CS685HW2PrachiJain

May 14, 2023

0.1 Homework 2, CS685 Spring 2023

0.1.1 This is due on May 13, 2023, submitted via Gradescope as a PDF (File>Print>Save as PDF). 100 points total.

IMPORTANT: After copying this notebook to your Google Drive, please paste a link to it below. To get a publicly-accessible link, hit the *Share* button at the top right, then click "Get shareable link" and copy over the result. If you fail to do this, you will receive no credit for this homework! *LINK:* https://colab.research.google.com/drive/12IwaBtqc65JzYFpVr-75V3WI93QIxcdZ?usp=sharing

---

How to do this problem set:

- Some questions require writing Python code and computing results, and the rest of them have written answers. For coding problems, you will have to fill out all code blocks that say `YOUR CODE HERE`.
- For text-based answers, you should replace the text that says "Write your answer here…" with your actual answer.
- This assignment is designed such that each cell takes a few minutes (if that) to run. If it is taking longer than that, you might have made a mistake in your code.

How to submit this problem set:

- Write all the answers in this Colab notebook. Once you are finished, generate a PDF via (File -> Print -> Save as PDF) and upload it to Gradescope.
- **Important:** check your PDF before you submit to Gradescope to make sure it exported correctly. If Colab gets confused about your syntax, it will sometimes terminate the PDF creation routine early.
- **Important:** on Gradescope, please make sure that you tag each page with the corresponding question(s). This makes it significantly easier for our graders to grade submissions, especially with the long outputs of many of these cells. We will take off points for submissions that are not tagged.

- When creating your final version of the PDF to hand in, please do a fresh restart and execute every cell in order. One handy way to do this is by clicking `Runtime -> Run All` in the notebook menu.

Academic honesty

- We will audit the Colab notebooks from a set number of students, chosen at random. The audits will check that the code you wrote actually generates the answers in your PDF. If you turn in correct answers on your PDF without code that actually generates those answers, we will consider this a serious case of cheating. See the course page for honesty policies.
- We will also run automatic checks of Colab notebooks for plagiarism. Copying code from others is also considered a serious case of cheating.
- There is no penalty for using AI assistance on this homework as long as you fully disclose it in the final cell of this notebook (this includes storing any prompts that you feed to large language models). That said, anyone caught using AI assistance without proper disclosure will receive a zero on the assignment (we have several automatic tools to detect such cases). We're literally allowing you to use it with no limitations, so there is no reason to lie!

---

1 Part 0: Setup

1.1 Adding a hardware accelerator

The purpose of this homework is to get you acquainted with using large-scale pretrained language models and decoding algorithms. Since we will be training large neural networks we will attach a GPU, otherwise training will take a very long time.

Please go to the menu and add a GPU as follows:

`Edit > Notebook Settings > Hardware accelerator > (GPU)`

Run the following cell to confirm that the GPU is detected.

[1]:
```python
import torch

# Confirm that the GPU is detected

assert torch.cuda.is_available()

# Get the GPU device name.
device_name = torch.cuda.get_device_name()
n_gpu = torch.cuda.device_count()
print(f"Found device: {device_name}, n_gpu: {n_gpu}")
```

```
Found device: Tesla T4, n_gpu: 1
```

1.2 Installing Hugging Face's Transformers library

We will use Hugging Face's Transformers (https://github.com/huggingface/transformers), an open-source library that provides general-purpose architectures for natural language understanding and generation with a collection of various pretrained models made by the NLP community. This library will allow us to easily use pretrained models like `BERT` and perform experiments on top of them. We can use these models to solve downstream target tasks, such as text classification, question answering, and sequence labeling.

Run the following cell to install Hugging Face's Transformers library, download data and supporting code for the homework, and install some additional packages. Note that you will be asked to link with your Google Drive account to download some of these files. If you're concerned about security risks (there have not been any issues in previous semesters), feel free to make a new Google account and use it for this homework!

