Lab 9: Finetuning GPT-2 with LoRA

Open in Colab

Open in Kaggle

In this lab we will use GPT-2 for the task of text generation. We'll first quickly compare Greedy Search and (Diverse) Beam Search with GPT-2. Then we'll finetune GPT-2 to generate text that is more explicitly infused with knowledge of Hemingway's book, "*The Sun also Rises*", and can generate text in the style of the book.

Lab 9 Assignment/Task

There are three questions in this lab. As an added bonus, try downloading your own book from Project Gutenberg to finetune GPT-2 to generate text following your chosen book/author (see this script) for help to convert it to a .csv file of sentences).

```
import torch

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
```

```
import numpy as np
import pandas as pd

from transformers import GPT2Tokenizer, AutoModelForCausalLM, TrainingArguments, Trainer
from torch.utils.data import Dataset, random_split
from peft import LoraModel, LoraConfig
```

```
tokenizer = GPT2Tokenizer.from_pretrained("gpt2")
model = AutoModelForCausalLM.from_pretrained("openai-community/gpt2")
```

```
/usr/local/lib/python3.11/dist-packages/huggingface_hub/utils/_auth.py:94: UserWarnir
The secret `HF_TOKEN` does not exist in your Colab secrets.
To authenticate with the Hugging Face Hub, create a token in your settings tab (https
You will be able to reuse this secret in all of your notebooks.
Please note that authentication is recommended but still optional to access public mc
  warnings.warn(
```

```
tokenizer_config.json:  100%                         26.0/26.0  [00:00<00:00,  1.58kB/
                                                   s]

vocab.json:  100%                                1.04M/1.04M  [00:00<00:00,  31.9MB/
                                               s]

merges.txt:  100%                                456k/456k  [00:00<00:00,  3.91MB/s]
merges.txt:  100%                                    456k/456k  [00:00<00:00,  2.01MB/s]

tokenizer.json:  100%                                1.36M/1.36M  [00:00<00:00,  5.84MB/
                                                     s]

config.json:  100%                                   665/665  [00:00<00:00,  58.9kB/s]

config.json:  100%                                   665/665  [00:00<00:00,  64.4kB/s]
Xet Storage is enabled for this repo, but the 'hf_xet' package is not installed. Fall
WARNING:huggingface_hub.file_download:Xet Storage is enabled for this repo, but the '
```

```
inputs = tokenizer(["Today is"], return_tensors="pt")
inputs
```

```
{'input_ids': tensor([[8888,  318]]), 'attention_mask': tensor([[1, 1]])}
```

Let's generate some text from the model using regular Greedy Search (here is the HuggingFace example documenting this).

```
# Example 1: Print the scores for each token generated with Greedy Search
#tokenizer.pad_token_id = tokenizer.eso_token_id
outputs = model.generate(**inputs, max_new_tokens=10, return_dict_in_generate=True, outpu
transition_scores = model.compute_transition_scores(
    outputs.sequences, outputs.scores, normalize_logits=True
)
# input_length is the length of the input prompt for decoder-only models, like the GPT fa
# encoder-decoder models, like BART or T5.
input_length = 1 if model.config.is_encoder_decoder else inputs.input_ids.shape[1]
generated_tokens = outputs.sequences[:, input_length:]
for tok, score in zip(generated_tokens[0], transition_scores[0]):
    # | token | token string | log probability | probability
    print(f"| {tok:5d} | {tokenizer.decode(tok):8s} | {score.numpy():.3f} | {np.exp(score
```

```
Setting `pad_token_id` to `eos_token_id`:50256 for open-end generation.
|   262 |  the     | -1.414 | 24.33%
|  1110 |  day     | -2.609 | 7.36%
|   618 |  when    | -2.010 | 13.41%
|   356 |  we      | -1.859 | 15.58%
|   460 |  can     | -2.508 | 8.14%
|   477 |  all     | -2.752 | 6.38%
|   307 |  be      | -2.960 | 5.18%
|  6613 |  proud   | -2.135 | 11.82%
|   286 |  of      | -0.558 | 57.21%
|   674 |  our     | -1.472 | 22.96%
```

```
outputs['sequences']
```

```
tensor([[8888,  318,  262, 1110,  618,  356,  460,  477,  307, 6613,  286,  674]])
```

