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

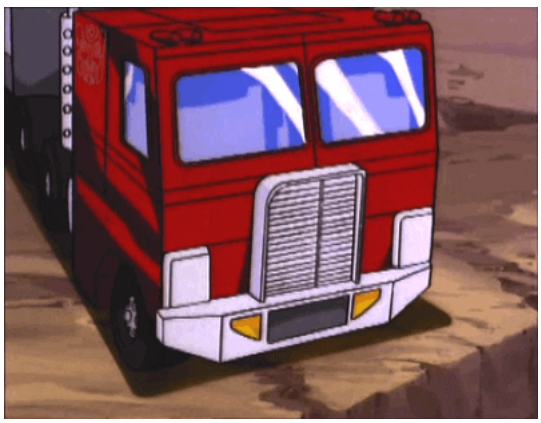
For this homework, we will take ideas from the entire class: language models, text generation, vector-based word representations, syntactic analysis, and neural networks. We'll be using large, pre-trained language models to generate text, and studying how we can fine-tune these large language models 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](). 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 mechanism that enables each position in the input sequence to attend to/consider all positions in the previous layer of the model simultaneously. This helps the model to understand contexts and relationships within the input data. This allows transformers to be do dynamic and context aware preocessing as copared to earlier sequence models. The name comes from the fact that contrary to "regular" attention, self attention refers to the same sequence which is being currently encoded.

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

> Generate Query, Key, Value Vectors: For each word in the input, generate query (Q), key (K), and value (V) vectors through learned linear transformations. Calculate Attention Scores: Compute dot product of Q and K for each word, scale down by square root of dimension of K. Apply Softmax: Run the scores through softmax to get attention weights. Compute Weighted Sum: Multiply each V by its softmax score (attention weight) and sum up.

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

> Query Vectors (Q): They represent the word that's currently being focused on, essentially asking questions about other words. Key Vectors (K): These are like identifiers for each word, providing the information that the query vectors interact with. Value Vectors (V): Once a query and a key interact (through attention scores), value vectors provide the actual content that's going to be processed further.

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

> An attention "head" in the Transformer model is a single attention mechanism in a layer. Each head independently performs the attention computation, resulting in different sets of Q, K, and V vectors. Using multiple heads allows the model to simultaneously focus on different types of information from different representational spaces at each position in the sequence. This parallel processing enhances the model's ability to capture various aspects of the data, leading to more robust and nuanced understanding.

5. (2 pts) What are positional embeddings?

> Positional embeddings in the Transformer model are added to the input embeddings to provide information about the position of each word in the sequence. Since the Transformer lacks inherent sequential processing like RNNs, positional embeddings are crucial for understanding the order of words. They enable the model to distinguish between words based on their position in the sequence, which is essential for understanding the meaning and context in language processing tasks.

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

> Positional embeddings are crucial in Transformers for incorporating the sequence context. While Transformers excel at parallel processing of words, they don't inherently recognize word order, which is vital for understanding phrases and sentences. Positional embeddings encode the position of each word, ensuring the model appreciates the sequential

> nature of language, such as differentiating "dog bites man" from "man bites dog." This positional information is integral for accurate language understanding and generation.

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:

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, 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 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?

> The difference is primarily in its structure and intended use. GPT-2 is a decoder-only model, meaning it consists only of the decoder part of the Transformer architecture. This design makes it highly effective for language generation tasks. In contrast, the standard Transformer model includes both encoder and decoder components, facilitating tasks involving understanding input data (encoding) and generating output data (decoding), like translation.

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

> "Self-Attention" is a mechanism in Transformer models allowing each position in the input sequence to attend to all positions in the previous layer of the model. It helps the model to understand context and relationships within the input.

"Masked Self-Attention" is a variant where certain positions are masked or hidden during training. This setup allows the model to predict missing words or tokens, making it effective for tasks like language understanding and context filling.

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 output scores from a model's final layer. In GPT-2, logits are computed using the Transformer's decoder architecture. Each token's logit represents how likely it is to be the next token in the sequence. GPT-2 uses a softmax function to convert these logits into probabilities. The model then uses those probabilities to select the next word in the generated text, though techniques like sampling (from all or top-k) or beam search or just selecting the highest one!(wont have diversity in outcomes then)

Aside: GPT-3

On June 11th 2020, OpenAI released GPT-3 (paper) (wikipedia). 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)

Here's a NYT article about how GPT-3 can write code, poetry, and argue

Here's an article GPT-3 wrote for The Guardian about how it loves humans and would never subjugate humanity

You may optionally choose to read Jay Alammar's most recent blog post "How GPT3 Works - Visualizations and Animations" from July 2020 if you're curious as to how GPT-3 differs from GPT-2