[2]:
```
!pip install transformers
!pip install -U -q PyDrive

from pydrive.auth import GoogleAuth
from pydrive.drive import GoogleDrive
from google.colab import auth
from oauth2client.client import GoogleCredentials
# Authenticate and create the PyDrive client.
auth.authenticate_user()
gauth = GoogleAuth()
gauth.credentials = GoogleCredentials.get_application_default()
drive = GoogleDrive(gauth)
print('success!')

import os
import zipfile

data_file = drive.CreateFile({'id': '1zeo8FcaNUnhN66OmGMNEAPvxOE4DPOnE'})
data_file.GetContentFile('hw1.zip')

# Extract data from the zipfile and put it into the current directory
with zipfile.ZipFile('hw1.zip', 'r') as zip_file:
    zip_file.extractall('./')
os.remove('hw1.zip')
# We will use hw1 as our working directory
os.chdir('hw1')
print("Data and supporting code downloaded!")

pretrained_models_dir = './pretrained_models_dir'
if not os.path.isdir(pretrained_models_dir):
  os.mkdir(pretrained_models_dir)   # directory to save pretrained models
print('model directory created')

!pip install -r requirements.txt
```

```python
print('everything set up!')
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-
wheels/public/simple/
Collecting transformers
  Downloading transformers-4.29.1-py3-none-any.whl (7.1 MB)
                           7.1/7.1 MB
88.8 MB/s eta 0:00:00
Requirement already satisfied: filelock in /usr/local/lib/python3.10/dist-
packages (from transformers) (3.12.0)
Collecting huggingface-hub<1.0,>=0.14.1 (from transformers)
  Downloading huggingface_hub-0.14.1-py3-none-any.whl (224 kB)
                          224.5/224.5 kB
25.4 MB/s eta 0:00:00
Requirement already satisfied: numpy>=1.17 in
/usr/local/lib/python3.10/dist-packages (from transformers) (1.22.4)
Requirement already satisfied: packaging>=20.0 in
/usr/local/lib/python3.10/dist-packages (from transformers) (23.1)
Requirement already satisfied: pyyaml>=5.1 in /usr/local/lib/python3.10/dist-
packages (from transformers) (6.0)
Requirement already satisfied: regex!=2019.12.17 in
/usr/local/lib/python3.10/dist-packages (from transformers) (2022.10.31)
Requirement already satisfied: requests in /usr/local/lib/python3.10/dist-
packages (from transformers) (2.27.1)
Collecting tokenizers!=0.11.3,<0.14,>=0.11.1 (from transformers)
  Downloading
tokenizers-0.13.3-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.whl
(7.8 MB)
                           7.8/7.8 MB
103.5 MB/s eta 0:00:00
Requirement already satisfied: tqdm>=4.27 in
/usr/local/lib/python3.10/dist-packages (from transformers) (4.65.0)
Requirement already satisfied: fsspec in /usr/local/lib/python3.10/dist-packages
(from huggingface-hub<1.0,>=0.14.1->transformers) (2023.4.0)
Requirement already satisfied: typing-extensions>=3.7.4.3 in
/usr/local/lib/python3.10/dist-packages (from huggingface-
hub<1.0,>=0.14.1->transformers) (4.5.0)
Requirement already satisfied: urllib3<1.27,>=1.21.1 in
/usr/local/lib/python3.10/dist-packages (from requests->transformers) (1.26.15)
Requirement already satisfied: certifi>=2017.4.17 in
/usr/local/lib/python3.10/dist-packages (from requests->transformers)
(2022.12.7)
Requirement already satisfied: charset-normalizer~=2.0.0 in
/usr/local/lib/python3.10/dist-packages (from requests->transformers) (2.0.12)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-
packages (from requests->transformers) (3.4)
Installing collected packages: tokenizers, huggingface-hub, transformers
```

```
Successfully installed huggingface-hub-0.14.1 tokenizers-0.13.3
transformers-4.29.1
success!
Data and supporting code downloaded!
model directory created
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-
wheels/public/simple/
Collecting seqeval (from -r requirements.txt (line 1))
  Downloading seqeval-1.2.2.tar.gz (43 kB)
                           43.6/43.6 kB
4.0 MB/s eta 0:00:00
  Preparing metadata (setup.py) … done
Collecting conllu (from -r requirements.txt (line 2))
  Downloading conllu-4.5.2-py2.py3-none-any.whl (16 kB)
Collecting googletrans (from -r requirements.txt (line 3))
  Downloading googletrans-3.0.0.tar.gz (17 kB)
  Preparing metadata (setup.py) … done
Collecting langdetect (from -r requirements.txt (line 4))
  Downloading langdetect-1.0.9.tar.gz (981 kB)
                          981.5/981.5 kB
49.4 MB/s eta 0:00:00
  Preparing metadata (setup.py) … done
Requirement already satisfied: numpy>=1.14.0 in /usr/local/lib/python3.10/dist-
packages (from seqeval->-r requirements.txt (line 1)) (1.22.4)
Requirement already satisfied: scikit-learn>=0.21.3 in
/usr/local/lib/python3.10/dist-packages (from seqeval->-r requirements.txt (line
1)) (1.2.2)
Collecting httpx==0.13.3 (from googletrans->-r requirements.txt (line 3))
  Downloading httpx-0.13.3-py3-none-any.whl (55 kB)
                           55.1/55.1 kB
6.4 MB/s eta 0:00:00
Requirement already satisfied: certifi in /usr/local/lib/python3.10/dist-
packages (from httpx==0.13.3->googletrans->-r requirements.txt (line 3))
(2022.12.7)
Collecting hstspreload (from httpx==0.13.3->googletrans->-r requirements.txt
(line 3))
  Downloading hstspreload-2023.1.1-py3-none-any.whl (1.5 MB)
                           1.5/1.5 MB
71.4 MB/s eta 0:00:00
Requirement already satisfied: sniffio in /usr/local/lib/python3.10/dist-
packages (from httpx==0.13.3->googletrans->-r requirements.txt (line 3)) (1.3.0)
Collecting chardet==3.* (from httpx==0.13.3->googletrans->-r requirements.txt
(line 3))
  Downloading chardet-3.0.4-py2.py3-none-any.whl (133 kB)
                          133.4/133.4 kB
15.7 MB/s eta 0:00:00
Collecting idna==2.* (from httpx==0.13.3->googletrans->-r requirements.txt
(line 3))
```

```
  Downloading idna-2.10-py2.py3-none-any.whl (58 kB)
                           58.8/58.8 kB
6.4 MB/s eta 0:00:00
Collecting rfc3986<2,>=1.3 (from httpx==0.13.3->googletrans->-r
requirements.txt (line 3))
  Downloading rfc3986-1.5.0-py2.py3-none-any.whl (31 kB)
Collecting httpcore==0.9.* (from httpx==0.13.3->googletrans->-r requirements.txt
(line 3))
  Downloading httpcore-0.9.1-py3-none-any.whl (42 kB)
                           42.6/42.6 kB
4.4 MB/s eta 0:00:00
Collecting h11<0.10,>=0.8 (from
httpcore==0.9.*->httpx==0.13.3->googletrans->-r requirements.txt (line 3))
  Downloading h11-0.9.0-py2.py3-none-any.whl (53 kB)
                           53.6/53.6 kB
6.7 MB/s eta 0:00:00
Collecting h2==3.* (from httpcore==0.9.*->httpx==0.13.3->googletrans->-r
requirements.txt (line 3))
  Downloading h2-3.2.0-py2.py3-none-any.whl (65 kB)
                           65.0/65.0 kB
8.5 MB/s eta 0:00:00
Collecting hyperframe<6,>=5.2.0 (from
h2==3.*->httpcore==0.9.*->httpx==0.13.3->googletrans->-r requirements.txt (line
3))
  Downloading hyperframe-5.2.0-py2.py3-none-any.whl (12 kB)
Collecting hpack<4,>=3.0 (from
h2==3.*->httpcore==0.9.*->httpx==0.13.3->googletrans->-r requirements.txt (line
3))
  Downloading hpack-3.0.0-py2.py3-none-any.whl (38 kB)
Requirement already satisfied: six in /usr/local/lib/python3.10/dist-packages
(from langdetect->-r requirements.txt (line 4)) (1.16.0)
Requirement already satisfied: scipy>=1.3.2 in /usr/local/lib/python3.10/dist-
packages (from scikit-learn>=0.21.3->seqeval->-r requirements.txt (line 1))
(1.10.1)
Requirement already satisfied: joblib>=1.1.1 in /usr/local/lib/python3.10/dist-
packages (from scikit-learn>=0.21.3->seqeval->-r requirements.txt (line 1))
(1.2.0)
Requirement already satisfied: threadpoolctl>=2.0.0 in
/usr/local/lib/python3.10/dist-packages (from scikit-learn>=0.21.3->seqeval->-r
requirements.txt (line 1)) (3.1.0)
Building wheels for collected packages: seqeval, googletrans, langdetect
  Building wheel for seqeval (setup.py) … done
  Created wheel for seqeval: filename=seqeval-1.2.2-py3-none-any.whl size=16165
sha256=3b8bc97447e95bc0a6baca6b0c411898a8c0aca37547a9606030ce416923db77
  Stored in directory: /root/.cache/pip/wheels/1a/67/4a/ad4082dd7dfc30f2abfe4d80
a2ed5926a506eb8a972b4767fa
  Building wheel for googletrans (setup.py) … done
  Created wheel for googletrans: filename=googletrans-3.0.0-py3-none-any.whl
```

```
size=15721
sha256=dcd6caa59f75243731ccbc4a562f702e02a9ebc43a5f750967606819d28d3aab
  Stored in directory: /root/.cache/pip/wheels/b3/81/ea/8b030407f8ebfc2f857814e0
86bb22ca2d4fea1a7be63652ab
  Building wheel for langdetect (setup.py) … done
  Created wheel for langdetect: filename=langdetect-1.0.9-py3-none-any.whl
size=993224
sha256=6dddb9348b42a3404681c400b71b2b4ba556fb4e1b562fb53c0693c06e008827
  Stored in directory: /root/.cache/pip/wheels/95/03/7d/59ea870c70ce4e5a370638b5
462a7711ab78fba2f655d05106
Successfully built seqeval googletrans langdetect
Installing collected packages: rfc3986, hyperframe, hpack, h11, chardet,
langdetect, idna, hstspreload, h2, conllu, httpcore, seqeval, httpx, googletrans
  Attempting uninstall: chardet
    Found existing installation: chardet 4.0.0
    Uninstalling chardet-4.0.0:
      Successfully uninstalled chardet-4.0.0
  Attempting uninstall: idna
    Found existing installation: idna 3.4
    Uninstalling idna-3.4:
      Successfully uninstalled idna-3.4
Successfully installed chardet-3.0.4 conllu-4.5.2 googletrans-3.0.0 h11-0.9.0
h2-3.2.0 hpack-3.0.0 hstspreload-2023.1.1 httpcore-0.9.1 httpx-0.13.3
hyperframe-5.2.0 idna-2.10 langdetect-1.0.9 rfc3986-1.5.0 seqeval-1.2.2
```

```
everything set up!
```

