Text Summarization Model

This repository contains code for training a T5-based model for text summarization using the SimpleT5 library. The model is trained on the CNN/DailyMail dataset and can generate abstractive summaries of input texts.

Pre-Requisites

Before running the code, ensure you have the following dependencies installed:

- numpy
- pandas
- scikit-learn
- simplet5
- transformers
- torch
- rouge-score

Importing Libraries

```
import numpy as np
import pandas as pd
import os
from sklearn.model_selection import train_test_split
from simplet5 import SimpleT5
from transformers import T5Tokenizer
import torch
from rouge import Rouge
```

The code begins by importing the required libraries for data handling, model training, and evaluation.

Data Preparation

```
input_dat = "cnn_dailymail/train.csv"
df = pd.read_csv(input_dat, encoding='latin-1')
df = df.rename(columns={"highlights":"target_text", "article":"source_text"})
df = df[['source_text', 'target_text']]
df['source_text'] = "summarize: " + df['source_text']
```

- The code reads the dataset from a CSV file (train.csv) using pandas.
- The columns of the dataframe are renamed to match the required input format for summarization.
- The dataframe is filtered to include only the relevant columns: source_text and target_text.
- A prefix, "summarize:", is added to each source_text to indicate summarization.

Train-Test Split

```
train_df, test_df = train_test_split(df, test_size=0.2)
```

The dataset is split into training and testing sets using the train_test_split function from scikit-learn. 80% of the data is used for training, and 20% is used for testing.

Tokenization and Model Initialization

```
tokenizer = T5Tokenizer.from_pretrained("t5-base")
model = SimpleT5()
model.from_pretrained(model_type="t5", model_name="t5-base")
```

- The T5 tokenizer is loaded from the "t5-base" pretrained model.
  This is the class provided by the transformers library for tokenizing text specifically for T5 models. It implements the tokenization process required to convert text into a sequence of tokens suitable for input to a T5 model.

- An instance of the SimpleT5 model is created and initialized with the "t5-base" model weights. "t5-base" refers to the base variant of the T5 model, which is a transformer-based language model pre-trained on a large corpus of text data.

- Example:
  ```
  text = "Hello, how are you?"
  tokens = tokenizer.tokenize(text)
  ```
  This will tokenize the text into a list of tokens: ['Hello', ',' , 'how', 'are', 'you', '?']. You can then use these tokens for further processing or input to the T5 model.

- By calling 'model.from_pretrained(model_type="t5", model_name="t5-base")', the code initializes a T5 model object and loads the pre-trained weights of the "t5-base" model. This allows you to use the model for various natural language processing tasks, such as text generation, summarization, or translation. After this line of code, you can use the model object to perform operations specific to the T5 model, such as generating text from prompts or fine-tuning the model on specific downstream tasks.

- "model.from_pretrained": This is a class method that is typically available in models provided by the transformers library. It allows you to initialize a model and load the pre-trained weights associated with a specific model configuration.

- 'model_type="t5"': This argument specifies the type of the model to initialize. In this case, the model type is set to "t5", indicating that we want to initialize a T5 model.

- 'model_name="t5-base"': This argument specifies the name of the pre-trained model to load. In this case, "t5-base" refers to the base variant of the T5 model, which is a transformer-based language model pre-trained on a large corpus of text data.

Model Training

```
model.train(train_df=train_df[:1000],
            eval_df=test_df[:100],
            source_max_token_len=400,
            target_max_token_len=120,
            batch_size=8, max_epochs=2, use_gpu=True,
            early_stopping_patience_epochs=2)
```

The model is trained using the train method of the SimpleT5 model. The arguments include:

- model.train: This calls the train method of the model object. It indicates that we want to train the model using the provided data and settings.