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, 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.

```
!pip install transformers
```

```
    Requirement already satisfied: transformers in /usr/local/lib/python3.10/dist-packages (4.35.2)
    Requirement already satisfied: filelock in /usr/local/lib/python3.10/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 transformer
    Requirement already satisfied: numpy>=1.17 in /usr/local/lib/python3.10/dist-packages (from transformers) (1.23.5)
    Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.10/dist-packages (from transformers) (23.2)
    Requirement already satisfied: pyyaml>=5.1 in /usr/local/lib/python3.10/dist-packages (from transformers) (6.0.1)
    Requirement already satisfied: regex!=2019.12.17 in /usr/local/lib/python3.10/dist-packages (from transformers) (2023.6.
    Requirement already satisfied: requests in /usr/local/lib/python3.10/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.
    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/python3.10/dist-packages (from huggingface-hub<1.0,>=0
    Requirement already satisfied: typing-extensions>=3.7.4.3 in /usr/local/lib/python3.10/dist-packages (from huggingface-h
    Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/dist-packages (from requests->trans
    Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (from requests->transformers) (3.
    Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.10/dist-packages (from requests->transformer
    Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-packages (from requests->transformer
```

2.1 The 'Pipeline' Interface

The simplest way to use the HuggingFace library is to use their Pipeline interface

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

```
from transformers import pipeline


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

```
config.json: 100%                                   665/665 [00:00<00:00, 41.5kB/s]
model.safetensors: 100%                             548M/548M [00:03<00:00, 180MB/s]
generation_config.json: 100%                        124/124 [00:00<00:00, 8.37kB/s]
vocab.json: 100%                                    1.04M/1.04M [00:00<00:00, 13.9MB/s]
merges.txt: 100%                                    456k/456k [00:00<00:00, 17.8MB/s]
```

```
outputs = generator('I wonder what I will generate?')
print(outputs)
```

```
Setting `pad_token_id` to `eos_token_id`:50256 for open-end generation.
[{'generated_text': "I wonder what I will generate? I don't really know what that is, which is why I can't answer. Maybe
```

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.

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)**

Limiting the max output to 50

```
## YOUR CODE HERE FOR MODEL 1
# Use a larger variant of GPT-2
gpt2_large_generator = pipeline('text-generation', model='gpt2-large')
prompt = "I wonder what I will generate?"
generated_text_gpt2 = gpt2_large_generator(prompt, max_length=50)
print(generated_text_gpt2[0]['generated_text'])
```

```
Setting `pad_token_id` to `eos_token_id`:50256 for open-end generation.
I wonder what I will generate? I want to see what happens when I mix my two kinds of energy."

The results from our experiments with the TEM, and the new understanding that the energy produced is actually heat, have
```

XLNet is a generalized autoregressive pretraining method.

```
## YOUR CODE HERE FOR MODEL 2
# Use the XLNet model
xlnet_generator = pipeline('text-generation', model='xlnet-base-cased')
generated_text_xlnet = xlnet_generator(prompt, max_length=50)
print(generated_text_xlnet[0]['generated_text'])
```

```
I wonder what I will generate? You never know. Maybe a new word for your local brand of slang. Or maybe it will get to m
```

The output from these two models would likely exhibit different characteristics. GPT-2 variants tend to generate text that is coherent and follows the narrative style of the prompt closely. XLNet, on the other hand, may provide a different perspective or style due to its different training approach and architecture.

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.

```
from transformers import GPT2Tokenizer, GPT2LMHeadModel# AutoTokenizer, AutoModelForCausalLM

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


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

```
Input Token IDs: tensor([[15496,    11,   703,   389,   345,    30]], device='cuda:0')
```

```
#### Step 2 and 3: Feed in the integer token IDs and get out a sequence of token IDs as output
outputs = model.generate(inputs)
print("Output Token IDs: " + str(outputs))
```

```
The attention mask and the pad token id were not set. As a consequence, you may observe unexpected behavior. Please pass
Setting `pad_token_id` to `eos_token_id`:50256 for open-end generation.
/usr/local/lib/python3.10/dist-packages/transformers/generation/utils.py:1273: UserWarning: Using the model-agnostic def
  warnings.warn(
Output Token IDs: tensor([[15496,    11,   703,   389,   345,    30,   198,   198,    40,  1101,
           257,  1310,  1643,   286,   257, 34712,    13,   314,  1101,   257]],
       device='cuda:0')
```

```
#### Step 4: Feed in the integer token IDs and get out a sequence of token IDs as output
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 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

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

```
inputs = tokenizer.encode("Hello, how are you?", return_tensors='pt').to(model.device)
outputs = model.generate(
      inputs,
      do_sample=True,          # Randomly sample from the logits instead of greedily picking next word with highest probabil
      top_k=50,                 # Only sample from the top 50 most likely words
      repetition_penalty=1.2,    # Downweights the probability of all previously generated tokens by a factor of 1.2
      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 consequence, you may observe unexpected behavior. Please pass
Setting `pad_token_id` to `eos_token_id`:50256 for open-end generation.
Hello, how are you?
The main thing I want to do is make this website as simple and efficient for web developers. It's not that old-school bu
```

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. :)

```
## YOUR CODE HERE FOR HYPERPARAMETER VARIATION 1
inputs = tokenizer.encode("Hello, how are you?", return_tensors='pt').to(model.device)
output_example1 = model.generate(
    inputs,
    do_sample=True,
    top_k=30,
    temperature=0.7,
    max_length=512
)
print("Example 1 Output: " + tokenizer.decode(output_example1[0], skip_special_tokens=True))
```

```
The attention mask and the pad token id were not set. As a consequence, you may observe unexpected behavior. Please pass
Setting `pad_token_id` to `eos_token_id`:50256 for open-end generation.
Example 1 Output: Hello, how are you?