2 Part 1. Masked Language Modeling (60 points)

In this part, we will see what happens when using large-scale pretrained encoder models (e.g., `BERT`) for text generation. You'll be implementing different decoding algorithms for the masked language modeling setup. Let's begin!

We'll use `BERT` (Devlin et al., 2019) for the task of masked word completion: given an input sentence with some words masked out, predict the masked word(s) based on its context. Run the following cell to download the pretrained `BERT` base model (cased) and tell PyTorch to use the GPU to run it.

[3]:
```python
from transformers import AutoTokenizer, AutoModelForMaskedLM

model_name_or_path = "bert-base-cased"
cache_dir = os.path.join(pretrained_models_dir, model_name_or_path)
tokenizer = AutoTokenizer.from_pretrained(model_name_or_path,␣
 ↪cache_dir=cache_dir)
model = AutoModelForMaskedLM.from_pretrained(model_name_or_path,␣
 ↪cache_dir=cache_dir)
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model = model.to(device)
```

```
print('success!')
```

```
Downloading (…)okenizer_config.json:   0%|          | 0.00/29.0 [00:00<?, ?B/s]

Downloading (…)lve/main/config.json:   0%|          | 0.00/570 [00:00<?, ?B/s]

Downloading (…)solve/main/vocab.txt:   0%|          | 0.00/213k [00:00<?, ?B/s]

Downloading (…)/main/tokenizer.json:   0%|          | 0.00/436k [00:00<?, ?B/s]

Downloading pytorch_model.bin:   0%|          | 0.00/436M [00:00<?, ?B/s]

Some weights of the model checkpoint at bert-base-cased were not used when
initializing BertForMaskedLM: ['cls.seq_relationship.bias',
'cls.seq_relationship.weight']
- This IS expected if you are initializing BertForMaskedLM from the checkpoint
of a model trained on another task or with another architecture (e.g.
initializing a BertForSequenceClassification model from a BertForPreTraining
model).
- This IS NOT expected if you are initializing BertForMaskedLM from the
checkpoint of a model that you expect to be exactly identical (initializing a
BertForSequenceClassification model from a BertForSequenceClassification model).

success!
```

2.0.1 Question 1.1 (10 points)

The below cell passes a single sentence with a [MASK] token into BERT, and returns the logits (i.e., the unnormalized probabilities) of the token prediction at each position of this sequence. Write some code in this cell that prints out the five most probable words for the masked position from the `token_logits` variable. If you did it right, you'll notice that these words mostly will make sense in the given context.

Hints

- Use `torch.where` to find the index of a masked token within the input tensor (note that `tokenizer.mask_token_id` gives us the index of the mask token in the vocabulary).
- Use `torch.topk` to get the `k` largest elements of a given tensor along a given dimension.
- Use `tokenizer.decode([token_id])` to convert a single integer `token_id` to a token string.

[4]:
```
sentence = f"""Pita Pockets is the best {tokenizer.mask_token} in
the town of Amherst, and everyone should try their falafel!""".replace('\n',' ')

# Encode the input sentence and get the model's output
input = tokenizer.encode(sentence, return_tensors="pt").to(device)
# The model outputs the masked language modeling logits of shape
# [batch_size, sequence_length, vocab_size]
token_logits = model(input)[0]

# YOUR CODE HERE!

# 1. Get index of Masked token
```

```python
maskTokenID = torch.where(input[0] == tokenizer.mask_token_id)[0].item()

# 2. Get Top 5 most probable elements from model's output given a masked token index
values, id = token_logits[0, maskTokenID].topk(5, dim=0)

# 3. Decode most probable values
decodeMask = tokenizer.decode(id).split()

print("    Input Sentence :", sentence)
print("="*150)
for id, word  in enumerate(decodeMask):
  print("Predicted Word ", (id + 1)," :", word)
  newSent = sentence.replace(tokenizer.mask_token, word)
  print("Contexual Sentence :", newSent)
  print("="*150)
```

```
    Input Sentence : Pita Pockets is the best [MASK] in the town of Amherst, and
everyone should try their falafel!
================================================================================
======================================================================
Predicted Word  1  : restaurant
Contexual Sentence : Pita Pockets is the best restaurant in the town of Amherst,
and everyone should try their falafel!
================================================================================
======================================================================
Predicted Word  2  : place
Contexual Sentence : Pita Pockets is the best place in the town of Amherst, and
everyone should try their falafel!
================================================================================
======================================================================
Predicted Word  3  : pub
Contexual Sentence : Pita Pockets is the best pub in the town of Amherst, and
everyone should try their falafel!
================================================================================
======================================================================
Predicted Word  4  : food
Contexual Sentence : Pita Pockets is the best food in the town of Amherst, and
everyone should try their falafel!
================================================================================
======================================================================
Predicted Word  5  : bar
Contexual Sentence : Pita Pockets is the best bar in the town of Amherst, and
everyone should try their falafel!
================================================================================
======================================================================
```

2.0.2 Question 1.2 (5 points)

The below cell contains the same context as before but with an increasing number of contiguous [MASK] tokens (run the cell to print out each context). For each input, replace each [MASK] token with the most probable token (i.e., the argmax of the probability distribution) as predicted by BERT, and then print out the resulting unmasked string. To be clear, your output should be five strings without any [MASK] tokens.

