Assignment 3

Install the Transformers, Datasets, and Evaluate libraries to run this notebook.

```
!pip install datasets evaluate transformers[sentencepiece]
!apt install git-lfs
```

```
#Mount Drive
from google.colab import drive
drive.mount("/content/drive/")
```

```
Drive already mounted at /content/drive/; to attempt to forcibly remount, call drive.mount("/content/drive/", force_remount=True).
```

```
#Import the train and test data from huggingface
from datasets import load_dataset, DatasetDict

ds_train = load_dataset("huggingface-course/codeparrot-ds-train", split="train")
ds_valid = load_dataset("huggingface-course/codeparrot-ds-valid", split="validation")
```

```
#Select 50k from the imported train data randomly to be as the our train data and 500 for validation
raw_datasets = DatasetDict(
    {
        "train": ds_train.shuffle(seed=42).select(range(50000)),
        "valid": ds_valid.shuffle(seed=42).select(range(500))
    }
)

raw_datasets
```

```
DatasetDict({
    train: Dataset({
        features: ['repo_name', 'path', 'copies', 'size', 'content', 'license'],
        num_rows: 50000
    })
    valid: Dataset({
        features: ['repo_name', 'path', 'copies', 'size', 'content', 'license'],
        num_rows: 500
    })
})
```

```
# show the first 200 characters of each field:
for key in raw_datasets["train"][0]:
    print(f"{key.upper()}: {raw_datasets['train'][0][key][:200]}")
```

```
#Show Sample of working

from transformers import AutoTokenizer

context_length = 128
tokenizer = AutoTokenizer.from_pretrained("huggingface-course/code-search-net-tokenizer")

outputs = tokenizer(
    raw_datasets["train"][:2]["content"],
    truncation=True,
    max_length=context_length,
    return_overflowing_tokens=True,
    return_length=True,
)

print(f"Input IDs length: {len(outputs['input_ids'])}")
print(f"Input chunk lengths: {(outputs['length'])}")
print(f"Chunk mapping: {outputs['overflow_to_sample_mapping']}")
```

```
#Tokenizition Process
def tokenize(element):
    outputs = tokenizer(
        element["content"],
        truncation=True,
        max_length=context_length,
        return_overflowing_tokens=True,
        return_length=True,
    )
    input_batch = []
    for length, input_ids in zip(outputs["length"], outputs["input_ids"]):
        if length == context_length:
            input_batch.append(input_ids)
    return {"input_ids": input_batch}
```

```
tokenized_datasets = raw_datasets.map(
    tokenize, batched=True, remove_columns=raw_datasets["train"].column_names
)
tokenized_datasets
```

```
100%    50/50 [07:14<00:00, 9.13s/ba]

100%    1/1 [00:04<00:00, 4.51s/ba]
DatasetDict({
    train: Dataset({
        features: ['input_ids'],
        num_rows: 1375550
    })
    valid: Dataset({
        features: ['input_ids'],
        num_rows: 13617
    })
})
```

```
from transformers import AutoTokenizer, GPT2LMHeadModel, AutoConfig

config = AutoConfig.from_pretrained(
    "gpt2",
    vocab_sizse=len(tokenizer),
    n_ctx=context_length,
    bos_token_id=tokenizer.bos_token_id,
    eos_token_id=tokenizer.eos_token_id,
)
```

```
#Initializing a new GPT model and print model parameters
model = GPT2LMHeadModel(config)
model_size = sum(t.numel() for t in model.parameters())
print(f"GPT-2 size: {model_size/1000**2:.1f}M parameters")
```

```
GPT-2 size: 124.4M parameters
```

```
#We can use the DataCollatorForLanguageModeling collator, which is designed specifically for language modeling.
from transformers import DataCollatorForLanguageModeling

tokenizer.pad_token = tokenizer.eos_token
data_collator = DataCollatorForLanguageModeling(tokenizer, mlm=False)
```

```
out = data_collator([tokenized_datasets["train"][i] for i in range(5)])
for key in out:
    print(f"{key} shape: {out[key].shape}")
```

```
You're using a GPT2TokenizerFast tokenizer. Please note that with a fast tokenizer, using the `__call__` method is faster than using a method to encode
input_ids shape: torch.Size([5, 128])
attention_mask shape: torch.Size([5, 128])
labels shape: torch.Size([5, 128])
```

Training

Possible Optimizers to try Optimizers = adamw_hf, adamw_torch, adamw_apex_fused, adamw_anyprecision or adafactor.