- train_df: The training dataframe. This argument specifies the training dataset to use. train_df refers to a pandas DataFrame containing the training data, and train_df[:1000] selects the first 1000 rows from the DataFrame to use as the training data.
- eval_df: The evaluation dataframe. This argument specifies the evaluation dataset to use. eval_df refers to a pandas DataFrame containing the evaluation data, and testl_df[:100] selects the first 100 rows from the DataFrame to use as the evaluation data.
- source_max_token_len: This argument sets the maximum token length for the source inputs. source_max_token_len=400 limits the number of tokens in the source input sequences to 400 tokens.
- target_max_token_len: This argument sets the maximum token length for the target outputs. target_max_token_len=200 limits the number of tokens in the target output sequences to 120 tokens.
- batch_size: This argument sets the batch size for training. It specifies the number of training examples that will be processed together in each training iteration or batch.
- max_epochs: This argument sets the maximum number of training epochs. An epoch is a complete pass through the entire training dataset.
- use_gpu: This argument specifies whether to use a GPU for training. If set to True, it indicates that a GPU should be used if available for faster training.
- early_stopping_patience_epochs:This argument sets the number of epochs to wait for early stopping. If the evaluation metric does not improve for this number of epochs, training will be stopped early.

Model Saving and Loading

```
filepath = "model.pth"
torch.save(model, filepath)
model1 = torch.load('model.pth', map_location=torch.device('cpu'))
model1.device= torch.device("cpu")
```

- The trained model is saved to a file using torch.save.
- The saved model can be loaded back using torch.load.
- The device is set to CPU for inference.
- model1 = torch.load('model.pth', map_location=torch.device('cpu')) is used to load a saved PyTorch model from a file. The torch.load() function is used to load the model object, and the map_location argument is set to torch.device('cpu') to ensure that the model is loaded onto the CPU device. Here's an explanation of what the code does:
- torch.load('model.pth', map_location=torch.device('cpu')): This code loads the model saved in the file 'model.pth' using the torch.load() function. The map_location argument is set to torch.device('cpu') to ensure that the model is loaded onto the CPU, even if the model was originally trained on a different device, such as a GPU. This allows you to load the model on a machine that might not have a compatible GPU.
- model1 = torch.load('model.pth', map_location=torch.device('cpu')) The loaded model is assigned to the variable model1. This variable represents the loaded model object and can be used for further operations or inference.
- By setting map_location=torch.device('cpu'), you ensure that the model is loaded onto the CPU device regardless of the device it was trained on. This is helpful when you want to load the model on a machine without a compatible GPU or when you want to perform inference on the CPU.
- Lastly, model1.device= torch.device("cpu") sets the device attribute of the model1 object to torch.device("cpu"). This explicitly confirms that the model is now on the CPU device.

Summarization using the Trained Model

```
model1 = torch.load('model.pth', map_location=torch.device("cuda"))
model1.device= torch.device("cuda")

text_input = "<input text to summarize>"
text_to_summarize = """summarize: """ + text_input
predicted_output = model1.predict(text_to_summarize)
predicted_output
```

- torch.load('model.pth', map_location=torch.device("cuda")): This line of code loads the model saved in the file 'model.pth' using the torch.load() function. The map_location argument is set to torch.device("cuda"), indicating that the model should be loaded onto the CUDA device (if available). This is useful when you have trained the model on a GPU and want to use it for inference on the GPU.
- model1 = torch.load('model.pth', map_location=torch.device("cuda")) The loaded model is assigned to the variable model1. This variable represents the loaded model object and can be used for further operations or inference.
- To generate a summary, provide the input text in the text_input variable.
- text_to_summarize = """summarize: """ + text_input: This line of code prepares the input text for summarization by concatenating it with the prefix "summarize: ". The specific format and requirements of the input text may depend on the model's input format and the task it was trained for.
- predicted_output = model1.predict(text_to_summarize): This line of code uses the model1 object to generate a predicted output for the summarization task. The exact method or function used for prediction may vary depending on the specific model or library being used. Here, model1.predict() is called with text_to_summarize as the input to generate the predicted summary.

