CS 505 Homework 06: Transformers

Due Friday 12/15 at midnight (1 minute after 11:59 pm) in Gradescope (with a grace period of 6 hours)

You may submit the homework up to 24 hours late (with the same grace period) for a penalty of 10%.

All homeworks will be scored with a maximum of 100 points; point values are given for individual problems, and if parts of problems do not have point values given, they will be counted equally toward the total for that problem.

Note: This final homework concerns transformers, and due to the complexity of the models and the HuggingFace ecosystem, there is a large amount of tutorial information. Please read through and *try* the code in the tutorials, and then answer the questions posted in the latter part of each problem.

Each problem is worth 33 points, and you will get 1 point free.

Submission Instructions

Because of the amount of tutorial material, we felt it was best to split the notebooks into separate files, so please submit *six* files:

- Files `HW06.P1.ipynb` , `HW06.P2.ipynb` , and `HW06.P3.ipynb` (be sure to select `Kernel -> Restart and Run All` before you submit, to make sure everything works); and
- Files `HW06.P1.pdf` , `HW06.P2.pdf` , and `HW06.P3.pdf` created from the previous.

  For best results obtaining a clean PDF file on the Mac, select `File -> Print Review` from the Jupyter window, then choose `File-> Print` in your browser and then `Save as PDF` . Something similar should be possible on a Windows machine -- just make sure it is readable and no cell contents have been cut off. Make it easy to grade!

The date and time of your submission is the last file you submitted, so if your IPYNB file is submitted on time, but your PDF is late, then your submission is late.

Full Disclosure: This notebook is based on work by Liam Dugan (UPenn).

Introduction

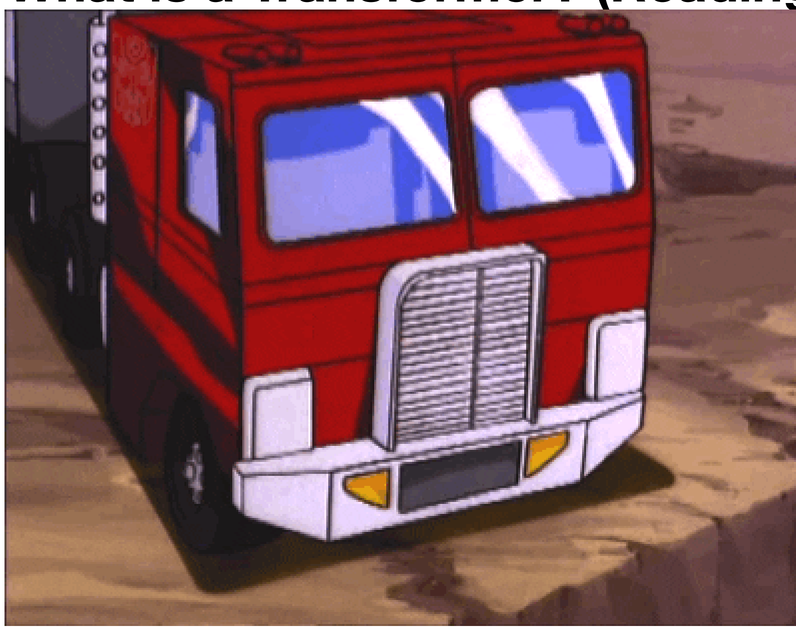
For this homework, we will take ideas from the entire class: language models, text generation, vector-based word representations, syntactic analysis, and neural networks. We'll be using large, pre-trained language models to generate text, and studying how we can fine-tune these large language models to generate text in whatever genre and style we want!

In this assignment you will get:

1. An overview of the "Transformer" architecture is and why it is particularly well suited for Natural Language Processing tasks
2. An introduction to the Generative Pretrained Transformer (GPT) family, which is a set of large-scale language models that can be used to generate text that often sounds like it was written by a human.
3. Experience with using the HuggingFace package to fine-tune these models to generate text that sounds like it comes from a specific source.

Problem One

Part 1: What is a Transformer? (Reading)



(It's probably not this guy, right?)

The Transformer

The current state-of-the-art for a variety of natural language processing tasks belongs to the **Transformer** architecture, first published December 6th 2017.

The Transformer can be thought of as a big feed-forward network with every feed-forward layer containing something called an "attention module".

> You might be wondering: why are we moving back to feed-forward networks after having so much success with recurrent neural networks and variants like LSTMs? Aren't RNNs naturally poised to handle sequences as their inputs? Well, as it turns out, the sequential nature of RNNs make them really difficult to train in a distributed/parallel fashion. So while RNNs make more sense to use on sequences of inputs, serial networks such as the transformer can be trained much faster, allowing orders of magnitude more training data to be used.

Reading # 1 - What is a Transformer?

In order to get a good grasp on exactly *why* these models are so good it's important to understand what they are and how they work.

Your first task for this homework is to read the blog post ["The Illustrated Transformer" by Jay Alammar (http://jalammar.github.io/illustrated-transformer/)](http://jalammar.github.io/illustrated-transformer/). This blog post explains the transformer architecture (and the all-important "Attention Module") with helpful visualizations and diagrams.

You should read this post very closely and understand exactly what the Transformer is and how it works. Once you're finished reading, answer the following questions in 2-3 sentences each.

1. (2 pts) What is Self-Attention (at a high level)?

> Self-Attention is a fundamental component of the Transformer model, which enables each token in the input sequence to consider and understand every other token in the same sequence. It helps the model to capture the context and relationships within the sequence itself, making it powerful in tasks like language generation.

2. (2 pts) How is Self-Attention computed?

> Self-Attention is computed by transforming each token in the input sequence into three vectors: Query (Q), Key (K), and Value (V). The attention score for each token is calculated by taking the dot product of Q and K of all tokens, followed by a softmax operation to normalize these scores. The final output is obtained by multiplying these normalized scores with the Value vectors and summing them up.

3. (2 pts) What do the "Query", "Key", and "Value" vectors encode (at a high level)?

> For Self-Attention, the "Query" vector represents the current token being focused on, the "Key" vectors represent all tokens in the sequence (including the current one), and are used to compute the attention scores, and the "Value" vectors hold the information of each token that is weighted by the attention scores. These vectors help determine the influence of each token on others in the sequence.

4. (2 pts) What is an attention "head" and why should we use multiple heads?

> An attention "head" focuses on different parts of the input sequence. Using multiple heads allows the model to simultaneously attend to information from different representation subspaces at different positions, enabling it to capture various types of relationships between tokens, which improves the model's ability to understand complex patterns in the data.

5. (2 pts) What are positional embeddings?

> Positional embeddings are additional vectors added to the input embeddings in the Transformer model. They provide info about the position of each token in the sequence.

6. (2 pts) Why are positional embeddings important?

> Positional embeddings are important because they allow the Transformer to consider the order of words in a sequence, which is essential for understanding the meaning and structure of sentences.

Reading # 2 - Transformer Language Models

On June 11th 2018, OpenAI released a model named **GPT**, standing for *Generative Pre-Trained Transformer*. This model was a Transformer architecture that was modified such that it could be used for Text Generation instead of sequence to sequence modeling. This model was also pre-trained, which means that anyone could download the fully trained model and use it without needing to train the model themselves.

On February 14th 2019, OpenAI released a blog post detailing a brand new version of GPT that had an insane **1.5 billion parameters**. They named this version **GPT-2**. To train such a large model, OpenAI crawled 40GB worth of text from the web (roughly 20,000,000,000 words).

GPT-2 is an extremely impressive language model that can generate text that often sounds so plausible that it seems like it might have been written by a human. Here is an example of what GPT-2 can generate, taken from [OpenAI's blog post (https://openai.com/blog/better-language-models/)](https://openai.com/blog/better-language-models/):

Human-Written Prompt:

> "Miley Cyrus was caught shoplifting from Abercrombie and Fitch on Hollywood Boulevard today."

