Text Summarization with Open LLMs

Subrata Mondal

- **transformers:** a library that provides state-of-the-art natural language processing models and tools
- **datasets:** a library that provides access to various natural language processing datasets
- **rouge-score:** a library that implements the ROUGE metrics for evaluating text summarization systems

```
# Use the pip command to install the following libraries:
# Use the -q option to suppress the output messages
!pip install transformers datasets rouge-score -q
```

```
                         7.7/7.7 MB 25.5 MB/s eta 0:00:00
                         493.7/493.7 kB 34.0 MB/s eta 0:00:00
Preparing metadata (setup.py) ... done
                         302.0/302.0 kB 30.5 MB/s eta 0:00:00
                         3.8/3.8 MB 66.2 MB/s eta 0:00:00
                         1.3/1.3 MB 55.3 MB/s eta 0:00:00
                         115.3/115.3 kB 11.8 MB/s eta 0:00:00
                         134.8/134.8 kB 13.1 MB/s eta 0:00:00
                         295.0/295.0 kB 22.8 MB/s eta 0:00:00
Building wheel for rouge-score (setup.py) ... done
```

```
# Import the datasets library from Hugging Face
from datasets import load_dataset

# Import the torch library for deep learning
import torch

# Check if a GPU is available and set the device accordingly
device = "cuda" if torch.cuda.is_available() else "cpu"
# Print the device name
print(device)
```

```
cuda
```

Load Dataset

- Use the `load_dataset` function to load the CNN/Daily Mail dataset. This dataset contains news articles and their summaries

```
# Use the version 3.0.0 of the dataset, which has better quality and consistency
dataset = load_dataset("cnn_dailymail", version="3.0.0")
```

```
Downloading builder script:   0%|          | 0.00/8.33k [00:00<?, ?B/s]

Downloading metadata:   0%|          | 0.00/9.88k [00:00<?, ?B/s]

Downloading readme:   0%|          | 0.00/15.1k [00:00<?, ?B/s]

Downloading data files:   0%|          | 0/5 [00:00<?, ?it/s]

Downloading data:   0%|          | 0.00/159M [00:00<?, ?B/s]

Downloading data:   0%|          | 0.00/376M [00:00<?, ?B/s]

Downloading data:   0%|          | 0.00/12.3M [00:00<?, ?B/s]

Downloading data:   0%|          | 0.00/661k [00:00<?, ?B/s]

Downloading data:   0%|          | 0.00/572k [00:00<?, ?B/s]

Generating train split: 0 examples [00:00, ? examples/s]

Generating validation split: 0 examples [00:00, ? examples/s]

Generating test split: 0 examples [00:00, ? examples/s]
```

```
print(f"Features: {dataset['train'].column_names}")
```

```
Features: ['article', 'highlights', 'id']
```

The dataset has three columns: * **article:** which contains the news articles * **highlights:** which contains the summaries, and * **id** to uniquely identify each article

```python
sample = dataset["train"][1]
print(f"""
Article (excerpt of 500 characters, total length: {len(sample["article"])}):
""")
print(sample["article"][:500])
print(f'\nSummary (length: {len(sample["highlights"])}):')
print(sample["highlights"])
```

```
Article (excerpt of 500 characters, total length: 4051):

Editor's note: In our Behind the Scenes series, CNN correspondents share their experiences i

Summary (length: 281):
Mentally ill inmates in Miami are housed on the "forgotten floor"
Judge Steven Leifman says most are there as a result of "avoidable felonies"
While CNN tours facility, patient shouts: "I am the son of the president"
Leifman says the system is unjust and he's fighting for change .
```

Some articles are very long and we want to make them shorter. For example, one article is 14 times longer than the summary we want. This is hard for some models that use transformers. Transformers can only read a small part of the text at a time, like a few paragraphs. A simple but not very good way to make summaries is to cut off the text after the part that the model can read. But this might leave out some important things for the summary that are at the end of the text. We have to accept this problem for now because of how the models are built.