[5]:
```python
sentence = f"""Pita Pockets is the best {tokenizer.mask_token} in
the town of Amherst, and everyone should try their falafel!""".replace('\n',' ')
sentenceList = []
logit_list = []
for idx in range(1,6):
    x = sentence.split()
    for mask_idx in range(idx):
        x[5+mask_idx] = tokenizer.mask_token
    x = ' '.join(x)
    print('input %d: %s' %(idx, x))
    input = tokenizer.encode(x, return_tensors="pt").to(device)
    token_logits = model(input)[0]
    logit_list.append((input, token_logits))
    sentenceList.append(x)

# YOUR CODE HERE!
def Unmasking(input, token_logits):
    #1. Get masked token index
    maskTokenId = torch.where(input[0]==tokenizer.mask_token_id)[0][0:]

    #2. Get most probable element from model output for masked index and decode it
    out = []
    for maskID in maskTokenId:
        maskWord = token_logits[0, maskID, :]
        decodeMask = tokenizer.decode([torch.argmax(maskWord)])
        out.append(decodeMask)
    return maskTokenId, out

newSentenceList = []
i = 0
print("="*150)
for input, tokenLogits in logit_list:
  maskTokenId, out = Unmasking(input, tokenLogits)
  outWords = " ".join(out)
  newSentence = str(sentenceList[i]).split()
  i += 1
  print(f"    Input Sentence {i} :", " ".join(newSentence))
  for j in range(len(out)):
    newSentence[5 + j] = out[j]
```

```python
    newSent = " ".join(newSentence)
    print("    Unmask String(s) :", outWords)
    print(f"Contexual Sentence {i} :", newSent)
    print("="*150)
```

```
input 1: Pita Pockets is the best [MASK] in the town of Amherst, and everyone
should try their falafel!
input 2: Pita Pockets is the best [MASK] [MASK] the town of Amherst, and
everyone should try their falafel!
input 3: Pita Pockets is the best [MASK] [MASK] [MASK] town of Amherst, and
everyone should try their falafel!
input 4: Pita Pockets is the best [MASK] [MASK] [MASK] [MASK] of Amherst, and
everyone should try their falafel!
input 5: Pita Pockets is the best [MASK] [MASK] [MASK] [MASK] [MASK] Amherst,
and everyone should try their falafel!
================================================================================
======================================================================
    Input Sentence 1 : Pita Pockets is the best [MASK] in the town of Amherst,
and everyone should try their falafel!
    Unmask String(s) : restaurant
Contexual Sentence 1 : Pita Pockets is the best restaurant in the town of
Amherst, and everyone should try their falafel!
================================================================================
======================================================================
    Input Sentence 2 : Pita Pockets is the best [MASK] [MASK] the town of
Amherst, and everyone should try their falafel!
    Unmask String(s) : restaurant in
Contexual Sentence 2 : Pita Pockets is the best restaurant in the town of
Amherst, and everyone should try their falafel!
================================================================================
======================================================================
    Input Sentence 3 : Pita Pockets is the best [MASK] [MASK] [MASK] town of
Amherst, and everyone should try their falafel!
    Unmask String(s) : in in the
Contexual Sentence 3 : Pita Pockets is the best in in the town of Amherst, and
everyone should try their falafel!
================================================================================
======================================================================
    Input Sentence 4 : Pita Pockets is the best [MASK] [MASK] [MASK] [MASK] of
Amherst, and everyone should try their falafel!
    Unmask String(s) : - in the out
Contexual Sentence 4 : Pita Pockets is the best - in the out of Amherst, and
everyone should try their falafel!
================================================================================
======================================================================
    Input Sentence 5 : Pita Pockets is the best [MASK] [MASK] [MASK] [MASK]
[MASK] Amherst, and everyone should try their falafel!
```

```
    Unmask String(s) : food ever in in to
Contexual Sentence 5 : Pita Pockets is the best food ever in in to Amherst, and
everyone should try their falafel!
================================================================================
======================================================================
```

2.0.3 Question 1.3 (10 points)

What do you notice about your outputs as the size of the masked span increases? Explain why this is happening.

[6]:
```python
from transformers import pipeline
FillMask = pipeline('fill-mask', model='bert-base-cased')
FillMask(sentence)
```

```
Downloading (…)lve/main/config.json:   0%|          | 0.00/570 [00:00<?, ?B/s]

Downloading pytorch_model.bin:   0%|          | 0.00/436M [00:00<?, ?B/s]

Some weights of the model checkpoint at bert-base-cased were not used when
initializing BertForMaskedLM: ['cls.seq_relationship.bias',
'cls.seq_relationship.weight']
- This IS expected if you are initializing BertForMaskedLM from the checkpoint
of a model trained on another task or with another architecture (e.g.
initializing a BertForSequenceClassification model from a BertForPreTraining
model).
- This IS NOT expected if you are initializing BertForMaskedLM from the
checkpoint of a model that you expect to be exactly identical (initializing a
BertForSequenceClassification model from a BertForSequenceClassification model).

Downloading (…)okenizer_config.json:   0%|          | 0.00/29.0 [00:00<?, ?B/s]

Downloading (…)solve/main/vocab.txt:   0%|          | 0.00/213k [00:00<?, ?B/s]

Downloading (…)/main/tokenizer.json:   0%|          | 0.00/436k [00:00<?, ?B/s]
```

[6]:
```
[{'score': 0.3161855638027191,
  'token': 4382,
  'token_str': 'restaurant',
  'sequence': 'Pita Pockets is the best restaurant in the town of Amherst, and
everyone should try their falafel!'},
 {'score': 0.08139079064130783,
  'token': 1282,
  'token_str': 'place',
  'sequence': 'Pita Pockets is the best place in the town of Amherst, and
everyone should try their falafel!'},
 {'score': 0.045339133590459824,
  'token': 11030,
  'token_str': 'pub',
  'sequence': 'Pita Pockets is the best pub in the town of Amherst, and everyone
should try their falafel!'},
```

```
 {'score': 0.042720239609479904,
  'token': 2094,
  'token_str': 'food',
  'sequence': 'Pita Pockets is the best food in the town of Amherst, and
everyone should try their falafel!'},
 {'score': 0.03440569341182709,
  'token': 2927,
  'token_str': 'bar',
  'sequence': 'Pita Pockets is the best bar in the town of Amherst, and everyone
should try their falafel!'}]
```

As the size of the masked span increases, the predictions for the masked tokens start to make less sense in the context of the sentence. The sentences become less grammatically correct and less semantically meaningful. In the beginning, when there was only one [MASK], it was able to predict "restaurant" for the sentence, and for two [MASK] tokens, it was able to predict "restaurant in" which made sense in the context, but afterward, the meaning became less clear and in the end it started repeating the predicted words as well getting nonsensical predictions like "in in the" and "-in the out".