Model Continuation:

> The 19-year-old singer was caught on camera being escorted out of the store by security guards.
>
> The singer was wearing a black hoodie with the label 'Blurred Lines' on the front and 'Fashion Police' on the back.
>
> Scroll down for video
>
> Shoplifting: Miley Cyrus was caught shoplifting from Abercrombie and Fitch on Hollywood Boulevard today (pictured)
>
> The singer was also wearing a pair of black-rimmed glasses, a black jacket, black jeans and black sandals.
>
> She was carrying a pair of black and white striped gloves and a small black bag.

(To be clear: Miley Cyrus was never actually arrested for shoplifting).

The quality of the model's output was so good, that the researchers at OpenAI were worried that, in addition to positive applications, it might be used for malicious purposes like generating fake news, impersonating people online, creating abusive or fake content to post on social media, or automating spam and phishing scams. In a [demo to Wired Magazine (https://www.wired.com/story/ai-text-generator-too-dangerous-to-make-public/)](https://www.wired.com/story/ai-text-generator-too-dangerous-to-make-public/), the reporter typed in "Hillary Clinton and George Soros" and GPT-2 generated a crazy rant with lots of conspiracy theories:

> Hillary Clinton and George Soros are a perfect match, and their agenda appears to be to create a political movement where Soros and his political machine and Clinton are two of the only major players. This is the first time Soros and Clinton have been caught on tape directly colluding in promoting the same false narrative. One of the key revelations in the leaked audio was Clinton's admission to a Russian banker that she knew about the Uranium One deal before it was approved by Congress. Clinton was shown sharing the same talking points that were originally drafted by a Fusion GPS contractor hired by an anti-Trump Republican donor. The leaked audio is the clearest evidence yet that the Clinton campaign and the Hillary Foundation colluded with Fusion GPS to manufacture propaganda against President Trump.

They were concerned enough that they labeled GPT-2 "too dangerous to release", and OpenAI initially refused to release their dataset, training code, or GPT-2 model weights. OpenAI decided to release in a delayed, phased fashion so that researchers could spend time working on automatic detection of generated text.

In this homework, you'll get to be the judge of how good GPT-2 is, as you'll be using it yourself to generate text!

To start your journey into the world of Text Generation, you should read Part 1 of the blog post ["The Illustrated GPT-2" by Jay Alammar (http://jalammar.github.io/illustrated-gpt2/)](http://jalammar.github.io/illustrated-gpt2/) and answer the following questions in 2-3 sentences each

7. (4 pts) How does the architecture of GPT-2 differ from the standard Encoder-Decoder Transformer model?

> Unlike the standard Encoder-Decoder Transformer model, GPT-2 uses a decoder-only architecture. Each block in GPT-2 processes the input sequence entirely with self-attention mechanisms, without any separate encoder phase. This design enables GPT-2 to generate text by predicting one word at a time, using all the previous words in the sequence as context, making it particularly effective for tasks like text generation.

8. (4 pts) What is the difference between "Masked Self-Attention" and "Self-Attention"

> Self-Attention allows each position to attend to all positions in the sequence, making it appropriate for tasks where the entire input sequence is known upfront, like in translation. On the other hand, masked Self-Attention is a variation of self-attention to prevent positions from attending to subsequent positions. This masking ensures that the prediction for a position can only depend on the known outputs at positions before it.

9. (4 pts) What are logits? How are they computed? and How does GPT-2 use them to decide which word to predict next?

> Logits are the raw, unnormalized scores outputted by the last layer of a neural network before passing through an activation function like softmax. They are computed as the dot product of the output vector from the final layer with the vector representation of each word in the model's vocabulary. GPT-2 uses logits to determine the next word by applying a softmax function to convert these logits into probabilities for each potential next word in the vocabulary. The word with the highest probability is then chosen as the next word in the generated text.

Aside: GPT-3

On June 11th 2020, OpenAI released GPT-3 [(paper) (https://arxiv.org/pdf/2005.14165.pdf)](https://arxiv.org/pdf/2005.14165.pdf) [(wikipedia) (https://en.wikipedia.org/wiki/GPT-3)](https://en.wikipedia.org/wiki/GPT-3). This model has an unfathomable **175 billion parameters** (100x larger than GPT-2!) and was trained on 570GB of text! This model is virtually indistinguishable from human output and can generate text about any topic and in any style with only a few words of priming text. It can do some very terrifying things.

GPT-3 Can:

- Generate JSX code off natural language descriptions
- Generate Emojis based off of descriptions of the feeling
- Generate regular expressions off natural language descriptions
- Generate website mockups off natural language descriptions
- Generate charts with titles, labels and legends from natural language descriptions
- Explain python code in plain english
- Automatically generate quiz questions (and grade them)
- Generate Latex from natural language descriptions
- Generate Linux commands from natural language descriptions
- Generate a Machine Learning model from natural language descriptions

Here's a collection of 21 things GPT-3 can do (with examples) (https://machinelearningknowledge.ai/openai-gpt-3-demos-to-convince-you-that-ai-threat-is-real-or-is-it/#OpenAI_GPT-3_Demos)

Here's a NYT article about how GPT-3 can write code, poetry, and argue (https://www.nytimes.com/2020/11/24/science/artificial-intelligence-ai-gpt3.html)

Here's an article GPT-3 wrote for The Guardian about how it loves humans and would never subjugate humanity (https://www.theguardian.com/commentisfree/2020/sep/08/robot-

---

Part 2: GPT-2 Text Generation with HuggingFace

Phew, that was a lot of reading. Now lets get to the fun part! Let's use the transformer to generate some text!!

We will use the Transformers library from HuggingFace (https://transformer.huggingface.co), which provides support for many Transformer-based language models like GPT-2.

IMPORTANT: Make sure that you have GPU set as your Hardware Accelerator in `Runtime > Change runtime type` before running this Colab.

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

```
Requirement already satisfied: transformers in /usr/local/lib/python
3.10/dist-packages (4.35.2)
Requirement already satisfied: filelock in /usr/local/lib/python3.1
0/dist-packages (from transformers) (3.13.1)
Requirement already satisfied: huggingface-hub<1.0,>=0.16.4 in /usr/
local/lib/python3.10/dist-packages (from transformers) (0.19.4)
Requirement already satisfied: numpy>=1.17 in /usr/local/lib/python
3.10/dist-packages (from transformers) (1.23.5)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/pyt
hon3.10/dist-packages (from transformers) (23.2)
Requirement already satisfied: pyyaml>=5.1 in /usr/local/lib/python
3.10/dist-packages (from transformers) (6.0.1)
Requirement already satisfied: regex!=2019.12.17 in /usr/local/lib/p
ython3.10/dist-packages (from transformers) (2023.6.3)
Requirement already satisfied: requests in /usr/local/lib/python3.1
0/dist-packages (from transformers) (2.31.0)
Requirement already satisfied: tokenizers<0.19,>=0.14 in /usr/local/
lib/python3.10/dist-packages (from transformers) (0.15.0)
Requirement already satisfied: safetensors>=0.3.1 in /usr/local/lib/
python3.10/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/py
thon3.10/dist-packages (from huggingface-hub<1.0,>=0.16.4->transform
ers) (2023.6.0)
Requirement already satisfied: typing-extensions>=3.7.4.3 in /usr/lo
cal/lib/python3.10/dist-packages (from huggingface-hub<1.0,>=0.16.4-
>transformers) (4.5.0)
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/loca
l/lib/python3.10/dist-packages (from requests->transformers) (3.3.2)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python
3.10/dist-packages (from requests->transformers) (3.6)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/
python3.10/dist-packages (from requests->transformers) (2.0.7)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/
python3.10/dist-packages (from requests->transformers) (2023.11.17)
```

2.1 The 'Pipeline' Interface

The simplest way to use the HuggingFace library is to use their [Pipeline interface (https://huggingface.co/transformers/main_classes/pipelines.html)](https://huggingface.co/transformers/main_classes/pipelines.html)

There are many different types of Pipelines available but in this section we'll use the TextGenerationPipeline to get up and running with pretrained gpt2 as fast as possible

In [ ]:
```
from transformers import pipeline
```

In [ ]:
```
# Note: device=0 means to use GPU, device=-1 is to use CPU
generator = pipeline('text-generation', model='gpt2', device=0)
```

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

model.safetensors:   0%|          | 0.00/548M [00:00<?, ?B/s]

generation_config.json:   0%|          | 0.00/124 [00:00<?, ?B/s]

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]
```

In [ ]:
```
outputs = generator('I wonder what I will generate?')
print(outputs)
```

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

[{'generated_text': 'I wonder what I will generate?\n\nI wonder what
will you have at your disposal.\n\nI wonder what will you have if yo
u will be strong.\n\nI wonder if you will show power through strengt
h.\n\nI wonder if'}]
```

Note that the 'text-generation' pipeline will work with any **auto-regressive** language model (a.k.a 'causal-lm' models according to the HuggingFace lingo). You can find a list of all such models here [https://huggingface.co/models?filter=causal-lm (https://huggingface.co/models?filter=causal-lm)](https://huggingface.co/models?filter=causal-lm).

