training

October 31, 2023

[1]:

```python
import torch
from torch.utils.data import DataLoader, Dataset
from transformers import T5Tokenizer, T5ForConditionalGeneration, AdamW
```

[2]:

```python
# Sample data
data = [
    {
```

```
        "question": "What is small intestine cancer?",
        "answer": "Key Points - Small intestine cancer is a rare disease in␣
 ↪which malignant (cancer) cells form in the tissues of the small intestine.  ␣
 ↪ - There are five types of small intestine cancer.    - Diet and health␣
 ↪history can affect the risk of developing small intestine cancer.    - Signs␣
 ↪and symptoms of small intestine cancer include unexplained weight loss and␣
 ↪abdominal pain.    - Tests that examine the small intestine are used to␣
 ↪detect (find), diagnose, and stage small intestine cancer.    - Certain␣
 ↪factors affect prognosis (chance of recovery) and treatment options. Small␣
 ↪intestine cancer is a rare disease in which malignant (cancer) cells form in␣
 ↪the tissues of the small intestine. The small intestine is part of the bodys␣
 ↪digestive system, which also includes the esophagus, stomach, and large␣
 ↪intestine. The digestive system removes and processes nutrients (vitamins,␣
 ↪minerals, carbohydrates, fats, proteins, and water) from foods and helps␣
 ↪pass waste material out of the body. The small intestine is a long tube that␣
 ↪connects the stomach to the large intestine. It folds many times to fit␣
 ↪inside the abdomen. There are five types of small intestine cancer. The␣
 ↪types of cancer found in the small intestine are adenocarcinoma, sarcoma,␣
 ↪carcinoid tumors, gastrointestinal stromal tumor, and lymphoma. This summary␣
 ↪discusses adenocarcinoma and leiomyosarcoma (a type of sarcoma).   ␣
 ↪Adenocarcinoma starts in glandular cells in the lining of the small␣
 ↪intestine and is the most common type of small intestine cancer. Most of␣
 ↪these tumors occur in the part of the small intestine near the stomach. They␣
 ↪may grow and block the intestine.   Leiomyosarcoma starts in the smooth␣
 ↪muscle cells of the small intestine. Most of these tumors occur in the part␣
 ↪of the small intestine near the large intestine.    See the following PDQ␣
 ↪summaries for more information on small intestine cancer:          -  Adult␣
 ↪Soft Tissue Sarcoma Treatment     -  Childhood Soft Tissue Sarcoma Treatment␣
 ↪    -  Adult Non-Hodgkin Lymphoma Treatment     -  Childhood Non-Hodgkin␣
 ↪Lymphoma Treatment     -  Gastrointestinal Carcinoid Tumors Treatment     - ␣
 ↪Gastrointestinal Stromal Tumors Treatment"
    },
    # Add more data points as needed
]
```

[26]:

```
!pip install sentencepiece
```

```
Defaulting to user installation because normal site-packages is not writeable
Collecting sentencepiece
  Downloading
sentencepiece-0.1.99-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.whl
(1.3 MB)
                           1.3/1.3 MB
36.4 MB/s eta 0:00:0000:01
Installing collected packages: sentencepiece
Successfully installed sentencepiece-0.1.99
```

[3]:

```python
# Initialize tokenizer and model
tokenizer = T5Tokenizer.from_pretrained('t5-small')
model = T5ForConditionalGeneration.from_pretrained('t5-small')
```

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

Downloading (…)okenizer_config.json:   0%|          | 0.00/2.32k [00:00<?, ?B/s]

You are using the default legacy behaviour of the <class
'transformers.models.t5.tokenization_t5.T5Tokenizer'>. If you see this, DO NOT
PANIC! This is expected, and simply means that the `legacy` (previous) behavior
will be used so nothing changes for you. If you want to use the new behaviour,
set `legacy=False`. This should only be set if you understand what it means, and
thouroughly read the reason why this was added as explained in
https://github.com/huggingface/transformers/pull/24565

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

Downloading model.safetensors:   0%|          | 0.00/242M [00:00<?, ?B/s]

