Review Summarization using GPT2

Mahisha Ramesh
MT23121

Introduction:

Review summarization plays a crucial role in extracting the key information from lengthy reviews, aiding users in making informed decisions. In this report, we present our approach to review summarization using GPT-2 (Generative Pre-trained Transformer 2) on the Amazon Fine Food Reviews dataset. We detail the steps taken to preprocess the data, train the model, and evaluate its performance using ROUGE scores.

PART-1

1. Use the Amazon Fine Food Reviews dataset

```
[4]: data_subset.head()
```

[4]:

	Summary	Text
0	Good Quality Dog Food	I have bought several of the Vitality canned d...
1	Not as Advertised	Product arrived labeled as Jumbo Salted Peanut...
2	"Delight" says it all	This is a confection that has been around a fe...
3	Cough Medicine	If you are looking for the secret ingredient i...
4	Great taffy	Great taffy at a great price. There was a wid...

2. Clean and preprocess the 'Text' and 'Summary' column from the dataset.
Model Training

Data Preprocessing:

1. Handle Missing Values: Check for and handle missing values in the dataset's 'Text' and 'Summary' columns.
2. Text Cleaning:

- Remove HTML tags, special characters, punctuation, symbols, and extra spaces.
- Convert all text to lowercase for uniformity.
- Expand contractions and remove stopwords to improve tokenization accuracy.
- Apply lemmatization or stemming to reduce words to their base form.

```python
# Define a function for text preprocessing using spaCy
def preprocess_text(text):
    # Check if the input is a string
    if isinstance(text, str):
        # Convert text to lowercase
        text = text.lower()

        # Remove punctuation
        text = text.translate(str.maketrans('', '', string.punctuation))

        # Tokenize text
        doc = nlp(text)

        # Remove stopwords and lemmatize
        tokens = [token.lemma_ for token in doc if token.text.lower() not in STOP_WORDS]

        # Join tokens back into a string
        preprocessed_text = ' '.join(tokens)

        return preprocessed_text
    else:
        # If the input is not a string, return an empty string
        return ""
```

PART-2

1. Initialize a GPT-2 tokenizer and model from Hugging Face.

Tokenization: Use the GPT-2 tokenizer provided by the Hugging Face library to tokenize the text data.

[10]:

```
from transformers import TextDataset, DataCollatorForLanguageModeling
from transformers import GPT2Tokenizer, GPT2LMHeadModel
from transformers import Trainer, TrainingArguments
```

```
2024-04-23 04:42:50.042383: E external/local_xla/xla/stream_executor/cuda/cuda_dnn.cc:9261] Unable to register cuDNN factor
y: Attempting to register factory for plugin cuDNN when one has already been registered
2024-04-23 04:42:50.042517: E external/local_xla/xla/stream_executor/cuda/cuda_fft.cc:607] Unable to register cuFFT factory:
Attempting to register factory for plugin cuFFT when one has already been registered
2024-04-23 04:42:50.176016: E external/local_xla/xla/stream_executor/cuda/cuda_blas.cc:1515] Unable to register cuBLAS facto
ry: Attempting to register factory for plugin cuBLAS when one has already been registered
```

2. Divide the dataset into training and testing (75:25)

```
def split_csv(input_file, output_file1, output_file2, split_ratio=0.75):
    # Read CSV file
    with open(input_file, 'r', newline='') as csv_file:
        csv_reader = csv.reader(csv_file)
        header = next(csv_reader)  # Assuming the first row is header
        data = list(csv_reader)

    # Shuffle data randomly
    random.shuffle(data)

    # Calculate split indices
    split_index = int(len(data) * split_ratio)

    # Split data
    data1 = data[:split_index]
    data2 = data[split_index:]

    # Write to CSV files
    with open(output_file1, 'w', newline='') as csv_file1:
        csv_writer1 = csv.writer(csv_file1)
        csv_writer1.writerow(header)
        csv_writer1.writerows(data1)
```

3. Implement a custom dataset class to prepare the data for training.

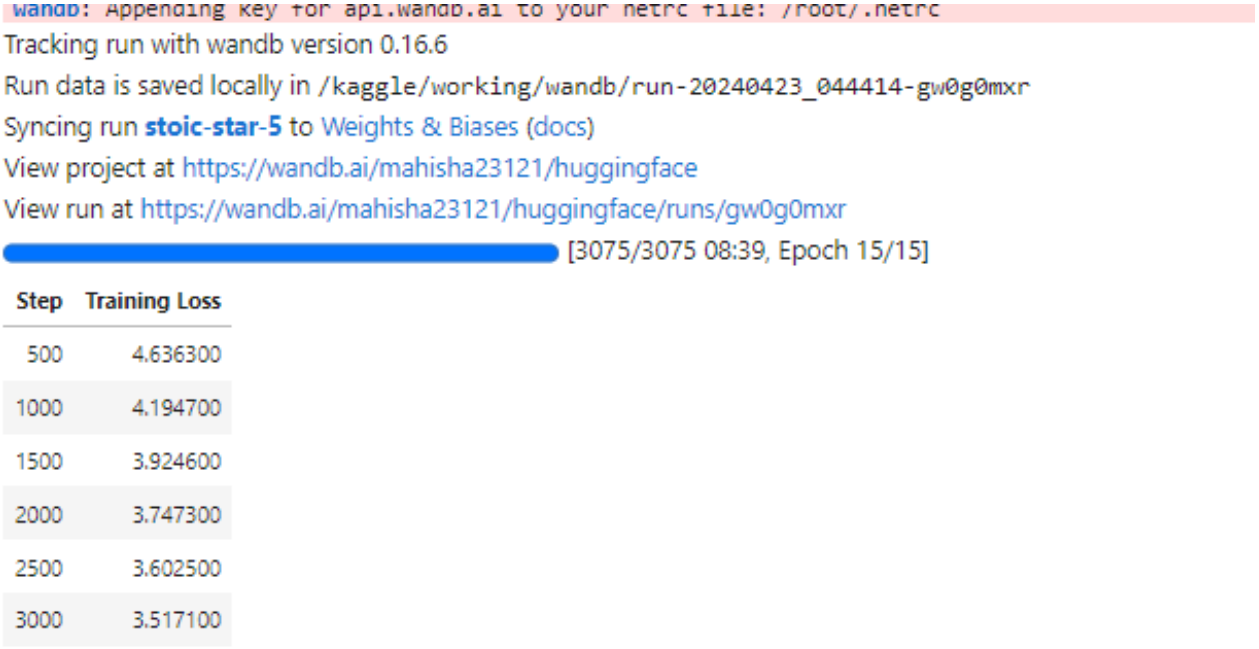
```
def train(train_file_path, model_name,
          output_dir,
          overwrite_output_dir,
          per_device_train_batch_size,
          num_train_epochs,
          save_steps):
    tokenizer = GPT2Tokenizer.from_pretrained(model_name)
    train_dataset = load_dataset(train_file_path, tokenizer)
    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,
    )
```