A: I'm not sure. I mean, I love her, and I think I know what she wants.

Q: But you're a bit of a dickhead.

A: Oh yeah, I mean, I'm definitely a dickhead. I know, I know.

Q: I think it's just a matter of time. But I guess you'll be fine.

A: Yeah, I think it's fine. I think I'll be fine.

Q: All right, you're in good spirits. So, who will be the next president of the United States?

A: I think we'll see. I mean, I mean, I think it'll be a little bit of a question. You know, I'm a little bit of a dickh

Q: I think you'll be president.

A: Well, I don't know. But I think I'll be president.

Q: What kind of president are you?

A: Uh-huh. Well, I mean, it's an interesting question. So, where are you from?

Q: Where are you from?

A: Uh-huh.

Q: So, how did you get here?

A: I got here from my brother, so I'm from Chicago, so I'm from Chicago, right?

Q: Why do you guys want to join the military?

A: Well, I guess I've got some friends out there, I guess.

Q: So, you guys are going to be there for a while, right?

A: I guess it's going to be a little bit of a shock to them.

Q: You're going to be there for a while?

A: Well, I guess they're going to be going to the military.

Q: And the military is going to be the one that will be going there?

A: So, it's going to be the Marine Corps.

Q: And the Marines?

A: Well, I guess they're going to be there, right?

Q: And the people in the Marines?

A: Yeah?
```

(4 pts) YOUR ANSWER HERE - EXPLANATION FOR HPARAM VARIATION 1 In Example 1, using top_k=30 and temperature=0.7, the generated output is a dialogue format that appears to be part of a casual conversation. The top_k=30 parameter makes the model focus on the top 30 most likely words, leading to a more predictable and coherent text, while temperature=0.7 slightly reduces randomness, giving a more realistic conversational tone. The conversation touches on personal opinions and experiences, reflecting a natural flow and contextually relevant responses, characteristic of the controlled randomness and coherence imposed by these hyperparameters.

```
## YOUR CODE HERE FOR HYPERPARAMETER VARIATION 2
inputs = tokenizer.encode("Hello, how are you?", return_tensors='pt').to(model.device)
output_example2 = model.generate(
    inputs,
    do_sample=True,
    top_p=0.9,
    temperature=1.2,
    max_length=512
)
print("Example 2 Output: " + tokenizer.decode(output_example2[0], skip_special_tokens=True))
```

```
The attention mask and the pad token id were not set. As a consequence, you may observe unexpected behavior. Please pass
Setting `pad_token_id` to `eos_token_id`:50256 for open-end generation.
Example 2 Output: Hello, how are you? Is your mom really going to leave? It's sad for us, we were worried she'd give you

"I did, and that scared him," Blake sighed.

"Is she?" Yang asked.

"No," Blake said, a little confused. "The guy was just sitting there, so I didn't believe him when he asked. So I'm not

"Yeah? Yeah. So why was he not the one?" Yang asked.

"Yeah, and I didn't know about it," Blake lied.

Yang asked what exactly made the man's mom afraid she might not give her up or something.

"Maybe it was something she knew someone was doing at home or something. Maybe it was something she just wasn't comforta

Yang started telling her story about her mom and what she had witnessed and how her mom had gone crazy because she was n

"Is that possible? So there were no fights or anything?" Blake asked.

"No, no," Yang lied.

Yang shook her head. "You're an alcoholic. Are you kidding me, this man doesn't drink and talk like he would be sober if

The man's mother said something about how she loved him but didn't understand why he thought her would not do anything i

"You just believe he can walk through the door and just sit there. How would you explain what his purpose was?" Yang ask

"He says he was an alcoholic in his teens. He's not. He's a really good man. He came across a guy like this in college w

Yang sighed. "She can talk and talk, if she wants," she said.

This was going to
```

(4 pts) YOUR ANSWER HERE -- EXPLANATION FOR HPARAM VARIATION 2 In Example 2, with top_p=0.9 and temperature=1.2, the generated output is a complex, emotionally charged conversation. The use of top_p=0.9 for nucleus sampling allows the model to generate more diverse and less predictable text by sampling from a broader range of possible next words. The temperature=1.2 increases the randomness further, leading to unexpected twists and more creative elements in the dialogue. This results in a narrative that is less coherent but more intriguing and varied, demonstrating the impact of these hyperparameters on the style and content of the generated 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 library.

