Вот пример кода для обучения модели LLaMA (Large Language Model) с использованием библиотеки Hugging Face Transformers. Как и в случае с BART, убедитесь, что у вас есть доступ к необходимым данным и что вы ознакомлены с архитектурой LLaMA.

import pandas as pd

import torch

from sklearn.model\_selection import train\_test\_split

from transformers import LlamaTokenizer, LlamaForCausalLM, Trainer, TrainingArguments

# Загрузка данных

# Предположим, что у вас есть CSV-файл с колонкой 'text'

df = pd.read\_csv('data.csv') # Убедитесь, что у вас есть файл data.csv с вашими данными

# Делим данные на обучающую и валидационную выборки

train\_texts, val\_texts = train\_test\_split(df['text'], test\_size=0.2, random\_state=42)

# Инициализация токенизатора

tokenizer = LlamaTokenizer.from\_pretrained('meta-llama/LLaMA-7B')

# Токенизация текста

train\_encodings = tokenizer(list(train\_texts), truncation=True, padding=True, max\_length=512, return\_tensors="pt")

val\_encodings = tokenizer(list(val\_texts), truncation=True, padding=True, max\_length=512, return\_tensors="pt")

# Преобразование данных в PyTorch Dataset

class TextDataset(torch.utils.data.Dataset):

def \_\_init\_\_(self, encodings):

self.encodings = encodings

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

return {key: val[idx] for key, val in self.encodings.items()}

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

return len(self.encodings['input\_ids'])

# Создание датасетов

train\_dataset = TextDataset(train\_encodings)

val\_dataset = TextDataset(val\_encodings)

# Инициализация модели

model = LlamaForCausalLM.from\_pretrained('meta-llama/LLaMA-7B')

# Параметры обучения

training\_args = TrainingArguments(

output\_dir='./results', # выходная директория

num\_train\_epochs=3, # количество эпох

per\_device\_train\_batch\_size=4, # размер батча

per\_device\_eval\_batch\_size=4, # размер батча для валидации

warmup\_steps=500, # количество разогревочных шагов

weight\_decay=0.01, # коэффициент L2 регуляризации

logging\_dir='./logs', # директория для логов

logging\_steps=10,

)

# Создание Trainer

trainer = Trainer(

model=model,

args=training\_args,

train\_dataset=train\_dataset,

eval\_dataset=val\_dataset,

)

# Обучение модели

trainer.train()

# Сохранение модели

model.save\_pretrained('./llama\_model')

tokenizer.save\_pretrained('./llama\_model')

### Обзор шагов:

1. Загрузка и подготовка данных: Загрузите данные с текстами, которые вы хотите использовать для обучения.

2. Токенизация: С помощью токенизатора LLaMA преобразуются ваши тексты в подходящий формат.

3. Создание кастомного Dataset: Создается класс Dataset для работы с PyTorch.

4. Инициализация модели LLaMA: Загружается предобученная модель LLaMA.

5. Настройка параметров обучения: Задаются параметры обучения, такие как количество эпох и размер батча.

6. Обучение модели: Запускается процесс обучения с использованием класса Trainer.

7. Сохранение модели: Модель и токенизатор сохраняются после завершения обучения.

### Примечания:

- Убедитесь, что у вас установлены пакеты transformers и torch. Их можно установить с помощью pip:

pip install transformers torch

- Проверьте путь к файлу CSV и убедитесь, что ваши метки правильно структурированы.

- Настройте параметры обучения с учетом вашего оборудования и объема данных.

- Учтите, что модели LLaMA требуют значительных вычислительных ресурсов, чтобы эффективно обучаться.

Вот полный код для обучения модели BART (Bidirectional and Auto-Regressive Transformers) для задачи текстовой генерации или обработки (например, для суммирования текста) с использованием библиотеки Hugging Face Transformers и PyTorch.

import pandas as pd

import torch

from sklearn.model\_selection import train\_test\_split

from transformers import BartTokenizer, BartForConditionalGeneration, Trainer, TrainingArguments

# Загрузка данных

# Предположим, что у вас есть CSV-файл с колонками 'text' и 'summary'

df = pd.read\_csv('data.csv') # Убедитесь, что у вас есть файл data.csv с вашими данными

# Делим данные на обучающую и валидационную выборки

train\_texts, val\_texts, train\_summaries, val\_summaries = train\_test\_split(df['text'], df['summary'], test\_size=0.2, random\_state=42)

# Инициализация токенизатора

tokenizer = BartTokenizer.from\_pretrained('facebook/bart-base')

# Токенизация текста и суммаризации

train\_encodings = tokenizer(list(train\_texts), truncation=True, padding=True, max\_length=1024, return\_tensors="pt")

train\_labels = tokenizer(list(train\_summaries), truncation=True, padding=True, max\_length=128, return\_tensors="pt").input\_ids

val\_encodings = tokenizer(list(val\_texts), truncation=True, padding=True, max\_length=1024, return\_tensors="pt")

val\_labels = tokenizer(list(val\_summaries), truncation=True, padding=True, max\_length=128, return\_tensors="pt").input\_ids

# Преобразование данных в PyTorch Dataset

class SummarizationDataset(torch.utils.data.Dataset):

def \_\_init\_\_(self, encodings, labels):

self.encodings = encodings

self.labels = labels

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

item = {key: val[idx] for key, val in self.encodings.items()}

item['labels'] = self.labels[idx]

return item

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

return len(self.labels)

# Создание датасетов

train\_dataset = SummarizationDataset(train\_encodings, train\_labels)

val\_dataset = SummarizationDataset(val\_encodings, val\_labels)