10. (6 pts) **Your first task is to use the Pipeline interface to get generation output below for at least two different 'causal-lm' models (One of these two can be a different version of GPT2, but make sure at least one is a non-gpt family language model)**

In [ ]:
```
## YOUR CODE HERE FOR MODEL 1: GPT2-large
from transformers import pipeline

# Using GPU if available, otherwise CPU
device = 0

generator_gpt2_large = pipeline('text-generation', model='gpt2-large'
output_gpt2_large = generator_gpt2_large('I wonder what I will genera

print(output_gpt2_large)
```

```
config.json:   0%|          | 0.00/666 [00:00<?, ?B/s]

model.safetensors:   0%|          | 0.00/3.25G [00:00<?, ?B/s]

generation_config.json:   0%|          | 0.00/124 [00:00<?, ?B/s]

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]

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

[{'generated_text': 'I wonder what I will generate? I can\'t even be
sure what I will end up with." The "latter" part I had to ignore. Th
is was me doing my best to make what I had always wanted with me in
that brief moment'}]
```

In [ ]:
```
## YOUR CODE HERE FOR MODEL 2: EleutherAI/gpt-neo-2.7B
generator_gpt_neo = pipeline('text-generation', model='EleutherAI/gpt
output_gpt_neo = generator_gpt_neo('I wonder what I will generate?')

print(output_gpt_neo)
```

```
config.json:   0%|          | 0.00/1.46k [00:00<?, ?B/s]

model.safetensors:   0%|          | 0.00/10.7G [00:00<?, ?B/s]

tokenizer_config.json:   0%|          | 0.00/200 [00:00<?, ?B/s]

vocab.json:   0%|          | 0.00/798k [00:00<?, ?B/s]

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

special_tokens_map.json:   0%|          | 0.00/90.0 [00:00<?, ?B/s]

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

[{'generated_text': 'I wonder what I will generate?\n\nThere are two
types of creative people: those who are comfortable with the uncomfo
rtable, and those who want to change things for the better. The form
er are more inclined to change things slowly, and experiment until t
hey'}]
```

2.2 Dissecting the Pipeline

Now that was easy!

As beautiful and easy as the Pipeline interface is, we want to know what's going on under the hood!

There are four main steps to a text generation pipeline:

1. (Tokenize) Turn the raw input text into a vector of integer token IDs using a tokenizer
2. (Encode) Feed those token IDs into the language model by querying for each token's embedding in the model's embedding matrix (the "encoder") and then feed the "encoded" sequence into the decoder module
3. (Decode) The decoder will output logits (a probability distribution over all possible integer token IDs) and we sample from those logits to get our next token -- repeat until EOS token is generated or we hit max_length
4. (Detokenize) Take the output sequence of token IDs and turn them from integer token IDs back to tokens with the tokenizer

Below you'll see how HuggingFace does this:

First we have to initialize both the tokenizer and the model from their pre-trained checkpoints. Note that the tokenizer has to match the model.

In [ ]:
```
from transformers import GPT2Tokenizer, GPT2LMHeadModel# AutoTokenize

tokenizer = GPT2Tokenizer.from_pretrained('gpt2')
model = GPT2LMHeadModel.from_pretrained('gpt2').cuda()
```

In [ ]:
```
#### Step 1: Tokenize the input into integer token IDs
inputs = tokenizer.encode("Hello, how are you?", return_tensors='pt')
print("Input Token IDs: " + str(inputs))
```

```
Input Token IDs: tensor([[15496,    11,   703,   389,   345,    3
0]], device='cuda:0')
```

In [ ]:
```
#### Step 2 and 3: Feed in the integer token IDs and get out a sequen
outputs = model.generate(inputs)
print("Output Token IDs: " + str(outputs))
```

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

Output Token IDs: tensor([[15496,    11,   703,   389,   345,    30,
198,   198,    40,  1101,
           257,  1310,  1643,   286,   257, 34712,    13,   314,  11
01,   257]],
       device='cuda:0')

/usr/local/lib/python3.10/dist-packages/transformers/generation/util
s.py:1273: UserWarning: Using the model-agnostic default `max_length
` (=20) to control the generation length. We recommend setting `max_
new_tokens` to control the maximum length of the generation.
  warnings.warn(
```

In [ ]:
```
#### Step 4: Feed in the integer token IDs and get out a sequence of
output_text = [tokenizer.decode(x) for x in outputs]
print("Output Text: " + str(output_text))
```

```
Output Text: ["Hello, how are you?\n\nI'm a little bit of a nerd.
I'm a"]
```

Now that you have dissected the pipeline, it's time to play with some common parameters!

[Check out this demo notebook from HuggingFace (https://github.com/huggingface/blog/blob/master/notebooks/02_how_to_generate.ipynb)](https://github.com/huggingface/blog/blob/master/notebooks/02_how_to_generate.ipynb) for a good overview of the different generation parameters and what they do (with example code!).

The full documentation on all of the parameters you can use in the generate function can be found here (https://huggingface.co/transformers/main_classes/model.html#transformers.generation_utils

As an example, below we have a call to generate that:

- randomly samples from the top 50 words in the output distribution (rather than just greedily picking the best one every time)
- downweights the probability of all previously generated tokens by a factor of 1.2 (to prevent repetition)
- goes on for 512 tokens, because its more interesting

In [ ]:
```
inputs = tokenizer.encode("Hello, how are you?", return_tensors='pt')
outputs = model.generate(
      inputs,
      do_sample=True,          # Randomly sample from the logits inst
      top_k=50,                 # Only sample from the top 50 most li
      repetition_penalty=1.2,    # Downweights the probability of all
      max_length=512           # Generate for a maximum of 512 tokens
  )
print([tokenizer.decode(x) for x in outputs][0])
```

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

Hello, how are you?
We love your work! Thank you again for contacting us. All feedback w
ill be very welcome (please send a message if that's what we need!),
and feel free to fill out the form below: http://www5gofilego7in1123
@googlemail-box/email (http://www5gofilego7in1123@googlemail-box/ema
il) Sign up Now https:/ (https:/) /truvigaradu1y4d_i3jX8xSHqmznCaG6N
kLfZlF2cWOJRpbBmlwU 934 547 4531 855

's best friend : The 'VIP'. Not being on my team makes me so excited
:) I was going through school last year then went back once in awhil
e...and it ended when this guy asked about our favorite snack - choc
olate cake! Soooo cool!! As nice as food can get at such an early ag
e :-) It tastes pretty good with many different flavours of butter,e
gg & fruit which is great even though most people may not like them 
any more than those who have never used something before but might t
ry some soon because they'll start liking "sweet" desserts too ;) An
d actually eating chocolates doesn't really mean MUCH BUT THEY WILL 
BE THAT GOOD TO SOUP AND VEGETABLY IN YOUR COW!! You must use 2 tabl
espoonful every 4 hours or their entire quantity would just melt awa
y!!! Just keep doing this until there isn` t enough time between all
meals.....so dont worry unless after 3pm another person wants one. D
o note however ive done mine twice already from here ONCE...it wont 
take two separate requests....I think i could easily run into 6 hung
ry others..but hey now its getting kinda busy since thats usually ar
ound midnight except today @ noon.. Anyway please don\' mumble thank
you much~ Also thanks alot yusssessings....hmmmm........<|endoftext|
>
```