Summarization Pipeline

We want to see how good some transformer models are at making summaries. We will look at the summaries they make for the example we saw before. The models we will look at are different and can read different amounts of text. But we will only give them the same amount of text, which is 2,000 letters, so we can compare their summaries better.

```python
sample_text = dataset["train"][1]["article"][:2000]
print(sample_text)
```

```
Editor's note: In our Behind the Scenes series, CNN correspondents share their experiences i
```

```
# We'll collect the generated summaries of each model in a dictionary
summaries = {}
```

A convention in summarization is to separate the summary sentences by a newline.

```
import nltk
from nltk.tokenize import sent_tokenize
nltk.download("punkt")
```

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt.zip.
```

```
True
```

```
string = "The U.S. are a country. The U.N. is an organization."
sent_tokenize(string)
```

```
['The U.S. are a country.', 'The U.N. is an organization.']
```

GPT2

GPT2 LLM is a model that can perform text summarization and other natural language processing tasks. Text summarization is the task of creating a short and concise summary of a longer text. GPT2 LLM stands for generative pre-trained transformer 2 language model, which describes its neural network architecture and pre-training strategy.

- GPT2 LLM uses a standard sequence-to-sequence model with a decoder only. The decoder is a transformer, which is a neural network that can process long sequences of text using attention mechanisms. The decoder generates text one word at a time, using the previous words and the input text as input.
- GPT2 LLM is pre-trained on a large corpus of text using a language modeling objective. This means that it learns to predict the next word in a sequence of words, given the previous words. This helps GPT2 LLM to learn general language skills and to generate fluent and coherent texts.
- GPT2 LLM is particularly effective when fine-tuned for text generation tasks, such as summarization, translation, or dialogue. It can match or surpass the performance of other models, such as BERT or T5, on various benchmarks, such as GLUE, SQuAD, XSum, CNN/Daily Mail, ELI5, or ConvAI2.

If you want to learn more about GPT2 LLM and how to use it for text summarization, you can check out some of these resources:

- Generating Text Summaries Using GPT-2 on PyTorch - Paperspace Blog: A blog post that shows how to use GPT2 LLM to generate summaries for different texts using PyTorch.
- How to Use LLMs to Generate Concise Summaries - Medium: A blog post that shows how to use GPT2 LLM to generate summaries for different texts using Google Cloud Platform.
- Language Models are Unsupervised Multitask Learners: A research paper that introduces the GPT2 LLM and its variants and shows its capabilities in various natural language processing tasks.

```
# Import the pipeline and set_seed functions from the transformers library
from transformers import pipeline, set_seed
```

- The pipeline function creates a pipeline object that can perform various natural language processing tasks, such as text generation, summarization, translation, etc.
- The set_seed function sets the random seed for the pipeline object, which makes the results reproducible
- The **gpt2-xl model** is a large and powerful model that can generate fluent and coherent texts
- **TL;DR** stands for Too Long; Didn't Read, and it is a way of asking for a short summary of a long text
- The **pipe** object will return a list of dictionaries, each containing a generated text and its score

```
# Set the random seed for the pipeline object to 42, which makes the results reproducible
set_seed(42)

# Create a pipeline object that can perform text generation using the gpt2-xl model
pipe = pipeline("text-generation", model="gpt2-xl")

# Define a query for the text generation task, which consists of a sample text and a TL;DR
gpt2_query = sample_text + "\nTL;DR:\n"

# Use the pipe object to generate a text from the query, with a maximum length of 512 toke
pipe_out = pipe(gpt2_query, max_length=512, clean_up_tokenization_spaces=True)
```

```
# Extract the generated text from the first dictionary in the list and remove the query pa
# Use the sent_tokenize function to split the generated text into sentences and join them
# Store the result in a dictionary called summaries, with the key "gpt2"
summaries["gpt2"] = "\n".join(sent_tokenize(pipe_out[0]["generated_text"][len(gpt2_query)
```

```
Downloading (…)lve/main/config.json:   0%|          | 0.00/689 [00:00<?, ?B/s]
```

```
Downloading model.safetensors:   0%|          | 0.00/6.43G [00:00<?, ?B/s]
```

```
Downloading (…)neration_config.json:   0%|          | 0.00/124 [00:00<?, ?B/s]
```

```
Downloading (…)olve/main/vocab.json:   0%|          | 0.00/1.04M [00:00<?, ?B/s]
```

```
Downloading (…)olve/main/merges.txt:   0%|          | 0.00/456k [00:00<?, ?B/s]
```

```
Downloading (…)/main/tokenizer.json:   0%|          | 0.00/1.36M [00:00<?, ?B/s]
```

```
Setting `pad_token_id` to `eos_token_id`:50256 for open-end generation.
```

Here we just store the summaries of the generated text by slicing off the input query and keep the result in a Python dictionary for later comparison.