This is happening due to **contextual understanding**. Language models, like the BERT base cased model trained using masked language modeling (MLM), understand and generate text based on the surrounding context. In our case, BERT was trained to predict only one mask token, and when we give multiple mask tokens, it has very little context, which it can use to predict the mask token. When a large span of text is masked out, the model has less context to work with, making it harder to predict what the masked words should be accurate.

In summary, as the span of masked tokens increases, the amount of contextual information available to BERT decreases, affecting the quality of the predictions made by the model.

Also, another observation is that our current implementation predicts each masked word individually, not considering the co-dependency between them. Certain models, like XLNet and RoBERTa, implement a different pre-training strategy that is more robust to masking large text spans. They could potentially provide better results for this kind of task.

2.0.4 Question 1.4 (15 points)

Let's focus on the sentence above with three mask tokens inserted in the middle. Instead of decoding all of them at once, in this problem, we will decode the mask tokens from left to right, one token at a time. For this problem, we'll rely on *greedy decoding*, so at each timestep, we will choose the highest probability token to replace the corresponding mask. Your code should do the following: encode the sentence, and then decode the left-most remaining mask token. Then encode the new sentence (with two mask tokens this time, not three) and decode the left-most remaining mask token. Repeat until all mask tokens have been decoded. Your code should **print the resulting sentence** with all mask tokens decoded. Note that there is only one correct output!

[13]:
```
sentence = f"""Pita Pockets is the best {tokenizer.mask_token} in
the town of Amherst, and everyone should try their falafel!""".replace('\n',' ')
x = sentence.split()
mask_idxs = [5+i for i in range(3)]
```

```python
for idx in mask_idxs:
    x[idx] = tokenizer.mask_token
sentence = ' '.join(x)
print(f"   Input Sentence: {sentence}")
print("="*150)
# YOUR CODE HERE!
for i in range(3):  # 1. => Number of Masks: 3
    input = tokenizer.encode(sentence, return_tensors='pt').to(device)
    maskTokenIndex = torch.where(input == tokenizer.mask_token_id)[1]

    # 2. Get the model predictions
    with torch.no_grad():
        predictions = model(input)[0]

    # 3. Get the predicted token ID
    predictedTokenID = torch.argmax(predictions[0, maskTokenIndex[0], :]).item()
    predictedToken = tokenizer.decode([predictedTokenID])

    # 4. Replace the first mask token in the sentence
    sentence = sentence.replace(tokenizer.mask_token, predictedToken, 1)
    print(f"Greedy Decoding {i+1}:", sentence)
print("="*150)
print(f"  Result Sentence: {sentence}")
```

```
   Input Sentence: Pita Pockets is the best [MASK] [MASK] [MASK] town of
Amherst, and everyone should try their falafel!
================================================================================
======================================================================
Greedy Decoding 1: Pita Pockets is the best in [MASK] [MASK] town of Amherst,
and everyone should try their falafel!
Greedy Decoding 2: Pita Pockets is the best in the [MASK] town of Amherst, and
everyone should try their falafel!
Greedy Decoding 3: Pita Pockets is the best in the small town of Amherst, and
everyone should try their falafel!
================================================================================
======================================================================
  Result Sentence: Pita Pockets is the best in the small town of Amherst, and
everyone should try their falafel!
```

2.0.5 Question 1.5 (10 points)

In the below cell, modify your code from Q1.4 to implement ancestral sampling, in which instead of choosing the highest probability token at every position, you sample a token from the *entire* distribution. Your code should **print out 5 different samples** obtained via ancestral sampling.

[18]:
```python
import random

sentence = f"""Pita Pockets is the best {tokenizer.mask_token} in
```

```python
the town of Amherst, and everyone should try their falafel!""".replace('\n',' ')
x = sentence.split()
mask_idxs = [5+i for i in range(3)]
for idx in mask_idxs:
    x[idx] = tokenizer.mask_token
sentence = ' '.join(x)
print(f"              Input Sentence: {sentence}")

# YOUR CODE HERE!
print("="*150)
# 1. Number of samples
num_samples = 5
i = 0
for _ in range(num_samples):
    inputSentence = sentence
    for _ in range(3):  #2. Number of Masks: 3
        input = tokenizer.encode(inputSentence, return_tensors='pt').to(device)
        maskTokenIndex = torch.where(input == tokenizer.mask_token_id)[1]

        # 3. Get the model predictions
        with torch.no_grad():
            predictions = model(input)[0]

        # 4. Get the predicted token ID through ancestral sampling
        probabilities = torch.softmax(predictions[0, maskTokenIndex[0], :],␣
 ↪dim=-1)
        predictedTokenID = torch.multinomial(probabilities, 1).item()
        predictedToken = tokenizer.decode([predictedTokenID])

        # Replace the first mask token in the inputSentence
        inputSentence = inputSentence.replace(tokenizer.mask_token,␣
 ↪predictedToken, 1)
    i += 1
    print(f"Ancestral Sampled Sentence {i}: {inputSentence}")
```

```
              Input Sentence: Pita Pockets is the best [MASK] [MASK] [MASK] town
of Amherst, and everyone should try their falafel!
================================================================================
======================================================================
Ancestral Sampled Sentence 1: Pita Pockets is the best from the entire town of
Amherst, and everyone should try their falafel!
Ancestral Sampled Sentence 2: Pita Pockets is the best in the entire town of
Amherst, and everyone should try their falafel!
Ancestral Sampled Sentence 3: Pita Pockets is the best and finest wine town of
Amherst, and everyone should try their falafel!
Ancestral Sampled Sentence 4: Pita Pockets is the best in the whole town of
Amherst, and everyone should try their falafel!
```

`Ancestral Sampled Sentence 5: Pita Pockets is the best in the small town of Amherst, and everyone should try their falafel!`

2.0.6 Question 1.6 (10 points)

Describe qualitatively the differences between the outputs you observed from ancestral sampling (Q1.5) and greedy decoding (Q1.4).

From the given outputs, one can observe a few differences between ancestral sampling and greedy decoding: 1. **Variability:** Ancestral sampling generates more diverse outputs by using random words such as “entire”, “diverse”,“whole”, “small” and “finest” by selecting tokens from the entire distribution rather than just choosing the most probable token. This leads to different output sentences for each sample. On the other hand, greedy decoding always chooses the most probable token “in the small”, resulting in the same output for each decoding. 2. **Semantic Consistency:** In these outputs, the sentences generated through ancestral sampling seem to be more semantically consistent and diverse, while the sentence generated through greedy decoding appears to be more predictable and less variable. 3. **Coherence and Grammatical Correctness:** Both methods seem to maintain grammatical correctness and produce coherent sentences. However, the quality of the output varies depending on the complexity of the sentence and the number of masked tokens. the sentences generated from ancestral sampling are less meaningful wrt to the context of the sentence compared to greedy decoding. 4. **Risk of Falling into Local Optima:** Greedy decoding can sometimes result in suboptimal results because it makes locally optimal choices at each step without considering the overall sentence structure. In contrast, due to its probabilistic nature, ancestral sampling is more likely to explore different paths and potentially avoid local optima.

Overall, the method one chooses between ancestral sampling and greedy decoding depends on the specific requirements of our task. For example, if one needs more diverse outputs, ancestral sampling might be a better choice, while if one is seeking the most probable completion, greedy decoding could be preferred.

3 Part 2. Write questions that large language models cannot answer (40 pts)

In this part, we will explore the limits of modern LLMs. You will be asked to first write 3 questions that ChatGPT (or a similar instruction/RLHF-tuned LLM such as GPT-4, BingChat, Bard, or YouChat) answers incorrectly. Note that some/all of these services require you to make accounts, and some may also require you to get off a waitlist. Then, you’ll try to get these models to answer the questions correctly via various prompting strategies. Finally, you’ll be asked to describe your experience working with these models.

3.0.1 Question 2.1 (15 points)

Pick an LLM API from the ones mentioned above, or any others that fit the criteria. Write three questions whose answers are *long-form* text (i.e., multiple sentences long, such as those found on r/explainlikeimfive), and which the LLM answers *incorrectly.* For this question, you must prompt the LLM in *zero-shot* mode without any instructions or any other fancier prompt engineering!

Q1: Suppose I have an 8x8 grid. The columns are labeled 1-8 from left to right, and the rows are labeled A-H from top to bottom. All cells are empty except for cell B-3, which has a rat, and cell

F-6, which has cheese. Suppose the following cells are impassable(the rat cannot go through them): D4, D5, D6, D7, E4, F4. Also, assume that the rat can only move up-down-left-right to adjacent cells, not diagonally. What is the exact series of cells the rat can move through to get to the cheese as quickly as possible?

- **Human Answer**:

Based on the given grid and impassable cells, the shortest path from B-3 to F-6 is as follows: B-3 -> C-3 -> D-3 -> E-3 -> F-3 -> G-3 -> G-4 -> G-5 -> F5 -> F-6 (9 cells) using any shortest path algorithm.

- **LLM Answer (ChatGPT)**:

To find the shortest path for the rat to reach the cheese, we can use a pathfinding algorithm such as Breadth-First Search (BFS) or Dijkstra's algorithm. In this case, since the grid is relatively small, we can use BFS.

Here's the step-by-step process for finding the shortest path from B-3 to F-6:

1. Create a queue and enqueue the starting cell B-3.
2. Create a visited set to keep track of cells already visited.
3. Create a dictionary to store the parent of each cell, to reconstruct the path later.
4. While the queue is not empty, do the following:

- Dequeue a cell from the front of the queue.
- If the dequeued cell is the target cell F-6, break out of the loop
- Otherwise, for each valid adjacent cell (up, down, left, right), do the following:
  - Check if the adjacent cell is within the grid and is not impassable (D4, D5, D6, D7, E4, F4).
  - If the adjacent cell is valid and not visited, enqueue it, mark it as visited, and store its parent as the current cell.
- Once the target cell is found or the queue becomes empty, check if the target cell was reached.
  - If the target cell was not reached, there is no valid path to the cheese.
  - If the target cell was reached, reconstruct the path from the target cell to the starting cell using the parent dictionary.

Based on the given grid and impassable cells, the shortest path from B-3 to F-6 is as follows:

B-3 -> C-3 -> C-4 -> C-5 -> C-6 -> D-6 -> E-6 -> F-6

So, the rat needs to move through the following cells to reach the cheese as quickly as possible: B-3, C-3, C-4, C-5, C-6, D-6, E-6, F-6.

Q2: A house has windows on all the four sides, all pointing south. A bear walks by the window. What color is the bear and why?

- **Human Answer**:

The color of the bear is white. It's a polar bear. All windows on the house point south because we're at the North Pole.

- **LLM Answer (ChatGPT)**:

Based on the given information, we can conclude that the bear would most likely be brown or black in color. The color of the bear has nothing to do with the orientation of the windows or the direction they are pointing. The color of a bear is determined by its species and geographical location.

Brown bears and black bears are common bear species found in various parts of the world. Brown bears can vary in color from light brown to dark brown, while black bears are typically black in color. The specific coloration of the bear in this scenario would depend on the species of bear and the region in which the story is set.

Q3: Answer with reasoning. Is Jupiter more dense than Saturn?

- **Human Answer**:

Jupiter is denser than Saturn because it 3 times more massive. Gravity makes it denser. Jovian planets are not quite spherical because of their rapid rotation. The layers are different for the different planets. * **LLM Answer (ChatGPT)**:

No, Jupiter is not more dense than Saturn. In fact, Saturn is less dense than Jupiter. Both Jupiter and Saturn are gas giants, composed primarily of hydrogen and helium, but their densities differ due to variations in their internal structures.

Jupiter has a density of about 1.33 grams per cubic centimeter, making it one of the least dense planets in the solar system. It has a massive and compact core surrounded by layers of metallic hydrogen and molecular hydrogen. The immense pressure in Jupiter's core compresses its materials, resulting in a relatively high overall density.

On the other hand, Saturn has a density of approximately 0.69 grams per cubic centimeter, making it less dense than Jupiter. Saturn has a similar internal structure to Jupiter, with a core, metallic hydrogen, and molecular hydrogen layers. However, Saturn's core is less massive and its overall structure is less compressed, leading to a lower density compared to Jupiter.

So, in summary, Jupiter is denser than Saturn, despite both planets being composed primarily of gases.

3.0.2 Question 2.2 (5 points)

Overall, what are the biggest issues with the generated answers? Some aspects you might consider are factuality, completeness, and ease of understanding.

LLM Generated Answer 1suffers from limited ease of understanding in terms of spatial reasoning and visualizing abilities. Spatial reasoning skills are important for many tasks and activities, such as navigation, problem-solving, and visualizing graphs in mind. It suffers from some factual incorrectness, especially in terms of identifying the impassable blocks, which can be more of a complex sentence understanding issue. In terms of completeness, it gave, although a wrong answer, it gives a complete justification/ reasoning to the best of its understanding of the answer generated.