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

[4]:

```python
# Preprocess data and create dataset
class QADataset(Dataset):
    def __init__(self, data, tokenizer, max_length=512):
        self.data = data
        self.tokenizer = tokenizer
        self.max_length = max_length

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

    def __getitem__(self, idx):
        context = self.data[idx]["context"]
        question = self.data[idx]["question"]
        answer = self.data[idx]["answer"]

        input_text = f"context: {context} question: {question}"
        target_text = answer

        input_ids = self.tokenizer.encode(input_text, add_special_tokens=True,␣
 ↪max_length=self.max_length, padding='max_length', truncation=True,␣
 ↪return_tensors='pt')
        target_ids = self.tokenizer.encode(target_text,␣
 ↪add_special_tokens=True, max_length=self.max_length, padding='max_length',␣
 ↪truncation=True, return_tensors='pt')

        return {
            'input_ids': input_ids.squeeze(),
            'attention_mask': input_ids.squeeze().gt(0),
            'labels': target_ids.squeeze()
        }
```

[5]:

```
dataset = QADataset(data, tokenizer)
train_loader = DataLoader(dataset, batch_size=2, shuffle=True)
```

[6]:

```
# Initialize optimizer
optimizer = AdamW(model.parameters(), lr=1e-4)

# Training loop
num_epochs = 5
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
model.to(device)
model.train()
```

```
/opt/miniconda3/lib/python3.10/site-packages/transformers/optimization.py:411:
FutureWarning: This implementation of AdamW is deprecated and will be removed in
a future version. Use the PyTorch implementation torch.optim.AdamW instead, or
set `no_deprecation_warning=True` to disable this warning
  warnings.warn(
```