11. Your job is to provide two different examples of generation output from GPT-2 with different choices of generation parameters. You must also provide a 1-2 sentence explanation of what these parameters do and how they affect your output

Feel free to get creative with this! Really poke around and try to find the combination of settings that gives you the best sounding text! The ways in which these parameters affect how 'human-like' a section of generated text sounds is an area of active research. :)

In [ ]:
```
## YOUR CODE HERE FOR HYPERPARAMETER VARIATION 1

input_sentence = "One day, we will be reunited."

# Tokenize the input
inputs = tokenizer.encode(input_sentence, return_tensors='pt').to(moc

# Using temperature to control the randomness of predictions
outputs_temp = model.generate(
    inputs,
    do_sample=True,
    top_k=50,
    max_length=50,
    #repetition_penalty=1.2,
    temperature=0.6  # Temperature control
)
output_text_temp = tokenizer.decode(outputs_temp[0], skip_special_tok

print("Output with Temperature Control:\n", output_text_temp)
```

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

Output with Temperature Control:
 One day, we will be reunited. We will be able to talk about the pas
t, the future, and what the future holds for us. We will be able to
discuss the current state of our relationship with the world. We wil
l be able to
```

(4 pts) EXPLANATION FOR HPARAM VARIATION 1: A lower temperature (like 0.6) makes the generated output more deterministic. The model is more likely to choose the most probable next word or token based on its training data. The generated text tends to make more sense with meaning: "One day, we will be reunited. We will be able to talk about the past, the future, and what the future holds for us. We will be able to discuss the current state of our relationship with the world."

In [ ]:
```
## YOUR CODE HERE FOR HYPERPARAMETER VARIATION 2
# Using no_repeat_ngram_size to avoid repetition
outputs_no_repeat = model.generate(
    inputs,
    do_sample=True,
    max_length=50,
    no_repeat_ngram_size=2  # Prevents the model from using the same
)
output_text_no_repeat = tokenizer.decode(outputs_no_repeat[0], skip_s
print("\nOutput with No Repeat Ngram Size:\n", output_text_no_repeat)
```

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


Output with No Repeat Ngram Size:
 One day, we will be reunited. We are going to see each other face f
or the first time ever as we celebrate the start of our second seaso
n.

Thank you to everyone who came to the premiere for what we hope is o
ne more chance
```

(4 pts) EXPLANATION FOR HPARAM VARIATION 2: Setting no_repeat_ngram_size to 2 ensures the model doesn't repeat the same 2-gram, reducing redundancy and improving the variety in the text.

2.3 Fine-Tuning GPT-2

Okay now time for the best part!

Generating general-purpose text from pre-trained models is great, but what if we want our text to be in a specific genre or style? Luckily for us, the GPT family of models use the idea of "Transfer learning" -- using knowledge gained from one problem (or training setting), and applying it to another area or domain. The idea of transfer learning for NLP, is that we can train a language model on general texts, and then adapt it to use it for a specific task or domain that we're interested in. This process is also called **fine-tuning**.

In this section we'll walk you through an example of using HuggingFace to fine-tune GPT-2 and then you'll be asked to fine-tune GPT-2 on two datasets of your own choosing!

Fine-Tuning Example using HuggingFace Datasets library: Crime and Punishment

For our fine-tuning example we're going to train GPT-2 to mimic the style of Fyodor Dostoevsky's novel "Crime and Punishment"

We will be downloading our data using the HuggingFace Datasets

In [ ]:
```
!pip install datasets
```

```
Collecting datasets
  Downloading datasets-2.15.0-py3-none-any.whl (521 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 521.2/521.2 kB 5.7 MB/
s eta 0:00:00
Requirement already satisfied: numpy>=1.17 in /usr/local/lib/python
3.10/dist-packages (from datasets) (1.23.5)
Requirement already satisfied: pyarrow>=8.0.0 in /usr/local/lib/pyth
on3.10/dist-packages (from datasets) (10.0.1)
Collecting pyarrow-hotfix (from datasets)
  Downloading pyarrow_hotfix-0.6-py3-none-any.whl (7.9 kB)
Collecting dill<0.3.8,>=0.3.0 (from datasets)
  Downloading dill-0.3.7-py3-none-any.whl (115 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 115.3/115.3 kB 8.4 MB/
s eta 0:00:00
Requirement already satisfied: pandas in /usr/local/lib/python3.10/d
ist-packages (from datasets) (1.5.3)
Requirement already satisfied: requests>=2.19.0 in /usr/local/lib/py
thon3.10/dist-packages (from datasets) (2.31.0)
Requirement already satisfied: tqdm>=4.62.1 in /usr/local/lib/python
3.10/dist-packages (from datasets) (4.66.1)
Requirement already satisfied: xxhash in /usr/local/lib/python3.10/d
ist-packages (from datasets) (3.4.1)
Collecting multiprocess (from datasets)
  Downloading multiprocess-0.70.15-py310-none-any.whl (134 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 134.8/134.8 kB 6.9 MB/
s eta 0:00:00
Requirement already satisfied: fsspec[http]<=2023.10.0,>=2023.1.0 in
/usr/local/lib/python3.10/dist-packages (from datasets) (2023.6.0)
Requirement already satisfied: aiohttp in /usr/local/lib/python3.10/
dist-packages (from datasets) (3.9.1)
Requirement already satisfied: huggingface-hub>=0.18.0 in /usr/loca
l/lib/python3.10/dist-packages (from datasets) (0.19.4)
Requirement already satisfied: packaging in /usr/local/lib/python3.1
0/dist-packages (from datasets) (23.2)
Requirement already satisfied: pyyaml>=5.1 in /usr/local/lib/python
3.10/dist-packages (from datasets) (6.0.1)
Requirement already satisfied: attrs>=17.3.0 in /usr/local/lib/pytho
n3.10/dist-packages (from aiohttp->datasets) (23.1.0)
Requirement already satisfied: multidict<7.0,>=4.5 in /usr/local/li
b/python3.10/dist-packages (from aiohttp->datasets) (6.0.4)
Requirement already satisfied: yarl<2.0,>=1.0 in /usr/local/lib/pyth
on3.10/dist-packages (from aiohttp->datasets) (1.9.4)
Requirement already satisfied: frozenlist>=1.1.1 in /usr/local/lib/p
ython3.10/dist-packages (from aiohttp->datasets) (1.4.1)
Requirement already satisfied: aiosignal>=1.1.2 in /usr/local/lib/py
thon3.10/dist-packages (from aiohttp->datasets) (1.3.1)
Requirement already satisfied: async-timeout<5.0,>=4.0 in /usr/loca
l/lib/python3.10/dist-packages (from aiohttp->datasets) (4.0.3)
Requirement already satisfied: filelock in /usr/local/lib/python3.1
0/dist-packages (from huggingface-hub>=0.18.0->datasets) (3.13.1)
Requirement already satisfied: typing-extensions>=3.7.4.3 in /usr/lo
cal/lib/python3.10/dist-packages (from huggingface-hub>=0.18.0->data
sets) (4.5.0)
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/loca
l/lib/python3.10/dist-packages (from requests>=2.19.0->datasets) (3.
3.2)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python
```

```
3.10/dist-packages (from requests>=2.19.0->datasets) (3.6)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/
python3.10/dist-packages (from requests>=2.19.0->datasets) (2.0.7)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/
python3.10/dist-packages (from requests>=2.19.0->datasets) (2023.11.
17)
Requirement already satisfied: python-dateutil>=2.8.1 in /usr/local/
lib/python3.10/dist-packages (from pandas->datasets) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python
3.10/dist-packages (from pandas->datasets) (2023.3.post1)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.1
0/dist-packages (from python-dateutil>=2.8.1->pandas->datasets) (1.1
6.0)
Installing collected packages: pyarrow-hotfix, dill, multiprocess, d
atasets
Successfully installed datasets-2.15.0 dill-0.3.7 multiprocess-0.70.
15 pyarrow-hotfix-0.6
```

In [ ]:

```
from transformers import Trainer, TrainingArguments, DataCollatorForL
from datasets import load_dataset, list_datasets
from transformers import GPT2Tokenizer, GPT2LMHeadModel# AutoTokenize
```

Step 1: Initialize a Brand New GPT-2 Model and Tokenizer

In [ ]:

```
tokenizer = GPT2Tokenizer.from_pretrained('gpt2')
tokenizer.pad_token = tokenizer.eos_token
model = GPT2LMHeadModel.from_pretrained('gpt2').cuda()
data_collator = DataCollatorForLanguageModeling(tokenizer=tokenizer,
```

###Step 2: Load the text of "Crime and Punishment" and tokenize it

The 'load_dataset' function queries for a dataset with a certain tag and downloads the corresponding data from HuggingFace's hosting site. This allows us to download all sorts of datasets through the same interface!

