Running into a `CUDA out of memory` error is common when working with large models or datasets. Here are several strategies you can employ to mitigate this issue:

### Strategies to Fix CUDA Out of Memory Error:

1. \*\*Reduce Batch Size:\*\*

The batch size directly impacts memory usage. Reducing the batch size can significantly lower the memory requirements.

```python

training\_args = TrainingArguments(

output\_dir='supervised-summarize-checkpoint',

learning\_rate=1e-5,

per\_device\_train\_batch\_size=8, # Reduce batch size

fp16=False,

gradient\_accumulation\_steps=1,

num\_train\_epochs=2,

warmup\_steps=100,

logging\_steps=10,

)

```

2. \*\*Enable Mixed Precision Training:\*\*

Mixed precision training uses both 16-bit and 32-bit floating point numbers, which can reduce memory usage and speed up training.

```python

training\_args = TrainingArguments(

output\_dir='supervised-summarize-checkpoint',

learning\_rate=1e-5,

per\_device\_train\_batch\_size=16,

fp16=True, # Enable mixed precision training

gradient\_accumulation\_steps=1,

num\_train\_epochs=2,

warmup\_steps=100,

logging\_steps=10,

)

```

3. \*\*Gradient Accumulation:\*\*

Gradient accumulation allows you to effectively increase the batch size without increasing memory usage by accumulating gradients over multiple steps.

```python

training\_args = TrainingArguments(

output\_dir='supervised-summarize-checkpoint',

learning\_rate=1e-5,

per\_device\_train\_batch\_size=8,

gradient\_accumulation\_steps=2, # Accumulate gradients over multiple steps

fp16=False,

num\_train\_epochs=2,

warmup\_steps=100,

logging\_steps=10,

)

```

4. \*\*Clear GPU Cache:\*\*

Clear the GPU cache before starting training to ensure there are no unnecessary memory allocations.

```python

torch.cuda.empty\_cache()

```

5. \*\*Use Smaller Model:\*\*

If feasible, use a smaller version of the model.

6. \*\*Ensure Proper Device Placement:\*\*

Ensure that the model and data are properly placed on the GPU.

```python

model = AutoModelForCausalLM.from\_pretrained("bigcode/tiny\_starcoder\_py", use\_cache=False).to("cuda:0")

```

Here is your updated code with reduced batch size, gradient accumulation, and clearing GPU cache:

```python

import random

import numpy as np

import torch

import pandas as pd

from transformers import (

AutoModelForCausalLM,

AutoTokenizer,

Trainer,

TrainingArguments,

default\_data\_collator,

)

def set\_seed(seed\_val=42):

random.seed(seed\_val)

np.random.seed(seed\_val)

torch.manual\_seed(seed\_val)

torch.cuda.manual\_seed\_all(seed\_val)

train\_batch\_size = 8 # Reduced batch size

gradient\_accumulation\_steps = 2 # Accumulate gradients

learning\_rate = 1e-5

eval\_batch\_size = 1

eval\_steps = 500

max\_input\_length = 550

save\_steps = 1000

num\_train\_epochs = 20

random.seed(42)

from datasets import load\_dataset

import json

import pandas as pd

import torch

from datasets import load\_dataset

from torch.utils.data import Dataset

class TLDRDataset(Dataset):

def \_\_init\_\_(self, train\_path, tokenizer, split, max\_length=256):

self.post\_list = []

dataset = pd.read\_parquet(train\_path)

self.labels = []

for sample in dataset.iterrows():

self.post\_list.append(sample[1]["prompt"])

self.labels.append(sample[1]["label"])

self.tokenizer = tokenizer

self.max\_length = max\_length

self.input\_ids = []

self.attn\_masks = []

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

return len(self.post\_list)

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

txt = self.post\_list[idx]

label = self.labels[idx]

encodings\_dict = self.tokenizer(txt, truncation=True, max\_length=self.max\_length, padding="max\_length")

encodings\_dict\_label = self.tokenizer(label, truncation=True, max\_length=self.max\_length, padding="max\_length")

input\_ids = torch.tensor(encodings\_dict["input\_ids"])

attn\_masks = torch.tensor(encodings\_dict["attention\_mask"])

labels\_ids = torch.tensor(encodings\_dict\_label["input\_ids"])

return {

"input\_ids": input\_ids,

"attention\_mask": attn\_masks,

"labels": labels\_ids,

}

print(torch.cuda.is\_available()) # Check if CUDA is available

print(torch.cuda.device\_count()) # Check the number of available CUDA devices

tokenizer = AutoTokenizer.from\_pretrained("bigcode/tiny\_starcoder\_py")

model = AutoModelForCausalLM.from\_pretrained("bigcode/tiny\_starcoder\_py", use\_cache=False).to("cuda:0")

tokenizer.pad\_token = tokenizer.eos\_token

model.resize\_token\_embeddings(len(tokenizer))

tokenizer.pad\_token\_id = tokenizer.eos\_token\_id

model.config.end\_token\_id = tokenizer.eos\_token\_id

model.config.pad\_token\_id = model.config.eos\_token\_id

data\_path = "data/test-00000-of-00001-59ffb27399371eac.parquet"

train\_dataset = TLDRDataset(

data\_path,

tokenizer,

"train",

max\_length=256,

)

for i in train\_dataset:

print(i["input\_ids"], i["labels"])

break

torch.cuda.set\_device(0)

torch.cuda.empty\_cache() # Clear GPU cache

# Prepare the trainer and start training

training\_args = TrainingArguments(

output\_dir='supervised-summarize-checkpoint',

learning\_rate=1e-5,

per\_device\_train\_batch\_size=train\_batch\_size,

fp16=True, # Enable mixed precision training

gradient\_accumulation\_steps=gradient\_accumulation\_steps,

num\_train\_epochs=2,

warmup\_steps=100,

logging\_steps=10,

)

trainer = Trainer(

model=model,

args=training\_args,

train\_dataset=train\_dataset,

)

trainer.train()

```

This code reduces the batch size, enables gradient accumulation, and clears the GPU cache before training. Additionally, it enables mixed precision training to further reduce memory usage.