4. Fine-tune the GPT-2 model on the review dataset to generate summaries.



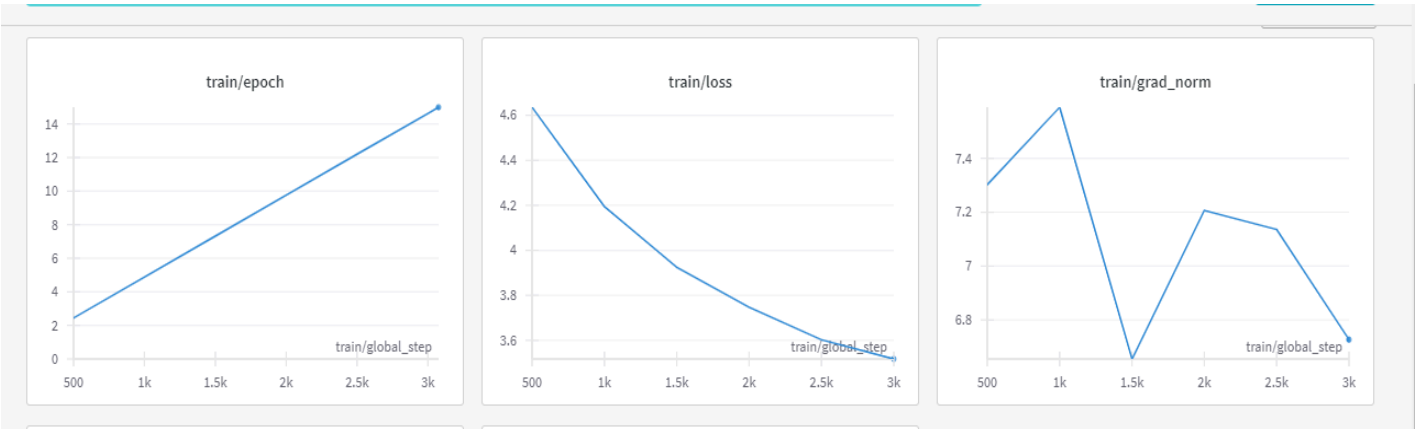
Step	Training Loss
500	4.636300
1000	4.194700
1500	3.924600
2000	3.747300
2500	3.602500
3000	3.517100

5. Experiment with different hyperparameters such as learning rate, batch size, and number of epochs to optimize the model's performance.

14]:

```
train_file_path = "/kaggle/working/reviews_train.txt"
model_name = 'gpt2'
output_dir = '/kaggle/working/result'
overwrite_output_dir = False
per_device_train_batch_size = 8
num_train_epochs = 15
save_steps = 500
```

Training and Validation Loss



Evaluation

After training, compute ROUGE scores on the test set to assess the model's overall performance i.e. compute ROUGE score for every predicted summary vs

the actual summary.

- Summaries Generation: Using the fine-tuned GPT-2 model, summaries were generated for review texts in the test set.
- ROUGE Score Calculation: ROUGE scores were computed by comparing the generated summaries with the actual summaries. ROUGE-1, ROUGE-2, and ROUGE-L scores were calculated to assess precision, recall, and F1-score.
- Example ROUGE Scores: For instance, the ROUGE scores for a given review text and its summary were as follows:

[25]:

```
from nltk.translate.bleu_score import corpus_bleu
from nltk.translate.bleu_score import sentence_bleu
from rouge import Rouge

# Example generated and reference texts
generated_text = "great kcup"
reference_text = "hot cocoa kcup"

# Initialize ROUGE
rouge = Rouge()

# Calculate ROUGE scores
scores = rouge.get_scores(generated_text, reference_text)

# Print ROUGE scores
print(scores)
```

```
[{'rouge-1': {'r': 0.3333333333333333, 'p': 0.5, 'f': 0.39999999520000007}, 'rouge-2': {'r': 0.0, 'p': 0.0, 'f': 0.0}, 'roug
e-l': {'r': 0.3333333333333333, 'p': 0.5, 'f': 0.39999999520000007}}]
```

+ Code + Markdown