Let's now use Beam Search (again using this example from HF).

```
inputs = tokenizer(["Today is"], return_tensors="pt")

# Approach 2: Beam Search
outputs = model.generate(
    **inputs,
    max_new_tokens=10,
    num_beams=6,
    num_beam_groups=3,
    diversity_penalty=5.0,
    num_return_sequences=6,
    return_dict_in_generate=True,
    output_scores=True,
)
transition_scores = model.compute_transition_scores(
    outputs.sequences, outputs.scores, outputs.beam_indices, normalize_logits=False
)
```

```
Setting `pad_token_id` to `eos_token_id`:50256 for open-end generation.
```

```
outputs['sequences']
```

```
tensor([[8888,  318,  407,  262,  886,  286,  262,  995,   11,  475,  340,  318],
        [8888,  318,  407,  262,  886,  286,  262,  995,   11,  475,  340,  338],
        [8888,  318,  262,  640,  329,  514,  284, 1011,  257, 1302, 1028,  262],
        [8888,  318,  262,  640,  329,  514,  284, 1011,  257, 1302, 1028,  428],
        [8888,  318,  618,  345,  423,  257, 1256,  286,  640,  284,  892,  546],
        [8888,  318,  618,  345,  423,  257, 1256,  286,  640,  284,  892,   13]])
```

```
for s, seq in enumerate(outputs['sequences']):
  print(f"seq {s}: {tokenizer.decode(seq)}")
```

```
seq 0: Today is not the end of the world, but it is
seq 1: Today is not the end of the world, but it's
seq 2: Today is the time for us to take a stand against the
seq 3: Today is the time for us to take a stand against this
seq 4: Today is when you have a lot of time to think about
seq 5: Today is when you have a lot of time to think.
```

Q1: Does the Beam Search above use Diverse Beam Search? If not, change it to use Diverse Beam Search and describe how the output differs.

(Hint: Look a few cells down at the next use of Beam Search, there are two parameters you will need to add, `num_beam_groups`, and `diversity_penalty`)

The outputs differ vastly when num_beams_groups and diversity_penatly are added. Without them, the sentences had very few differences-the last few words. However, those added variables added 3 separate scenarios of possible words after "Today is".

```
prompt = ["Cohn confronted the bullfighter and "]
inputs = tokenizer(prompt, return_tensors="pt")

max_new_toks = 15
# Example 1: Print the scores for each token generated with Greedy Search
#outputs = model.generate(**inputs, max_new_tokens=max_new_toks, return_dict_in_generate=
outputs = model.generate(**inputs, max_new_tokens=max_new_toks, return_dict_in_generate=T
transition_scores = model.compute_transition_scores(
    outputs.sequences, outputs.scores, normalize_logits=True
)
# input_length is the length of the input prompt for decoder-only models, like the GPT fa
# encoder-decoder models, like BART or T5.
input_length = 1 if model.config.is_encoder_decoder else inputs.input_ids.shape[1]
generated_tokens = outputs.sequences[:, input_length:]