```
print(summaries)
```

```
{'gpt2': "I'm not an expert on mental illness and would be happy to learn of an expert who ca
```

T5

T5 LLM is a model that can perform text summarization and other natural language processing tasks. Text summarization is the task of creating a short and concise summary of a longer text. T5 LLM stands for text-to-text transfer transformer language model, which describes its neural network architecture and pre-training strategy.

- T5 LLM uses a standard sequence-to-sequence model with an encoder and a decoder. The encoder and the decoder are both transformers, which are neural networks that can process long sequences of text using attention mechanisms. The encoder converts the input text into a sequence of hidden states, and the decoder generates the output text from the hidden states.

- T5 LLM is pre-trained on a large corpus of text using a text-to-text objective. This means that it learns to convert any input text into any output text, such as summarization, translation, question-answering, etc. This helps T5 LLM to learn general language skills and to handle various tasks with the same model.
- T5 LLM is particularly effective when fine-tuned for text generation tasks, such as summarization, translation, or dialogue. It can match or surpass the performance of other models, such as BERT or GPT-2, on various benchmarks, such as GLUE, SQuAD, XSum, CNN/Daily Mail, ELI5, or ConvAI2.
- The **t5-large model** is a large and powerful model that can generate summaries for various texts
- The **pipe** object will return a list of dictionaries, each containing a summary text and its score

```
# Import the pipeline function from the transformers library
from transformers import pipeline
# Create a pipeline object that can perform summarization using the t5-large model
pipe = pipeline("summarization", model="t5-large")

# Use the pipe object to generate a summary for the sample text
pipe_out = pipe(sample_text)

# Extract the summary text from the first dictionary in the list
# Use the sent_tokenize function to split the summary text into sentences and join them wi
# Store the result in a dictionary called summaries, with the key "t5"
summaries["t5"] = "\n".join(sent_tokenize(pipe_out[0]["summary_text"]))
```

```
Downloading (…)lve/main/config.json:   0%|          | 0.00/1.21k [00:00<?, ?B/s]

Downloading model.safetensors:   0%|          | 0.00/2.95G [00:00<?, ?B/s]

Downloading (…)neration_config.json:   0%|          | 0.00/147 [00:00<?, ?B/s]

Downloading (…)ve/main/spiece.model:   0%|          | 0.00/792k [00:00<?, ?B/s]

Downloading (…)/main/tokenizer.json:   0%|          | 0.00/1.39M [00:00<?, ?B/s]
```

```
/usr/local/lib/python3.10/dist-packages/transformers/models/t5/tokenization_t5_fast.py:158: I
For now, this behavior is kept to avoid breaking backwards compatibility when padding/encodin
- Be aware that you SHOULD NOT rely on t5-large automatically truncating your input to 512 wh
- If you want to encode/pad to sequences longer than 512 you can either instantiate this toke
- To avoid this warning, please instantiate this tokenizer with `model_max_length` set to you
  warnings.warn(
```

BART

BART LLM is a model that can perform text summarization and other natural language processing tasks. Text summarization is the task of creating a short and concise summary of a longer text. BART LLM stands for bidirectional and autoregressive transformer language model, which describes its neural network architecture and pre-training strategy.

- BART LLM uses a standard sequence-to-sequence model with an encoder and a decoder. The encoder is bidirectional, which means it can use the context from both sides of a word. The decoder is autoregressive, which means it generates text one word at a time, using the previous words as input.
- BART LLM is pre-trained on a large corpus of text using a denoising objective. This means that it learns to reconstruct corrupted texts by applying various transformations, such as masking, deleting, or permuting words or sentences. This helps BART LLM to learn general language skills and to handle noisy or incomplete inputs.
- BART LLM is particularly effective when fine-tuned for text generation tasks, such as summarization, translation, or dialogue. It can match or surpass the performance of other models, such as RoBERTa or GPT-2, on various benchmarks, such as GLUE, SQuAD, XSum, CNN/Daily Mail, ELI5, or ConvAI2.
- The **facebook/bart-large-cnn model** is a large and powerful model that can generate summaries for various texts
- The **pipe** object will return a list of dictionaries, each containing a summary text and its score

```python
# Import the pipeline function from the transformers library
from transformers import pipeline
# Create a pipeline object that can perform summarization using the facebook/bart-large-cn
pipe = pipeline("summarization", model="facebook/bart-large-cnn")

# Use the pipe object to generate a summary for the sample text
pipe_out = pipe(sample_text)

# Extract the summary text from the first dictionary in the list
# Use the sent_tokenize function to split the summary text into sentences and join them wi
# Store the result in a dictionary called summaries, with the key "bart"
summaries["bart"] = "\n".join(sent_tokenize(pipe_out[0]["summary_text"]))
```

```
Downloading (…)lve/main/config.json:   0%|          | 0.00/1.58k [00:00<?, ?B/s]

Downloading pytorch_model.bin:   0%|          | 0.00/1.63G [00:00<?, ?B/s]

Downloading (…)neration_config.json:   0%|          | 0.00/363 [00:00<?, ?B/s]

Downloading (…)olve/main/vocab.json:   0%|          | 0.00/899k [00:00<?, ?B/s]

Downloading (…)olve/main/merges.txt:   0%|          | 0.00/456k [00:00<?, ?B/s]

Downloading (…)/main/tokenizer.json:   0%|          | 0.00/1.36M [00:00<?, ?B/s]
```

Comparing Different Summaries

Now that we have generated summaries with four different models, let's compare the results. Keep in mind that one model has not been trained on the dataset at all (GPT-2), one model has been fine-tuned on this task among others (T5), and two models have exclusively been fine-tuned on this task (BART and PEGASUS).

```
print("GROUND TRUTH")
print(dataset["train"][1]["highlights"])
print("")

for model_name in summaries:
  print(model_name.upper())
  print(summaries[model_name])
  print("")
```

```
GROUND TRUTH
Mentally ill inmates in Miami are housed on the "forgotten floor"
Judge Steven Leifman says most are there as a result of "avoidable felonies"
While CNN tours facility, patient shouts: "I am the son of the president"
Leifman says the system is unjust and he's fighting for change .
```

```
GPT2
I'm not an expert on mental illness and would be happy to learn of an expert who can vouch fo
Update #2 : The article has now been updated, please use the links below.
Corrections to earlier sections in the report: I was wrong on some minor details regarding ho
I corrected those errors in

T5
mentally ill inmates are housed on the ninth floor of a florida jail .
most face drug charges or charges of assaulting an officer .
judge says arrests often result from confrontations with police .
one-third of all people in Miami-dade county jails are mental ill .

BART
Mentally ill inmates are housed on the "forgotten floor" of Miami-Dade jail.
Most often, they face drug charges or charges of assaulting an officer.
Judge Steven Leifman says the arrests often result from confrontations with police.
He says about one-third of all people in the county jails are mentally ill.
```

The first thing we notice by looking at the model outputs is that the summary generated by GPT-2 is quite different from the others. Instead of giving a summary of the text, it summarizes the characters. Often the GPT-2 model “hallucinates” or invents facts, since it was not explicitly trained to generate truthful summaries. For example, at the time of writing, Nesta is not the fastest man in the world, but sits in ninth place.

Measuring the Quality of Generated Text

Good evaluation metrics are important, since we use them to measure the performance of models not only when we train them but also later, in production. If we have bad metrics we might be blind to model degradation, and if they are misaligned with the business goals we might not create any value.

ROUGE

ROUGE metrics are a way of measuring how good a text summarization system is. Text summarization is the task of creating a short and concise summary of a longer text. ROUGE stands for Recall-Oriented Understudy for Gisting Evaluation, and it compares the summary produced by a system (called the candidate) with one or more summaries written by humans (called the references).

- ROUGE metrics use both recall and precision to evaluate the quality of summaries. Recall measures how much of the important information in the reference summaries is captured by the candidate summary. Precision measures how much of the information in the candidate summary is relevant and needed.

- ROUGE metrics can use different levels of granularity to compare the summaries, such as words, n-grams, sentences, or longest common subsequence. For example, ROUGE-1 uses unigrams (single words), ROUGE-2 uses bigrams (two-word phrases), and ROUGE-L uses the longest common subsequence (the longest sequence of words that appear in both summaries in the same order).
- ROUGE metrics can also use different variants to adjust for the length of the summaries, such as ROUGE-SU4, which uses skip-bigrams with a maximum gap of four words, or ROUGE-Lsum, which gives more weight to longer summaries.
- ROUGE metrics are widely used in natural language processing research and applications, such as text summarization and machine translation. They are easy to implement and interpret, and they can provide a quantitative measure of the performance of different systems.
- The **evaluate library** provides a unified API for loading and using various evaluation metrics, such as ROUGE, BLEU, etc.