LLM Generated Answer 2suffers from limited intuitiveness around logical and physical reasoning. (ease of understanding) In ML, physical reasoning is challenging and requires specialized models to learn and reason about physical/geographic systems. Moreover, inductive reasoning needs models to conclude by going from the specific to the general (Biased to Generalised), which is again challenging. Because of limited ease of understanding, the model concludes with factually incorrect answers. In terms of completeness, it gave, although a wrong answer, it gives a complete justification/ reasoning to the best of its understanding of the answer generated.

LLM Generated Answer 3 suffers from factual error inaccuracies and ambiguity. It has some degree of completeness as ChatGPT mentions both the densities of Jupiter and Saturn, which intuitively suggests that Jupiter is denser than Saturn. With respect to ease of understanding, the answer has some significant wrong reasoning sentences surrounding Jupiter's planet understanding comparable to its counterpart Saturn.

3.0.3 Question 2.3 (20 points)

Now, use any prompt engineering technique you like to help the LLM answer your questions correctly. You are obviously not allowed to include your questions and their correct answers in the prompt. If you are unable to prompt a model to answer one or more of your questions correctly, please list all of the strategies you tried, and then paste the prompt that led to the highest-quality generated answer.

Q1: Suppose I have an 8x8 grid. The columns are labeled 1-8 from left to right, and the rows are labeled A-H from top to bottom. All cells are empty except for cell B-3, which has a rat, and cell F-6, which has cheese. Suppose the following cells are impassable(the rat cannot go through them): D4, D5, D6, D7, E4, F4. Also, assume that the rat can only move up-down-left-right to adjacent cells, not diagonally. What is the exact series of cells the rat can move through to get to the cheese as quickly as possible?

- **Human Answer**:

Based on the given grid and impassable cells, the shortest path from B-3 to F-6 is as follows: B-3 -> C-3 -> D-3 -> E-3 -> F-3 -> G-3 -> G-4 -> G-5 -> F5 -> F-6 (9 cells) using any shortest path algorithm.

- **Zero-shot LLM Answer**:

To find the shortest path for the rat to reach the cheese, we can use a pathfinding algorithm such as Breadth-First Search (BFS) or Dijkstra's algorithm. In this case, since the grid is relatively small, we can use BFS.

Here's the step-by-step process for finding the shortest path from B-3 to F-6:

1. Create a queue and enqueue the starting cell B-3.
2. Create a visited set to keep track of cells already visited.
3. Create a dictionary to store the parent of each cell, to reconstruct the path later.
4. While the queue is not empty, do the following:

- Dequeue a cell from the front of the queue.
- If the dequeued cell is the target cell F-6, break out of the loop
- Otherwise, for each valid adjacent cell (up, down, left, right), do the following:
  - Check if the adjacent cell is within the grid and is not impassable (D4, D5, D6, D7, E4, F4).
  - If the adjacent cell is valid and not visited, enqueue it, mark it as visited, and store its parent as the current cell.
- Once the target cell is found or the queue becomes empty, check if the target cell was reached.
  - If the target cell was not reached, there is no valid path to the cheese.
  - If the target cell was reached, reconstruct the path from the target cell to the starting cell using the parent dictionary.

Based on the given grid and impassable cells, the shortest path from B-3 to F-6 is as follows:

B-3 -> C-3 -> C-4 -> C-5 -> C-6 -> D-6 -> E-6 -> F-6

So, the rat needs to move through the following cells to reach the cheese as quickly as possible: B-3, C-3, C-4, C-5, C-6, D-6, E-6, F-6.

- **Your prompt -> Papraphrased Prompt**:

What is the exact series of cells the rat (placed at B-3) can move through horizontally or vertically to get to the cheese (placed at F-6) as quickly as possible in an 8x8 grid where the rat is not allowed to pass through the following cells: D4, D5, D6, D7, E4, F4 and the grid is labeled 1-8 from left to right, and A-H from top to bottom?

- **Prompt-engineered answer (ChatGPT)**:

To determine the exact series of cells the rat can move through to reach the cheese as quickly as possible, we can use an algorithm such as breadth-first search (BFS) to find the shortest path while avoiding the blocked cells. Here's the path from B-3 to F-6:

Convert the grid positions into numerical coordinates:

B-3: (2, 3) F-6: (6, 6) Apply the BFS algorithm to find the shortest path:

Start at (2, 3) and explore neighboring cells in a queue. The possible moves are up, down, left, and right, as long as they are within the grid boundaries and not blocked by the specified cells. Keep track of visited cells to avoid revisiting them. Repeat the process until reaching the target cell (6, 6) or until all reachable cells have been explored. Here is the sequence of cells the rat can move through, represented as grid positions:

B-3 (2, 3) → B-4 (2, 4) → B-5 (2, 5) → B-6 (2, 6) → B-7 (2, 7) → C-7 (3, 7) → D-7 (4, 7) → E-7 (5, 7) → F-7 (6, 7) → F-6 (6, 6)

Thus, the rat can reach the cheese at F-6 by moving through the above sequence of cel

- **Strategies you tried**:

Some of the paraphrasing prompt engineering strategies I tried for this question are in zero-shot setting are:

1. I asked a related question that would lead ChatGPT to think about the answer, for example: "Can you represent the grid for the before question as a graph? Can you mark the impassable cells in such a grid?"
2. I gave a clue or a suggestion that would help the ChatGPT to find the answer. For example: "You can use a shortest path algorithm to solve this problem."
3. I gave a list of possible answers and asked the ChatGPT to choose one and provide reasoning, for example: "Is the first cell in the path B-4, C-3, or B-2?" "Why did you choose B-4 as the first cell in the path?"

I settled on paraphrasing because it allows me to restate the question more simply and clearly, making it easier for the ChatGPT to understand and answer. For example: "There is a rat at B-3 and cheese at F-6 in an 8x8 grid. The rat can only move up, down, left, or right. Some cells are blocked and the rat cannot go through them. The grid has numbers from 1 to 8 on the columns and letters from A to H on the rows. What is the shortest way for the rat to reach the cheese? List all the cells in order."

Q2: A house has windows on all the four sides, all pointing south. A bear walks by the window. What color is the bear and why?

- **Human Answer**:

The color of the bear is white. It's a polar bear. All windows on the house point south because we're at the North Pole. * **Zero-shot LLM Answer**:

Based on the given information, we can conclude that the bear would most likely be brown or black in color. The color of the bear has nothing to do with the orientation of the windows or the direction they are pointing. The color of a bear is determined by its species and geographical location.

Brown bears and black bears are common bear species found in various parts of the world. Brown bears can vary in color from light brown to dark brown, while black bears are typically black in color. The specific coloration of the bear in this scenario would depend on the species of bear and the region in which the story is set.