```
!pip install -U datasets
```

```
Collecting datasets
  Downloading datasets-2.16.0-py3-none-any.whl (507 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 507.1/507.1 kB 3.4 MB/s eta 0:00:00
Requirement already satisfied: filelock in /usr/local/lib/python3.10/dist-packages (from datasets) (3.13.1)
Requirement already satisfied: numpy>=1.17 in /usr/local/lib/python3.10/dist-packages (from datasets) (1.23.5)
```

```
Requirement already satisfied: pyarrow>=8.0.0 in /usr/local/lib/python3.10/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 11.0 MB/s eta 0:00:00
Requirement already satisfied: pandas in /usr/local/lib/python3.10/dist-packages (from datasets) (1.5.3)
Requirement already satisfied: requests>=2.19.0 in /usr/local/lib/python3.10/dist-packages (from datasets) (2.31.0)
Requirement already satisfied: tqdm>=4.62.1 in /usr/local/lib/python3.10/dist-packages (from datasets) (4.66.1)
Requirement already satisfied: xxhash in /usr/local/lib/python3.10/dist-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 16.3 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 datas
Requirement already satisfied: aiohttp in /usr/local/lib/python3.10/dist-packages (from datasets) (3.9.1)
Requirement already satisfied: huggingface-hub>=0.19.4 in /usr/local/lib/python3.10/dist-packages (from datasets) (0.19.
Requirement already satisfied: packaging in /usr/local/lib/python3.10/dist-packages (from datasets) (23.2)
Requirement already satisfied: pyyaml>=5.1 in /usr/local/lib/python3.10/dist-packages (from datasets) (6.0.1)
Requirement already satisfied: attrs>=17.3.0 in /usr/local/lib/python3.10/dist-packages (from aiohttp->datasets) (23.1.0
Requirement already satisfied: multidict<7.0,>=4.5 in /usr/local/lib/python3.10/dist-packages (from aiohttp->datasets) (
Requirement already satisfied: yarl<2.0,>=1.0 in /usr/local/lib/python3.10/dist-packages (from aiohttp->datasets) (1.9.4
Requirement already satisfied: frozenlist>=1.1.1 in /usr/local/lib/python3.10/dist-packages (from aiohttp->datasets) (1.
Requirement already satisfied: aiosignal>=1.1.2 in /usr/local/lib/python3.10/dist-packages (from aiohttp->datasets) (1.3
Requirement already satisfied: async-timeout<5.0,>=4.0 in /usr/local/lib/python3.10/dist-packages (from aiohttp->dataset
Requirement already satisfied: typing-extensions>=3.7.4.3 in /usr/local/lib/python3.10/dist-packages (from huggingface-h
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/dist-packages (from requests>=2.19.
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (from requests>=2.19.0->datasets)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.10/dist-packages (from requests>=2.19.0->dat
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-packages (from requests>=2.19.0->dat
Requirement already satisfied: python-dateutil>=2.8.1 in /usr/local/lib/python3.10/dist-packages (from pandas->datasets)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-packages (from pandas->datasets) (2023.3.p
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/dist-packages (from python-dateutil>=2.8.1->pandas-
Installing collected packages: pyarrow-hotfix, dill, multiprocess, datasets
Successfully installed datasets-2.16.0 dill-0.3.7 multiprocess-0.70.15 pyarrow-hotfix-0.6
```

```
# export LC_ALL=C.UTF-8
# export LANG=C.UTF-8
```

```
from transformers import Trainer, TrainingArguments, DataCollatorForLanguageModeling
import datasets
from datasets import load_dataset, list_datasets
```

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

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

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

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.

```
def encode(batch): return tokenizer([x.strip('\n\r') for x in batch['line']], truncation=True, padding=True)

crime_and_punishment = load_dataset('crime_and_punish', split='train')
processed = crime_and_punishment.map(encode, batched=True, batch_size=len(crime_and_punishment))
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).

```
!pip install accelerate -U
```

```
Collecting accelerate
  Downloading accelerate-0.25.0-py3-none-any.whl (265 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 265.7/265.7 kB 2.1 MB/s eta 0:00:00
Requirement already satisfied: numpy>=1.17 in /usr/local/lib/python3.10/dist-packages (from accelerate) (1.23.5)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.10/dist-packages (from accelerate) (23.2)
Requirement already satisfied: psutil in /usr/local/lib/python3.10/dist-packages (from accelerate) (5.9.5)
```

```
Requirement already satisfied: pyyaml in /usr/local/lib/python3.10/dist-packages (from accelerate) (6.0.1)
Requirement already satisfied: torch>=1.10.0 in /usr/local/lib/python3.10/dist-packages (from accelerate) (2.1.0+cu121)
Requirement already satisfied: huggingface-hub in /usr/local/lib/python3.10/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/python3.10/dist-packages (from torch>=1.10.0->accelerate) (3.1
Requirement already satisfied: typing-extensions in /usr/local/lib/python3.10/dist-packages (from torch>=1.10.0->acceler
Requirement already satisfied: sympy in /usr/local/lib/python3.10/dist-packages (from torch>=1.10.0->accelerate) (1.12)
Requirement already satisfied: networkx in /usr/local/lib/python3.10/dist-packages (from torch>=1.10.0->accelerate) (3.2
Requirement already satisfied: jinja2 in /usr/local/lib/python3.10/dist-packages (from torch>=1.10.0->accelerate) (3.1.2
Requirement already satisfied: fsspec in /usr/local/lib/python3.10/dist-packages (from torch>=1.10.0->accelerate) (2023.
Requirement already satisfied: triton==2.1.0 in /usr/local/lib/python3.10/dist-packages (from torch>=1.10.0->accelerate)
Requirement already satisfied: requests in /usr/local/lib/python3.10/dist-packages (from huggingface-hub->accelerate) (2
Requirement already satisfied: tqdm>=4.42.1 in /usr/local/lib/python3.10/dist-packages (from huggingface-hub->accelerate
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.10/dist-packages (from jinja2->torch>=1.10.0->a
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/dist-packages (from requests->huggi
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (from requests->huggingface-hub->
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.10/dist-packages (from requests->huggingface
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-packages (from requests->huggingface
Requirement already satisfied: mpmath>=0.19 in /usr/local/lib/python3.10/dist-packages (from sympy->torch>=1.10.0->accel
Installing collected packages: accelerate
Successfully installed accelerate-0.25.0
```

```
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!