[6]:

```
T5ForConditionalGeneration(
  (shared): Embedding(32128, 512)
  (encoder): T5Stack(
    (embed_tokens): Embedding(32128, 512)
    (block): ModuleList(
      (0): T5Block(
        (layer): ModuleList(
          (0): T5LayerSelfAttention(
            (SelfAttention): T5Attention(
              (q): Linear(in_features=512, out_features=512, bias=False)
              (k): Linear(in_features=512, out_features=512, bias=False)
              (v): Linear(in_features=512, out_features=512, bias=False)
              (o): Linear(in_features=512, out_features=512, bias=False)
              (relative_attention_bias): Embedding(32, 8)
            )
            (layer_norm): T5LayerNorm()
            (dropout): Dropout(p=0.1, inplace=False)
          )
          (1): T5LayerFF(
            (DenseReluDense): T5DenseActDense(
              (wi): Linear(in_features=512, out_features=2048, bias=False)
              (wo): Linear(in_features=2048, out_features=512, bias=False)
              (dropout): Dropout(p=0.1, inplace=False)
              (act): ReLU()
            )
            (layer_norm): T5LayerNorm()
            (dropout): Dropout(p=0.1, inplace=False)
          )
        )
      )
      (1-5): 5 x T5Block(
        (layer): ModuleList(
          (0): T5LayerSelfAttention(
            (SelfAttention): T5Attention(
              (q): Linear(in_features=512, out_features=512, bias=False)
              (k): Linear(in_features=512, out_features=512, bias=False)
              (v): Linear(in_features=512, out_features=512, bias=False)
              (o): Linear(in_features=512, out_features=512, bias=False)
            )
            (layer_norm): T5LayerNorm()
            (dropout): Dropout(p=0.1, inplace=False)
          )
          (1): T5LayerFF(
            (DenseReluDense): T5DenseActDense(
              (wi): Linear(in_features=512, out_features=2048, bias=False)
              (wo): Linear(in_features=2048, out_features=512, bias=False)
              (dropout): Dropout(p=0.1, inplace=False)
              (act): ReLU()
            )
            (layer_norm): T5LayerNorm()
            (dropout): Dropout(p=0.1, inplace=False)
          )
        )
      )
    )
    (final_layer_norm): T5LayerNorm()
    (dropout): Dropout(p=0.1, inplace=False)
  )
  (decoder): T5Stack(
    (embed_tokens): Embedding(32128, 512)
    (block): ModuleList(
      (0): T5Block(
        (layer): ModuleList(
          (0): T5LayerSelfAttention(
            (SelfAttention): T5Attention(
              (q): Linear(in_features=512, out_features=512, bias=False)
              (k): Linear(in_features=512, out_features=512, bias=False)
              (v): Linear(in_features=512, out_features=512, bias=False)
              (o): Linear(in_features=512, out_features=512, bias=False)
              (relative_attention_bias): Embedding(32, 8)
            )
            (layer_norm): T5LayerNorm()
            (dropout): Dropout(p=0.1, inplace=False)
          )
          (1): T5LayerCrossAttention(
            (EncDecAttention): T5Attention(
              (q): Linear(in_features=512, out_features=512, bias=False)
              (k): Linear(in_features=512, out_features=512, bias=False)
              (v): Linear(in_features=512, out_features=512, bias=False)
              (o): Linear(in_features=512, out_features=512, bias=False)
            )
            (layer_norm): T5LayerNorm()
            (dropout): Dropout(p=0.1, inplace=False)
          )
          (2): T5LayerFF(
            (DenseReluDense): T5DenseActDense(
              (wi): Linear(in_features=512, out_features=2048, bias=False)
              (wo): Linear(in_features=2048, out_features=512, bias=False)
              (dropout): Dropout(p=0.1, inplace=False)
              (act): ReLU()
            )
            (layer_norm): T5LayerNorm()
            (dropout): Dropout(p=0.1, inplace=False)
          )
        )
      )
      (1-5): 5 x T5Block(
        (layer): ModuleList(
          (0): T5LayerSelfAttention(
            (SelfAttention): T5Attention(
              (q): Linear(in_features=512, out_features=512, bias=False)
              (k): Linear(in_features=512, out_features=512, bias=False)
              (v): Linear(in_features=512, out_features=512, bias=False)
              (o): Linear(in_features=512, out_features=512, bias=False)
            )
            (layer_norm): T5LayerNorm()
            (dropout): Dropout(p=0.1, inplace=False)
          )
          (1): T5LayerCrossAttention(
            (EncDecAttention): T5Attention(
              (q): Linear(in_features=512, out_features=512, bias=False)
              (k): Linear(in_features=512, out_features=512, bias=False)
              (v): Linear(in_features=512, out_features=512, bias=False)
              (o): Linear(in_features=512, out_features=512, bias=False)
            )
            (layer_norm): T5LayerNorm()
            (dropout): Dropout(p=0.1, inplace=False)
          )
          (2): T5LayerFF(
            (DenseReluDense): T5DenseActDense(
              (wi): Linear(in_features=512, out_features=2048, bias=False)
              (wo): Linear(in_features=2048, out_features=512, bias=False)
              (dropout): Dropout(p=0.1, inplace=False)
              (act): ReLU()
            )
            (layer_norm): T5LayerNorm()
            (dropout): Dropout(p=0.1, inplace=False)
          )
        )
      )
    )
    (final_layer_norm): T5LayerNorm()
    (dropout): Dropout(p=0.1, inplace=False)
  )
  (lm_head): Linear(in_features=512, out_features=32128, bias=False)
)
```

[7]:
```
for epoch in range(num_epochs):
    total_loss = 0
    for batch in train_loader:
        input_ids = batch['input_ids'].to(device)
        attention_mask = batch['attention_mask'].to(device)
        labels = batch['labels'].to(device)

        outputs = model(input_ids=input_ids, attention_mask=attention_mask,␣
 ↪labels=labels)
        loss = outputs.loss
        total_loss += loss.item()

        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

    average_loss = total_loss / len(train_loader)
    print(f'Epoch {epoch + 1}/{num_epochs}, Loss: {average_loss:.4f}')
```

```
Epoch 1/5, Loss: 10.3419
Epoch 2/5, Loss: 7.3311
Epoch 3/5, Loss: 4.7834
Epoch 4/5, Loss: 3.1484
Epoch 5/5, Loss: 3.0300
```

[8]:
```
# Save the trained model
model.save_pretrained('generative_qa_model')
tokenizer.save_pretrained('generative_qa_model')
```

[8]:

```
('generative_qa_model/tokenizer_config.json',
 'generative_qa_model/special_tokens_map.json',
 'generative_qa_model/spiece.model',
 'generative_qa_model/added_tokens.json')
```

