```
import re
import random
import pandas as pd
import numpy as np
import torch
from torch.utils.data import Dataset, DataLoader
from transformers import AutoTokenizer, AutoModelWithLMHead
import torch.optim as optim
from nltk.translate.bleu_score import sentence_bleu
from rouge import Rouge
from tqdm import tqdm
```

```
device = torch.device('cuda:0' if torch.cuda.is_available() else 'cpu')
device
```

```
device(type='cuda', index=0)
```

```
reviews = pd.read_csv( 'C:/Users/28360/Downloads/gpt3_result.csv')
reviews['training'] = reviews['cleaned_lyrics'] + " TL; DR " + reviews['gpt_summaries']
reviews = reviews['training'].to_list()
```

```
len(reviews)
```

```
6679
```

```
max_length = 1000
```

```
tokenizer = AutoTokenizer.from_pretrained("gpt2")
model = AutoModelWithLMHead.from_pretrained("gpt2")
```

```
c:\Users\28360\AppData\Local\Programs\Python\Python312\Lib\site-packages\transformers\models\auto\modeling_auto.py:1581: FutureWarnin
  warnings.warn(
```

```
model = model.to(device)
model
```

```
GPT2LMHeadModel(
  (transformer): GPT2Model(
    (wte): Embedding(50257, 768)
    (wpe): Embedding(1024, 768)
    (drop): Dropout(p=0.1, inplace=False)
    (h): ModuleList(
      (0-11): 12 x GPT2Block(
        (ln_1): LayerNorm((768,), eps=1e-05, elementwise_affine=True)
        (attn): GPT2Attention(
          (c_attn): Conv1D()
          (c_proj): Conv1D()
          (attn_dropout): Dropout(p=0.1, inplace=False)
          (resid_dropout): Dropout(p=0.1, inplace=False)
        )
        (ln_2): LayerNorm((768,), eps=1e-05, elementwise_affine=True)
        (mlp): GPT2MLP(
          (c_fc): Conv1D()
          (c_proj): Conv1D()
          (act): NewGELUActivation()
          (dropout): Dropout(p=0.1, inplace=False)
        )
      )
    )
    (ln_f): LayerNorm((768,), eps=1e-05, elementwise_affine=True)
  )
  (lm_head): Linear(in_features=768, out_features=50257, bias=False)
)
```

```
optimizer = optim.AdamW(model.parameters(), lr=3e-4)
```

```
tokenizer.encode(" TL;DR ")
```

```
[24811, 26, 7707, 220]
```

```
extra_length = len(tokenizer.encode(" TL;DR "))
```

```
def calculate_rouge_scores(reference, hypothesis):
    rouge = Rouge()
    scores = rouge.get_scores(hypothesis, reference)[0]
    score1, score2, scorel = scores['rouge-1']['p'], scores['rouge-2']['p'], scores['rouge-l']['p']
    return score1, score2, scorel
```

```
# Define a function to calculate BLEU score
def calculate_bleu_score(reference, hypothesis):
    return sentence_bleu([reference], hypothesis)
```

```
class ReviewDataset(Dataset):
    def __init__(self, tokenizer, reviews, max_len):
        self.max_len = max_len
        self.tokenizer = tokenizer
        self.eos = self.tokenizer.eos_token
        self.eos_id = self.tokenizer.eos_token_id
        self.reviews = reviews
        self.result = []

        for review in self.reviews:
            # Encode the text using tokenizer.encode(). We add EOS at the end
            tokenized = self.tokenizer.encode(review + self.eos)

            # Padding/truncating the encoded sequence to max_len
            padded = self.pad_truncate(tokenized)

            # Creating a tensor and adding to the result
            self.result.append(torch.tensor(padded))

    def __len__(self):
        return len(self.result)


    def __getitem__(self, item):
        return self.result[item]

    def pad_truncate(self, name):
        name_length = len(name) - extra_length
        if name_length < self.max_len:
            difference = self.max_len - name_length
            result = name + [self.eos_id] * difference
        elif name_length > self.max_len:
            result = name[:self.max_len + 3]+[self.eos_id]
        else:
            result = name
        return result
```

```
dataset = ReviewDataset(tokenizer, reviews, max_length)
```

```
    Token indices sequence length is longer than the specified maximum sequence length for this model (1049 > 1024). Running this sequenc
```

```
dataloader = DataLoader(dataset, batch_size=5, shuffle=True, drop_last=True)
```

```
len(dataset)
```

```
    6679
```

```
def train_model(model, optimizer, dl, epochs):
    for epoch in range(epochs):
        epoch_loss = 0.0
        total_bleu_score = 0.0

        with tqdm(total=len(dl), desc=f'Epoch {epoch + 1}/{epochs}', unit='batch') as pbar:
            for idx, batch in enumerate(dl):
                with torch.set_grad_enabled(True):
                    optimizer.zero_grad()
                    batch = batch.to(device)
                    output = model(batch, labels=batch)
                    loss = output[0]
                    loss.backward()
                    optimizer.step()

                    # Calculate BLEU score for the first example in the batch
                    reference_summary = reviews[idx]
                    generated_ids = torch.argmax(output.logits, dim=-1)
                    generated_summary = tokenizer.decode(generated_ids[0].tolist(), skip_special_tokens=True)


                    # bleu_score = calculate_bleu_score(reference_summary, generated_summary)
                    bleu_score = 0

                    epoch_loss += loss.item()
                    total_bleu_score += bleu_score
                    pbar.set_postfix({'Loss': loss.item(), 'Bleu': bleu_score})
                    pbar.update(1)

        # Print the average loss for the epoch
        avg_epoch_loss = epoch_loss / len(dl)
        avg_bleu_score = total_bleu_score / len(dl)
        print(f"Epoch {epoch + 1}/{epochs}, Average Loss: {avg_epoch_loss}, Average Bleu Score: {avg_bleu_score}")
```

```
train_model(model=model, optimizer=optimizer, dl=dataloader, epochs=10)
```

```
Epoch 1/10: 100%|██████████| 1335/1335 [22:35<00:00,  1.02s/batch, Loss=2.13, Bleu=0]
Epoch 1/10, Average Loss: 2.2048028082436835, Average Bleu Score: 0.0
```

```
Epoch 2/10: 100%|██████████| 1335/1335 [26:24<00:00,  1.19s/batch, Loss=1.79, Bleu=0]
Epoch 2/10, Average Loss: 2.0267192449462548, Average Bleu Score: 0.0
Epoch 3/10: 100%|██████████| 1335/1335 [25:52<00:00,  1.16s/batch, Loss=2.66, Bleu=0]
Epoch 3/10, Average Loss: 1.8995960906650242, Average Bleu Score: 0.0
Epoch 4/10: 100%|██████████| 1335/1335 [23:27<00:00,  1.05s/batch, Loss=2, Bleu=0]
Epoch 4/10, Average Loss: 1.759524801473939, Average Bleu Score: 0.0
Epoch 5/10: 100%|██████████| 1335/1335 [25:27<00:00,  1.14s/batch, Loss=1.61, Bleu=0]
Epoch 5/10, Average Loss: 1.5875817490427682, Average Bleu Score: 0.0
Epoch 6/10: 100%|██████████| 1335/1335 [26:54<00:00,  1.21s/batch, Loss=1.25, Bleu=0]
Epoch 6/10, Average Loss: 1.3811730338839556, Average Bleu Score: 0.0
Epoch 7/10: 100%|██████████| 1335/1335 [29:43<00:00,  1.34s/batch, Loss=0.862, Bleu=0]
Epoch 7/10, Average Loss: 1.1454374385237247, Average Bleu Score: 0.0
Epoch 8/10: 100%|██████████| 1335/1335 [31:46<00:00,  1.43s/batch, Loss=0.995, Bleu=0]
Epoch 8/10, Average Loss: 0.8990422778584984, Average Bleu Score: 0.0
Epoch 9/10: 100%|██████████| 1335/1335 [24:55<00:00,  1.12s/batch, Loss=0.6, Bleu=0]
Epoch 9/10, Average Loss: 0.6670633290367627, Average Bleu Score: 0.0
Epoch 10/10: 100%|██████████| 1335/1335 [22:54<00:00,  1.03s/batch, Loss=0.707, Bleu=0]Epoch 10/10, Average Loss: 0.47405111833234853
```

```
path = 'C:/Users/28360/Downloads/model/GPT2_summary_model.pt'

torch.save(model.state_dict(), path)
```