for tok, score in zip(generated_tokens[0], transition_scores[0]):
    # | token | token string | log probability | probability
    print(f"| {tok:5d} | {tokenizer.decode(tok):8s} | {score.numpy():.3f} | {np.exp(score
```

```
Setting `pad_token_id` to `eos_token_id`:50256 for open-end generation.
|  3711 | iced     | -0.646 | 52.43%
|   340 |  it      | -1.618 | 19.82%
|   510 |  up      | -0.897 | 40.80%
|    13 | .        | -1.199 | 30.16%
|   198 | 
        | -1.543 | 21.38%
|   198 | 
        | -0.018 | 98.22%
|     1 | "        | -0.677 | 50.79%
|    40 | I        | -1.864 | 15.50%
|  1101 | 'm       | -2.006 | 13.45%
|   407 |  not     | -1.525 | 21.76%
|  1016 |  going   | -1.388 | 24.95%
|   284 |  to      | -0.038 | 96.27%
|  1309 |  let     | -2.831 | 5.90%
|   345 |  you     | -0.957 | 38.42%
|   651 |  get     | -2.149 | 11.66%
```

```
inputs = tokenizer(prompt, return_tensors="pt")
```

```
# Approach 2: Reconstruct the sequence scores from Beam Search
outputs = model.generate(
    **inputs,
    max_new_tokens=max_new_toks,
    num_beams=6,
    num_beam_groups=3,
    diversity_penalty=5.0,
    num_return_sequences=6,
    return_dict_in_generate=True,
    output_scores=True,
    temperature=1.0,
    #do_sample=True
)
transition_scores = model.compute_transition_scores(
    outputs.sequences, outputs.scores, outputs.beam_indices, normalize_logits=False
)
# If you sum the generated tokens' scores and apply the length penalty, you'll get the se
# Tip 1: recomputing the scores is only guaranteed to match with `normalize_logits=False`
# use case, you might want to recompute it with `normalize_logits=True`.
# Tip 2: the output length does NOT include the input length
output_length = np.sum(transition_scores.numpy() < 0, axis=1)
length_penalty = model.generation_config.length_penalty
reconstructed_scores = transition_scores.sum(axis=1) / (output_length**length_penalty)

print(np.allclose(outputs.sequences_scores, reconstructed_scores))

for s, seq in enumerate(outputs['sequences']):
  print(f"seq {s}: {tokenizer.decode(seq)}")
```

```
Setting `pad_token_id` to `eos_token_id`:50256 for open-end generation.
False
seq 0: Cohn confronted the bullfighter and iced him up.

"I'm not going to let you get
seq 1: Cohn confronted the bullfighter and iced it up.

"I'm not going to let you get
seq 2: Cohn confronted the bullfighter and urchin, who had been in a state of shock.

"
seq 3: Cohn confronted the bullfighter and ichthyologist, who had been working on the
seq 4: Cohn confronted the bullfighter and urchin, who had been in a state of shock.

The
seq 5: Cohn confronted the bullfighter and ichthyologist, who had been working on the
<ipython-input-10-2896640e2900>:25: DeprecationWarning: __array_wrap__ must accept co
  reconstructed_scores = transition_scores.sum(axis=1) / (output_length**length_penal
```