- **Your prompt -> Self-Ask Prompting**:
  - Prompt 1: Given a house covered with windows from all sides and all such windows pointing south where is the house located on Earth?
  - Prompt 2: If a bear walks by the window of such a house, what color is the bear and why?
- **Prompt-engineered answer (ChatGPT)** :
  - Prompt 1 Answer: Based on the information provided, if a house has windows on all sides pointing south, it suggests that the house is located near the Earth's North Pole. In polar regions, such as the Arctic, there are times when the sun remains low in the southern sky, and buildings may have windows oriented towards the south to maximize sunlight and heat gain.

  The positioning of windows in this manner would be beneficial for capturing sunlight during the limited daylight hours in polar regions, where the sun remains low on the horizon. However, it's important to note that this is just a general assumption based on the given information, and it's possible for such window placement to exist in other contexts as well.

  - Prompt 2 Answer:

  Given that the house is located near the Earth's North Pole, it is reasonable to assume that the bear in question could be a polar bear. Polar bears are native to the Arctic region and are well-adapted to the cold, icy environments found there. Their fur is uniquely suited for camouflage in the Arctic environment, and it appears white.

  The white coloration of polar bears helps them blend in with the snow and ice, providing them with effective camouflage for hunting and avoiding predators. This adaptation allows them to approach their prey more stealthily.

  Therefore, based on the assumption that the house is located near the North Pole, it is likely that the bear passing by the window would be a polar bear, and its color would be white.

- **Strategies you tried**: Some of the prompt engineering strategies I tried for this question are:

1. **Instruction prompting**:

I gave a list of possible answers and asked the ChatGPT to choose one. For example: "Is the bear white, brown, black, or polar for a geographical place like Massachusetts, Alaska, or California."

2. **Few Shot Chain of Thought prompting**: I asked ChatGPT to explain its answer or reasoning in more detail. For example: "Question: Given a house in California and a bear walks by the window of such a house. What color is the bear, and why? Answer: The color of the bear is brown." "Question: Given a house in England and a bear walks by the window of such a house. What color is the bear, and why? Answer: The color of the bear is black"

I settled on the Self-Ask prompting strategy because it allows the user to generate their own questions and answers, enhancing their learning and understanding of the problem. For example: "What does it mean to have windows pointing south on all four sides? What kind of place has such a house? What kind of bears live in that place?"

Finally I settled for Self Ask Prompting Strategy mentioned in **MEASURING AND NARROWING THE COMPOSITIONALITY GAP IN LANGUAGE MODELS** paper by Ofir Press.

Q3: Answer with reasoning. Is Jupiter more dense than Saturn?

- **Human Answer**:

Jupiter is denser than Saturn because it 3 times more massive. Gravity makes it denser. Jovian planets are not quite spherical because of their rapid rotation. The layers are different for the different planets. * **Zero-shot LLM Answer**:

No, Jupiter is not more dense than Saturn. In fact, Saturn is less dense than Jupiter. Both Jupiter and Saturn are gas giants, composed primarily of hydrogen and helium, but their densities differ due to variations in their internal structures.

Jupiter has a density of about 1.33 grams per cubic centimeter, making it one of the least dense planets in the solar system. It has a massive and compact core surrounded by layers of metallic hydrogen and molecular hydrogen. The immense pressure in Jupiter's core compresses its materials, resulting in a relatively high overall density.

On the other hand, Saturn has a density of approximately 0.69 grams per cubic centimeter, making it less dense than Jupiter. Saturn has a similar internal structure to Jupiter, with a core, metallic hydrogen, and molecular hydrogen layers. However, Saturn's core is less massive and its overall structure is less compressed, leading to a lower density compared to Jupiter.

So, in summary, Jupiter is denser than Saturn, despite both planets being composed primarily of gases.

- **Your prompt -> Zero Shot Chain of Thought prompting**: What is the density of Jupiter? What is the the density of Saturn? Answer with reasoning. Is Jupiter more dense than Saturn?
- **Prompt-engineered answer**:

Jupiter is indeed more dense than Saturn. The density of Jupiter is approximately 1.33 grams per cubic centimeter, while the density of Saturn is about 0.69 grams per cubic centimeter.

Jupiter, with its higher density, has a more massive and compact core compared to Saturn. The intense gravitational forces compress the materials in Jupiter, resulting in a higher overall density. Saturn, although it has a similar structure with core, metallic hydrogen, and molecular hydrogen layers, has a less massive core and is less compressed, leading to a lower density.

To summarize, Jupiter is more dense than Saturn. * **Strategies you tried**:

Some of the prompt engineering strategies I tried for this question are: Paraphrasing: I gave a list of possible answers and asked ChatGPT to choose one. For example: "Is Jupiter more dense than Saturn, less dense than Saturn, or equally dense as Saturn?" SelfAsk: I asked the ChatGPT to explain their answer or reasoning in more detail. For example: "How did you calculate the densities of Jupiter and Saturn?" I settled on Zero Shot Chain of Thought prompting on ChatGPT because it allows me to generate a natural and engaging conversation with the user, where I can guide them to find the answer by asking and answering questions along the way.

3.1 AI Disclosure

- Did you use any AI assistance to complete this homework? If so, please also specify what AI you used.
  - *Yes: ChatGPT*

(only complete the below questions if you answered yes above)

- If you used a large language model to assist you, please paste *all* of the prompts that you used below. Add a separate bullet for each prompt, and specify which problem is associated with which prompt.
  - Suppose I have an 8x8 grid. The columns are labeled 1-8 from left to right, and the rows are labeled A-H from top to bottom. All cells are empty except for cell B-3, which has a rat, and cell F-6, which has cheese. Suppose the following cells are impassable(the rat cannot go through them): D4, D5, D6, D7, E4, F4. Also, assume that the rat can only move up-down-left-right to adjacent cells, not diagonally. What is the exact series of cells the rat can move through to get to the cheese as quickly as possible?
  - A house has windows on all the four sides, all pointing south. A bear walks by the window. What color is the bear and why?
  - Answer with reasoning. Is Jupiter more dense than Saturn?
  - What is the exact series of cells the rat (placed at B-3) can move through horizontally or vertically to get to the cheese (placed at F-6) as quickly as possible in an 8x8 grid where the rat is not allowed to pass through the following cells: D4, D5, D6, D7, E4, F4 and the grid is labeled 1-8 from left to right, and A-H from top to bottom?
  - Self Ask Prompt 1: Given a house covered with windows from all sides and all such windows pointing south where is the house located on Earth? Self Ask Prompt 2: If a bear walks by the window of such a house, what color is the bear and why?
  - What is the density of Jupiter? What is the the density of Saturn? Answer with reasoning. Is Jupiter more dense than Saturn?
- **Free response**: For each problem for which you used assistance, describe your overall experience with the AI. How helpful was it? Did it just directly give you a good answer, or did you have to edit it? Was its output ever obviously wrong or irrelevant? Did you use it to get the answer or check your own answer?

– *your response here*

```
[ ]:
```