# Sample Exercise: Basic Text Generation with GPT-2

This sample exercise demonstrates the basic use of the GPT-2 model for text generation using the Hugging Face Transformers library. Learners will complete a script that generates text based on a given input prompt. The aim is to familiarise students with the loading of models, text generation, and understanding the impact of parameter changes.

# Importing necessary libraries from Hugging Face

from transformers import 1)\_\_\_ as

from transformers import 2)\_\_\_ as

# Load pre-trained GPT-2 model and tokenizer

tokenizer = 2)\_\_\_.from\_pretrained('gpt2')

model = 1)\_\_\_.from\_pretrained('gpt2')

# Encode some input text

inputs = tokenizer.encode("Today's weather is", add\_special\_tokens=True, return\_tensors='pt')

# Generate text using the model

outputs = model.generate(inputs, max\_length=3)\_\_\_, num\_return\_sequences=4)\_\_\_)

print("Generated text:", tokenizer.decode(outputs[0], skip\_special\_tokens=True))

### **Instructions:**

1. **Complete the Imports**: Replace 1)\_\_\_ with the name of the class for loading the model and 2)\_\_\_ with the name of the tokenizer class from the Transformers library.
2. **Set max\_length**: Fill in 3)\_\_\_ with the maximum number of tokens for the generated text (suggest 50 as a starting point).
3. **Adjust num\_return\_sequences**: Input a suitable number in 4)\_\_\_ to define how many different text sequences you want to generate based on the input (try starting with 1).
4. **Experiment and Discuss**: After completing the script, run it with various inputs and parameters. Discuss how changes in max\_length and num\_return\_sequences impact the output in terms of diversity and relevance of the generated text.

### **Learning Outcome:**

By completing this exercise, learners will engage directly with the mechanics of a state-of-the-art text generation model. They will learn how different settings influence the model's output and will be able to articulate their understanding of the practical aspects of using such models in real-world applications.

Here are the answers for the gaps specified in the SampleExercise.md:

**1**: GPT2LMHeadModel  
**2**: GPT2Tokenizer  
**3**: 50 (as a suggested starting point for max\_length)  
**4**: 1 (as a suggested starting value for num\_return\_sequences)

### 

### **Sample Exercise 1: Understanding Hugging Face Components**

**File:** UnderstandingComponents.md

#### **Python Code**

python

Copy code

# Import the essential components from the Hugging Face library

from transformers import \_\_1\_\_, \_\_2\_\_, \_\_3\_\_

# Check versions to ensure compatibility

print("Transformer version:", \_\_1\_\_.\_\_version\_\_)

print("Tokenizer version:", \_\_2\_\_.\_\_version\_\_)

print("Datasets version:", \_\_3\_\_.\_\_version\_\_)

#### **Instructions:**

1. **Fill in \_\_1\_\_**: Replace \_\_1\_\_ with the module responsible for the transformer models.
2. **Fill in \_\_2\_\_**: Replace \_\_2\_\_ with the module used for tokenization.
3. **Fill in \_\_3\_\_**: Replace \_\_3\_\_ with the module for handling datasets.

#### **Answers:**

1. **1**: transformers
2. **2**: transformers
3. **3**: datasets

### **Sample Exercise 3: Applying Advanced Sampling Strategies**

**File:** AdvancedSamplingStrategies.md

# Import GPT2 model and tokenizer from transformers

from transformers import GPT2LMHeadModel, GPT2Tokenizer

# Load tokenizer and model

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

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

# Generate text with advanced sampling techniques

inputs = tokenizer.encode("The future of AI in:", add\_special\_tokens=True, return\_tensors='pt')

outputs = model.generate(inputs, max\_length=50, \_\_1\_\_=\_\_2\_\_, \_\_3\_\_=\_\_4\_\_)

# Print the generated text

print("Generated text:", tokenizer.decode(outputs[0]))

#### **Instructions:**

1. **Fill in \_\_1\_\_**: Replace this with the parameter that specifies the sampling strategy to limit the selection pool to the most likely tokens.
2. **Fill in \_\_2\_\_**: Provide an integer value to define the size of the selection pool.
3. **Fill in \_\_3\_\_**: Specify the parameter to define the cumulative probability cutoff for the sampling pool.
4. **Fill in \_\_4\_\_**: Provide a decimal value representing the cutoff for cumulative probabilities.

#### **Answers:**

1. **1**: top\_k
2. **2**: 10
3. **3**: top\_p
4. **4**: 0.95

### **Sample Exercise: Fine-Tuning GPT Models for Genre-Specific Text Generation**

# Import the necessary tools from transformers

from transformers import GPT2Tokenizer, GPT2LMHeadModel, Trainer, TrainingArguments

# Load a pre-trained GPT-2 model and tokenizer

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

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

# Prepare a dataset (assume dataset is loaded and named 'dataset')

train\_dataset = dataset['train']

test\_dataset = dataset['test']

# Define training arguments for fine-tuning

training\_args = TrainingArguments(

output\_dir='./results', # output directory

num\_train\_epochs=\_\_1\_\_, # total number of training epochs

per\_device\_train\_batch\_size=\_\_2\_\_, # batch size for training

per\_device\_eval\_batch\_size=\_\_3\_\_, # batch size for evaluation

warmup\_steps=\_\_4\_\_, # number of warmup steps for learning rate scheduler

evaluation\_strategy='epoch',

logging\_dir='./logs', # directory for storing logs

)

# Initialize Trainer to fine-tune the model

trainer = Trainer(

model=model,

args=training\_args,

train\_dataset=train\_dataset,

eval\_dataset=test\_dataset

)

# Start fine-tuning

trainer.train()

#### **Instructions:**

1. **Fill in \_\_1\_\_**: Replace this with the number of epochs to train the model for fine-tuning.
2. **Fill in \_\_2\_\_ and \_\_3\_\_**: Set the appropriate batch sizes for training and evaluation, respectively.
3. **Fill in \_\_4\_\_**: Input the number of steps to perform during the warmup phase of the learning rate scheduler.

#### **Answers:**

1. **1**: 3 (number of epochs; can be adjusted based on the size of the dataset and desired training duration)
2. **2**: 8 (batch size for training; depends on available compute resources)
3. **3**: 16 (batch size for evaluation; typically larger if compute allows as it speeds up the process)
4. **4**: 500 (warmup steps; a common practice to help stabilise the model's training early on)