Let's now load the raw text from Hemingway's book, *"The Sun also Rises"*.

```
heming = pd.read_csv("https://raw.githubusercontent.com/sgeinitz/DSML4220/main/data/sunal
heming.head()
```

	sentence
0	Robert Cohn was once middleweight boxing champ...
1	Do not think that I am very much impressed by ...
2	He cared nothing for boxing, in fact he dislik...
3	There was a certain inner comfort in knowing h...
4	He was Spider Kelly's star pupil.

Next steps: Generate code with heming View recommended plots New interactive sheet

```
sentences = heming['sentence']
sentences.head()
```

	sentence
0	Robert Cohn was once middleweight boxing champ...
1	Do not think that I am very much impressed by ...
2	He cared nothing for boxing, in fact he dislik...
3	There was a certain inner comfort in knowing h...
4	He was Spider Kelly's star pupil.

dtype: object

```
print(f"        sentence: '{sentences[0]}' \n is tokenized as: {tokenizer.encode(sentence
```

```
         sentence: 'Robert Cohn was once middleweight boxing champion of Princeton.'
 is tokenized as: [19156, 45005, 373, 1752, 3504, 6551, 21576, 8783, 286, 23173, 13]
```

```
max_length = max([len(tokenizer.encode(sentence)) for sentence in sentences])
max_length
```

```
224
```

```
class HemingwayDataset(Dataset):
    def __init__(self, txt_list, tokenizer, max_length):
        self.input_ids = []
        self.attn_masks = []
        self.labels = []
        for txt in txt_list:
            encodings_dict = tokenizer('<|startoftext|>' + txt + '<|endoftext|>', truncat
                                       max_length=max_length, padding="max_length")
            self.input_ids.append(torch.tensor(encodings_dict['input_ids']))
            self.attn_masks.append(torch.tensor(encodings_dict['attention_mask']))

    def __len__(self): # overload the len() Python built-in function
        return len(self.input_ids)

    def __getitem__(self, idx): # overload the [] operator
        return self.input_ids[idx], self.attn_masks[idx]

tokenizer.pad_token_id = tokenizer.eos_token_id


dataset = HemingwayDataset(sentences, tokenizer, max_length=max_length)
train_size = int(0.9 * len(dataset))
train_dataset, val_dataset = random_split(dataset, [train_size, len(dataset) - train_size
```

```
train_dataset[0]
```

```
(tensor([   27,    91,  9688,  1659,  5239,    91,    29,   447,   251,   564,
           250,    54,   261,   447,   247,    83,   340,   307, 37196,    11,
           447,   251, 18726,   531,    13, 50256, 50256, 50256, 50256, 50256,
         50256, 50256, 50256, 50256, 50256, 50256, 50256, 50256, 50256, 50256,
         50256, 50256, 50256, 50256, 50256, 50256, 50256, 50256, 50256, 50256,
         50256, 50256, 50256, 50256, 50256, 50256, 50256, 50256, 50256, 50256,
         50256, 50256, 50256, 50256, 50256, 50256, 50256, 50256, 50256, 50256,
         50256, 50256, 50256, 50256, 50256, 50256, 50256, 50256, 50256, 50256,
         50256, 50256, 50256, 50256, 50256, 50256, 50256, 50256, 50256, 50256,
         50256, 50256, 50256, 50256, 50256, 50256, 50256, 50256, 50256, 50256,
         50256, 50256, 50256, 50256, 50256, 50256, 50256, 50256, 50256, 50256,
         50256, 50256, 50256, 50256, 50256, 50256, 50256, 50256, 50256, 50256,
         50256, 50256, 50256, 50256, 50256, 50256, 50256, 50256, 50256, 50256,
         50256, 50256, 50256, 50256, 50256, 50256, 50256, 50256, 50256, 50256,
         50256, 50256, 50256, 50256, 50256, 50256, 50256, 50256, 50256, 50256,
         50256, 50256, 50256, 50256, 50256, 50256, 50256, 50256, 50256, 50256,
         50256, 50256, 50256, 50256, 50256, 50256, 50256, 50256, 50256, 50256,
         50256, 50256, 50256, 50256, 50256, 50256, 50256, 50256, 50256, 50256,
         50256, 50256, 50256, 50256, 50256, 50256, 50256, 50256, 50256, 50256,
         50256, 50256, 50256, 50256, 50256, 50256, 50256, 50256, 50256, 50256,
         50256, 50256, 50256, 50256, 50256, 50256, 50256, 50256, 50256, 50256,
         50256, 50256, 50256, 50256, 50256, 50256, 50256, 50256, 50256, 50256,
         50256, 50256, 50256, 50256]),
 tensor([1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
         1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
         0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
         0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
         0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
         0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
         0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
         0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
         0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
         0, 0, 0, 0, 0, 0, 0, 0]))
```

Notice that above we set the `pad_token_id` to be the same as the `eos_token_id` (i.e. end-of-stream token id). So all of those `50256` entries above are being used as end-of-stream, or end-of-sequence tokens (except the first one, which is denoting the end of the sequence).

```
tokenizer.decode([50256])
```

```
'<|endoftext|>'
```

```
batch_size = 4
n_epochs = 2
training_args = TrainingArguments(output_dir='~/hemingway_generation', num_train_epochs=n
                                  eval_steps=20, per_device_train_batch_size=batch_size,
                                  warmup_steps=10, weight_decay=0.05, logging_dir='~/hemi
```