```
# Use the pip command to install the evaluate library from Hugging Face
# Use the -q option to suppress the output messages
!pip install evaluate -q
```

```
                                0.0/84.1 kB ? eta -:--:--                              84.1/84.1 kB
```

- The **ROUGE metric** is a set of metrics for evaluating automatic summarization and machine translation systems

```
# Import the evaluate library from Hugging Face
import evaluate
# Use the load function to load the ROUGE metric from the Hugging Face Hub
rouge_metric = evaluate.load("rouge")
# Use the description attribute to get a brief description of the metric
rouge_metric.description
```

```
'ROUGE, or Recall-Oriented Understudy for Gisting Evaluation, is a set of metrics and a softw
```

```
# Import the pandas library for data analysis
import pandas as pd
```

```
# Get the reference summary from the dataset
reference = dataset["train"][1]["highlights"]
# Create an empty list to store the records of the ROUGE scores
records = []

# Define a list of the ROUGE metrics to use
rouge_names = ["rouge1", "rouge2", "rougeL", "rougeLsum"]
# Loop through the summaries dictionary, which contains the summaries generated by differe
for model_name in summaries:
  print(model_name)
  # Add the prediction and reference summary to the rouge_metric object, which computes th
  rouge_metric.add(prediction=summaries[model_name], reference=reference)
  # Compute and get the ROUGE scores as a dictionary
  score = rouge_metric.compute()
  # Create a dictionary that contains only the ROUGE scores for each metric, without the p
  rouge_dict = dict((rn, score[rn]) for rn in rouge_names)
  # Append the rouge_dict to the records list
  records.append(rouge_dict)

# Create a pandas DataFrame from the records list, using the model names as the index
pd.DataFrame.from_records(records, index=summaries.keys())
```

```
gpt2
t5
bart
```

	rouge1	rouge2	rougeL	rougeLsum
gpt2	0.188034	0.017391	0.102564	0.188034
t5	0.382979	0.130435	0.255319	0.382979
bart	0.475248	0.222222	0.316832	0.415842

Final Thoughts

Best Model - BART

	rouge1	rouge2	rougeL	rougeLsum
gpt2	0.188034	0.017391	0.102564	0.188034
t5	0.382979	0.130435	0.255319	0.382979

	rouge1	rouge2	rougeL	rougeLsum
bart	0.475248	0.222222	0.316832	0.415842

Based on the ROUGE metrics, the best text summarization model to choose is **BART**. BART has the highest scores for ROUGE-1, ROUGE-2, and ROUGE-L, which means it has the best performance in terms of matching the n-grams, the longest common subsequence, and the summary length of the reference summary. BART also has a high score for ROUGE-Lsum, which is a variant of ROUGE-L that gives more weight to longer summaries.

Some key insights from the results are:

- **GPT-2** has the lowest scores for all metrics, which means it has the worst performance in terms of summarizing the text. GPT-2 is a generative model that can produce fluent and coherent texts, but it is not specifically trained for summarization tasks. Therefore, it may not capture the main points or the structure of the original text.
- **T5** has better scores than GPT-2, but lower scores than BART. T5 is a text-to-text model that can perform various natural language processing tasks, including summarization. T5 is trained on a large corpus of text and can generate summaries that are relevant and concise. However, T5 may not be able to handle complex or long texts as well as BART.
- **BART** has the best scores for all metrics, which means it has the best performance in terms of summarizing the text. BART is a denoising autoencoder model that can reconstruct corrupted texts. BART is trained on a large corpus of text and can generate summaries that are accurate and informative. BART can also handle complex or long texts better than T5 or GPT-2.