**Response 2**

If you still encounter `CUDA out of memory` errors despite the previous adjustments, here are a few more strategies you can try:

### Further Strategies:

1. \*\*Use Smaller Batch Size:\*\*

Further reduce the batch size. Even smaller values like 2 or 4 can help.

2. \*\*Reduce Sequence Length:\*\*

Lower the `max\_input\_length` if possible, which reduces the memory footprint per sample.

3. \*\*Check GPU Usage:\*\*

Ensure no other processes are using GPU memory.

4. \*\*Clear Cache Regularly:\*\*

Regularly clear the cache during training.

5. \*\*Use CPU for Evaluation:\*\*

If memory is still insufficient, consider using the CPU for evaluation steps.

### Updated Code with Additional Adjustments:

```python

import random

import numpy as np

import torch

import pandas as pd

from transformers import (

AutoModelForCausalLM,

AutoTokenizer,

Trainer,

TrainingArguments,

default\_data\_collator,

)

def set\_seed(seed\_val=42):

random.seed(seed\_val)

np.random.seed(seed\_val)

torch.manual\_seed(seed\_val)

torch.cuda.manual\_seed\_all(seed\_val)

# Reduce batch size and max input length

train\_batch\_size = 4 # Further reduced batch size

gradient\_accumulation\_steps = 4 # Increase accumulation steps to maintain effective batch size

learning\_rate = 1e-5

eval\_batch\_size = 1

eval\_steps = 500

max\_input\_length = 256 # Reduce sequence length

save\_steps = 1000

num\_train\_epochs = 20

random.seed(42)

from datasets import load\_dataset

import json

import pandas as pd

import torch

from datasets import load\_dataset

from torch.utils.data import Dataset

class TLDRDataset(Dataset):

def \_\_init\_\_(self, train\_path, tokenizer, split, max\_length=256):

self.post\_list = []

dataset = pd.read\_parquet(train\_path)

self.labels = []

for sample in dataset.iterrows():

self.post\_list.append(sample[1]["prompt"])

self.labels.append(sample[1]["label"])

self.tokenizer = tokenizer

self.max\_length = max\_length

self.input\_ids = []

self.attn\_masks = []

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

return len(self.post\_list)

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

txt = self.post\_list[idx]

label = self.labels[idx]

encodings\_dict = self.tokenizer(txt, truncation=True, max\_length=self.max\_length, padding="max\_length")

encodings\_dict\_label = self.tokenizer(label, truncation=True, max\_length=self.max\_length, padding="max\_length")

input\_ids = torch.tensor(encodings\_dict["input\_ids"])

attn\_masks = torch.tensor(encodings\_dict["attention\_mask"])

labels\_ids = torch.tensor(encodings\_dict\_label["input\_ids"])

return {

"input\_ids": input\_ids,

"attention\_mask": attn\_masks,

"labels": labels\_ids,

}

print(torch.cuda.is\_available()) # Check if CUDA is available

print(torch.cuda.device\_count()) # Check the number of available CUDA devices

tokenizer = AutoTokenizer.from\_pretrained("bigcode/tiny\_starcoder\_py")

model = AutoModelForCausalLM.from\_pretrained("bigcode/tiny\_starcoder\_py", use\_cache=False).to("cuda:0")

tokenizer.pad\_token = tokenizer.eos\_token

model.resize\_token\_embeddings(len(tokenizer))

tokenizer.pad\_token\_id = tokenizer.eos\_token\_id

model.config.end\_token\_id = tokenizer.eos\_token\_id

model.config.pad\_token\_id = model.config.eos\_token\_id

data\_path = "data/test-00000-of-00001-59ffb27399371eac.parquet"

train\_dataset = TLDRDataset(

data\_path,

tokenizer,

"train",

max\_length=max\_input\_length,

)

for i in train\_dataset:

print(i["input\_ids"], i["labels"])

break

torch.cuda.set\_device(0)

torch.cuda.empty\_cache() # Clear GPU cache

# Prepare the trainer and start training

training\_args = TrainingArguments(

output\_dir='supervised-summarize-checkpoint',

learning\_rate=1e-5,

per\_device\_train\_batch\_size=train\_batch\_size,

fp16=True, # Enable mixed precision training

gradient\_accumulation\_steps=gradient\_accumulation\_steps,

num\_train\_epochs=2,

warmup\_steps=100,

logging\_steps=10,

)

trainer = Trainer(

model=model,

args=training\_args,

train\_dataset=train\_dataset,

)

trainer.train()

```

### Explanation of Adjustments:

1. \*\*Reduced Batch Size\*\*: Lowered to 4 to significantly reduce memory usage.

2. \*\*Increased Gradient Accumulation\*\*: Set to 4 to maintain effective batch size without increasing memory usage.

3. \*\*Reduced Max Input Length\*\*: Reduced to 256 to lower the memory footprint per sample.

4. \*\*Clearing Cache Regularly\*\*: Ensuring GPU cache is cleared before training.

If you continue to face issues, consider these additional steps:

- Monitor GPU usage to ensure no other processes are consuming memory.

- Restart the kernel to free up memory.

- Use a smaller model if available.

These steps should help mitigate the CUDA memory issues.