```
trainer.train()
```

[1374/1374 04:28, Epoch 1/1]

Step	Training Loss
100	4.015900
200	3.740000
300	3.703800
400	3.566000
500	3.617300
600	3.590300
700	3.528100
800	3.518200
900	3.452900
1000	3.462200
1100	3.477500
1200	3.470900
1300	3.469500

```
TrainOutput(global_step=1374, training_loss=3.579586334450623, metrics=
{'train_runtime': 269.5391, 'train_samples_per_second': 81.506,
'train_steps_per_second': 5.098, 'total_flos': 392405005440000.0,
```

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!

```
trainer.save_model('./dostoevskypt2')


dostoevskypt2 = pipeline('text-generation', model='./dostoevskypt2', device=0)
gpt2 = pipeline('text-generation', model='gpt2', device=0)


print(dostoevskypt2('Saint Petersburg is'))
print(gpt2('Saint Petersburg is'))
```

```
Setting `pad_token_id` to `eos_token_id`:50256 for open-end generation.
Setting `pad_token_id` to `eos_token_id`:50256 for open-end generation.
[{'generated_text': 'Saint Petersburg is no stranger to such men. One can hardly believe it, that people are living amon
[{'generated_text': "Saint Petersburg is home again for a very strong first-ever game and you should make no mistake abo
```

PERPLEXITY

12. (2 pts) Using the pointer here, compute the perplexity of the GPT2 pre-trained model on the Wikipedia test set (you can keep the same hyperparameters as in the link)

```
## YOUR CODE HERE - FOR COMPUTING PERPLEXITY OF GPT2 ON WIKIPEDIA TEST SET
# Load tokenizer and model
tokenizer = GPT2Tokenizer.from_pretrained('gpt2')
model = GPT2LMHeadModel.from_pretrained('gpt2').cuda()



# 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 loop
      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
ppl(model, encodings.input_ids, stride).item()
```

```
Token indices sequence length is longer than the specified maximum sequence length for this model (287644 > 1024). Runni
100%|██████████| 562/562 [00:59<00:00,  9.43it/s]
25.170494079589844
```

> YOUR PERPLEXITY ANSWER HERE: 25.17

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

```
## YOUR CODE HERE - FOR COMPUTING PERPLEXITY OF DOSTOEVSKYPT2 ON WIKIPEDIA TEST SET
# Assuming 'dostoevskypt2' is the name of your fine-tuned model
tokenizer = GPT2Tokenizer.from_pretrained('dostoevskypt2')
model = GPT2LMHeadModel.from_pretrained('dostoevskypt2').cuda()

# Load the Wikipedia test set
test = load_dataset("wikitext", "wikitext-2-raw-v1", split="test")
encodings = tokenizer("\n\n".join(test["text"]), return_tensors="pt")

# Define the perplexity calculation function
def ppl(model, input_ids_all, stride=512):
    nlls = []
    for i in tqdm(range(0, input_ids_all.size(1), stride)):
        begin_loc = max(i + stride - model.config.n_positions, 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.device)
        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 and print the perplexity
perplexity = ppl(model, encodings.input_ids)
print("Perplexity:", perplexity.item())
```

```
Token indices sequence length is longer than the specified maximum sequence length for this model (287644 > 1024). Runni
100%|██████████| 562/562 [01:03<00:00,  8.83it/s]Perplexity: 68.33443450927734
```

> YOUR PERPLEXITY ANSWER HERE 68.33

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

```
crime_and_punishment = load_dataset('crime_and_punish', split='train')
print(crime_and_punishment.column_names)
```

```
['line']
```

```
## YOUR CODE HERE - FOR COMPUTING PERPLEXITY OF GPT2 ON CRIME AND PUNISHMENT TRAIN DATASET
tokenizer = GPT2Tokenizer.from_pretrained('gpt2')
model = GPT2LMHeadModel.from_pretrained('gpt2').cuda()

