Here is an example Python code that uses the GPT-2 transformer-based language model to train a language model on the Hamlet corpus

import torch

import torch.nn as nn

from torch.utils.data import Dataset, DataLoader

from transformers import GPT2LMHeadModel, GPT2Tokenizer

# Load the Hamlet corpus

with open("hamlet.txt", "r", encoding="utf-8") as f:

text = f.read()

# Initialize the tokenizer and encode the text

tokenizer = GPT2Tokenizer.from\_pretrained("gpt2")

encoded = tokenizer.encode(text)

# Define a custom PyTorch dataset to handle the encoded data

class TextDataset(Dataset):

def \_\_init\_\_(self, data, block\_size):

self.data = data

self.block\_size = block\_size

def \_\_len\_\_(self):

return len(self.data) - self.block\_size

def \_\_getitem\_\_(self, idx):

return torch.tensor(self.data[idx:idx+self.block\_size])

# Set the training parameters

block\_size = 128

batch\_size = 16

epochs = 10

# Initialize the GPT-2 model

model = GPT2LMHeadModel.from\_pretrained("gpt2")

# Define the loss function and optimizer

criterion = nn.CrossEntropyLoss()

optimizer = torch.optim.Adam(model.parameters(), lr=1e-4)

# Initialize the data loader

dataset = TextDataset(encoded, block\_size)

loader = DataLoader(dataset, batch\_size=batch\_size, shuffle=True)

# Train the model

for epoch in range(epochs):

running\_loss = 0.0

for batch in loader:

inputs, labels = batch[:, :-1], batch[:, 1:]

inputs, labels = inputs.to(device), labels.to(device)

optimizer.zero\_grad()

outputs = model(inputs)[0]

loss = criterion(outputs.view(-1, outputs.size(-1)), labels.view(-1))

loss.backward()

optimizer.step()

running\_loss += loss.item()

print(f"Epoch {epoch+1} loss: {running\_loss/len(loader):.4f}")

This code loads the Hamlet corpus from a text file, encodes the text using the GPT-2 tokenizer, and trains the GPT-2 model on the encoded data using a custom PyTorch dataset and data loader. The model is trained using cross-entropy loss and the Adam optimizer, and the training progress is printed to the console after each epoch.

Note that this code assumes that you have already installed the **transformers** package, which provides the GPT-2 tokenizer and model. If you have not installed this package yet, you can do so using pip:

pip install transformers

Also note that this code uses the pre-trained GPT-2 model as a starting point, and fine-tunes the model on the Hamlet corpus. This allows the model to adapt to the specific language and style of the corpus, and improve its performance on tasks such as language generation.

After training the model, you can use it for various language-related tasks such as text completion, summarization, and translation. For example, to generate text using the trained model, you can use the **generate** method of the GPT2LMHeadModel class:

# Generate some text

prompt = "To be or not to be"

input\_ids = tokenizer.encode(prompt, return\_tensors="pt")

output = model.generate(input\_ids, max\_length=50, do\_sample=True)

generated = tokenizer.decode(output[0], skip\_special\_tokens=True)

print(generated)

This code generates text starting with the prompt "To be or not to be" and prints the generated text to the console. The **generate** method takes as input a tensor of input IDs, which in this case is obtained by encoding the prompt using the GPT-2 tokenizer. The **max\_length** parameter specifies the maximum length of the generated text, and **do\_sample=True** enables random sampling of the model's output. The generated text is decoded using the tokenizer's **decode** method to obtain a human-readable string.

Overall, this code demonstrates how to train a language model using the GPT-2 transformer-based architecture on the Hamlet corpus, and how to use the trained model for various language-related tasks.

In order to save the trained model for future use, you can use PyTorch's built-in model serialization functionality. For example, you can save the model to a file using the **save\_pretrained** method of the GPT2LMHeadModel class:

# Save the trained model to a file

model\_path = "hamlet\_model"

model.save\_pretrained(model\_path)

This code saves the trained model to a directory named "hamlet\_model". You can later load the model from this directory using the **from\_pretrained** method of the GPT2LMHeadModel class:

# Load the trained model from a file

model = GPT2LMHeadModel.from\_pretrained(model\_path)

This code loads the model from the "hamlet\_model" directory and initializes a new instance of the GPT2LMHeadModel class with the saved weights and configuration.

In addition to saving the trained model, you can also save the tokenizer to a file using the **save\_pretrained** method of the GPT2Tokenizer class:

python

# Save the tokenizer to a file

tokenizer\_path = "hamlet\_tokenizer"

tokenizer.save\_pretrained(tokenizer\_path)

This code saves the tokenizer to a directory named "hamlet\_tokenizer". You can later load the tokenizer from this directory using the **from\_pretrained** method of the GPT2Tokenizer class:

This code loads the tokenizer from the "hamlet\_tokenizer" directory and initializes a new instance of the GPT2Tokenizer class with the saved vocabulary and configuration.

By saving both the trained model and the tokenizer to files, you can easily reuse the trained model for various language-related tasks, without having to retrain the model from scratch.

Full code

import torch

from transformers import GPT2Tokenizer, GPT2LMHeadModel

from torch.utils.data import Dataset, DataLoader

# Define a custom dataset for the Hamlet corpus

class HamletDataset(Dataset):

def \_\_init\_\_(self, file\_path, tokenizer):

self.examples = []

with open(file\_path, "r", encoding="utf-8") as f:

lines = f.readlines()

for line in lines:

line = line.strip()

if len(line) > 0:

self.examples.append(tokenizer.encode(line))

def \_\_len\_\_(self):

return len(self.examples)

def \_\_getitem\_\_(self, idx):

return torch.tensor(self.examples[idx], dtype=torch.long)

# Initialize the tokenizer and model

tokenizer = GPT2Tokenizer.from\_pretrained("gpt2")

model = GPT2LMHeadModel.from\_pretrained("gpt2")

# Load the Hamlet corpus as a custom dataset

dataset = HamletDataset("hamlet.txt", tokenizer)

# Define a data loader for the dataset

dataloader = DataLoader(dataset, batch\_size=4, shuffle=True)

# Set the model to training mode and specify the optimizer

model.train()

optimizer = torch.optim.Adam(model.parameters(), lr=1e-5)

# Train the model for a specified number of epochs

num\_epochs = 5

for epoch in range(num\_epochs):

for batch in dataloader:

# Move the batch to the device (GPU or CPU)

batch = batch.to(model.device)

# Compute the loss and perform backpropagation

outputs = model(batch, labels=batch)

loss = outputs[0]

loss.backward()

# Update the model parameters

optimizer.step()

optimizer.zero\_grad()

# Print the epoch number and the loss

print(f"Epoch {epoch+1}, Loss: {loss.item()}")

# Save the trained model and tokenizer to files

model\_path = "hamlet\_model"

tokenizer\_path = "hamlet\_tokenizer"

model.save\_pretrained(model\_path)

tokenizer.save\_pretrained(tokenizer\_path)

This code defines a custom dataset class for the Hamlet corpus, which loads the corpus from a file and tokenizes it using the GPT-2 tokenizer. It then defines a data loader for the dataset, which batches the examples and shuffles them randomly. The code sets the model to training mode and specifies the Adam optimizer with a learning rate of 1e-5. It then trains the model for 5 epochs on the Hamlet corpus, using batches from the data loader.

After training the model, the code saves the trained model and tokenizer to files, which can be loaded later for use in various language-related tasks.