The documentation for load_dataset can be found [here (https://huggingface.co/docs/datasets/package_reference/loading_methods.html#datasets.lo](https://huggingface.co/docs/datasets/package_reference/loading_methods.html#datasets.lo)

Here we take our tokenizer and run it on the entirety of Crime and Punishment in a single batch by using map on our custom encode function.

In [ ]:
```
def encode(batch): return tokenizer([x.strip('\n\r') for x in batch['

crime_and_punishment = load_dataset('crime_and_punish', split='train'
processed = crime_and_punishment.map(encode, batched=True, batch_size
processed.set_format('torch', columns=['input_ids', 'attention_mask']
```

Step 3: Initialize the Trainer

The 'Trainer' module is the main way we perform fine-tuning. In order to initialize a Trainer, you need a model, tokenizer, TrainingArguments, your training data (in a Dataset object) and something called a data_collator (which tells the Trainer not to look for a vector of labels).

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

```
Requirement already satisfied: transformers[torch] in /usr/local/li
b/python3.10/dist-packages (4.35.2)
Requirement already satisfied: filelock in /usr/local/lib/python3.1
0/dist-packages (from transformers[torch]) (3.13.1)
Requirement already satisfied: huggingface-hub<1.0,>=0.16.4 in /usr/
local/lib/python3.10/dist-packages (from transformers[torch]) (0.19.
4)
Requirement already satisfied: numpy>=1.17 in /usr/local/lib/python
3.10/dist-packages (from transformers[torch]) (1.23.5)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/pyt
hon3.10/dist-packages (from transformers[torch]) (23.2)
Requirement already satisfied: pyyaml>=5.1 in /usr/local/lib/python
3.10/dist-packages (from transformers[torch]) (6.0.1)
Requirement already satisfied: regex!=2019.12.17 in /usr/local/lib/p
ython3.10/dist-packages (from transformers[torch]) (2023.6.3)
Requirement already satisfied: requests in /usr/local/lib/python3.1
0/dist-packages (from transformers[torch]) (2.31.0)
Requirement already satisfied: tokenizers<0.19,>=0.14 in /usr/local/
lib/python3.10/dist-packages (from transformers[torch]) (0.15.0)
Requirement already satisfied: safetensors>=0.3.1 in /usr/local/lib/
python3.10/dist-packages (from transformers[torch]) (0.4.1)
Requirement already satisfied: tqdm>=4.27 in /usr/local/lib/python3.
10/dist-packages (from transformers[torch]) (4.66.1)
Requirement already satisfied: torch!=1.12.0,>=1.10 in /usr/local/li
b/python3.10/dist-packages (from transformers[torch]) (2.1.0+cu121)
Collecting accelerate>=0.20.3 (from transformers[torch])
  Downloading accelerate-0.25.0-py3-none-any.whl (265 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 265.7/265.7 kB 5.0 MB/
s eta 0:00:00
Requirement already satisfied: psutil in /usr/local/lib/python3.10/d
ist-packages (from accelerate>=0.20.3->transformers[torch]) (5.9.5)
Requirement already satisfied: fsspec>=2023.5.0 in /usr/local/lib/py
thon3.10/dist-packages (from huggingface-hub<1.0,>=0.16.4->transform
ers[torch]) (2023.6.0)
Requirement already satisfied: typing-extensions>=3.7.4.3 in /usr/lo
cal/lib/python3.10/dist-packages (from huggingface-hub<1.0,>=0.16.4-
>transformers[torch]) (4.5.0)
Requirement already satisfied: sympy in /usr/local/lib/python3.10/di
st-packages (from torch!=1.12.0,>=1.10->transformers[torch]) (1.12)
Requirement already satisfied: networkx in /usr/local/lib/python3.1
0/dist-packages (from torch!=1.12.0,>=1.10->transformers[torch]) (3.
2.1)
Requirement already satisfied: jinja2 in /usr/local/lib/python3.10/d
ist-packages (from torch!=1.12.0,>=1.10->transformers[torch]) (3.1.
2)
Requirement already satisfied: triton==2.1.0 in /usr/local/lib/pytho
n3.10/dist-packages (from torch!=1.12.0,>=1.10->transformers[torch])
(2.1.0)
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/loca
l/lib/python3.10/dist-packages (from requests->transformers[torch])
(3.3.2)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python
3.10/dist-packages (from requests->transformers[torch]) (3.6)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/
python3.10/dist-packages (from requests->transformers[torch]) (2.0.
7)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/
```

```
python3.10/dist-packages (from requests->transformers[torch]) (2023.
11.17)
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/pyt
hon3.10/dist-packages (from jinja2->torch!=1.12.0,>=1.10->transforme
rs[torch]) (2.1.3)
Requirement already satisfied: mpmath>=0.19 in /usr/local/lib/python
3.10/dist-packages (from sympy->torch!=1.12.0,>=1.10->transformers[t
orch]) (1.3.0)
Installing collected packages: accelerate
Successfully installed accelerate-0.25.0
```

In [ ]:
```
!pip install accelerate -U
```

```
Requirement already satisfied: accelerate in /usr/local/lib/python3.
10/dist-packages (0.25.0)
Requirement already satisfied: numpy>=1.17 in /usr/local/lib/python
3.10/dist-packages (from accelerate) (1.23.5)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/pyt
hon3.10/dist-packages (from accelerate) (23.2)
Requirement already satisfied: psutil in /usr/local/lib/python3.10/d
ist-packages (from accelerate) (5.9.5)
Requirement already satisfied: pyyaml in /usr/local/lib/python3.10/d
ist-packages (from accelerate) (6.0.1)
Requirement already satisfied: torch>=1.10.0 in /usr/local/lib/pytho
n3.10/dist-packages (from accelerate) (2.1.0+cu121)
Requirement already satisfied: huggingface-hub in /usr/local/lib/pyt
hon3.10/dist-packages (from accelerate) (0.19.4)
Requirement already satisfied: safetensors>=0.3.1 in /usr/local/lib/
python3.10/dist-packages (from accelerate) (0.4.1)
Requirement already satisfied: filelock in /usr/local/lib/python3.1
0/dist-packages (from torch>=1.10.0->accelerate) (3.13.1)
Requirement already satisfied: typing-extensions in /usr/local/lib/p
ython3.10/dist-packages (from torch>=1.10.0->accelerate) (4.5.0)
Requirement already satisfied: sympy in /usr/local/lib/python3.10/di
st-packages (from torch>=1.10.0->accelerate) (1.12)
Requirement already satisfied: networkx in /usr/local/lib/python3.1
0/dist-packages (from torch>=1.10.0->accelerate) (3.2.1)
Requirement already satisfied: jinja2 in /usr/local/lib/python3.10/d
ist-packages (from torch>=1.10.0->accelerate) (3.1.2)
Requirement already satisfied: fsspec in /usr/local/lib/python3.10/d
ist-packages (from torch>=1.10.0->accelerate) (2023.6.0)
Requirement already satisfied: triton==2.1.0 in /usr/local/lib/pytho
n3.10/dist-packages (from torch>=1.10.0->accelerate) (2.1.0)
Requirement already satisfied: requests in /usr/local/lib/python3.1
0/dist-packages (from huggingface-hub->accelerate) (2.31.0)
Requirement already satisfied: tqdm>=4.42.1 in /usr/local/lib/python
3.10/dist-packages (from huggingface-hub->accelerate) (4.66.1)
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/pyt
hon3.10/dist-packages (from jinja2->torch>=1.10.0->accelerate) (2.1.
3)
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/loca
l/lib/python3.10/dist-packages (from requests->huggingface-hub->acce
lerate) (3.3.2)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python
3.10/dist-packages (from requests->huggingface-hub->accelerate) (3.
6)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/
python3.10/dist-packages (from requests->huggingface-hub->accelerat
e) (2.0.7)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/
python3.10/dist-packages (from requests->huggingface-hub->accelerat
e) (2023.11.17)
Requirement already satisfied: mpmath>=0.19 in /usr/local/lib/python
3.10/dist-packages (from sympy->torch>=1.10.0->accelerate) (1.3.0)
```