[11]:

```
import torch
from transformers import T5Tokenizer, T5ForConditionalGeneration

# Load tokenizer and model
tokenizer = T5Tokenizer.from_pretrained('generative_qa_model')
model = T5ForConditionalGeneration.from_pretrained('generative_qa_model')

def generate_answer(context, question):
    # Split the input context into chunks of maximum sequence length
    max_seq_length = tokenizer.model_max_length
    chunks = [context[i:i+max_seq_length] for i in range(0, len(context),␣
 ↪max_seq_length)]

    # Generate answers for each chunk
    generated_answers = []
    for chunk in chunks:
        input_text = f"context: {chunk} question: {question}"
        input_ids = tokenizer.encode(input_text, return_tensors="pt",␣
 ↪max_length=max_seq_length, truncation=True)

        # Generate answer
        with torch.no_grad():
            output_ids = model.generate(input_ids, max_length=50,␣
 ↪num_return_sequences=1)

        # Decode and add the answer to the list
        answer = tokenizer.decode(output_ids[0], skip_special_tokens=True)
        generated_answers.append(answer)

    # Concatenate answers from chunks
    final_answer = " ".join(generated_answers)
    return final_answer

# Example context and question
```

```
question = "What is small intestine cancer?"

# Generate answer
answer = generate_answer(context, question)
print("Generated Answer:", answer)
```

```
Generated Answer: small intestine cancer is a long tube that connects the
stomach to the large intestine
```

[ ]:

```

```

[1]:

```
import pandas as pd
from sklearn.model_selection import train_test_split
import torch
from torch.utils.data import DataLoader, Dataset
from transformers import T5Tokenizer, T5ForConditionalGeneration, AdamW
```

[2]:

```
# Load data from CSV file
df = pd.read_csv('output_file.csv')
df = df.dropna(subset=['Answers'])
df = df.dropna(subset=['Contexts'])
df = df.dropna(subset=['Questions'])
df = df.head(10000)

# Split the dataset into train and test sets
train_df, test_df = train_test_split(df, test_size=0.2, random_state=42)
```

[3]:

```
df
```

[3]:

```
                                                Contexts  \
0      The small intestine is part of the\n\t\t  body…
1      The small intestine is part of the\n\t\t  body…
2      The small intestine is part of the\n\t\t  body…
3      The small intestine is part of the\n\t\t  body…
4      The small intestine is part of the\n\t\t  body…
…                                                    …
34547  MORE INFORMATION In the United States, more th…
34548  MORE INFORMATION In the United States, more th…
34549  MORE INFORMATION In the United States, more th…
34550  MORE INFORMATION In the United States, more th…
34551  MORE INFORMATION In the United States, more th…

                                               Questions  \
0                 What is (are) Small Intestine Cancer ?
1            Who is at risk for Small Intestine Cancer? ?
2      What are the symptoms of Small Intestine Cancer ?
3                How to diagnose Small Intestine Cancer ?
```

```
4        What is the outlook for Small Intestine Cancer ?
…                                                  …
34547    Who is at risk for Peripheral Artery Disease? ?
34548  What are the symptoms of Peripheral Artery Dis…
34549        How to diagnose Peripheral Artery Disease ?
34550  What are the treatments for Peripheral Artery …
34551         How to prevent Peripheral Artery Disease ?

                                                 Answers
0      Key Points\n                   - Small intest…
1      Diet and health history can affect the risk of…
2      Signs and symptoms of small intestine cancer i…
3      Tests that examine the small intestine are use…
4      Certain factors affect prognosis (chance of re…
…                                                  …
34547  Peripheral artery disease (P.A.D.) affects mil…
34548  Many people who have peripheral artery disease…
34549  Peripheral artery disease (P.A.D.) is diagnose…
34550  Treatments for peripheral artery disease (P.A…
34551  Taking action to control your risk factors can…

[8808 rows x 3 columns]
```

[4]:

```python
# Initialize tokenizer and model
tokenizer = T5Tokenizer.from_pretrained('t5-small')
model = T5ForConditionalGeneration.from_pretrained('t5-small')
```

```
You are using the default legacy behaviour of the <class
'transformers.models.t5.tokenization_t5.T5Tokenizer'>. If you see this, DO NOT
PANIC! This is expected, and simply means that the `legacy` (previous) behavior
will be used so nothing changes for you. If you want to use the new behaviour,
set `legacy=False`. This should only be set if you understand what it means, and
thouroughly read the reason why this was added as explained in
https://github.com/huggingface/transformers/pull/24565
```

