GPT-2

A Simple Guide to Fine-Tuning GPT-2

Hamza Fatnaoui

Introduction

In this guide, you will dive into the process of fine-tuning GPT-2 on your own dataset to use it for specific tasks. We will define GPT-2, explore how it works, and discuss the benefits of fine-tuning this model for various applications. By the end of this guide, you'll have a comprehensive understanding of how to tailor GPT-2 to meet your unique needs.

1-Overview of GPT-2

1. **Architecture**: GPT-2 is based on the Transformer architecture, which uses self-attention mechanisms to process input sequences efficiently.
2. **Pre-training**: The model is pre-trained on a diverse dataset containing a large portion of the internet's text, allowing it to learn a wide range of language patterns and knowledge.
3. **Generative Capabilities**: GPT-2 can generate human-like text based on a given prompt, making it useful for various applications, including content creation, chatbots, translation, and more.
4. **Fine-tuning**: take a look at the next

2-Benefits of Fine-Tuning GPT-2

Fine-tuning in deep learning is a form of transfer learning. It involves taking a pre-trained model, which has been trained on a large dataset for a general task such

as image recognition or natural language understanding, and making minor adjustments to its internal parameters. The goal is to optimize the model's performance on a new, related task without starting the training process from scratch.

Steps to Fine-Tune GPT-2

1. Preparing Your Dataset

1.1. Data Collection:

- Gather a dataset that is relevant to the specific task you want to fine-tune GPT-2 on. We are going to use The WikiQA corpus, which is a publicly available set of question and sentence pairs, collected and annotated for research on open-domain question answering. This is the link to the provided dataset on Hugging Face: WikiQA on Hugging Face.

1.2. Data Preprocessing:

- First we need to load the dataset using Hugging Face datasets library.

```
# Step 1: Install the Hugging Face datasets library
!pip install datasets

# Step 2: Load the dataset
from datasets import load_dataset

# Loading the 'glue' dataset
dataset = load_dataset("microsoft/wiki_qa")
```

- The data provided contains additional features, but we are only interested in the questions and the responses. We will extract these two features and store them in two files, `Q_A_trai.txt` and `Q_A_test.txt`, to fine-tune and test our model.

```
with open('Q_A_train.txt', 'w') as file:
    for row in dataset['train']:
        question = row['question']
        answer = row['answer']
        file.write(f"Question: {question}\nAnswer: {answer}\n\n")

with open('Q_A_test.txt', 'w') as file:
    for row in dataset['test']:
        question = row['question']
        answer = row['answer']
        file.write(f"Question: {question}\nAnswer: {answer}\n\n")
```

2. Setting Up the Environment

2.1. Install Required Libraries:

- Ensure you have the necessary libraries installed,

```
!pip install transformers
pip install accelerate -U
import pandas as pd
import numpy as np
import re
import os
```

pip install transformers: HuggingFace's Transformers library is a popular open-source library for natural language processing (NLP) tasks. It provides a unified API for various transformer models such as BERT, GPT-2

pip install accelerate -U: to make the training faster and more efficient.

2.2. Environment Setup:

- Using Google Colab, we faced countless session timeouts, so we switched to Lightning AI, which provides 22 free GPU hours monthly. You can try out the platform at Lightning AI.

3. Loading the Pre-trained GPT-2 Model

3.1. Code:

```
from transformers import GPT2LMHeadModel, GPT2Tokenizer
model = GPT2LMHeadModel.from_pretrained('gpt2')
tokenizer = GPT2Tokenizer.from_pretrained('gpt2')
```

GPT2LMHeadModel: loads the architecture and pre-trained parameters of the GPT-2 model, facilitating language modeling tasks by predicting the next word in a sequence based on context.

GPT2Tokenizer: prepares text inputs for the model by converting them into tokens

4.Preparing the Dataset

4.1. Loading Data:

- We have already loaded the dataset and converted it into a suitable format (questions and answers).

```
Question: how are glacier caves formed?
Answer: A partly submerged glacier cave on Perito Moreno Glacier .

Question: how are glacier caves formed?
Answer: The ice facade is approximately 60 m high

Question: how are glacier caves formed?
Answer: Ice formations in the Titlis glacier cave

Question: how are glacier caves formed?
Answer: A glacier cave is a cave formed within the ice of a glacier .
```

4.2. Tokenizing Data:

```
from transformers import TextDataset, DataCollatorForLanguageModeling
def load_dataset(file_path, tokenizer, block_size = 128):
    dataset = TextDataset(
        tokenizer = tokenizer,
        file_path = file_path,
        block_size = block_size,
    )
    return dataset
```

→Load_dataet function: tokenizes each piece of text using the tokenizer `GPT2Tokenizer` , and formats them into a dataset suitable for further processing or training in natural language processing tasks.