Let's load GPT-2 and then take a rough glance at the architecture of GPT (w/ ~130M parameters) by printing the model.

```
model = AutoModelForCausalLM.from_pretrained("openai-community/gpt2")
print(model)
```

```
GPT2LMHeadModel(
  (transformer): GPT2Model(
    (wte): Embedding(50257, 768)
    (wpe): Embedding(1024, 768)
    (drop): Dropout(p=0.1, inplace=False)
    (h): ModuleList(
      (0-11): 12 x GPT2Block(
        (ln_1): LayerNorm((768,), eps=1e-05, elementwise_affine=True)
        (attn): GPT2Attention(
          (c_attn): Conv1D(nf=2304, nx=768)
          (c_proj): Conv1D(nf=768, nx=768)
          (attn_dropout): Dropout(p=0.1, inplace=False)
          (resid_dropout): Dropout(p=0.1, inplace=False)
        )
        (ln_2): LayerNorm((768,), eps=1e-05, elementwise_affine=True)
        (mlp): GPT2MLP(
          (c_fc): Conv1D(nf=3072, nx=768)
          (c_proj): Conv1D(nf=768, nx=3072)
          (act): NewGELUActivation()
          (dropout): Dropout(p=0.1, inplace=False)
        )
      )
    )
    (ln_f): LayerNorm((768,), eps=1e-05, elementwise_affine=True)
  )
  (lm_head): Linear(in_features=768, out_features=50257, bias=False)
)
```

```
def count_trainable_parameters(mod):
    model_parameters = filter(lambda p: p.requires_grad, mod.parameters())
    params = sum([np.prod(p.size()) for p in model_parameters])
    return params

gpt2_params = count_trainable_parameters(model)
print(f"GPT-2 trainable parameters: {gpt2_params}")
```

```
GPT-2 trainable parameters: 124439808
```

```
trainer = Trainer(model=model,  args=training_args, train_dataset=train_dataset, eval_dat
                  data_collator=lambda data: {'input_ids': torch.stack([f[0] for f in dat
                                              'attention_mask': torch.stack([f[1] for f i
                                              'labels': torch.stack([f[0] for f in data])
# on Colab this will take 6+hrs w/ cpu or <10min w/ T4 GPU per epoch
trainer.train()
```

```
`loss_type=None` was set in the config but it is unrecognised.Using the default loss:
```

[3072/3072 17:49, Epoch 2/2]

Step	Training Loss
100	0.994900
200	0.230600
300	0.212100
400	0.222700
500	0.207800
600	0.206900
700	0.211300
800	0.204600
900	0.195000
1000	0.201500
1100	0.180500
1200	0.182700
1300	0.184300
1400	0.197900
1500	0.200600
1600	0.182100
1700	0.169800

```
1800    0.167700
1900    0.177600
2000    0.169400
2100    0.173700
2200    0.166900
2300    0.168800
2400    0.167100
2500    0.181400
2600    0.169900
2700    0.165900
2800    0.180000
2900    0.163500
3000    0.159600
```

```
TrainOutput(global_step=3072, training_loss=0.21227164069811502,
metrics={'train_runtime': 1071.2071, 'train_samples_per_second': 11.469,
'train_steps_per_second': 2.868, 'total_flos': 1404477333504000.0, 'train_loss':
```

```
inputs = tokenizer(prompt, return_tensors="pt").to(device)

# Use Diverse Beam Search
outputs = model.generate(
    **inputs,
    max_new_tokens=max_new_toks,
    num_beams=6,
    num_beam_groups=3,
    diversity_penalty=5.0,
    num_return_sequences=5,
    return_dict_in_generate=True,
    output_scores=True,
    temperature=2.0,
    #do_sample=True
)
transition_scores = model.compute_transition_scores(
    outputs.sequences, outputs.scores, outputs.beam_indices, normalize_logits=False
)

output_length = np.sum(transition_scores.cpu().numpy() < 0, axis=1)
length_penalty = model.generation_config.length_penalty
reconstructed_scores = transition_scores.cpu().sum(axis=1) / (output_length**length_penal

for s, seq in enumerate(outputs['sequences']):
  gen_text = tokenizer.decode(seq)
```

```
    # remove everything from '<|endoftext|>' on at the end of gen_text
    gen_text = gen_text[:gen_text.find('<|endoftext|>')]
    print(f"seq {s}: {gen_text}")
```

```
/usr/local/lib/python3.11/dist-packages/transformers/generation/configuration_utils.p
  warnings.warn(
Setting `pad_token_id` to `eos_token_id`:50256 for open-end generation.
seq 0: Cohn confronted the bullfighter and iced it.
seq 1: Cohn confronted the bullfighter and _______’s bullfighter,” he said.
seq 2: Cohn confronted the bullfighter and ” he said, “I don’t want to g
seq 3: Cohn confronted the bullfighter and _______’s bullfighter.
seq 4: Cohn confronted the bullfighter and __________’’s bull-fighter.
<ipython-input-35-be92d5bf19ba>:22: DeprecationWarning: __array_wrap__ must accept cc
  reconstructed_scores = transition_scores.cpu().sum(axis=1) / (output_length**length
```

Next, let's use LoRA to fine tune the model. We'll load the model again to ensure that the earlier finetuning is not included.

```
# load the model again so that we can use LoRA
model = AutoModelForCausalLM.from_pretrained("openai-community/gpt2")
print(model)
```

```
GPT2LMHeadModel(
  (transformer): GPT2Model(
    (wte): Embedding(50257, 768)
    (wpe): Embedding(1024, 768)
    (drop): Dropout(p=0.1, inplace=False)
    (h): ModuleList(
      (0-11): 12 x GPT2Block(
        (ln_1): LayerNorm((768,), eps=1e-05, elementwise_affine=True)
        (attn): GPT2Attention(
          (c_attn): Conv1D(nf=2304, nx=768)
          (c_proj): Conv1D(nf=768, nx=768)
          (attn_dropout): Dropout(p=0.1, inplace=False)
          (resid_dropout): Dropout(p=0.1, inplace=False)
        )
        (ln_2): LayerNorm((768,), eps=1e-05, elementwise_affine=True)
        (mlp): GPT2MLP(
          (c_fc): Conv1D(nf=3072, nx=768)
          (c_proj): Conv1D(nf=768, nx=3072)
          (act): NewGELUActivation()
          (dropout): Dropout(p=0.1, inplace=False)
        )
      )
    )
    (ln_f): LayerNorm((768,), eps=1e-05, elementwise_affine=True)
  )
  (lm_head): Linear(in_features=768, out_features=50257, bias=False)
)
```

```
target_modules = ["q_proj", "k_proj", "v_proj", "out_proj", "fc_in", "fc_out", "wte", "c_
```

```
lora_config = LoraConfig(
    task_type="CAUSAL_LM",
    inference_mode=False,
    r=16,
    lora_alpha=32,
    target_modules=target_modules,
    lora_dropout=0.01,
    fan_in_fan_out=True
)

lora_model = LoraModel(model, lora_config, "default")
lora_model.model.tie_weights()


print(lora_model)
```

```
LoraModel(
  (model): GPT2LMHeadModel(
    (transformer): GPT2Model(
      (wte): lora.Embedding(
        (base_layer): Embedding(50257, 768)
        (lora_dropout): ModuleDict(
          (default): Dropout(p=0.01, inplace=False)
        )
        (lora_A): ModuleDict()
        (lora_B): ModuleDict()
        (lora_embedding_A): ParameterDict(  (default): Parameter containing: [torch.F
        (lora_embedding_B): ParameterDict(  (default): Parameter containing: [torch.F
        (lora_magnitude_vector): ModuleDict()
      )
      (wpe): Embedding(1024, 768)
      (drop): Dropout(p=0.1, inplace=False)
      (h): ModuleList(
        (0-11): 12 x GPT2Block(
          (ln_1): LayerNorm((768,), eps=1e-05, elementwise_affine=True)
          (attn): GPT2Attention(
            (c_attn): Conv1D(nf=2304, nx=768)
            (c_proj): lora.Linear(
              (base_layer): Conv1D(nf=768, nx=768)
              (lora_dropout): ModuleDict(
                (default): Dropout(p=0.01, inplace=False)
              )
              (lora_A): ModuleDict(
                (default): Linear(in_features=768, out_features=16, bias=False)
              )
              (lora_B): ModuleDict(
                (default): Linear(in_features=16, out_features=768, bias=False)
              )
              (lora_embedding_A): ParameterDict()
              (lora_embedding_B): ParameterDict()
              (lora_magnitude_vector): ModuleDict()
            )
            (attn_dropout): Dropout(p=0.1, inplace=False)
            (resid_dropout): Dropout(p=0.1, inplace=False)
            (resid_dropout): Dropout(p=0.1, inplace=False)
          )
          (ln_2): LayerNorm((768,), eps=1e-05, elementwise_affine=True)
          (mlp): GPT2MLP(
            (c_fc): lora.Linear(
              (base_layer): Conv1D(nf=3072, nx=768)
              (lora_dropout): ModuleDict(
                (default): Dropout(p=0.01, inplace=False)
              )
              (lora_A): ModuleDict(
                (default): Linear(in_features=768, out_features=16, bias=False)
              )
              (lora_B): ModuleDict(
                (default): Linear(in_features=16, out_features=3072, bias=False)
              )
              (lora_embedding_A): ParameterDict()
              (lora_embedding_B): ParameterDict()
              (lora_magnitude_vector): ModuleDict()
            )
            (c_proj): lora.Linear(
              (base_layer): Conv1D(nf=768, nx=3072)
```

```
trainer = Trainer(model=lora_model,  args=training_args, train_dataset=train_dataset, eva
                  data_collator=lambda data: {'input_ids': torch.stack([f[0] for f in dat
                                              'attention_mask': torch.stack([f[1] for f i
                                              'labels': torch.stack([f[0] for f in data])
```

```
trainer.train()
```