# Load the "Crime and Punishment" dataset
crime_and_punishment = load_dataset('crime_and_punish', split='train')
encodings = tokenizer("\n\n".join(crime_and_punishment["line"]), return_tensors="pt")

# Define the perplexity calculation function
def ppl(model, input_ids_all, stride=512):
    nlls = []
    for i in tqdm(range(0, input_ids_all.size(1), stride)):
        begin_loc = max(i + stride - model.config.n_positions, 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.device)
        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 and print the perplexity
perplexity = ppl(model, encodings.input_ids)
print("Perplexity:", perplexity.item())
```

```
Token indices sequence length is longer than the specified maximum sequence length for this model (360591 > 1024). Runni
100%|██████████| 705/705 [01:13<00:00,  9.58it/s]Perplexity: 83.48881530761719
```

> YOUR PERPLEXITY ANSWER HERE 83.48

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

```
## YOUR CODE HERE - FOR COMPUTING PERPLEXITY OF DOSTOEVSKYPT2 ON CRIME AND PUNISHMENT TRAIN DATASET
# Load the fine-tuned model and tokenizer
tokenizer = GPT2Tokenizer.from_pretrained('dostoevskypt2')
model = GPT2LMHeadModel.from_pretrained('dostoevskypt2').cuda()

# Load the training dataset (replace 'your_dataset_name' with the actual name)
dataset = load_dataset('crime_and_punish', split='train')
encodings = tokenizer("\n\n".join(dataset['line']), return_tensors='pt')

# Define the perplexity calculation function
def calculate_perplexity(model, encodings, stride=512):
    nlls = []
    for i in tqdm(range(0, encodings.input_ids.size(1), stride)):
        begin_loc = max(i + stride - model.config.n_positions, 0)
        end_loc = min(i + stride, encodings.input_ids.size(1))
        trg_len = end_loc - i
        input_ids = encodings.input_ids[:, begin_loc:end_loc].to(model.device)
        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() / encodings.input_ids.size(1))
    return ppl

# Calculate perplexity
perplexity = calculate_perplexity(model, encodings)
print("Perplexity:", perplexity.item())
```

```
Token indices sequence length is longer than the specified maximum sequence length for this model (360591 > 1024). Runni
100%|██████████| 705/705 [01:15<00:00,  9.40it/s]Perplexity: 63.42562484741211
```

> YOUR PERPLEXITY ANSWER HERE 63.42

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

DOSTOEVSKYPT2 because it has a lower perplexity score

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

```
## YOUR CODE HERE - FOR COMPUTING PERPLEXITY OF GPT2 ON PRIDE AND PREJUDICE TEXT

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

# Read the Pride and Prejudice text file
file_path = '/content/prideAndPrejudice.txt'  # Update with your file path
with open(file_path, 'r', encoding='utf-8') as file:
    pride_and_prejudice_text = file.read()

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

# Define the perplexity calculation function
def calculate_perplexity(model, encodings, stride=512):
    nlls = []
    for i in tqdm(range(0, encodings.input_ids.size(1), stride)):
        begin_loc = max(i + stride - model.config.n_positions, 0)
        end_loc = min(i + stride, encodings.input_ids.size(1))
        trg_len = end_loc - i
        input_ids = encodings.input_ids[:, begin_loc:end_loc].to(model.device)
        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() / encodings.input_ids.size(1))
    return ppl

# Calculate perplexity
perplexity = calculate_perplexity(model, encodings)
print("Perplexity:", perplexity.item())
```

```
Token indices sequence length is longer than the specified maximum sequence length for this model (153493 > 1024). Runni
100%|██████████| 300/300 [00:29<00:00, 10.11it/s]Perplexity: 29.143468856811523
```

> YOUR PERPLEXITY ANSWER HERE 29.14

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

```
## YOUR CODE HERE - FOR COMPUTING PERPLEXITY OF dostoevskipt2 ON PRIDE AND PREJUDICE TEXT
# Load the fine-tuned dostoevskypt2 model and tokenizer
tokenizer = GPT2Tokenizer.from_pretrained('dostoevskypt2')
model = GPT2LMHeadModel.from_pretrained('dostoevskypt2').to('cuda')

# Read the Pride and Prejudice text file
file_path ='/content/prideAndPrejudice.txt'  # Update with your file path
with open(file_path, 'r', encoding='utf-8') as file:
    pride_and_prejudice_text = file.read()

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

# Define the perplexity calculation function
def calculate_perplexity(model, encodings, stride=512):
    nlls = []
    for i in tqdm(range(0, encodings.input_ids.size(1), stride)):
        begin_loc = max(i + stride - model.config.n_positions, 0)
        end_loc = min(i + stride, encodings.input_ids.size(1))
        trg_len = end_loc - i
        input_ids = encodings.input_ids[:, begin_loc:end_loc].to(model.device)
        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() / encodings.input_ids.size(1))
    return ppl

# Calculate perplexity
perplexity = calculate_perplexity(model, encodings)
print("Perplexity:", perplexity.item())
```

```
Token indices sequence length is longer than the specified maximum sequence length for this model (153493 > 1024). Runni
100%|██████████| 300/300 [00:29<00:00, 10.13it/s]
Perplexity: 42.67128372192383
```

> YOUR PERPLEXITY ANSWER HERE 42.67

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!

