CS 505 Homework 06: Transformers

Problem Three

See the problem one notebook for details on due date, submission, etc.

As with problems one and two , there is an extensive tutorial section, followed by some tasks at the end you need to complete.

Full Disclosure: This is based on Rey Farhan's post:

(https://reyfarhan.com/posts/easy-gpt2-finetuning-huggingface/ (https://reyfarhan.com/posts/easy-gpt2-finetuning-huggingface/))

Introduction

In this problem, we will learn about fine-tuning GPT2 using Hugging Face's Transformers library (https://huggingface.co/transformers/) and PyTorch on raw data.

We start with a simplified script for fine-tuning GPT-2, and at the end, your task will be to modify this assignment on raw text of your choice (I have put Pride and Prejudice as an example below) and include 10 sample generations from your chosen text that you find interesting.

Setup

In [1]:
```
!pip install transformers
```

```
Requirement already satisfied: transformers in /usr/local/lib/python3.1
0/dist-packages (4.35.2)
Requirement already satisfied: filelock in /usr/local/lib/python3.10/di
st-packages (from transformers) (3.13.1)
Requirement already satisfied: huggingface-hub<1.0,>=0.16.4 in /usr/loc
al/lib/python3.10/dist-packages (from transformers) (0.19.4)
Requirement already satisfied: numpy>=1.17 in /usr/local/lib/python3.1
0/dist-packages (from transformers) (1.23.5)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python
3.10/dist-packages (from transformers) (23.2)
Requirement already satisfied: pyyaml>=5.1 in /usr/local/lib/python3.1
0/dist-packages (from transformers) (6.0.1)
Requirement already satisfied: regex!=2019.12.17 in /usr/local/lib/pyth
on3.10/dist-packages (from transformers) (2023.6.3)
Requirement already satisfied: requests in /usr/local/lib/python3.10/di
st-packages (from transformers) (2.31.0)
Requirement already satisfied: tokenizers<0.19,>=0.14 in /usr/local/li
b/python3.10/dist-packages (from transformers) (0.15.0)
Requirement already satisfied: safetensors>=0.3.1 in /usr/local/lib/pyt
hon3.10/dist-packages (from transformers) (0.4.1)
Requirement already satisfied: tqdm>=4.27 in /usr/local/lib/python3.10/
dist-packages (from transformers) (4.66.1)
Requirement already satisfied: fsspec>=2023.5.0 in /usr/local/lib/pytho
n3.10/dist-packages (from huggingface-hub<1.0,>=0.16.4->transformers)
(2023.6.0)
Requirement already satisfied: typing-extensions>=3.7.4.3 in /usr/loca
l/lib/python3.10/dist-packages (from huggingface-hub<1.0,>=0.16.4->tran
sformers) (4.5.0)
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/l
ib/python3.10/dist-packages (from requests->transformers) (3.3.2)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.1
0/dist-packages (from requests->transformers) (3.6)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/pyt
hon3.10/dist-packages (from requests->transformers) (2.0.7)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/pyt
hon3.10/dist-packages (from requests->transformers) (2023.11.17)
```

In [2]:
```
import os
import time
import datetime
from google.colab import drive

import pandas as pd
import seaborn as sns
import numpy as np
import random

import matplotlib.pyplot as plt
%matplotlib inline

import torch
from torch.utils.data import Dataset, DataLoader, random_split, RandomSa
torch.manual_seed(42)

from transformers import GPT2LMHeadModel,  GPT2Tokenizer, GPT2Config, GP
from transformers import AdamW, get_linear_schedule_with_warmup

import nltk
nltk.download('punkt')
```

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt.zip.
```

Out[2]:
```
True
```

In [ ]:
```
!nvidia-smi
```

Create Training Set

In [4]:
```
# mount my Google Drive directory and access the training data located t
gdrive_dir = '/content/gdrive/'
data_dir = os.path.join(gdrive_dir, "'My Drive'")
filename = 'Jane_Eyre.txt'

drive.mount(gdrive_dir, force_remount=True)
```

```
Mounted at /content/gdrive/
```

In [5]:
```
# copy the data to the current Colab working directory
!cp $data_dir/$filename .
```