[5]:

```python
class QADataset(Dataset):
    def __init__(self, data, tokenizer, max_length=512):
        self.data = data.dropna(subset=['Answers'])
        self.tokenizer = tokenizer
        self.max_length = max_length

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

    def __getitem__(self, idx):
        context = self.data['Contexts'].iloc[idx]
        question = self.data['Questions'].iloc[idx]
        answer = self.data['Answers'].iloc[idx]

        input_text = f"context: {context} question: {question}"
        target_text = answer

        # Encode input and target texts without any maximum length limit
        input_ids = self.tokenizer.encode(input_text, add_special_tokens=True,␣
 ↪return_tensors='pt')
        target_ids = self.tokenizer.encode(target_text,␣
 ↪add_special_tokens=True, return_tensors='pt')

        return {
            'input_ids': input_ids.squeeze(),
            'attention_mask': input_ids.squeeze().gt(0),
            'labels': target_ids.squeeze()
        }
```

[6]:

```
from torch.nn.utils.rnn import pad_sequence

def collate_fn(batch):
    input_ids = [item['input_ids'] for item in batch]
    attention_masks = [item['attention_mask'] for item in batch]
    labels = [item['labels'] for item in batch]

    # Pad input_ids, attention_masks, and labels to the same length
    padded_input_ids = pad_sequence(input_ids, batch_first=True,␣
 ↪padding_value=tokenizer.pad_token_id)
    padded_attention_masks = pad_sequence(attention_masks, batch_first=True,␣
 ↪padding_value=0)  # Assuming 0 for padding token
    padded_labels = pad_sequence(labels, batch_first=True,␣
 ↪padding_value=tokenizer.pad_token_id)

    return {
        'input_ids': padded_input_ids,
        'attention_mask': padded_attention_masks,
        'labels': padded_labels
    }
```

[7]:

```
train_dataset = QADataset(train_df, tokenizer)
test_dataset = QADataset(test_df, tokenizer)
batch_size = 1
train_loader = DataLoader(train_dataset, batch_size=batch_size, shuffle=True,␣
 ↪collate_fn=collate_fn)
test_loader = DataLoader(test_dataset, batch_size=batch_size,␣
 ↪collate_fn=collate_fn)
```

[8]:

```python
# Initialize optimizer
optimizer = AdamW(model.parameters(), lr=1e-4)

# Training loop
num_epochs = 50
device = torch.device('cuda:1' if torch.cuda.is_available() else 'cpu')
model.to(device)
model.train()
```

```
/opt/miniconda3/lib/python3.10/site-packages/transformers/optimization.py:411:
FutureWarning: This implementation of AdamW is deprecated and will be removed in
a future version. Use the PyTorch implementation torch.optim.AdamW instead, or
set `no_deprecation_warning=True` to disable this warning
  warnings.warn(
```

