project2_part_a

November 14, 2023

[1]:
```
import numpy as np
import torch
import json
import random
from transformers import GPT2Tokenizer, GPT2LMHeadModel
from mingpt.model import GPT
from datasets import load_dataset
import os
import matplotlib.pyplot as plt
from torch.utils.data import Dataset, DataLoader
from datasets import load_dataset
```

[12]:
```
dataset = load_dataset("togethercomputer/RedPajama-Data-1T-Sample",␣
↪'plain_text', cache_dir='datasets')
```

[13]:
```
class RedPajamaGPTDataset(Dataset):
    def __init__(self, dataset, split, max_length=1024):
        self.data = dataset[split]
        self.tokenizer = GPT2Tokenizer.from_pretrained("gpt2")
        self.vocab_size = self.tokenizer.vocab_size

        self.max_length = max_length
        self.tokenizer.pad_token_id = self.vocab_size-1

        # Add padding token
        #self.tokenizer.add_special_tokens({'pad_token': 'PAD'})

    def __len__(self):
        return len(self.data)

    def __getitem__(self, idx):
        text = self.data[idx]['text']

        # Tokenize the text
        inputs = self.tokenizer(text, return_tensors="pt", truncation=True,␣
↪max_length=self.max_length, padding='max_length')

        # Split tokens into chunks
```

```
        input_ids = inputs.input_ids[:,:-1]
        input_chunks = torch.split(input_ids, self.max_length)

        # Generate targets
        target_chunks = []
        for chunk in input_chunks:
            target_chunk = chunk[:,1:].clone()  # Exclude [CLS] token
            target_chunks.append(target_chunk)
        result_targets = inputs.input_ids[:,1:]

        return input_ids, result_targets


red_pajama_gpt2_dataset = RedPajamaGPTDataset(dataset=dataset,␣
 ↪split='train',max_length=1024)
dataloader = DataLoader(red_pajama_gpt2_dataset, batch_size=2, shuffle=True)

#for input_chunks, attention_chunks, target_chunks in dataloader:
    # Process each batch here
    #print("Input Chunks:", input_chunks)
    #print("Attention Chunks:", attention_chunks)
    #print("Target Chunks:", target_chunks)
```

[2]:
```
data = []
with open('proj2_data.jsonl', 'r') as jsonl_file:
    for line in jsonl_file:
        data.append(json.loads(line))
```

[3]:
```
class CustomThePileDataset(Dataset):
    def __init__(self, dataset, max_length=1024):
        self.data = dataset
        self.tokenizer = GPT2Tokenizer.from_pretrained("gpt2")
        self.vocab_size = self.tokenizer.vocab_size
        self.max_length = max_length
        self.tokenizer.pad_token_id = self.vocab_size-1

    def __len__(self):
        return len(self.data)

    def __getitem__(self, idx):
        text = self.data[idx]['text']

        # Tokenize the text
        inputs = self.tokenizer(text, return_tensors="pt", truncation=True,␣
 ↪max_length=self.max_length, padding='max_length')

        # Split tokens into chunks
```

```
        input_ids = inputs.input_ids[:,:-1]
        input_chunks = torch.split(input_ids, self.max_length)

        # Generate targets
        target_chunks = []
        for chunk in input_chunks:
            target_chunk = chunk[:,1:].clone()  # Exclude [CLS] token
            target_chunks.append(target_chunk)
        result_targets = inputs.input_ids[:,1:]

        return input_ids, result_targets


thepile_dataset = CustomThePileDataset(dataset=data,max_length=1024)
```

[16]:
```
red_pajama_gpt2_dataset.__getitem__(1)
```

[16]:
```
(tensor([[  59, 5458,   90,  …,   82,  503,  326]]),
 tensor([[ 5458,    90, 21906,  …,   503,   326,   287]]))
```

[4]:
```
thepile_dataset.__getitem__(1)
```

[4]:
```
(tensor([[ 8818,  2010,   796,  …, 50256, 50256, 50256]]),
 tensor([[ 2010,   796,   651,  …, 50256, 50256, 50256]]))
```

1 From demo.py

1.1 RedPajama

[18]:
```
from mingpt.model import GPT
import os

checkpoint_dir = 'redpajama'
dir_path = f'./checkpoints/{checkpoint_dir}'

if not os.path.exists(dir_path):
    os.makedirs(dir_path)
    checkpoints = os.listdir(dir_path)
else:
    checkpoints = os.listdir(dir_path)

checkpoints.sort()

model_config = GPT.get_default_config()
model_config.model_type = 'gpt-nano'
model_config.vocab_size = red_pajama_gpt2_dataset.vocab_size
model_config.block_size = red_pajama_gpt2_dataset.max_length
```

```
model_config.checkpoint = f'checkpoints/{checkpoint_dir}/' + checkpoints[-1] if␣
 ↪checkpoints else None
model = GPT(model_config)
```

```
number of parameters: 2.55M
```

