Text Summarization using BART Model

ODII ODEH

In this project, we explore text summarization using the BART (Bidirectional and Auto-Regressive Transformers) model, a state-of-the-art sequence-to-sequence model developed by Facebook AI. We utilise the CNN-DailyMail dataset, which contains news articles paired with human-generated summaries.

Text summarization is the process of condensing large volumes of text into shorter, concise summaries while preserving the most important information. In this project, we leverage the BART (Bidirectional and Auto-Regressive Transformers) model, developed by Facebook AI, for text summarization tasks.

Using the CNN-DailyMail dataset, which consists of news articles paired with human-generated summaries, we train the BART model to generate abstractive summaries of input news articles. We fine-tune the model using the Hugging Face Transformers library and evaluate its performance using the ROUGE metric.

The trained model is capable of generating informative and concise summaries for news articles, providing a valuable tool for handling large volumes of text efficiently.

The project consists of the following steps:

1. Data Preparation
2. Model Training
3. Evaluation
4. Inference

The first thing we did it in the code is importing required libraries as we can see here in the code

**import pandas as pd**

**from io import StringIO**

**from transformers import (AutoModelForSeq2SeqLM,**

**AutoTokenizer,**

**DataCollatorForSeq2Seq,**

**TrainingArguments,**

**Trainer,**

**IntervalStrategy,**

**EarlyStoppingCallback,**

**)**

**from datasets import Dataset, DatasetDict, load\_metric**

**import torch**

**import nltk**

| Library | Used for |
| --- | --- |
| pandas | For data manipulation. |
| StringIO | For creating in-memory file objects. |
| transformers | For accessing pre-trained BART model and tokenizer. |
| Dataset, DatasetDict, load\_metric | For managing datasets and loading evaluation metrics |
| torch | For PyTorch operations |
| nltk | For text processing. |

**Downloading NLTK Data**

**nltk.download("punkt", quiet=True)**

### **Load Pre-trained Model and Tokenizer:**

cheakPoint = "facebook/bart-large-cnn"

model = AutoModelForSeq2SeqLM.from\_pretrained(cheakPoint).to(device)

tokenizer = AutoTokenizer.from\_pretrained(cheakPoint)

Explanation:

* checkPoint: The pre-trained BART model checkpoint to be used. Here, we are using the "facebook/bart-large-cnn" model.
* AutoModelForSeq2SeqLM : Loads the pre-trained BART model for sequence-to-sequence tasks.
* AutoTokenizer : Loads the pre-trained BART tokenizer.

### **Load and Prepare Data:**

# Load data

valid\_lines = []

with open("train.csv", "r", encoding="utf-8") as file:

for line in file:

try:

pd.read\_csv(StringIO(line))

valid\_lines.append(line)

except pd.errors.ParserError:

continue

train\_data = pd.read\_csv(StringIO('\n'.join(valid\_lines)))

train\_data.dropna(inplace=True)

Here what i did is to load the data , handling parsing errors. And drop the missing values

### **Preprocess Data:**

train\_dataset = Dataset.from\_pandas(train\_data)

train\_dataset = train\_dataset.shuffle(seed=42)

train, val = train\_dataset.select(range(400)), train\_dataset.select(range(400, 490))

dataset\_dict = DatasetDict({"train": train, "validation": val})

dataset\_dict.remove\_columns("id")

This process i convert the dataframe to a dataset object and after that i shuffled the dataset and split it into training and validation sets and last thing a removed the id col

### **Tokenization and Preprocessing:**

encoder\_max\_length = 1024

decoder\_max\_length = 128

def batch\_tokenize\_preprocess(batch, tokenizer, encoder\_max\_length, decoder\_max\_length):

source, target = batch["article"], batch["highlights"]

source\_tokenized = tokenizer(source, padding="max\_length", truncation=True, max\_length=encoder\_max\_length )

target\_tokenized = tokenizer(target, padding="max\_length", truncation=True, max\_length=decoder\_max\_length)

# Ignore padding in the loss

target\_labels = [

[-100 if token == tokenizer.pad\_token\_id else token for token in l]

for l in target\_tokenized["input\_ids"]

]