[8]:

```
T5ForConditionalGeneration(
  (shared): Embedding(32128, 512)
  (encoder): T5Stack(
    (embed_tokens): Embedding(32128, 512)
    (block): ModuleList(
      (0): T5Block(
        (layer): ModuleList(
          (0): T5LayerSelfAttention(
            (SelfAttention): T5Attention(
              (q): Linear(in_features=512, out_features=512, bias=False)
              (k): Linear(in_features=512, out_features=512, bias=False)
              (v): Linear(in_features=512, out_features=512, bias=False)
              (o): Linear(in_features=512, out_features=512, bias=False)
              (relative_attention_bias): Embedding(32, 8)
            )
            (layer_norm): T5LayerNorm()
            (dropout): Dropout(p=0.1, inplace=False)
          )
          (1): T5LayerFF(
            (DenseReluDense): T5DenseActDense(
              (wi): Linear(in_features=512, out_features=2048, bias=False)
              (wo): Linear(in_features=2048, out_features=512, bias=False)
              (dropout): Dropout(p=0.1, inplace=False)
              (act): ReLU()
            )
            (layer_norm): T5LayerNorm()
            (dropout): Dropout(p=0.1, inplace=False)
          )
        )
      )
      (1-5): 5 x T5Block(
        (layer): ModuleList(
          (0): T5LayerSelfAttention(
            (SelfAttention): T5Attention(
              (q): Linear(in_features=512, out_features=512, bias=False)
              (k): Linear(in_features=512, out_features=512, bias=False)
              (v): Linear(in_features=512, out_features=512, bias=False)
              (o): Linear(in_features=512, out_features=512, bias=False)
            )
            (layer_norm): T5LayerNorm()
            (dropout): Dropout(p=0.1, inplace=False)
          )
          (1): T5LayerFF(
            (DenseReluDense): T5DenseActDense(
              (wi): Linear(in_features=512, out_features=2048, bias=False)
              (wo): Linear(in_features=2048, out_features=512, bias=False)
              (dropout): Dropout(p=0.1, inplace=False)
              (act): ReLU()
            )
            (layer_norm): T5LayerNorm()
            (dropout): Dropout(p=0.1, inplace=False)
          )
        )
      )
    )
    (final_layer_norm): T5LayerNorm()
    (dropout): Dropout(p=0.1, inplace=False)
  )
  (decoder): T5Stack(
    (embed_tokens): Embedding(32128, 512)
    (block): ModuleList(
      (0): T5Block(
        (layer): ModuleList(
          (0): T5LayerSelfAttention(
            (SelfAttention): T5Attention(
              (q): Linear(in_features=512, out_features=512, bias=False)
              (k): Linear(in_features=512, out_features=512, bias=False)
              (v): Linear(in_features=512, out_features=512, bias=False)
              (o): Linear(in_features=512, out_features=512, bias=False)
              (relative_attention_bias): Embedding(32, 8)
            )
            (layer_norm): T5LayerNorm()
            (dropout): Dropout(p=0.1, inplace=False)
          )
          (1): T5LayerCrossAttention(
            (EncDecAttention): T5Attention(
              (q): Linear(in_features=512, out_features=512, bias=False)
              (k): Linear(in_features=512, out_features=512, bias=False)
              (v): Linear(in_features=512, out_features=512, bias=False)
```

```
              (o): Linear(in_features=512, out_features=512, bias=False)
            )
            (layer_norm): T5LayerNorm()
            (dropout): Dropout(p=0.1, inplace=False)
          )
          (2): T5LayerFF(
            (DenseReluDense): T5DenseActDense(
              (wi): Linear(in_features=512, out_features=2048, bias=False)
              (wo): Linear(in_features=2048, out_features=512, bias=False)
              (dropout): Dropout(p=0.1, inplace=False)
              (act): ReLU()
            )
            (layer_norm): T5LayerNorm()
            (dropout): Dropout(p=0.1, inplace=False)
          )
        )
      )
      (1-5): 5 x T5Block(
        (layer): ModuleList(
          (0): T5LayerSelfAttention(
            (SelfAttention): T5Attention(
              (q): Linear(in_features=512, out_features=512, bias=False)
              (k): Linear(in_features=512, out_features=512, bias=False)
              (v): Linear(in_features=512, out_features=512, bias=False)
              (o): Linear(in_features=512, out_features=512, bias=False)
            )
            (layer_norm): T5LayerNorm()
            (dropout): Dropout(p=0.1, inplace=False)
          )
          (1): T5LayerCrossAttention(
            (EncDecAttention): T5Attention(
              (q): Linear(in_features=512, out_features=512, bias=False)
              (k): Linear(in_features=512, out_features=512, bias=False)
              (v): Linear(in_features=512, out_features=512, bias=False)
              (o): Linear(in_features=512, out_features=512, bias=False)
            )
            (layer_norm): T5LayerNorm()
            (dropout): Dropout(p=0.1, inplace=False)
          )
          (2): T5LayerFF(
            (DenseReluDense): T5DenseActDense(
              (wi): Linear(in_features=512, out_features=2048, bias=False)
              (wo): Linear(in_features=2048, out_features=512, bias=False)
              (dropout): Dropout(p=0.1, inplace=False)
              (act): ReLU()
            )
            (layer_norm): T5LayerNorm()
            (dropout): Dropout(p=0.1, inplace=False)
          )
        )
      )
    )
    (final_layer_norm): T5LayerNorm()
    (dropout): Dropout(p=0.1, inplace=False)
  )
  (lm_head): Linear(in_features=512, out_features=32128, bias=False)
)
```