```
cp: cannot stat '/content/gdrive/My Drive/Jane_Eyre.txt': No such file
or directory
```

In [6]:
```
f=open(filename)
docs=f.readlines()
docs=[b.strip() for b in docs]
docs[:10]
```

Out[6]:
```
['There was no possibility of taking a walk that day. We had been',
 'wandering, indeed, in the leafless shrubbery an hour in the mornin
g;',
 'but since dinner (Mrs. Reed, when there was no company, dined earl
y)',
 'the cold winter wind had brought with it clouds so sombre, and a rai
n',
 'so penetrating, that further outdoor exercise was now out of the',
 'question.',
 '',
 'I was glad of it: I never liked long walks, especially on chilly',
 'afternoons: dreadful to me was the coming home in the raw twilight,',
 'with nipped fingers and toes, and a heart saddened by the chidings o
f']
```

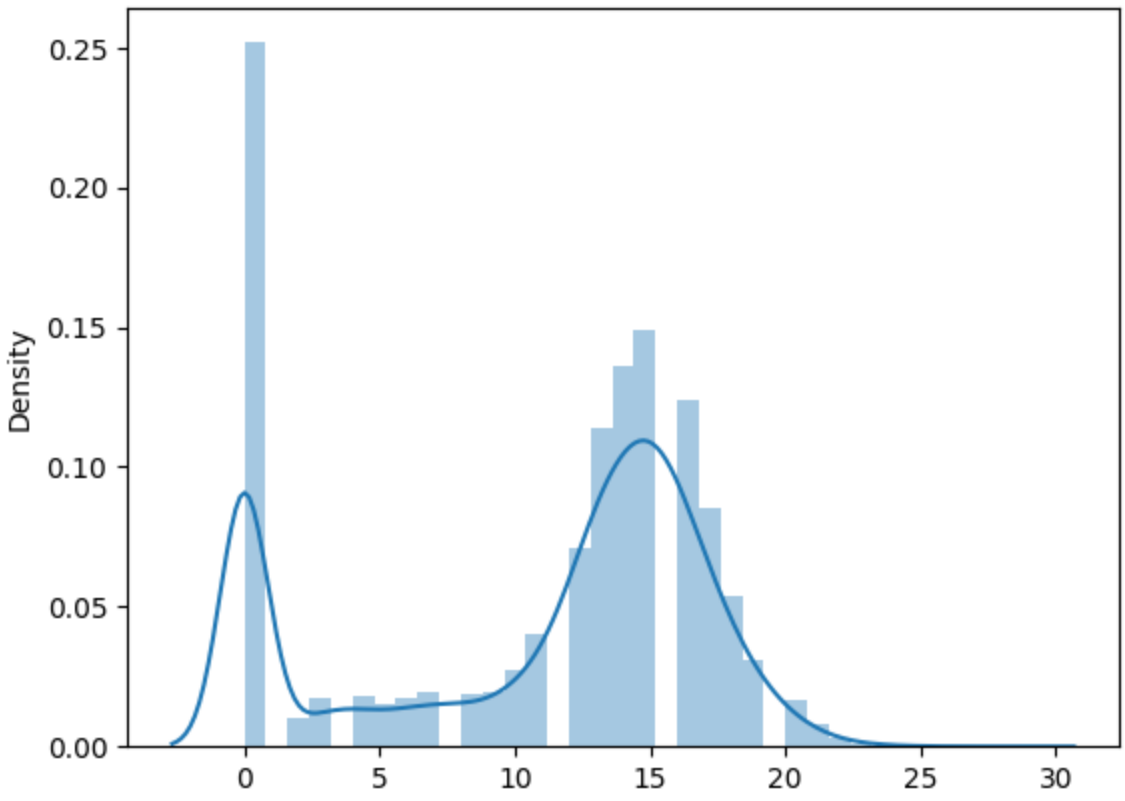
We need to get an idea of how long our training documents are.

I'm not going to use the same tokenizer as the GPT2 one, which is a byte pair encoding tokenizer (https://blog.floydhub.com/tokenization-nlp/). Instead, I'm using a simple one just to get a rough understanding.

In [7]:
```
doc_lengths = []

for doc in docs:

    # get rough token count distribution
    tokens = nltk.word_tokenize(doc)

    doc_lengths.append(len(tokens))

doc_lengths = np.array(doc_lengths)

sns.distplot(doc_lengths)
```

```
<ipython-input-7-0dc1d487ac7d>:12: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.1
4.0.

Please adapt your code to use either `displot` (a figure-level function
with
similar flexibility) or `histplot` (an axes-level function for histogra
ms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751 (http
s://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751)

  sns.distplot(doc_lengths)
```

Out[7]:
```
<Axes: ylabel='Density'>
```



In [8]:
```
# the max token length
len(doc_lengths[doc_lengths > 768])/len(doc_lengths)
```

Out[8]:
```
0.0
```

In [9]:
```
np.average(doc_lengths)
```

Out[9]:
```
10.867097458648791
```

Even though these token counts won't match up to the BPE tokenizer's, I'm confident that most lines will be fit under the 768 embedding size limit for the small GPT2 model.

GPT2 Tokenizer

Although the defaults take care of this,I thought I'd show that you can specify some of the special tokens.

In [10]:
```
# Load the GPT tokenizer.
tokenizer = GPT2Tokenizer.from_pretrained('gpt2', bos_token='<|startofte
```

```
vocab.json:   0%|          | 0.00/1.04M [00:00<?, ?B/s]

merges.txt:   0%|          | 0.00/456k [00:00<?, ?B/s]

tokenizer.json:   0%|          | 0.00/1.36M [00:00<?, ?B/s]

config.json:   0%|          | 0.00/665 [00:00<?, ?B/s]
```

In [11]:
```
print("The max model length is {} for this model, although the actual eml
print("The beginning of sequence token {} token has the id {}".format(to
print("The end of sequence token {} has the id {}".format(tokenizer.conv
print("The padding token {} has the id {}".format(tokenizer.convert_ids_
```

```
The max model length is 1024 for this model, although the actual embedd
ing size for GPT small is 768
The beginning of sequence token <|startoftext|> token has the id 50257
The end of sequence token <|endoftext|> has the id 50256
The padding token <|pad|> has the id 50258
```

PyTorch Datasets & Dataloaders

GPT2 is a large model. Increasing the batch size above 2 has lead to out of memory problems. This can be mitigated by accumulating the gradients but that is out of scope here.

In [12]:
```
batch_size = 2
```

I'm using the standard PyTorch approach of loading data in using a dataset class (https://pytorch.org/tutorials/beginner/data_loading_tutorial.html).

I'm passing in the tokenizer as an argument but normally I would instantiate it within the class.

In [13]:
```
class GPT2Dataset(Dataset):

  def __init__(self, txt_list, tokenizer, gpt2_type="gpt2", max_length=7

    self.tokenizer = tokenizer
    self.input_ids = []
    self.attn_masks = []

    for txt in txt_list:

      encodings_dict = tokenizer('<|startoftext|>'+ txt + '<|endoftext|>

      self.input_ids.append(torch.tensor(encodings_dict['input_ids']))
      self.attn_masks.append(torch.tensor(encodings_dict['attention_mask

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

  def __getitem__(self, idx):
    return self.input_ids[idx], self.attn_masks[idx]
```

To understand how I've used the tokenizer, it's worth reading the docs (https://huggingface.co/transformers/main_classes/tokenizer.html). I've wrapped each line in the bos and eos tokens.

Every tensor passed to the model should be the same length.

If the line is shorter than 768 tokens, it will be padded to a length of 768 using the padding token. In addition, an attention mask will be returned that needs to be passed to the model to tell it to ignore the padding tokens.

If the line is longer than 768 tokens, it will be truncated without the eos_token. This isn't a problem.

In [14]:
```
dataset = GPT2Dataset(docs, tokenizer, max_length=768)