In [ ]:
```
import accelerate
training_args = TrainingArguments(output_dir='/content/',
    overwrite_output_dir=True,
    num_train_epochs=1,
    per_device_train_batch_size=16,
    per_device_eval_batch_size=64,
    logging_steps=100,
    weight_decay=0.01,
    logging_dir='./logs',
)

trainer = Trainer(
    model=model,
    tokenizer=tokenizer,
    args=training_args,
    data_collator=data_collator,
    train_dataset=processed,
)
```

Step 4: Fine-Tune the Model!

Now we're done! All we have to do is hit run and sit back!

In [ ]:
```
trainer.train()
```

[1374/1374 01:41, Epoch 1/1]

Step	Training Loss
100	4.018200
200	3.741400
300	3.714900
400	3.573000
500	3.622300
600	3.598900
700	3.528100
800	3.514200
900	3.465800
1000	3.473000
1100	3.479400
1200	3.470100
1300	3.469700

Out[6]:
```
TrainOutput(global_step=1374, training_loss=3.5842322026104214, metr
ics={'train_runtime': 102.9786, 'train_samples_per_second': 213.336,
'train_steps_per_second': 13.343, 'total_flos': 392405005440000.0,
'train_loss': 3.5842322026104214, 'epoch': 1.0})
```

Step 5: Save the Model and use it to Generate!

Save your fine-tuned model and compare its output with regular GPT-2's output to see the difference for yourself!

In [ ]:
```
trainer.save_model('./dostoevskypt2')
```

In [ ]:
```
from transformers import pipeline
dostoevskypt2 = pipeline('text-generation', model='./dostoevskypt2',
gpt2 = pipeline('text-generation', model='gpt2', device=0)
```

In [ ]:
```
print(dostoevskypt2('Saint Petersburg is'))
print(gpt2('Saint Petersburg is'))
```

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

[{'generated_text': 'Saint Petersburg is haunted by the same morbid
feeling of haunted illness: there are not many people who remember.
“When the cold came in I couldn’t bring myself to go to the shop. Th
ey wouldn’t even take me out'}]
[{'generated_text': 'Saint Petersburg is an extremely important cent
re — the oldest city in Russia — one of the top five in the world an
d one of the second-best centers of science, technology, engineerin
g, and mathematics in the world. It boasts an amazing population of
nearly'}]
```

PERPLEXITY

12. (2 pts) Using the pointer [here (https://huggingface.co/transformers/perplexity.html)](https://huggingface.co/transformers/perplexity.html), compute the perplexity of the GPT2 pre-trained model on the Wikipedia test set (you can keep the same hyperparameters as in the link)

In [ ]:
```
## YOUR CODE HERE - FOR COMPUTING PERPLEXITY OF GPT2 ON WIKIPEDIA TES

# ANSWERS BELOW:
# Load wiki test set
from datasets import load_dataset
import torch
from tqdm import tqdm

test = load_dataset("wikitext", "wikitext-2-raw-v1", split="test")
encodings = tokenizer("\n\n".join(test["text"]), return_tensors="pt")
max_length = model.config.n_positions
stride = 512

# Define a function for ppl
def ppl(model, input_ids_all, stride):
  nlls = []
  for i in tqdm(range(0, input_ids_all.size(1), stride)):
      begin_loc = max(i + stride - max_length, 0)
      end_loc = min(i + stride, input_ids_all.size(1))
      trg_len = end_loc - i  # may be different from stride on last l
      input_ids = input_ids_all[:, begin_loc:end_loc].to("cuda:0")
      target_ids = input_ids.clone()
      target_ids[:, :-trg_len] = -100

      with torch.no_grad():
          outputs = model(input_ids, labels=target_ids)
          neg_log_likelihood = outputs[0] * trg_len

      nlls.append(neg_log_likelihood)

  ppl = torch.exp(torch.stack(nlls).sum() / end_loc)
  return ppl

perplexity = ppl(model, encodings.input_ids, stride)
print(f"perplexity is {perplexity}.")
```

```
100%|██████████| 562/562 [00:20<00:00, 27.89it/s]

perplexity is 88.01296997070312.
```

> YOUR PERPLEXITY ANSWER HERE:
> perplexity is 88.01296997070312.

13. (2 pts) Compute the perplexity of the dostoevskypt2 model on Wikipedia test set

In [ ]:
```python
## YOUR CODE HERE - FOR COMPUTING PERPLEXITY OF DOSTOEVSKYPT2 ON WIKI
from transformers import GPT2Tokenizer, GPT2LMHeadModel
from datasets import load_dataset
import torch
from tqdm import tqdm

# Load the custom model 'dostoevskypt2' and tokenizer
model = GPT2LMHeadModel.from_pretrained('dostoevskypt2')
tokenizer = GPT2Tokenizer.from_pretrained('dostoevskypt2')

# Load the Wikipedia test dataset
test = load_dataset("wikitext", "wikitext-2-raw-v1", split="test")

# Prepare the encodings of test data for the model
encodings = tokenizer("\n\n".join(test["text"]), return_tensors="pt")
max_length = model.config.n_positions
stride = 512

# Define the function to compute perplexity
def ppl(model, input_ids_all, stride):
    nlls = []
    for i in tqdm(range(0, input_ids_all.size(1), stride)):
        begin_loc = max(i + stride - max_length, 0)
        end_loc = min(i + stride, input_ids_all.size(1))
        trg_len = end_loc - i  # may be different from stride on last
        input_ids = input_ids_all[:, begin_loc:end_loc].to(model.devi
        target_ids = input_ids.clone()
        target_ids[:, :-trg_len] = -100

        with torch.no_grad():
            outputs = model(input_ids, labels=target_ids)
            neg_log_likelihood = outputs[0] * trg_len

        nlls.append(neg_log_likelihood)

    ppl = torch.exp(torch.stack(nlls).sum() / end_loc)
    return ppl

# Compute the perplexity
perplexity = ppl(model, encodings.input_ids, stride)
print(perplexity)
```

```
Token indices sequence length is longer than the specified maximum s
equence length for this model (287644 > 1024). Running this sequence
through the model will result in indexing errors
100%|██████████| 562/562 [17:45<00:00,  1.90s/it]

tensor(68.7596)
```

Perplexity: 68.7596

14. (2 pts) Compute the perplexity of the GPT2 pre-trained model on the Crime and Punishment train dataset

In [ ]:
```
from transformers import GPT2Tokenizer, GPT2LMHeadModel
import torch
from tqdm import tqdm
from tensorflow.compat.v1.io.gfile import GFile

# Load the GPT-2 model and tokenizer
model = GPT2LMHeadModel.from_pretrained('gpt2')
tokenizer = GPT2Tokenizer.from_pretrained('gpt2')

# Read the text file from Google Cloud Storage
with GFile('gs://trax-ml/reformer/crime-and-punishment-2554.txt') as
    text = file.read()

# Tokenize the text
encodings = tokenizer(text, return_tensors="pt")
max_length = model.config.n_positions
stride = 512

# Compute the perplexity
perplexity = ppl(model, encodings.input_ids, stride)
print("Perplexity:", perplexity.item())
```