Evaluation using ROUGE Scores

```
text_summaries = []  # reference summary
for x in range(20001, 20101):
    summary = str(df['target_text'][x])
    summary = summary.replace("\n", "")
    print(summary)
    print("\n")
    text_summaries.append(summary)

model_summaries = []
for x in range(20001, 20101):
    model_summary = model1.predict(df['source_text'][x])

    model_summaries.append(model_summary)
model_summaries = [str(summary) for summary in model_summaries]
text_summaries = [str(summary) for summary in text_summaries]

rouge = Rouge()
rouge.get_scores(model_summaries, text_summaries, avg=True, ignore_empty=True)
```

- summary = str(df['target_text'][x]): This retrieves the reference summary from the DataFrame (df) at index x and converts it to a string.
- summary = summary.replace("\n", ""): This removes any newline characters from the reference summary string.
- text_summaries.append(summary): This adds the cleaned reference summary to the text_summaries list.
- model_summary = model1.predict(df['source_text'][x]): This generates a model-generated summary by calling the model1.predict() method, passing in the source text from the DataFrame at index x.
- model_summaries.append(model_summary): This adds the model-generated summary to the model_summaries list.
- model_summaries = [str(summary) for summary in model_summaries]: This converts all the model-generated summaries in model_summaries to strings.
- text_summaries = [str(summary) for summary in text_summaries]: This converts all the reference summaries in text_summaries to strings.
- rouge = Rouge(): This creates an instance of the Rouge class, which is used for calculating ROUGE scores.
- rouge.get_scores(model_summaries, text_summaries, avg=True, ignore_empty=True): This computes the ROUGE scores for the model-generated summaries (model_summaries) compared to the reference summaries (text_summaries). The avg=True argument computes the average ROUGE scores, and ignore_empty=True indicates that empty summaries should be ignored in the calculation.

The result of this calculation will provide various ROUGE scores, such as ROUGE-1, ROUGE-2, and ROUGE-L. These scores assess the similarity between the model-generated summaries and the reference summaries, providing an evaluation of the quality of the generated summaries.

Conclusion

The provided code demonstrates the process of training a T5-based model for text summarization using the SimpleT5 library. It covers data preprocessing, model training, saving and loading the model, generating summaries, and evaluating the model using ROUGE scores.
Please note that the code assumes you have the necessary dataset and have made appropriate modifications to fit your specific use case.

Modules Used

Meaning and functionality of each module used in the code:

- **numpy (import numpy as np)**: NumPy is a fundamental package for scientific computing in Python. It provides support for large, multi-dimensional arrays and matrices, along with a collection of mathematical functions to operate on these arrays efficiently.
- **pandas (import pandas as pd)**: Pandas is a powerful library for data manipulation and analysis. It provides data structures like dataframes, which are tabular structures to hold and manipulate data. Pandas offers a wide range of functions to handle data, including reading and writing data from various file formats, data cleaning, filtering, and aggregation.
- **sklearn.model_selection (from sklearn.model_selection import train_test_split)**: This module from scikit-learn provides functions for model selection and evaluation. In this code, the train_test_split function is used to split the dataset into training and testing subsets. It randomly shuffles the data and splits it into two parts based on the specified test size or train size.
- **simplet5 (from simplet5 import SimpleT5)**: SimpleT5 is a Python library built on top of the Hugging Face's transformers library. It simplifies the process of using transformer models, such as

T5, for various natural language processing tasks. It provides a high-level API for training, fine-tuning, and using transformer models with a user-friendly interface.

- **transformers (from transformers import T5Tokenizer)**: The transformers library is a popular library developed by Hugging Face. It offers a wide range of pre-trained models and tools for natural language processing tasks. In this code, the T5Tokenizer class is used to tokenize text into subwords compatible with the T5 model. Tokenization is the process of breaking text into smaller units, such as words or subwords, for further processing.
- **torch (import torch)**: Torch is a powerful deep learning framework that provides tensor computation and deep neural network building blocks. It offers various tools and utilities for creating and training neural networks, optimizing models, and performing tensor operations efficiently. In this code, the torch module is used for saving and loading models, as well as performing inference.
- **rouge (from rouge import Rouge)**: Rouge is a popular evaluation metric for text summarization. The rouge module provides an implementation of this metric. It calculates the ROUGE scores, which measure the overlap between the generated summary and the reference summary. ROUGE scores are commonly used to evaluate the quality of text summarization systems.

These modules are essential for different aspects of the code. They provide functionality for data manipulation, model training, evaluation, and inference. Each module serves a specific purpose, such as handling data, tokenizing text, training models, and evaluating performance, making the code more efficient and easier to implement.