```
datasets_list = list_datasets()
print(', '.join(dataset for dataset in datasets_list))
```

```
<ipython-input-31-6bc899386cec>:1: FutureWarning: list_datasets is deprecated and will be removed in the next major vers
  datasets_list = list_datasets()
```

```
!pip install huggingface_hub
```

```
Requirement already satisfied: huggingface_hub in /usr/local/lib/python3.10/dist-packages (0.19.4)
Requirement already satisfied: filelock in /usr/local/lib/python3.10/dist-packages (from huggingface_hub) (3.13.1)
Requirement already satisfied: fsspec>=2023.5.0 in /usr/local/lib/python3.10/dist-packages (from huggingface_hub) (2023.
Requirement already satisfied: requests in /usr/local/lib/python3.10/dist-packages (from huggingface_hub) (2.31.0)
Requirement already satisfied: tqdm>=4.42.1 in /usr/local/lib/python3.10/dist-packages (from huggingface_hub) (4.66.1)
Requirement already satisfied: pyyaml>=5.1 in /usr/local/lib/python3.10/dist-packages (from huggingface_hub) (6.0.1)
Requirement already satisfied: typing-extensions>=3.7.4.3 in /usr/local/lib/python3.10/dist-packages (from huggingface_h
Requirement already satisfied: packaging>=20.9 in /usr/local/lib/python3.10/dist-packages (from huggingface_hub) (23.2)
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/dist-packages (from requests->huggi
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (from requests->huggingface_hub)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.10/dist-packages (from requests->huggingface
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-packages (from requests->huggingface
```

```
from huggingface_hub import list_datasets
datasets_list = list_datasets()
print(', '.join(dataset.id for dataset in datasets_list))
```

```
from datasets import load_dataset

dataset = load_dataset('arxiv_dataset')

# For demonstration, let's take a look at the first entry in the dataset
print(dataset['train'][0])
```

```
/usr/local/lib/python3.10/dist-packages/datasets/load.py:1429: FutureWarning: The repository for arxiv_dataset contains
You can avoid this message in future by passing the argument `trust_remote_code=True`.
Passing `trust_remote_code=True` will be mandatory to load this dataset from the next major release of `datasets`.
  warnings.warn(
```

```
Downloading builder script: 100%  4.67k/4.67k [00:00<00:00, 143kB/s]
Downloading metadata: 100%  1.83k/1.83k [00:00<00:00, 68.4kB/s]
Downloading readme: 100%  7.25k/7.25k [00:00<00:00, 123kB/s]
```

```
---------------------------------------------------------------------------
ManualDownloadError                       Traceback (most recent call last)
<ipython-input-36-f9872d5921f7> in <cell line: 3>()
      1 from datasets import load_dataset
      2
----> 3 dataset = load_dataset('arxiv_dataset')
      4
      5 # For demonstration, let's take a look at the first entry in the dataset

                                  2 frames
/usr/local/lib/python3.10/dist-packages/datasets/builder.py in _check_manual_download(self, dl_manager)
   1024     def _check_manual_download(self, dl_manager):
   1025         if self.manual_download_instructions is not None and dl_manager.manual_dir is None:
-> 1026             raise ManualDownloadError(
   1027                 textwrap.dedent(
   1028                     f"""\

ManualDownloadError:                The dataset arxiv_dataset with config default requires manual data.
               Please follow the manual download instructions:
                    You need to go to https://www.kaggle.com/Cornell-University/arxiv,
and manually download the dataset. Once it is completed,
a zip folder named archive.zip will be appeared in your Downloads folder
or whichever folder your browser chooses to save files to. Extract that folder
and you would get a arxiv-metadata-oai-snapshot.json file
You can then move that file under <path/to/folder>.
The <path/to/folder> can e.g. be "~/manual_data".
arxiv_dataset can then be loaded using the following command `datasets.load_dataset("arxiv_dataset", data_dir="
<path/to/folder>")`.