# Инициализация модели

model = BartForConditionalGeneration.from\_pretrained('facebook/bart-base')

# Параметры обучения

training\_args = TrainingArguments(

output\_dir='./results', # выходная директория

num\_train\_epochs=3, # количество эпох

per\_device\_train\_batch\_size=4, # размер батча

per\_device\_eval\_batch\_size=4, # размер батча для валидации

warmup\_steps=500, # количество разогревочных шагов

weight\_decay=0.01, # коэффициент L2 регуляризации

logging\_dir='./logs', # директория для логов

logging\_steps=10,

)

# Создание Trainer

trainer = Trainer(

model=model,

args=training\_args,

train\_dataset=train\_dataset,

eval\_dataset=val\_dataset,

)

# Обучение модели

trainer.train()

# Сохранение модели

model.save\_pretrained('./bart\_model')

tokenizer.save\_pretrained('./bart\_model')

### Обзор шагов:

1. Загрузка и подготовка данных: Вам нужно загрузить свои данные (тексты и их суммирования) и разделить их на обучающую и валидационную выборки.

2. Токенизация: С помощью токенизатора BART преобразуются ваши тексты и их суммирования в подходящий формат.

3. Создание кастомного Dataset: Создается класс Dataset для работы с PyTorch.

4. Инициализация модели BART: Загружается предобученная модель BART для последовательной генерации.

5. Настройка параметров обучения: Устанавливаются различные параметры, включая количество эпох и размер батча.

6. Обучение модели: Запускается процесс обучения на основе созданного класса Trainer.

7. Сохранение модели: В конце код сохраняет модель и токенизатор.

### Примечания:

- Убедитесь, что у вас установлены пакеты transformers и torch. Их можно установить с помощью pip:

pip install transformers torch

Обучение модели BERT (Bidirectional Encoder Representations from Transformers) требует использования специализированных библиотек, таких как Hugging Face Transformers и PyTorch. В данном примере я покажу, как можно обучить модель BERT для задачи классификации текстов.

Вот полный код для этой задачи:

import pandas as pd

import numpy as np

from sklearn.model\_selection import train\_test\_split

from transformers import BertTokenizer, BertForSequenceClassification

from transformers import Trainer, TrainingArguments

import torch

# Загрузка данных

# Предположим, что ваши данные находятся в формате CSV с колонками 'text' и 'label'

df = pd.read\_csv('data.csv') # Убедитесь, что у вас есть файл data.csv с вашими данными

# Делим данные на обучающую и тестовую выборки

train\_texts, val\_texts, train\_labels, val\_labels = train\_test\_split(df['text'], df['label'], test\_size=0.2, random\_state=42)

# Инициализация токенизатора

tokenizer = BertTokenizer.from\_pretrained('bert-base-uncased')

# Токенизация текста

train\_encodings = tokenizer(list(train\_texts), truncation=True, padding=True, max\_length=512)

val\_encodings = tokenizer(list(val\_texts), truncation=True, padding=True, max\_length=512)

# Преобразование данных в PyTorch Dataset

class NewsDataset(torch.utils.data.Dataset):

def \_\_init\_\_(self, encodings, labels):

self.encodings = encodings

self.labels = labels

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

item = {key: torch.tensor(val[idx]) for key, val in self.encodings.items()}

item['labels'] = torch.tensor(self.labels[idx])

return item

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

return len(self.labels)

# Создание датасетов

train\_dataset = NewsDataset(train\_encodings, train\_labels.to\_numpy())

val\_dataset = NewsDataset(val\_encodings, val\_labels.to\_numpy())

# Инициализация модели

model = BertForSequenceClassification.from\_pretrained('bert-base-uncased', num\_labels=len(df['label'].unique()))

# Параметры обучения

training\_args = TrainingArguments(

output\_dir='./results', # выходная директория

num\_train\_epochs=3, # количество эпох

per\_device\_train\_batch\_size=8, # размер батча

per\_device\_eval\_batch\_size=8, # размер батча для валидации

warmup\_steps=500, # количество разогревочных шагов

weight\_decay=0.01, # коэффициент L2 регуляризации

logging\_dir='./logs', # директория для логов

logging\_steps=10,

)

# Создание Trainer

trainer = Trainer(

model=model,

args=training\_args,

train\_dataset=train\_dataset,

eval\_dataset=val\_dataset,

)

# Обучение модели

trainer.train()

# Сохранение модели

model.save\_pretrained('./model')

tokenizer.save\_pretrained('./model')

### Обзор шагов:

1. Загрузка и предобработка данных: Вам нужно будет загрузить ваши данные и разделить их на обучающую и тестовую выборки.

2. Токенизация: Используя токенизатор BERT, ваши тексты будут преобразованы в подходящий формат.

3. Создание кастомного Dataset: Создается собственный класс Dataset для работы с PyTorch.

4. Инициализация модели BERT: Загружается предобученная модель BERT для классификации.

5. Настройка параметров обучения: Устанавливаются различные параметры, такие как количество эпох, размер батча и т. д.

6. Обучение модели: Запускается процесс обучения на основе созданного класса Trainer.

7. Сохранение модели: В конце код сохраняет модель и токенизатор.

### Примечания:

- Убедитесь, что у вас установлен пакет transformers и torch. Вы можете установить их с помощью pip:

pip install transformers torch

- Вам необходимо адаптировать код для работы с вашими данными (например, изменить путь к файлу CSV, если это необходимо).

- Данные должны быть подготовлены и высчитываться в числовом формате, соответствующем формату модели. Убедитесь, что ваши метки числовые и соответствуют количеству классов