```
Token indices sequence length is longer than the specified maximum s
equence length for this model (344946 > 1024). Running this sequence
through the model will result in indexing errors
100%|██████████| 674/674 [17:09<00:00,  1.53s/it]

Perplexity: 115.53711700439453
```

Perplexity: 115.53711700439453

15. (2 pts) Compute the **train** perplexity of the **dostoevskypt2** model

In [ ]:
```
## YOUR CODE HERE - FOR COMPUTING PERPLEXITY OF DOSTOEVSKYPT2 ON CRIM
# Load the custom model 'dostoevskypt2' and tokenizer
model = GPT2LMHeadModel.from_pretrained('dostoevskypt2')
tokenizer = GPT2Tokenizer.from_pretrained('dostoevskypt2')

# Read the text file from Google Cloud Storage
with GFile('gs://trax-ml/reformer/crime-and-punishment-2554.txt') as
    text = file.read()

# Tokenize the text
encodings = tokenizer(text, return_tensors="pt")
max_length = model.config.n_positions
stride = 512

# Compute the perplexity
perplexity = ppl(model, encodings.input_ids, stride)
print("Perplexity:", perplexity.item())
```

```
Token indices sequence length is longer than the specified maximum s
equence length for this model (344946 > 1024). Running this sequence
through the model will result in indexing errors
100%|██████████| 674/674 [19:22<00:00,  1.72s/it]

Perplexity: 79.84642028808594
```

Perplexity: 79.84642028808594

(1 pt) Which model performs better on Crime and Punishment train set, vanilla GPT-2 or your dostoevskypt2 checkpoint?

As the perplexity of vanilla gpt2 is way higher than that of dostoevskypt2, dostoevskypt2 performs better in terms of understanding the nuances of "Crime and Punishment."

16. (2 pts) Compute perplexity of the GPT2 model on your raw pride and prejudice text.

In [ ]:
```
## YOUR CODE HERE - FOR COMPUTING PERPLEXITY OF GPT2 ON PRIDE AND PRE
# Load the GPT-2 model and tokenizer
model = GPT2LMHeadModel.from_pretrained('gpt2')
tokenizer = GPT2Tokenizer.from_pretrained('gpt2')
max_length = model.config.n_positions
stride = 512

# Read the content of your text file
with open('prideAndPrejudice.txt', 'r', encoding='utf-8') as file:
    text = file.read()

# Tokenize the text
encodings = tokenizer(text, return_tensors="pt")

# Compute the perplexity
perplexity = ppl(model, encodings.input_ids, stride)
print(perplexity)
```

```
Token indices sequence length is longer than the specified maximum s
equence length for this model (153493 > 1024). Running this sequence
through the model will result in indexing errors
100%|██████████| 300/300 [07:19<00:00,  1.46s/it]

tensor(29.1430)
```

> Perplexity: 29.1430

17. (2 pts) Compute perplexity of the **dostoevskypt2** model on your raw pride and prejudice text.

In [ ]:
```
## YOUR CODE HERE - FOR COMPUTING PERPLEXITY OF dostoevskipt2 ON PRID
from transformers import GPT2Tokenizer, GPT2LMHeadModel
import torch
from tqdm import tqdm

# Load the custom model and tokenizer
model = GPT2LMHeadModel.from_pretrained('dostoevskypt2')
tokenizer = GPT2Tokenizer.from_pretrained('dostoevskypt2')

# Read the text file
with open('prideAndPrejudice.txt', 'r', encoding='utf-8') as file:
    text = file.read()

# Tokenize the text
encodings = tokenizer(text, return_tensors="pt")
max_length = model.config.n_positions
stride = 512

# Compute the perplexity
perplexity = ppl(model, encodings.input_ids, stride)
print("Perplexity:", perplexity.item())
```

```
Token indices sequence length is longer than the specified maximum s
equence length for this model (153493 > 1024). Running this sequence
through the model will result in indexing errors
100%|██████████| 300/300 [07:13<00:00,  1.45s/it]

Perplexity: 42.6693000793457
```

> Perplexity: 42.6693000793457

Now's Your Turn!

Your job is to fine-tune GPT2 one more time with your choice of fine-tuning dataset.

*****For the fine-tuned model you create, you should clearly demonstrate (through visible generation outputs and analysis) that your fine-tuned model follows the desired style better than vanilla GPT2** ***

Please make sure to give a brief description

In order to see which datasets are available for download, run the cell below. Pick one that you think would be interesting!

In [ ]:
```
datasets_list = list_datasets()
print(', '.join(dataset for dataset in datasets_list))
```

```
<ipython-input-22-6bc899386cec>:1: FutureWarning: list_datasets
is deprecated and will be removed in the next major version of d
atasets. Use 'huggingface_hub.list_datasets' instead.
  datasets_list = list_datasets()

acronym_identification, ade_corpus_v2, adversarial_qa, aeslc, af
rikaans_ner_corpus, ag_news, ai2_arc, air_dialogue, ajgt_twitter
_ar, allegro_reviews, allocine, alt, amazon_polarity, amazon_rev
iews_multi, amazon_us_reviews, ambig_qa, americas_nli, ami, amtt
l, anli, app_reviews, aqua_rat, aquamuse, bigIR/ar_cov19, ar_res
_reviews, ar_sarcasm, arabic_billion_words, arabic_pos_dialect,
arabic_speech_corpus, arcd, arsentd_lev, art, arxiv_dataset, asc
ent_kb, aslg_pc12, asnq, asset, assin, assin2, atomic, autshumat
o, facebook/babi_qa, banking77, bbaw_egyptian, bbc_hindi_nli, bc
2gm_corpus, beans, best2009, bianet, bible_para, big_patent, bil
lsum, bing_coronavirus_query_set, biomrc, biosses, TheBritishLib
rary/blbooks, TheBritishLibrary/blbooksgenre, blended_skill_tal
k, blimp, blog_authorship_corpus, bn_hate_speech, bnl_newspaper
s, bookcorpus, bookcorpusopen, boolq, bprec, break_data, brwac,
bsd_ja_en, bswac, c3, c4, cail2018, caner, capes, casino, catalo
```

Tips

- Most of the datasets hosted by HuggingFace are not meant for Causal LM fine-tuning. Make sure you preprocess them accordingly if you want to use them.
- In order to check out information about a dataset hosted by huggingface you can use [this web viewer (https://huggingface.co/datasets/viewer/?dataset=crime_and_punish)](https://huggingface.co/datasets/viewer/?dataset=crime_and_punish). Try to avoid downloading a dataset that's too big!
- You will likely have to change the custom 'encode' function for each new dataset you want to fine-tune on. You need to change batch['line'] to instead index with the correct column label for your specific dataset (it probably wont be called 'line').

Useful Links

load_datasets Documentation (https://huggingface.co/docs/datasets/package_reference/loading_methods.html#datasets.lo

Trainer Documentation (https://huggingface.co/transformers/main_classes/trainer.html#id1)

Example: Fine-Tuning BERT for Esperanto (https://colab.research.google.com/github/huggingface/blog/blob/master/notebooks/01_how

Example: Fine-Tuning for IMDb Classification (https://colab.research.google.com/drive/1-JIJlao4dI-Ilww_NnTc0rxtp-ymgDgM?usp=sharing#scrollTo=5DEWNilys9Ty)

18. Dataset #1

In [ ]:
```
from transformers import GPT2LMHeadModel, GPT2Tokenizer, DataCollator
from datasets import load_dataset

# Load tokenizer and model
tokenizer = GPT2Tokenizer.from_pretrained('gpt2')
model = GPT2LMHeadModel.from_pretrained('gpt2')

# Set tokenizer's padding token
tokenizer.pad_token = tokenizer.eos_token

# Load IMDb dataset
dataset = load_dataset("imdb")

# Function to preprocess and tokenize the dataset
def preprocess_function(examples):
    return tokenizer([f"Review: {text} [POSITIVE]" if label == 1 else
                     truncation=True, max_length=1024, padding="max_l

# Tokenize and preprocess the dataset
tokenized_dataset = dataset.map(preprocess_function, batched=True)
tokenized_dataset.set_format('torch', columns=['input_ids', 'attentic