               Manual data can be loaded with:
                datasets.load_dataset("arxiv_dataset", data_dir="<path/to/manual/data>")
```

SEARCH STACK OVERFLOW

Start coding or generate with AI.

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. 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

Trainer Documentation

Example: Fine-Tuning BERT for Esperanto

Example: Fine-Tuning for IMDb Classification

18. Dataset #1

```
from transformers import GPT2LMHeadModel, GPT2Tokenizer, Trainer, TrainingArguments
from datasets import load_dataset
import torch

# Check if GPU is available and set the device
device = 'cuda' if torch.cuda.is_available() else 'cpu'

# Load the tokenizer and model
tokenizer = GPT2Tokenizer.from_pretrained('gpt2')
tokenizer.pad_token = tokenizer.eos_token  # Set the EOS token as the padding token
model = GPT2LMHeadModel.from_pretrained('gpt2').to(device)

# Load the Shakespeare dataset
dataset = load_dataset('tiny_shakespeare')

# Function to tokenize the dataset and prepare labels
def tokenize_and_format(batch):
    tokenized_inputs = tokenizer(batch['text'], padding='max_length', truncation=True, max_length=64)
    tokenized_inputs['labels'] = tokenized_inputs['input_ids'].copy()
    return tokenized_inputs

# Tokenize and format the dataset
tokenized_dataset = dataset.map(tokenize_and_format, batched=True, remove_columns=['text'])

# Define the training arguments
training_args = TrainingArguments(
    output_dir='./gpt2-finetuned-shakespeare',
    num_train_epochs=150,  # Can be adjusted
    per_device_train_batch_size=16,  # Adjust based on your GPU memory
    per_device_eval_batch_size=64,
    warmup_steps=500,
    weight_decay=0.01,
    logging_dir='./logs',
    logging_steps=100,
)

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

# Train the model
trainer.train()

# Save the fine-tuned model
model.save_pretrained('./gpt2_finetuned-shakespeare')
```


[150/150 00:13, Epoch 150/150]

Step	Training Loss
100	2.710200

(4 pts) YOUR ANSWER HERE - BRIEF DESCRIPTION OF THE DATASET YOU CHOSE The tiny_shakespeare dataset is a compact collection of works by William Shakespeare. It's a curated subset derived from the complete works of Shakespeare, designed specifically for training and experimenting with machine learning models in natural language processing tasks, especially text generation. Here's a brief overview of this dataset: The dataset includes excerpts from Shakespeare's plays and poetry. It typically features a mixture of dialogues, monologues, and descriptions found in his works. As the name suggests, it's significantly smaller than the complete collection of Shakespeare's works. This reduced size makes it more manageable for training models, especially in environments with limited computational resources. It offers a concentrated dose of Shakespeare's unique writing style, making it ideal for training models to generate text that mimics his iconic use of language, including vocabulary, meter, and rhetorical devices.

```
## YOUR CODE HERE - FOR GENERATION WITH YOUR FINE-TUNED MODEL AND COMPARISON WITH REGULAR GPT2
from transformers import GPT2LMHeadModel, GPT2Tokenizer

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

# Load the regular GPT-2 model
model_gpt2 = GPT2LMHeadModel.from_pretrained('gpt2')

# Load your fine-tuned model
model_fine_tuned = GPT2LMHeadModel.from_pretrained('./gpt2_finetuned-shakespeare')

# Function to generate text
def generate_text(model, prompt, max_length=50):
    input_ids = tokenizer.encode(prompt, return_tensors='pt')
    output = model.generate(input_ids, max_length=max_length, temperature=0.9,
        top_k=40, no_repeat_ngram_size = 3, do_sample=True,num_return_sequences=1)
    return tokenizer.decode(output[0], skip_special_tokens=True)

# Set a prompt
prompt = "Hi, my name is Gaurav"

# Generate text with both models
output_regular = generate_text(model_gpt2, prompt)
output_fine_tuned = generate_text(model_fine_tuned, prompt)

# Print the outputs for comparison
print("Regular GPT-2 Output:\n", output_regular, "\n")
print("Fine-Tuned Model Output:\n", output_fine_tuned)
```

```
The attention mask and the pad token id were not set. As a consequence, you may observe unexpected behavior. Please pass
Setting `pad_token_id` to `eos_token_id`:50256 for open-end generation.
The attention mask and the pad token id were not set. As a consequence, you may observe unexpected behavior. Please pass
Setting `pad_token_id` to `eos_token_id`:50256 for open-end generation.
Regular GPT-2 Output:
```

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

GPT2 output : The regular GPT-2 model, which is trained on a diverse range of internet text, generates a contemporary and realistic sentence. It introduces a fictional character named Gaurav Shah and associates him with involvement in the Indian government. This output is straightforward, aligns with modern language use, and is typical of the diverse, general-purpose nature of GPT-2's training data. The output is coherent and could be part of a realistic introduction or bio.

The output from the fine-tuned model reflects the Shakespearean style, evident from the old English and dramatic dialogue format ("You are all right. Before we proceed any further, hear me speak."). This style is characteristic of Shakespeare's plays, which often involve direct address and dramatic monologues. The use of phrases like "hear me speak" and the repetition of "speak" are reminiscent of the poetic and rhetorical devices used by Shakespeare. The language is more formal and archaic compared to the modern English used by the regular GPT-2 model.

This comparison demonstrates the effectiveness of fine-tuning in shifting the style and language of a model's output. The fine-tuned model has successfully adapted to the Shakespearean style, showcasing the impact of domain-specific training on language model performance.

Double-click (or enter) to edit

.

.

Double-click (or enter) to edit

.

.

.

.

.

.

.