# Split into training and validation sets
train_size = int(0.9 * len(dataset))
val_size = len(dataset) - train_size

train_dataset, val_dataset = random_split(dataset, [train_size, val_size

print('{:>5,} training samples'.format(train_size))
print('{:>5,} validation samples'.format(val_size))
```

```
18,663 training samples
2,074 validation samples
```

In [15]:
```
# Create the DataLoaders for our training and validation datasets.
# We'll take training samples in random order.
train_dataloader = DataLoader(
            train_dataset,  # The training samples.
            sampler = RandomSampler(train_dataset), # Select batches ran
            batch_size = batch_size # Trains with this batch size.
        )

# For validation the order doesn't matter, so we'll just read them sequer
validation_dataloader = DataLoader(
            val_dataset, # The validation samples.
            sampler = SequentialSampler(val_dataset), # Pull out batches
            batch_size = batch_size # Evaluate with this batch size.
        )
```

Finetune GPT2 Language Model

In [26]:
```
configuration = GPT2Config.from_pretrained('gpt2', output_hidden_states=F

# instantiate the model
model = GPT2LMHeadModel.from_pretrained("gpt2", config=configuration)

# this step is necessary because I've added some tokens (bos_token, etc)
# otherwise the tokenizer and model tensors won't match up
model.resize_token_embeddings(len(tokenizer))

# Tell pytorch to run this model on the GPU.
device = torch.device("cuda")
model.cuda()

# Set the seed value all over the place to make this reproducible.
seed_val = 42
random.seed(seed_val)
np.random.seed(seed_val)
torch.manual_seed(seed_val)
torch.cuda.manual_seed_all(seed_val)

# Training parameters
epochs = 5
learning_rate = 3e-5  # Adjusted learning rate
warmup_steps = 1e2
epsilon = 1e-8
sample_every = 100
```

In [27]:
```
# Note: AdamW is a class from the huggingface library (as opposed to pyto
optimizer = AdamW(model.parameters(),
                  lr = learning_rate,
                  eps = epsilon
                )
```

In [28]:
```
# Total number of training steps is [number of batches] x [number of epo
# (Note that this is not the same as the number of training samples).
total_steps = len(train_dataloader) * epochs

# Create the learning rate scheduler.
# This changes the learning rate as the training loop progresses
scheduler = get_linear_schedule_with_warmup(optimizer,
                                            num_warmup_steps = warmup_st
                                            num_training_steps = total_s
```

In [29]:
```
def format_time(elapsed):
    return str(datetime.timedelta(seconds=int(round((elapsed)))))
```

In [30]:
```
total_t0 = time.time()

training_stats = []

model = model.to(device)

for epoch_i in range(0, epochs):

    # ========================================
    #               Training
    # ========================================

    print("")
    print('======== Epoch {:} / {:} ========'.format(epoch_i + 1, epochs
    print('Training...')

    t0 = time.time()

    total_train_loss = 0

    model.train()

    for step, batch in enumerate(train_dataloader):
        b_input_ids = batch[0].to(device)
        b_labels = batch[0].to(device)
        b_masks = batch[1].to(device)
        model.zero_grad()
        outputs = model(  b_input_ids,
                          labels=b_labels,
                          attention_mask = b_masks,
                          token_type_ids=None
                        )
        loss = outputs[0]
        batch_loss = loss.item()
        total_train_loss += batch_loss
        # Get sample every x batches.
        if step % sample_every == 0 and not step == 0:
            elapsed = format_time(time.time() - t0)
            print('  Batch {:>5,}  of  {:>5,}. Loss: {:>5,}.   Elapsed:
            model.eval()
            sample_outputs = model.generate(
                                    bos_token_id=random.randint(1,30000)
                                    do_sample=True,
                                    top_k=50,
                                    max_length = 200,
                                    top_p=0.95,
                                    num_return_sequences=1
                                )
            for i, sample_output in enumerate(sample_outputs):
                  print("{}: {}".format(i, tokenizer.decode(sample_outpu
            model.train()
        loss.backward()
        optimizer.step()
        scheduler.step()

    # Calculate the average loss over all of the batches.
    avg_train_loss = total_train_loss / len(train_dataloader)
```

```
    # Measure how long this epoch took.
    training_time = format_time(time.time() - t0)

    print("")
    print("  Average training loss: {0:.2f}".format(avg_train_loss))
    print("  Training epoch took: {:}".format(training_time))

    # ========================================
    #               Validation
    # ========================================

    print("")
    print("Running Validation...")

    t0 = time.time()

    model.eval()

    total_eval_loss = 0
    nb_eval_steps = 0

    # Evaluate data for one epoch
    for batch in validation_dataloader:

        b_input_ids = batch[0].to(device)
        b_labels = batch[0].to(device)
        b_masks = batch[1].to(device)

        with torch.no_grad():

            outputs  = model(b_input_ids,
#                             token_type_ids=None,
                             attention_mask = b_masks,
                            labels=b_labels)

            loss = outputs[0]

        batch_loss = loss.item()
        total_eval_loss += batch_loss

    avg_val_loss = total_eval_loss / len(validation_dataloader)

    validation_time = format_time(time.time() - t0)

    print("  Validation Loss: {0:.2f}".format(avg_val_loss))
    print("  Validation took: {:}".format(validation_time))

    # Record all statistics from this epoch.
    training_stats.append(
        {
            'epoch': epoch_i + 1,
            'Training Loss': avg_train_loss,
            'Valid. Loss': avg_val_loss,
            'Training Time': training_time,
            'Validation Time': validation_time
        }
```

```
    )

print("")
print("Training complete!")
print("Total training took {:} (h:mm:ss)".format(format_time(time.time()
```

```
======== Epoch 1 / 5 ========
Training...

The attention mask and the pad token id were not set. As a consequence,
you may observe unexpected behavior. Please pass your input's `attentio
n_mask` to obtain reliable results.
Setting `pad_token_id` to `eos_token_id`:50256 for open-end generation.

  Batch   100  of  9,332. Loss: 0.0737159252166748.   Elapsed: 0:00:16.
0:  bipartisan the the the the the the the the the
 the the
 the the the the the
 the the the the

The attention mask and the pad token id were not set. As a consequence,
you may observe unexpected behavior. Please pass your input's `attentio
n_mask` to obtain reliable results.
Setting `pad_token_id` to `eos_token_id`:50256 for open-end generation.
```

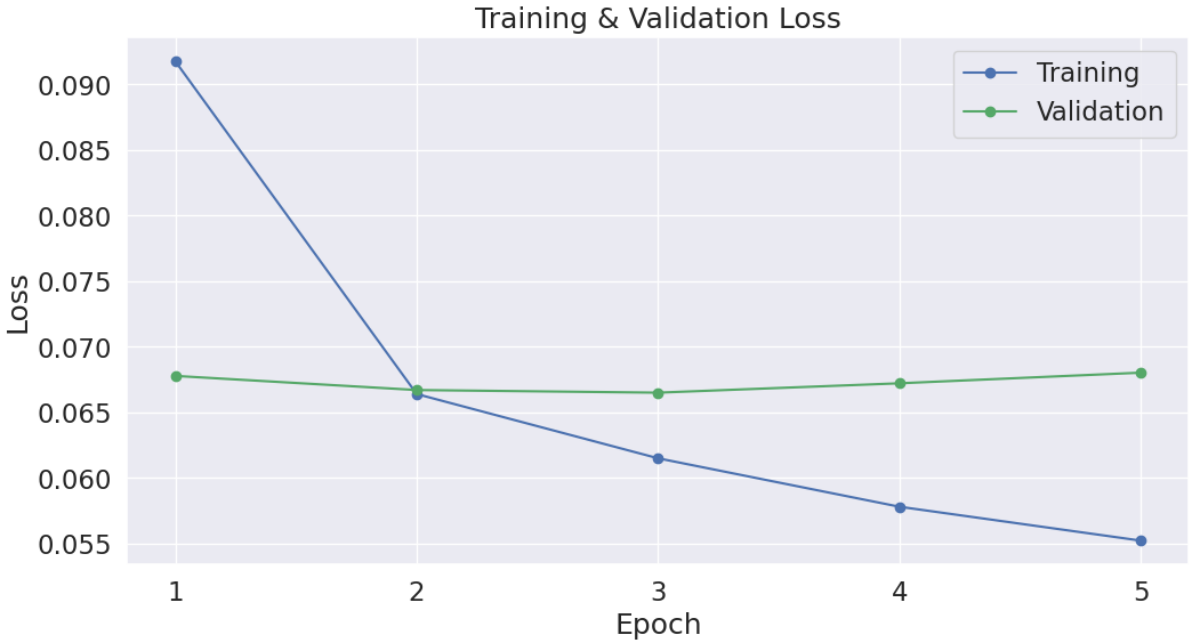
Let's view the summary of the training process.

In [31]:
```
# Display floats with two decimal places.
pd.set_option('display.precision', 2)

# Create a DataFrame from our training statistics.
df_stats = pd.DataFrame(data=training_stats)

# Use the 'epoch' as the row index.
df_stats = df_stats.set_index('epoch')

# A hack to force the column headers to wrap.
#df = df.style.set_table_styles([dict(selector="th",props=[('max-width',

# Display the table.
df_stats
```

Out[31]:

epoch	Training Loss	Valid. Loss	Training Time	Validation Time
1	0.09	0.07	0:24:33	0:00:51
2	0.07	0.07	0:24:23	0:00:51
3	0.06	0.07	0:24:23	0:00:51
4	0.06	0.07	0:24:24	0:00:51
5	0.06	0.07	0:24:24	0:00:51

In [32]:
```
# Use plot styling from seaborn.
sns.set(style='darkgrid')

# Increase the plot size and font size.
sns.set(font_scale=1.5)
plt.rcParams["figure.figsize"] = (12,6)

# Plot the learning curve.
plt.plot(df_stats['Training Loss'], 'b-o', label="Training")
plt.plot(df_stats['Valid. Loss'], 'g-o', label="Validation")

# Label the plot.
plt.title("Training & Validation Loss")
plt.xlabel("Epoch")
plt.ylabel("Loss")
plt.legend()
plt.xticks([1, 2, 3, 4, 5])

plt.show()
```



Display Model Info

In [33]:
```
# Get all of the model's parameters as a list of tuples.
params = list(model.named_parameters())

print('The GPT-2 model has {:} different named parameters.\n'.format(len

print('==== Embedding Layer ====\n')

for p in params[0:2]:
    print("{:<55} {:>12}".format(p[0], str(tuple(p[1].size()))))

print('\n==== First Transformer ====\n')

for p in params[2:14]:
    print("{:<55} {:>12}".format(p[0], str(tuple(p[1].size()))))

print('\n==== Output Layer ====\n')

for p in params[-2:]:
    print("{:<55} {:>12}".format(p[0], str(tuple(p[1].size()))))
```

```
The GPT-2 model has 148 different named parameters.

==== Embedding Layer ====

transformer.wte.weight                                  (50259, 768)
transformer.wpe.weight                                   (1024, 768)

==== First Transformer ====

transformer.h.0.ln_1.weight                                  (768,)
transformer.h.0.ln_1.bias                                    (768,)
transformer.h.0.attn.c_attn.weight                       (768, 2304)
transformer.h.0.attn.c_attn.bias                            (2304,)
transformer.h.0.attn.c_proj.weight                       (768, 768)
transformer.h.0.attn.c_proj.bias                             (768,)
transformer.h.0.ln_2.weight                                  (768,)
transformer.h.0.ln_2.bias                                    (768,)
transformer.h.0.mlp.c_fc.weight                          (768, 3072)
transformer.h.0.mlp.c_fc.bias                               (3072,)
transformer.h.0.mlp.c_proj.weight                        (3072, 768)
transformer.h.0.mlp.c_proj.bias                              (768,)

==== Output Layer ====

transformer.ln_f.weight                                      (768,)
transformer.ln_f.bias                                        (768,)
```

Saving & Loading Fine-Tuned Model

In [34]:
```
# Saving best-practices: if you use defaults names for the model, you ca

output_dir = './model_save/'

# Create output directory if needed
if not os.path.exists(output_dir):
    os.makedirs(output_dir)

print("Saving model to %s" % output_dir)

# Save a trained model, configuration and tokenizer using `save_pretraine
# They can then be reloaded using `from_pretrained()`
model_to_save = model.module if hasattr(model, 'module') else model  # Ta
model_to_save.save_pretrained(output_dir)
tokenizer.save_pretrained(output_dir)

# Good practice: save your training arguments together with the trained m
# torch.save(args, os.path.join(output_dir, 'training_args.bin'))
```

```
Saving model to ./model_save/
```

Out[34]:
```
('./model_save/tokenizer_config.json',
 './model_save/special_tokens_map.json',
 './model_save/vocab.json',
 './model_save/merges.txt',
 './model_save/added_tokens.json')
```

In [ ]:
```
!ls -l --block-size=K ./model_save/
```

In [ ]:
```
!ls -l --block-size=M ./model_save/pytorch_model.bin
```

```
-rw-r--r-- 1 root root 487M Mar 27 16:29 ./model_save/pytorch_model.bin
```

In [ ]:
```
# Copy the model files to a directory in your Google Drive.
!cp -r ./model_save/ $data_dir

# # Load a trained model and vocabulary that you have fine-tuned
#model = GPT2LMHeadModel.from_pretrained(output_dir)
#tokenizer = GPT2Tokenizer.from_pretrained(output_dir)
#model.to(device)
```

Generate Text

In [82]:
```
model.eval()

prompt = "<|startoftext|>"

generated = torch.tensor(tokenizer.encode(prompt)).unsqueeze(0)
generated = generated.to(device)

print(generated)

sample_outputs = model.generate(
                                generated,
                                #bos_token_id=random.randint(1,30000),
                                do_sample=True,
                                top_k=50,
                                max_length = 300,
                                top_p=0.95,
                                num_return_sequences=3
                                )

for i, sample_output in enumerate(sample_outputs):
  print("{}: {}\n\n".format(i, tokenizer.decode(sample_output, skip_spec
```

```
The attention mask and the pad token id were not set. As a consequence,
you may observe unexpected behavior. Please pass your input's `attentio
n_mask` to obtain reliable results.
Setting `pad_token_id` to `eos_token_id`:50256 for open-end generation.

tensor([[50257]], device='cuda:0')
0: “If you like, sir, I shall go to school.”


1:


2: “A woman has just arrived at Thornfield.”
```

YOUR TURN!

These aren't bad at all! Now train the model on your chosen raw text that is roughly comparable in size to pride and prejudice.

There are two things you need to do:

- Draw a figure tracking the training and validation losses as in previous homeworks.
- Print out some sample text from your chosen data and report 10 example generations that you think are interesting! Do your examples look like your training text?

> 10 interesting example generations (they are very similar to my training text--Jane Eyre by Charlotte Brontë)

1. “But no! let us be reconciled!” exclaimed Helen.
2. “No, Jane, I am not afraid of you.”
3. “Jane, you have a strange passion. And how is your brain?”
4. not even that of a lady, nor even that of a man; she never married.
5. The door was bolted; the clerk-door was locked. The door was again unclosed.
6. He stopped; his eye met my shoulder: he did not care to look up.
7. “She said she would give you a dinner-bell-fellow,” I observed.
8. I turned my head from St. John when I heard him speak.
9. I would not have liked it: it would not have been advisable.
10. “A woman has just arrived at Thornfield.”