[19]:
```
# create a Trainer object
from mingpt.trainer import Trainer

train_config = Trainer.get_default_config()
train_config.learning_rate = 5e-4 # the model we're using is so small that we␣
 ↪can go a bit faster
train_config.max_iters = 900 + model.iter_num if model_config.checkpoint else␣
 ↪900
train_config.num_workers = 0
train_config.checkpoint_iters = 200
train_config.batch_size = 1
train_config.checkpoint_name = f'{checkpoint_dir}/checkpoint'
trainer = Trainer(train_config, model, red_pajama_gpt2_dataset)
```

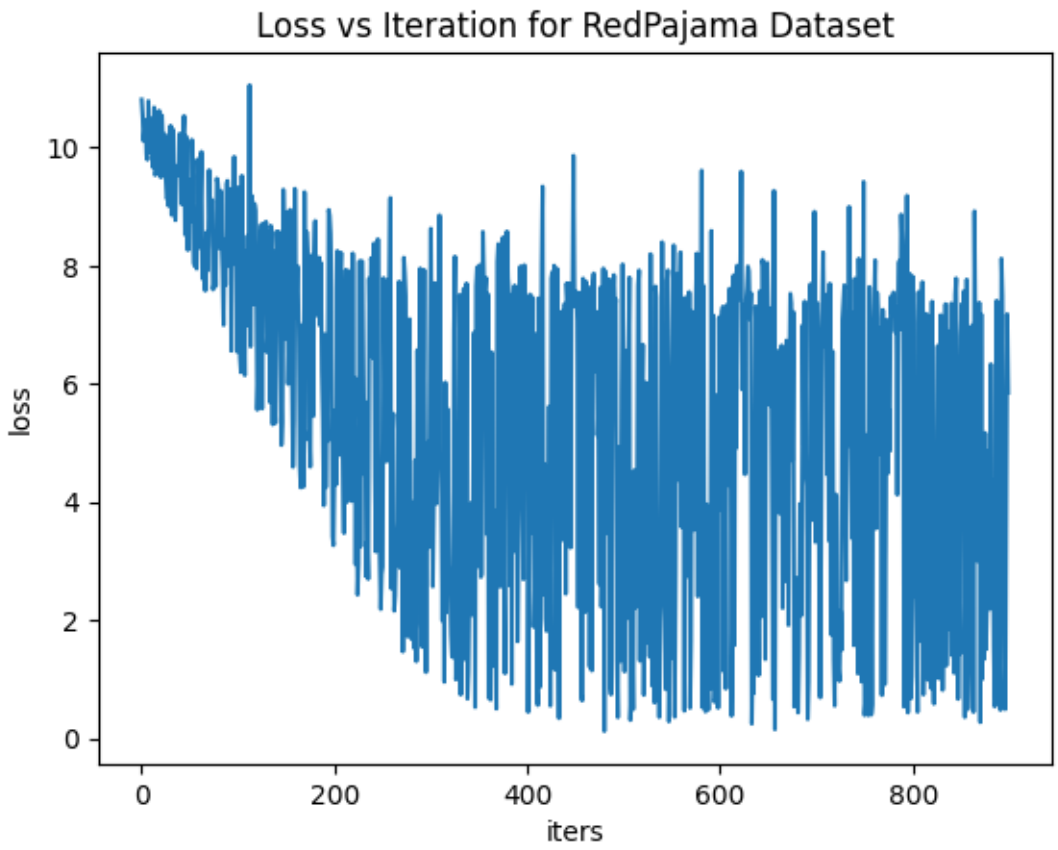
```
running on device cpu
```

[20]:
```
def batch_end_callback(trainer):
    if trainer.iter_num % 100 == 0:
        print(f"iter_dt {trainer.iter_dt * 1000:.2f}ms; iter {trainer.iter_num}:
 ↪ train loss {trainer.loss.item():.5f}")
trainer.set_callback('on_batch_end', batch_end_callback)

trainer.run()
```

```
iter_dt 0.00ms; iter 0: train loss 10.80981
iter_dt 803.53ms; iter 100: train loss 9.32378
iter_dt 794.02ms; iter 200: train loss 4.72923
iter_dt 812.34ms; iter 300: train loss 8.62620
iter_dt 828.25ms; iter 400: train loss 7.45123
iter_dt 963.42ms; iter 500: train loss 6.00072
iter_dt 888.50ms; iter 600: train loss 3.95060
iter_dt 938.74ms; iter 700: train loss 6.80541
iter_dt 1208.27ms; iter 800: train loss 7.81231
```

[21]:
```
plt.plot(np.arange(len(trainer.curr_loss)), trainer.curr_loss)
plt.title('Loss vs Iteration for RedPajama Dataset')
plt.xlabel('iters')
plt.ylabel('loss')
plt.show()
```



2 ThePile (chopped)

[5]:
```
from mingpt.model import GPT
import os

checkpoint_dir = 'thepile'
dir_path = f'./checkpoints/{checkpoint_dir}'

if not os.path.exists(dir_path):
    os.makedirs(dir_path)
    checkpoints = os.listdir(dir_path)
else:
    checkpoints = os.listdir(dir_path)

checkpoints.sort()

model_config = GPT.get_default_config()
model_config.model_type = 'gpt-nano'
model_config.vocab_size = thepile_dataset.vocab_size
```

```
model_config.block_size = thepile_dataset.max_length
model_config.checkpoint = f'checkpoints/{checkpoint_dir}/' + checkpoints[-1] if␣
 ↪checkpoints else None
model = GPT(model_config)
```

```
number of parameters: 2.55M
```