# Create a dictionary for the batch

batch\_dict = {

"input\_ids": source\_tokenized["input\_ids"],

"attention\_mask": source\_tokenized["attention\_mask"],

"labels": target\_labels,

}

return batch\_dict

train\_data = train.map(

lambda batch: batch\_tokenize\_preprocess(batch, tokenizer, encoder\_max\_length, decoder\_max\_length),

batched=True,

remove\_columns=train.column\_names,

)

validation\_data = val.map(

lambda batch: batch\_tokenize\_preprocess(batch, tokenizer, encoder\_max\_length, decoder\_max\_length),

batched=True,

remove\_columns=val.column\_names,

)

Here i defined a function called batch\_tokenize\_preprocess to tokenize and preprocess the data for training and validation , after word i Tokenize the input articles and target highlights using the tokenizer. Pad and truncate the sequences to specified maximum lengths.

### **Define Evaluation Metrics:**

def compute\_metrics(eval\_preds): preds, labels = eval\_preds if isinstance(preds, tuple): preds = preds[0] decoded\_preds = [tokenizer.batch\_decode(np.argmax(pred, axis=1), skip\_special\_tokens=True) for pred in preds] labels = np.where(labels != -100, labels, tokenizer.pad\_token\_id) decoded\_labels = [tokenizer.batch\_decode(label, skip\_special\_tokens=True) for label in labels] result = metric.compute(predictions=decoded\_preds, references=decoded\_labels, use\_stemmer=True) result = {key: value.mid.fmeasure \* 100 for key, value in result.items()} prediction\_lens = [ np.count\_nonzero(pred != tokenizer.pad\_token\_id) for pred in preds ] result["gen\_len"] = np.mean(prediction\_lens) result = { k: round(v, 4) for k, v in result.items() } return result

Here i Define a function compute\_metrics to compute evaluation metrics (ROUGE score) for the model.

### **Data Collator and Training Arguments :**

data\_collator = DataCollatorForSeq2Seq(tokenizer, model=model)

batch\_size = 4

training\_args = TrainingArguments(

output\_dir='bart\_CNN\_NLP',

num\_train\_epochs=4,

per\_device\_train\_batch\_size=batch\_size,

per\_device\_eval\_batch\_size=batch\_size,

warmup\_steps=500,

weight\_decay=0.1,

label\_smoothing\_factor=0.1,

logging\_dir="bart\_logs",

logging\_steps=20,

load\_best\_model\_at\_end=True,

evaluation\_strategy = "steps",

eval\_steps = 40,

save\_steps=1e6,

)

DataCollatorForSeq2Seq is a Data collator for sequence-to-sequence tasks.

TrainingArguments Define training arguments such as output directory, number of epochs, batch size, etc.

### **Define Trainer and Train the Model:**

trainer = Trainer(

model=model,

args=training\_args,

data\_collator=data\_collator,

train\_dataset=train\_data,

eval\_dataset=validation\_data,

tokenizer=tokenizer,

compute\_metrics=compute\_metrics,

callbacks = [EarlyStoppingCallback(early\_stopping\_patience=3)]

)

trainer.train()

Trainer: Initialize a trainer object for training the model.

### **Inference:**

def generate\_summary(test\_samples, model, max\_length):

inputs = tokenizer(

test\_samples,

padding=True,

truncation=True,

max\_length=max\_length,

return\_tensors="pt",

)

input\_ids = inputs.input\_ids.to(model.device)

attention\_mask = inputs.attention\_mask.to(model.device)

outputs = model.generate(input\_ids, attention\_mask=attention\_mask)

output\_str = tokenizer.batch\_decode(outputs, skip\_special\_tokens=True)

return output\_str

sample = "The tower is 324 metres (1,063 ft) tall, about the same height as an 81-storey building, and the tallest structure in Paris. Its base is square, measuring 125 metres (410 ft) on each side. During its construction, the Eiffel Tower surpassed the Washington Monument to become the tallest man-made structure in the world, a title it held for 41 years until the Chrysler Building in New York City was finished in 1930. It was the first structure to reach a height of 300 metres. Due to the addition of a broadcasting aerial at the top of the tower in 1957, it is now taller than the Chrysler Building by 5.2 metres (17 ft). Excluding transmitters, the Eiffel Tower is the second tallest free-standing structure in France after the Millau Viaduct."

res = generate\_summary(sample, trainer.model, max\_length=1028)

print(res)

Last this i Define a function generate\_summary to generate summaries using the trained model.

And Use the trained model to generate summaries for sample articles.