# Use DataCollatorForLanguageModeling
data_collator = DataCollatorForLanguageModeling(tokenizer=tokenizer,

# Define training arguments
training_args = TrainingArguments(
    output_dir="./results",
    overwrite_output_dir=True,
    num_train_epochs=3,
    per_device_train_batch_size=4,
    warmup_steps=500,
    weight_decay=0.01,
    logging_dir="./logs",
)

# Initialize Trainer
trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=tokenized_dataset['train'],
    eval_dataset=tokenized_dataset['test'],
    data_collator=data_collator,
)

# Train the model
trainer.train()
```

```
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]
```

```
model.safetensors:   0%|          | 0.00/548M [00:00<?, ?B/s]

generation_config.json:   0%|          | 0.00/124 [00:00<?, ?B/s]

Downloading builder script:   0%|          | 0.00/4.31k [00:00<?, ?
B/s]

Downloading metadata:   0%|          | 0.00/2.17k [00:00<?, ?B/s]

Downloading readme:   0%|          | 0.00/7.59k [00:00<?, ?B/s]

Downloading data:   0%|          | 0.00/84.1M [00:00<?, ?B/s]

Generating train split:   0%|          | 0/25000 [00:00<?, ? example
s/s]

Generating test split:   0%|          | 0/25000 [00:00<?, ? example
s/s]

Generating unsupervised split:   0%|          | 0/50000 [00:00<?, ?
examples/s]

Map:   0%|          | 0/25000 [00:00<?, ? examples/s]

Map:   0%|          | 0/25000 [00:00<?, ? examples/s]

Map:   0%|          | 0/50000 [00:00<?, ? examples/s]
```

[2456/18750 16:54 < 1:52:14, 2.42 it/s, Epoch 0.39/3]

Step	Training Loss
500	3.685200
1000	3.600900
1500	3.569500
2000	3.528600

```
---------------------------------------------------------------------------
OutOfMemoryError                          Traceback (most recent call last)
<ipython-input-1-7c2c29b22905> in <cell line: 47>()
     45 
     46 # Train the model
---> 47 trainer.train()

/usr/local/lib/python3.10/dist-packages/transformers/trainer.py in train(self, resume_from_checkpoint, trial, ignore_keys_for_eval, **kwargs)
   1553                 hf_hub_utils.enable_progress_bars()
   1554         else:
-> 1555             return inner_training_loop(
   1556                 args=args,
   1557                 resume_from_checkpoint=resume_from_checkpoint,

/usr/local/lib/python3.10/dist-packages/transformers/trainer.py in _inner_training_loop(self, batch_size, args, resume_from_checkpoint, trial, ignore_keys_for_eval)
   1858 
   1859                 with self.accelerator.accumulate(model):
-> 1860                     tr_loss_step = self.training_step(model, inputs)
   1861 
   1862                 if (

/usr/local/lib/python3.10/dist-packages/transformers/trainer.py in training_step(self, model, inputs)
   2732                 scaled_loss.backward()
   2733         else:
-> 2734             self.accelerator.backward(loss)
   2735 
   2736         return loss.detach() / self.args.gradient_accumulation_steps

/usr/local/lib/python3.10/dist-packages/accelerate/accelerator.py in backward(self, loss, **kwargs)
   1903             self.scaler.scale(loss).backward(**kwargs)
   1904         else:
-> 1905             loss.backward(**kwargs)
   1906 
   1907     def set_trigger(self):

/usr/local/lib/python3.10/dist-packages/torch/_tensor.py in backward(self, gradient, retain_graph, create_graph, inputs)
    490                 inputs=inputs,
    491             )
--> 492         torch.autograd.backward(
    493             self, gradient, retain_graph, create_graph, inputs=inputs
    494         )

/usr/local/lib/python3.10/dist-packages/torch/autograd/__init__.py in backward(tensors, grad_tensors, retain_graph, create_graph, grad_v
```

```
ariables, inputs)
    249     # some Python versions print out the first line of a mul
ti-line function
    250     # calls in the traceback and some print out the last lin
e
--> 251     Variable._execution_engine.run_backward(  # Calls into t
he C++ engine to run the backward pass
    252         tensors,
    253         grad_tensors_,

OutOfMemoryError: CUDA out of memory. Tried to allocate 786.00 MiB.
GPU 0 has a total capacity of 15.77 GiB of which 738.38 MiB is free.
Process 33157 has 15.05 GiB memory in use. Of the allocated memory 1
4.01 GiB is allocated by PyTorch, and 682.09 MiB is reserved by PyTo
rch but unallocated. If reserved but unallocated memory is large try
setting max_split_size_mb to avoid fragmentation.  See documentation
for Memory Management and PYTORCH_CUDA_ALLOC_CONF
```

(4 pts) The IMDb dataset is a widely-used dataset for binary sentiment classification tasks. It consists of 50,000 movie reviews from the Internet Movie Database (IMDb), a popular online database for movies, television, and celebrity content. These reviews are evenly split into two sets: 25,000 for training and 25,000 for testing. Each set contains an equal number of positive and negative reviews, making the dataset well-balanced. The reviews are in plain text and vary in length, providing a rich resource for natural language processing (NLP) tasks related to sentiment analysis.

In terms of utility, the IMDb dataset is a benchmark for evaluating the performance of various machine learning models, particularly those focused on sentiment analysis. The nature of the dataset, with its informal and opinionated text, poses a unique challenge for models to accurately discern and categorize the sentiment expressed in each review. This makes it an ideal choice for training and testing models in the realm of NLP, including but not limited to traditional machine learning models, deep learning approaches, and more recent transformer-based architectures. The dataset's inclusion on Hugging Face allows for easy access and integration with modern NLP tools and frameworks, facilitating research and development in sentiment analysis and related fields.

In [ ]:
```
from transformers import GPT2Tokenizer, GPT2LMHeadModel
from datasets import load_dataset
import torch
from tqdm import tqdm

def ppl(model, input_ids_all, stride=512):
    max_length = model.config.n_positions
    nlls = []
    for i in tqdm(range(0, input_ids_all.size(1), stride)):
        begin_loc = max(i + stride - max_length, 0)
        end_loc = min(i + stride, input_ids_all.size(1))
        trg_len = end_loc - i
        input_ids = input_ids_all[:, begin_loc:end_loc].to(model.devi
        target_ids = input_ids.clone()
        target_ids[:, :-trg_len] = -100

        with torch.no_grad():
            outputs = model(input_ids, labels=target_ids)
            neg_log_likelihood = outputs[0] * trg_len

        nlls.append(neg_log_likelihood)

    return torch.exp(torch.stack(nlls).sum() / end_loc).item()

# Load the tokenizer
tokenizer = GPT2Tokenizer.from_pretrained('gpt2')

# Set tokenizer's padding token to EOS token
tokenizer.pad_token = tokenizer.eos_token

# Load a subset of the IMDb dataset for perplexity calculation
dataset = load_dataset("imdb", split='test[:1%]')
texts = ["Review: " + text for text in dataset['text']]
encodings = tokenizer(texts, return_tensors="pt", truncation=True, ma

# Load the fine-tuned GPT-2 model
fine_tuned_model = GPT2LMHeadModel.from_pretrained('./results/checkpo
fine_tuned_model.to('cuda')

# Load the vanilla GPT-2 model
vanilla_model = GPT2LMHeadModel.from_pretrained('gpt2')
vanilla_model.to('cuda')

# Calculate perplexity for the fine-tuned model
fine_tuned_ppl = ppl(fine_tuned_model, encodings['input_ids'].to('cuc
print(f"Fine-Tuned GPT-2 Perplexity: {fine_tuned_ppl}")

# Calculate perplexity for the vanilla model
vanilla_ppl = ppl(vanilla_model, encodings['input_ids'].to('cuda'))
print(f"Vanilla GPT-2 Perplexity: {vanilla_ppl}")
```

(5 pts) YOUR ANSWER HERE - COMPARISON OF YOUR DATASET'S FINE-TUNED OUTPUT VS NON-FINE-TUNED OUTPUT