[9]:

```
!nvidia-smi
```

```
Tue Oct 31 15:58:07 2023
+-------------------------------------------------------------------------------
--------+
| NVIDIA-SMI 535.86.10              Driver Version: 535.86.10    CUDA Version:
12.2     |
|-----------------------------------------+----------------------+--------------
--------+
| GPU  Name                 Persistence-M | Bus-Id        Disp.A | Volatile
Uncorr. ECC |
| Fan  Temp   Perf          Pwr:Usage/Cap |         Memory-Usage | GPU-Util
Compute M. |
|                                         |                      |
MIG M. |
|=========================================+======================+==============
========|
|   0  Quadro RTX 8000                On  | 00000000:25:00.0 Off |
0 |
| N/A   39C    P0              59W / 250W |    505MiB / 46080MiB |      0%
Default |
|                                         |                      |
N/A |
+-----------------------------------------+----------------------+--------------
--------+
|   1  Quadro RTX 8000                On  | 00000000:81:00.0 Off |
0 |
| N/A   47C    P0              61W / 250W |    403MiB / 46080MiB |      0%
Default |
|                                         |                      |
N/A |
+-----------------------------------------+----------------------+--------------
--------+
|   2  Quadro RTX 8000                On  | 00000000:E2:00.0 Off |
0 |
| N/A   34C    P8              14W / 250W |      3MiB / 46080MiB |      0%
```

```
Default |
|                                         |                      |
N/A |
+-----------------------------------------+----------------------+--------------
--------+

+-------------------------------------------------------------------------------
--------+
| Processes:
|
|  GPU   GI   CI        PID   Type   Process name                           GPU
Memory |
|        ID   ID
Usage      |
|===============================================================================
========|
|    0   N/A  N/A    444072      C   /opt/miniconda3/bin/python
250MiB |
|    0   N/A  N/A    450552      C   /opt/miniconda3/bin/python
252MiB |
|    1   N/A  N/A    864480      C   /opt/miniconda3/bin/python
400MiB |
+-------------------------------------------------------------------------------
--------+
```

[10]:

```
device
```

[10]:

```
device(type='cuda', index=1)
```

[11]:

```python
from tqdm import tqdm

# Define the number of epochs
num_epochs = 10

# Create tqdm progress bars for training and testing loops
train_bar = tqdm(total=len(train_loader), desc='Training', position=0)
test_bar = tqdm(total=len(test_loader), desc='Testing', position=0)

for epoch in range(num_epochs):
    total_loss = 0
    train_bar.n = 0
    train_bar.last_print_n = 0
    train_bar.reset()

    for batch in train_loader:
        input_ids = batch['input_ids'].to(device)
        attention_mask = batch['attention_mask'].to(device)
        labels = batch['labels'].to(device)
        # Truncate input sequences to fit within the model's maximum sequence␣
↪length
        max_length = 512
        input_ids = input_ids[:, :max_length]
        attention_mask = attention_mask[:, :max_length]
        labels = labels[:, :max_length]
        outputs = model(input_ids=input_ids, attention_mask=attention_mask,␣
↪labels=labels)

        loss = outputs.loss
        total_loss += loss.item()

        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

        train_bar.set_postfix({'Epoch': epoch + 1, 'Loss': total_loss /␣
↪(train_bar.last_print_n + 1)})
        train_bar.update()

    train_bar.set_postfix({'Epoch': epoch + 1, 'Loss': total_loss /␣
↪len(train_loader)})
    train_bar.update()

    # Evaluate on test data
    model.eval()
    total_test_loss = 0
    test_bar.n = 0
    test_bar.last_print_n = 0
    test_bar.reset()

    with torch.no_grad():
        for batch in test_loader:
            input_ids = batch['input_ids'].to(device)
            attention_mask = batch['attention_mask'].to(device)
            labels = batch['labels'].to(device)

            outputs = model(input_ids=input_ids, attention_mask=attention_mask,␣
↪labels=labels)
            loss = outputs.loss
            total_test_loss += loss.item()

            test_bar.set_postfix({'Epoch': epoch + 1, 'Test Loss':␣
↪total_test_loss / (test_bar.last_print_n + 1)})
            test_bar.update()
    test_bar.set_postfix({'Epoch': epoch + 1, 'Test Loss': total_test_loss /␣
 ↪len(test_loader)})
    test_bar.update()

    model.train()

# Close the tqdm progress bars
train_bar.close()
test_bar.close()
```

```
Training:   0%|          | 0/7046 [00:00<?, ?it/s]Token indices sequence length
is longer than the specified maximum sequence length for this model (899 > 512).
Running this sequence through the model will result in indexing errors
Training: 7047it [07:01, 16.71it/s, Epoch=10, Loss=1.05]t/s, Epoch=10, Test
Loss=0.888]
Testing: 1763it [01:02, 28.35it/s, Epoch=10, Test Loss=0.888]
```

[ ]:

```
for epoch in range(num_epochs):
    total_loss = 0
    for batch in train_loader:
        input_ids = batch['input_ids'].to(device)
        attention_mask = batch['attention_mask'].to(device)
        labels = batch['labels'].to(device)
        # Truncate input sequences to fit within the model's maximum sequence␣
 ↪length
        max_length = 512
        input_ids = input_ids[:, :max_length]
        attention_mask = attention_mask[:, :max_length]
        labels = labels[:, :max_length]
        outputs = model(input_ids=input_ids, attention_mask=attention_mask,␣
 ↪labels=labels)

        loss = outputs.loss
        total_loss += loss.item()

        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

    average_loss = total_loss / len(train_loader)
    print(f'Training Epoch {epoch + 1}/{num_epochs}, Loss: {average_loss:.4f}')

    # Evaluate on test data
    model.eval()
    total_test_loss = 0
    with torch.no_grad():
        for batch in test_loader:
            input_ids = batch['input_ids'].to(device)
            attention_mask = batch['attention_mask'].to(device)
            labels = batch['labels'].to(device)

            outputs = model(input_ids=input_ids, attention_mask=attention_mask,␣
 ↪labels=labels)
            loss = outputs.loss
            total_test_loss += loss.item()

    average_test_loss = total_test_loss / len(test_loader)
    print(f'Test Loss: {average_test_loss:.4f}')
    model.train()
```

[12]:

```
# Save the trained model
model.save_pretrained('generative_qa_model')
tokenizer.save_pretrained('generative_qa_model')
```

[12]:

```
('generative_qa_model/tokenizer_config.json',
 'generative_qa_model/special_tokens_map.json',
 'generative_qa_model/spiece.model',
 'generative_qa_model/added_tokens.json')
```

[13]:

```
import torch
from transformers import T5Tokenizer, T5ForConditionalGeneration

# Load tokenizer and model
tokenizer = T5Tokenizer.from_pretrained('generative_qa_model')
model = T5ForConditionalGeneration.from_pretrained('generative_qa_model')

def generate_answer(context, question):
    # Split the input context into chunks of maximum sequence length
    max_seq_length = tokenizer.model_max_length
    chunks = [context[i:i+max_seq_length] for i in range(0, len(context),␣
 ↪max_seq_length)]

    # Generate answers for each chunk
    generated_answers = []
    for chunk in chunks:
        input_text = f"context: {chunk} question: {question}"
        input_ids = tokenizer.encode(input_text, return_tensors="pt",␣
 ↪max_length=max_seq_length, truncation=True)

        # Generate answer
        with torch.no_grad():
            output_ids = model.generate(input_ids, max_length=50,␣
 ↪num_return_sequences=1)
```

```
        # Decode and add the answer to the list
        answer = tokenizer.decode(output_ids[0], skip_special_tokens=True)
        generated_answers.append(answer)

    # Concatenate answers from chunks
    final_answer = " ".join(generated_answers)
    return final_answer

# Example context and question
```

```python
question = "What to do for Financial Help for Diabetes Care ?"

# Generate answer
answer = generate_answer(context, question)
print("Generated Answer:", answer)
```

```
Generated Answer: Signs and symptoms of small intestine cancer include fever,
weight loss, and weight loss. These and other signs and symptoms may be caused
by small intestine cancer or by other conditions. Check with your doctor if you
have any
```

```
[ ]:
```