4.3. Handling the padding:

```
def load_data_collator(tokenizer, mlm = False):
    data_collator = DataCollatorForLanguageModeling(
        tokenizer=tokenizer,
        mlm=mlm,
    )
    return data_collator
```

→Load_data_collator: It takes care of the intricate details of data batching, padding

5. Training

5.1. Training function:

```
from transformers import Trainer, TrainingArguments
def train(train_file_path,model_name,
          output_dir,
          overwrite_output_dir,
          per_device_train_batch_size,
          num_train_epochs,
          save_steps):
  tokenizer = GPT2Tokenizer.from_pretrained(model_name)
  train_dataset = load_dataset(train_file_path, tokenizer)
  data_collator = load_data_collator(tokenizer)

  tokenizer.save_pretrained(output_dir)

  model = GPT2LMHeadModel.from_pretrained(model_name)

  model.save_pretrained(output_dir)

  training_args = TrainingArguments(
          output_dir=output_dir,
          overwrite_output_dir=overwrite_output_dir,
          per_device_train_batch_size=per_device_train_batch_size,
          num_train_epochs=num_train_epochs,
      )

  trainer = Trainer(
          model=model,
          args=training_args,
          data_collator=data_collator,
          train_dataset=train_dataset,
  )

  trainer.train()
  trainer.save_model()
```

→overwrite_output_dir: setting this variable to 'false' allows you to ensure that any existing models or checkpoints in the specified output directory are not overwritten. This way, you can keep multiple versions of your models for comparison or backup purposes.

→per_device_train_batch_size: This means the model will process 4 sentences (or training examples) in parallel on each device (GPU) during training. This helps to speed up the training process and make it more efficient.

→Epoch: An epoch is one complete pass through the entire training dataset. During one epoch, the model sees every training example exactly once.

→output_dir: where the checkpoint and the architecture is going to be stored

5.2. Training the model:

- Initilizing our variables

```
train_file_path = "Q_A_train.txt"
model_name = 'gpt2'
overwrite_output_dir = False
per_device_train_batch_size = 8
num_train_epochs = 50.0
save_steps = 50000
output_dir = 'Chat_Model/'
```

- training

```
train(
    train_file_path=train_file_path,
    model_name=model_name,
    output_dir=output_dir,
    overwrite_output_dir=overwrite_output_dir,
    per_device_train_batch_size=per_device_train_batch_size,
    num_train_epochs=num_train_epochs,
    save_steps=save_steps
)
```

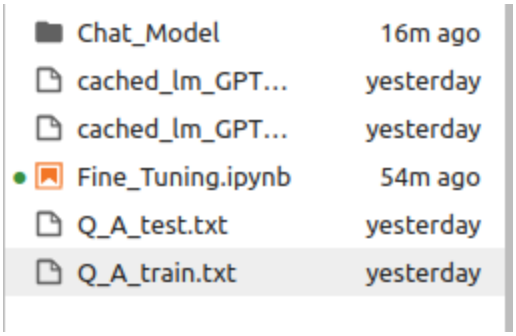
- noticing the epochs

Step	Training Loss
500	2.602000
1000	2.427600
1500	2.258600
2000	2.149300
2500	2.067000
3000	1.930000
3500	1.902700

- the completion of the fine tuning with training loss equal to 0.2

39500	0.248200
40000	0.248300
40500	0.246400
41000	0.242100
41500	0.242600
42000	0.240200
42500	0.241000
43000	0.239200

- So finally, your bar will look like this, containing these files



6. Inference

- Now, after fine tuning the model we are going to use these learned patterns to predict outcomes for new data.

6.1. code

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

```python
def load_model(model_path):
    model = GPT2LMHeadModel.from_pretrained(model_path)
    return model


def load_tokenizer(tokenizer_path):
    tokenizer = GPT2Tokenizer.from_pretrained(tokenizer_path)
    return tokenizer

def generate_text(model_path, sequence, max_length):

    model = load_model(model_path)
    tokenizer = load_tokenizer(model_path)
    ids = tokenizer.encode(f'{sequence}', return_tensors='pt')
    final_outputs = model.generate(
        ids,
        do_sample=True,
        max_length=max_length,
        pad_token_id=model.config.eos_token_id,
        top_k=50,
        top_p=0.95,
    )
    print(tokenizer.decode(final_outputs[0], skip_special_tokens=True))
```

→max_length: Once the sequence reaches this length, generation stops.

→pad_token_id: Specifies the token ID used for padding the sequence.

→sampling:

- **Greedy Decoding**: Always picking the most likely next word.
- **Sampling**: Randomly picking from a group of likely words

→top_k: Limits the sampling pool to the top `k` most likely next tokens.

→top_p: Enables nucleus sampling, also known as top-p sampling.

6.2. Example

- The example code with the generated output

```python
model_path = "Chat_Model"
sequence2 = "Question: how a water pump works"
max_len = 50
generate_text(model_path, sequence2, max_len)
```

```
Question: how a water pump works
Answer: Unlike some mechanical pump systems, like those for heavy machinery, a public water supply or desalination plant will discharge any substances that enter it, eliminating the need to remove water once it's in place
```

6.3 Your guide

- Here is the entire code in this file,

Fine_Tuning_GPT2.ipynb