[3072/3072 15:11, Epoch 2/2]

Step	Training Loss
100	3.876500
200	0.338900
300	0.258100
400	0.260600
500	0.243400
600	0.238700
700	0.243500
800	0.233500
900	0.225900
1000	0.230300
1100	0.205000
1200	0.207400

```
1300    0.209100
1400    0.223200
1500    0.226600
1600    0.224000
1700    0.214500
1800    0.212300
1900    0.223900
2000    0.214400
2100    0.218300
2200    0.209300
2300    0.212900
2400    0.210100
2500    0.230000
2600    0.212500
2700    0.211300
2800    0.227100
2900    0.205400
3000    0.201500
```

```
TrainOutput(global_step=3072, training_loss=0.3452523198599617,
metrics={'train_runtime': 912.0541, 'train_samples_per_second': 13.471,
'train_steps_per_second': 3.368, 'total_flos': 1447176244942848.0, 'train_loss':
```

Q2: How many more `training_samples_per_second` could the LoRA model get through during finetuning than the original GPT-2 model could?

The LoRA was able to get through 2 more training_samples_per_second than the GPT-2 model.

```
inputs = tokenizer(prompt, return_tensors="pt").to(device)

# Use Diverse Beam Search
outputs = lora_model.generate(
    **inputs,
    max_new_tokens=max_new_toks,
    num_beams=6,
    num_beam_groups=3,
    diversity_penalty=5.0,
    num_return_sequences=5,
    return_dict_in_generate=True,
    output_scores=True,
    temperature=1.5,
    #do_sample=True
)
transition_scores = lora_model.compute_transition_scores(
    outputs.sequences, outputs.scores, outputs.beam_indices, normalize_logits=False
)

output_length = np.sum(transition_scores.cpu().numpy() < 0, axis=1)
length_penalty = lora_model.generation_config.length_penalty
reconstructed_scores = transition_scores.cpu().sum(axis=1) / (output_length**length_penal

for s, seq in enumerate(outputs['sequences']):
  gen_text = tokenizer.decode(seq)
  # remove everything from '<|endoftext|>' to the end from gen_text
  gen_text = gen_text[:gen_text.find('<|endoftext|>')]
  print(f"seq {s}: {gen_text}")
```

```
/usr/local/lib/python3.11/dist-packages/transformers/generation/configuration_utils.p
  warnings.warn(
Setting `pad_token_id` to `eos_token_id`:50256 for open-end generation.
seq 0: Cohn confronted the bullfighter and iced it.
seq 1: Cohn confronted the bullfighter and iced the bull.
seq 2: Cohn confronted the bullfighter and  said, ''I don’t care about the bullfighte
seq 3: Cohn confronted the bullfighter and  said, ''I don’t care about the bullfight
seq 4: Cohn confronted the bullfighter and ’d the bullfighter,” the bullfighter said,
<ipython-input-36-88065fb89208>:22: DeprecationWarning: __array_wrap__ must accept cc
  reconstructed_scores = transition_scores.cpu().sum(axis=1) / (output_length**lengtl
```

```
lora_params = count_trainable_parameters(lora_model)
print(f"LoRA trainable parameters: {lora_params} ({(100*lora_params/gpt2_params):.2f}% of
```

```
LoRA trainable parameters: 2585872 (2.08% of GPT-2's trainable parameters)
```

Q3: How many fewer parameters did the LoRA model need to train/tune than the full GPT-2 model did?

(Hint: See output from above cell)

It used 2.08% of GPT-2's trainable parameters.