modify max_steps to stop after a number of iterations

modify batch size to fit into memory modify save every n steps to modify how often save occurs

modify output_dir to a google drive path to save and load the model correctly

```
# Prepare the model for  training by traning args

from transformers import Trainer, TrainingArguments

args = TrainingArguments(
    output_dir="/content/drive/MyDrive/DL3/MAX_STEP/",
    optim= 'adamw_hf',
    per_device_train_batch_size=16,
    per_device_eval_batch_size=16,
    evaluation_strategy="steps",
    eval_steps=5_000,
    logging_steps=1,
    gradient_accumulation_steps=8,
    num_train_epochs=1,
    weight_decay=0.1,
    warmup_steps=100,
    lr_scheduler_type="cosine",
    learning_rate=5e-4,
```

```
    save_steps=2000,
    fp16=True,
    max_steps=3000,
)
trainer = Trainer(
    model=model,
    tokenizer=tokenizer,
    args=args,
    data_collator=data_collator,
    train_dataset=tokenized_datasets["train"],
    eval_dataset=tokenized_datasets["valid"],

)
```

```
PyTorch: setting up devices
The default value for the training argument `--report_to` will change in v5 (from all installed integrations to none). In v5, you will need to use `--re
max_steps is given, it will override any value given in num_train_epochs
Using cuda_amp half precision backend
```

```
# Start Training
result = trainer.train()
```

```
***** Running training *****
  Num examples = 1375550
  Num Epochs = 1
  Instantaneous batch size per device = 16
  Total train batch size (w. parallel, distributed & accumulation) = 128
  Gradient Accumulation steps = 8
  Total optimization steps = 3000
  Number of trainable parameters = 124439808
```

[3000/3000 1:29:16, Epoch 0/1]

Step	Training Loss	Validation Loss

```
Saving model checkpoint to /content/drive/MyDrive/DL3/MAX_STEP/checkpoint-2000
Configuration saved in /content/drive/MyDrive/DL3/MAX_STEP/checkpoint-2000/config.json
Model weights saved in /content/drive/MyDrive/DL3/MAX_STEP/checkpoint-2000/pytorch_model.bin
tokenizer config file saved in /content/drive/MyDrive/DL3/MAX_STEP/checkpoint-2000/tokenizer_config.json
Special tokens file saved in /content/drive/MyDrive/DL3/MAX_STEP/checkpoint-2000/special_tokens_map.json


Training completed. Do not forget to share your model on huggingface.co/models =)
```

```
#Start Evaluation
eval_results = trainer.evaluate()
```

```
***** Running Evaluation *****
  Num examples = 13617
  Batch size = 16
```

[852/852 3:22:40]

Report Perplexity and eval_results number with each experiment

```
#Perplexity is a measurement of how well a probability distribution or probability model predicts a sample

import numpy as np
print(f"Perplexity: {np.exp(eval_results['eval_loss']):.2f}")
```

```
Perplexity: 6.71
```

```
result
```

```
TrainOutput(global_step=3000, training_loss=2.024006205002467, metrics={'train_runtime': 5358.2878, 'train_samples_per_second': 71.665,
'train_steps_per_second': 0.56, 'total_flos': 2.5084035072e+16, 'train_loss': 2.024006205002467, 'epoch': 0.28})
```

```
trainer.state.log_history
```

Example to load from checkpoint Note: move to Drive and get Drive path first

```
#trainer.train(resume_from_checkpoint='/content/drive/MyDrive/DL3/LR/checkpoint-2100')
```

Test Code Prompts

Model and Tokenizer must be present

```
import torch
from transformers import pipeline

device =  torch.device("cuda:0") if torch.cuda.is_available() else torch.device("cpu")
print(device)
pipe = pipeline(
    "text-generation",
     model=model,
     tokenizer=tokenizer,
      device=device
)
```

```
cuda:0
```

```
txt = """\
# create some data
x = np.random.randn(100)
y = np.random.randn(100)

# create scatter plot with x, y
"""
print(pipe(txt, num_return_sequences=1)[0]["generated_text"])
```

```
Setting `pad_token_id` to `eos_token_id`:0 for open-end generation.
# create some data
x = np.random.randn(100)
y = np.random.randn(100)

# create scatter plot with x, y
ax = plt.subplot(111)
ax.scatter(
/usr/local/lib/python3.8/dist-packages/transformers/generation/utils.py:1387: UserWarning: Neither `max_length` nor `max_new_tokens` has been set, `max_
  warnings.warn(
```

```
txt = """\
# create some data
x = np.random.randn(100)
y = np.random.randn(100)

# create dataframe from x and y
"""
print(pipe(txt, num_return_sequences=1)[0]["generated_text"])
```

```
Setting `pad_token_id` to `eos_token_id`:0 for open-end generation.
# create some data
x = np.random.randn(100)
y = np.random.randn(100)

# create dataframe from x and y
X = []
y_data = []
y_data_
```

```
txt = """\
# dataframe with profession, income and name
df = pd.DataFrame({'profession': x, 'income':y, 'name': z})

# calculate the mean income per profession
"""
print(pipe(txt, num_return_sequences=1)[0]["generated_text"])
```

```
Setting `pad_token_id` to `eos_token_id`:0 for open-end generation.
# dataframe with profession, income and name
df = pd.DataFrame({'profession': x, 'income':y, 'name': z})

# calculate the mean income per profession
df.mean(0
```

```
txt = """
# import random forest regressor from scikit-learn
from sklearn.ensemble import RandomForestRegressor

# fit random forest model with 300 estimators on X, y:
"""
print(pipe(txt, num_return_sequences=1)[0]["generated_text"])
```

```
Setting `pad_token_id` to `eos_token_id`:0 for open-end generation.

# import random forest regressor from scikit-learn
from sklearn.ensemble import RandomForestRegressor

# fit random forest model with 300 estimators on X, y:
X, y = datasets.make_classification(n_samples=2000
```

✓ 0s completed at 9:34 PM