning job and save metrics to CSV
    monitor_and_save(job_id, output_csv)

# Execute the main function
if __name__ == "__main__":
    main()
```

Appendix

fine_tuning_metrics.csv

fine_tuning_metrics.csv > data

```csv
step,train_loss,total_steps,train_mean_token_accuracy
54,0.1215573251247406,99,0.9523809552192688
53,0.19870059192180634,99,0.9166666865348816
52,0.07533347606658936,99,0.9375
51,0.0041697025299072266,99,1.0
50,0.09862694144248962,99,1.0
49,0.2384372055530548,99,0.8823529481887817
48,0.17555296421051025,99,0.9473684430122375
47,0.16324013471603394,99,1.0
46,0.1669486165046692,99,1.0
45,0.35301464796066284,99,0.8999999761581421
44,0.31195369362831116,99,0.9523809552192688
43,0.42186903953552246,99,0.8823529481887817
42,0.059812165796756744,99,0.949999988079071
41,0.31486380100250244,99,0.8947368264198303
40,0.357208251953125,99,0.9230769276618958
39,0.3834072947502136,99,0.8461538553237915
38,0.26170656085014343,99,0.8461538553237915
37,0.03665351867675781,99,1.0
36,0.5882760882377625,99,0.8500000238418579
35,0.1965879201889038,99,1.0
34,0.4812479019165039,99,0.8333333134651184
33,0.7063129544258118,99,0.8500000238418579
32,0.5363132357597351,99,0.8947368264198303
31,0.6552855372428894,99,0.7916666865348816
30,0.6168771982192993,99,0.7692307829856873
29,0.6708090305328369,99,0.8823529481887817
28,0.12238538265228271,99,1.0
27,0.7210091948509216,99,0.7692307829856873
26,0.5629405975341797,99,0.8125
25,0.42375800013542175,99,0.7692307829856873
24,0.6360242962837219,99,0.8095238208770752
```

Appendix

```
(venv) vaishnavi@DESKTOP-9V8KJG2:/mnt/c/Users/Mohit/Desktop/Gen AI/Week 8$ python3 Fine_Tune.py
Conversion successful! data.json has been converted to data_prepared.jsonl
File uploaded: file-gUpBN7H9NmgE06KPipPD1WYP
Fine-tuning job started: ftjob-5rrKtbTTUUiHnoVNd2SNyEmq
Fine-tuning in progress... (status: validating_files)
Fine-tuning in progress... (status: running)
Fine-tuning in progress... (status: running)
Fine-tuning in progress... (status: running)
Fine-tuning in progress... (status: running)
Fine-tuning in progress... (status: running)
Fine-tuning in progress... (status: running)
Fine-tuning completed successfully!
Metrics successfully saved to fine_tuning_metrics.csv
(venv) vaishnavi@DESKTOP-9V8KJG2:/mnt/c/Users/Mohit/Desktop/Gen AI/Week 8$
```