[6]:
```
# create a Trainer object
from mingpt.trainer import Trainer

train_config = Trainer.get_default_config()
train_config.learning_rate = 5e-4 # the model we're using is so small that we␣
 ↪can go a bit faster
train_config.max_iters = 600 + model.iter_num if model_config.checkpoint else␣
 ↪600
train_config.num_workers = 0
train_config.checkpoint_iters = 200
train_config.batch_size = 1
train_config.checkpoint_name = f'{checkpoint_dir}/checkpoint'
trainer = Trainer(train_config, model, thepile_dataset)
```

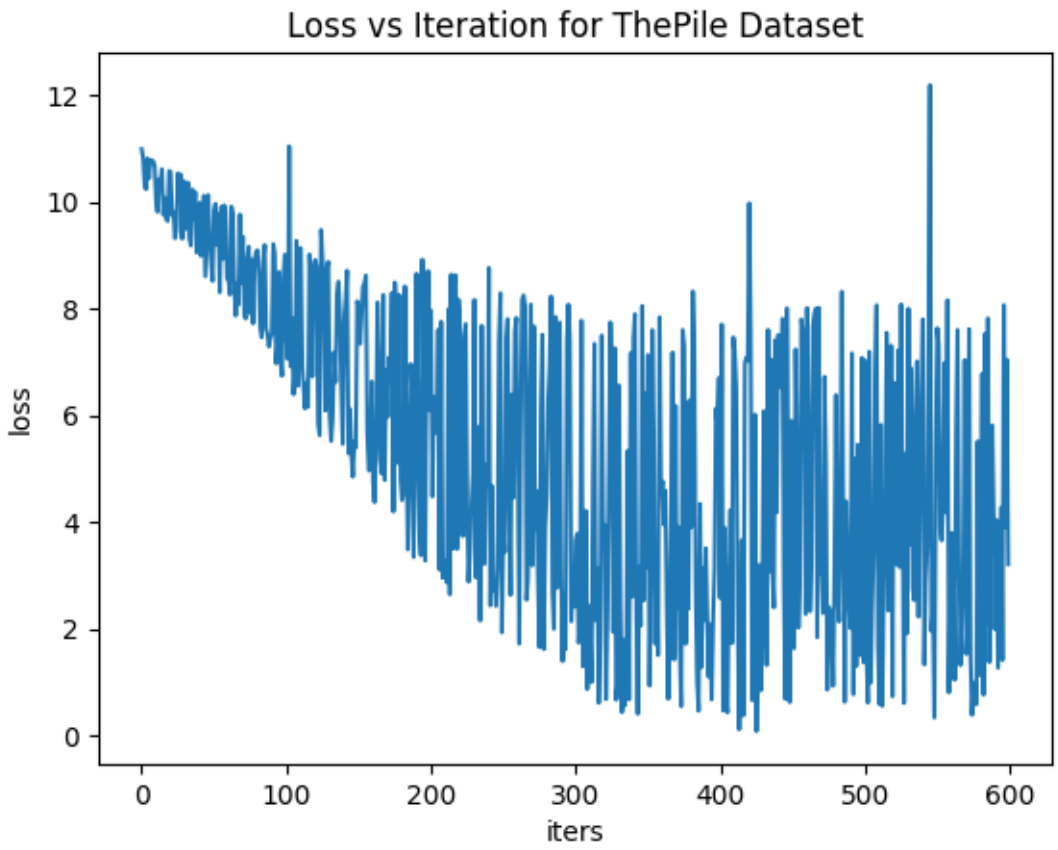
```
running on device cpu
```

[7]:
```
def batch_end_callback(trainer):
    if trainer.iter_num % 100 == 0:
        print(f"iter_dt {trainer.iter_dt * 1000:.2f}ms; iter {trainer.iter_num}:
 ↪ train loss {trainer.loss.item():.5f}")
trainer.set_callback('on_batch_end', batch_end_callback)

trainer.run()
```

```
iter_dt 0.00ms; iter 0: train loss 10.99453
iter_dt 1007.05ms; iter 100: train loss 7.48162
iter_dt 934.71ms; iter 200: train loss 7.96684
iter_dt 1032.47ms; iter 300: train loss 3.65589
iter_dt 1063.77ms; iter 400: train loss 2.57136
iter_dt 1149.47ms; iter 500: train loss 7.02844
```

[11]:
```
plt.plot(np.arange(len(trainer.curr_loss)), trainer.curr_loss)
plt.title('Loss vs Iteration for ThePile Dataset')
plt.xlabel('iters')
plt.ylabel('loss')
plt.show()
```


Loss vs Iteration for ThePile Dataset
loss
iters
0
2
4
6
8
10
12
0
100
200
300
400
500
600

[ ]: