Assignment 3: RNN, Transformer and LLM-based Agent Design

With your environment activated int the terminal, run:

```
mamba env create -n cs5293-3 python=3.10
pip install -r requirements.txt
##Your VSCode may complain sometime you need to install ipykernel
using the following commands. If not, then  just ignore this.
#mamba install -n cs5293-3 ipykernel  --force-reinstall
```

In this assignment, you have to submit two things:

- (1) The whole folder with your code
- (2) A report to summarize what your experienements in part 2 and part 3

Part 1: Reading Assignment on Transformer Jupernotebook (20')

There two excellent transformer juternote books at large, which covers a great amount details of transformer that we could cannot cover more in the class: "The Illustrated Transfomer" by Jay Alammar and "The Annotated Transformer" by harvardnlp lab.

Some one combined this two and create a single jupternotebook here. You need to do the following:

1. Adjust your own requirements.txt (if necessary to your environment) to make that notebook all runnable
2. Understand the details of the key components of transformer, run through the code, and take any notes you want to take on this notebook.
3. This note book could be printed out for the final exam, but if you don't understand them before the exam. It will be hard.

What to turn in: your own annotations on the above notebook that you will bring with you to the final exam.

Part 2: Programming Assignment (Total: 60)

In Assignment 2, you have worked on a 5-way deep average network on sentiment dataset (SST-5), which has 5 labels: very positive, positive, neutral, negative, very negative. However, the performance is realtive low.

In this part, based on the same SST-5 dataset, you have two tasks:

- Section 2.1. Using builtin pytorch 1-layer LSTM with the same given word embedding to improve the performance. (30')
- Section 2.2 Using the library of Huggingface Transformers to improve your model with BERT Finetuning.(30' + 10')

Important Hints:

1. You will find a ton of existing code for this two tasks, it is ok to refer or reuse some of thsoe code.
2. But some of those code are too old. You SHOULD use the lastest stable pytorch(2.6.0) and huggingface tranformers(>4.50.0) for this.
3. To run a batch job on OSCER, you need to reassemble the code from the notebook into regular source code to submit a slurm job.

What to turn in: You need to turn in the code and a pdf report to show the performance of your two new models on SST-5 dataset. Performance Metrics:

- Classification Report in Sklearn (https://scikit-learn.org/stable/modules/generated/sklearn.metrics.classification_report.html)
- Confusion Matrix (https://scikit-learn.org/stable/modules/generated/sklearn.metrics.confusion_matrix.html)

Section 2.1 Pytorch LSTM (30)

Your goal: Add a new pytorch model(nn.module) using the 1-layer LSTM module for your SST-5 sentiment classifier in assignment 2.

Hint: There are tons of tutorial of using pytorch LSTM, even for this SST-5 dataset...

1. https://github.com/doslim/Sentiment-Analysis-SST5/blob/8f041635a3b959a405e2105bc9037b6af77aa7ba/semtiment%20analysis/codes/model.py#L45 This one use both lstm and attention. You just need to play with lstm with hyper parameters (such as hidden layer size, learning rate, weight decay, etc.)
2. https://colab.research.google.com/gist/SauravMaheshkar/168f0817f0cd29dd4048868fb0dd4401/lstms-in-pytorch.ipynb#scrollTo=BKAA2rR0-B-3 This also introduce the code how to use wandb to track your experiments, which will be very useful for your project.

What to Report:

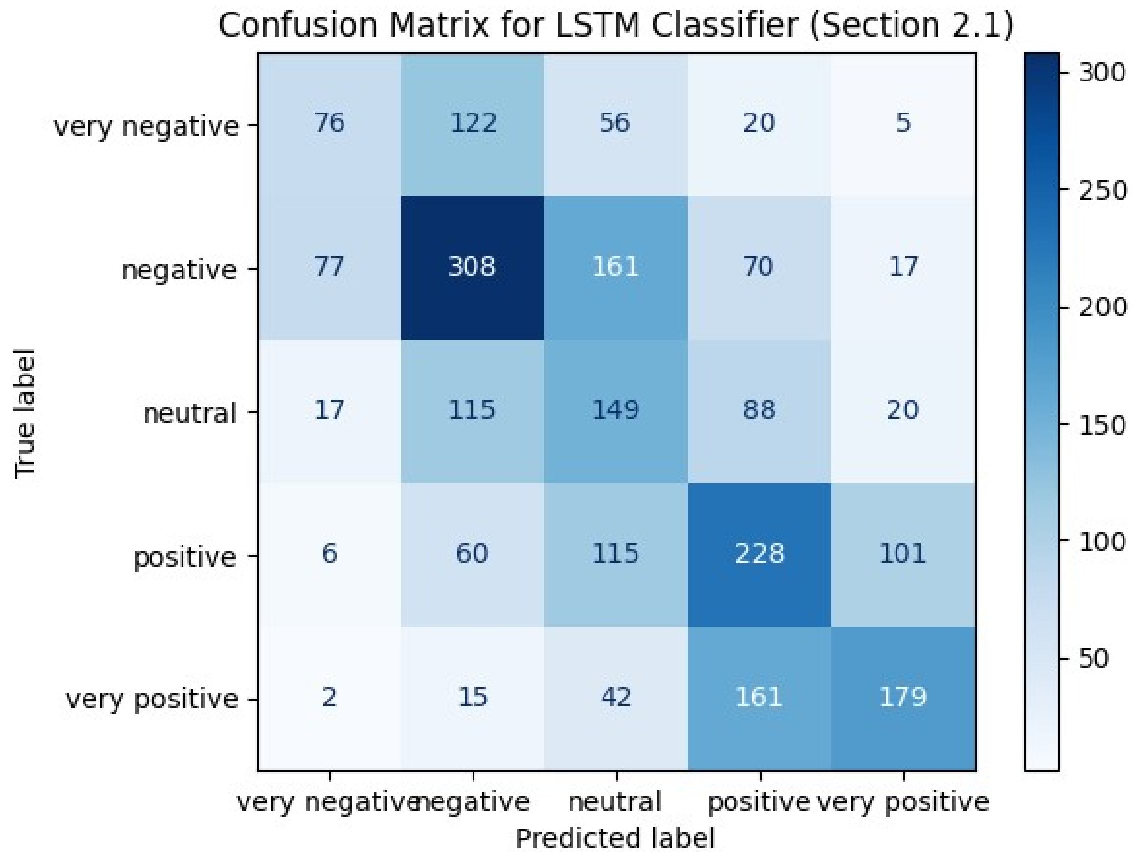
- Your LSTM hyperparameters (https://medium.com/geekculture/10-hyperparameters-to-keep-an-eye-on-for-your-lstm-model-and-other-tips-f0ff5b63fcd4)
- Performance Metrics

```
# This cell runs the sentiment_classifier.py script, which loads the
SST-5 sentiment dataset and
# GloVe embeddings, builds a 1-layer LSTM classifier for 5-class
sentiment classification, trains
# the model on CPU, and prints the training loss and evaluation
report.

%run ../src/sentiment_classifier.py
```

```
Repo card metadata block was not found. Setting CardData to empty.
```

```
Training on CPU...
Epoch 1/50 - Loss: 1.5727
Epoch 2/50 - Loss: 1.5686
Epoch 3/50 - Loss: 1.5682
Epoch 4/50 - Loss: 1.5676
Epoch 5/50 - Loss: 1.5657
Epoch 6/50 - Loss: 1.5643
Epoch 7/50 - Loss: 1.5537
Epoch 8/50 - Loss: 1.5644
Epoch 9/50 - Loss: 1.5601
Epoch 10/50 - Loss: 1.5574
Epoch 11/50 - Loss: 1.5566
Epoch 12/50 - Loss: 1.5563
Epoch 13/50 - Loss: 1.5529
Epoch 14/50 - Loss: 1.5508
Epoch 15/50 - Loss: 1.5483
Epoch 16/50 - Loss: 1.5425
Epoch 17/50 - Loss: 1.5390
Epoch 18/50 - Loss: 1.5248
Epoch 19/50 - Loss: 1.4658
Epoch 20/50 - Loss: 1.4646
Epoch 21/50 - Loss: 1.3565
Epoch 22/50 - Loss: 1.2793
Epoch 23/50 - Loss: 1.2253
Epoch 24/50 - Loss: 1.1703
Epoch 25/50 - Loss: 1.1236
Epoch 26/50 - Loss: 1.0580
Epoch 27/50 - Loss: 0.9962
Epoch 28/50 - Loss: 0.9187
Epoch 29/50 - Loss: 0.8570
Epoch 30/50 - Loss: 0.7818
Epoch 31/50 - Loss: 0.7008
Epoch 32/50 - Loss: 0.6192
Epoch 33/50 - Loss: 0.5619
Epoch 34/50 - Loss: 0.5058
Epoch 35/50 - Loss: 0.4592
Epoch 36/50 - Loss: 0.4381
Epoch 37/50 - Loss: 0.3757
Epoch 38/50 - Loss: 0.3400
```

```
Epoch 39/50 - Loss: 0.3123
Epoch 40/50 - Loss: 0.2957
Epoch 41/50 - Loss: 0.2776
Epoch 42/50 - Loss: 0.2501
Epoch 43/50 - Loss: 0.2540
Epoch 44/50 - Loss: 0.2339
Epoch 45/50 - Loss: 0.2171
Epoch 46/50 - Loss: 0.2131
Epoch 47/50 - Loss: 0.2201
Epoch 48/50 - Loss: 0.1780
Epoch 49/50 - Loss: 0.1831
Epoch 50/50 - Loss: 0.1747
Classification Report:
               precision    recall  f1-score   support

very negative       0.43      0.27      0.33       279
     negative       0.50      0.49      0.49       633
      neutral       0.28      0.38      0.33       389
     positive       0.40      0.45      0.42       510
very positive       0.56      0.45      0.50       399

     accuracy                           0.43      2210
    macro avg       0.43      0.41      0.41      2210
 weighted avg       0.44      0.43      0.43      2210

Classification Report:
               precision    recall  f1-score   support

very negative       0.43      0.27      0.33       279
     negative       0.50      0.49      0.49       633
      neutral       0.28      0.38      0.33       389
     positive       0.40      0.45      0.42       510
very positive       0.56      0.45      0.50       399

     accuracy                           0.43      2210
    macro avg       0.43      0.41      0.41      2210
 weighted avg       0.44      0.43      0.43      2210


<Figure size 800x600 with 0 Axes>
```



Section 2.2 Huggingface Transformer, Various LMs (30)

Your goal: use the huggingface transformers library to rebuilt your sentiment classifier There are a lot of old version codes of using BERT for SST-5, DON"T use them but read them will help you understand. They are obsolete but they have more details. Such as this one. https://github.com/munikarmanish/bert-sentiment/blob/master/bert_sentiment/train.py

The most recent version of huggingface is easy to use but hide too many details. I hope to make your life easier, please use the eaiser pytorch version of this following tutoral for SST-5. https://huggingface.co/docs/transformers/en/tasks/sequence_classification

When you run this jupternobook, you will get prompted to two accounts:

1. Create a huggingface account and create a access token to login in this note book.
2. Create a wandb account to keep the log into the wandb, which has been automaitically integrated into hugginface trainer to track your experiments. See the details here. https://docs.wandb.ai/guides/integrations/huggingface/

The above tutorial will take you about 40 minutes to run on an old Nvidia Tesla T4 GPU with the free version of Colab, and sometimes it is slow. So try the Oscer first, then paid Colab GPU.

Your Task:

- See if you could replace the dataset with SST-5, and the model with "bert-base-cased" and get familiar with this new framework for your sentiment classifier, report the final performance.

(HINT: you almost only need to change the parameters in AutoTokenizer, and AutoModelXX, the learning rate for finetuning is often small (around 10^-3 to 10^-5), the epoch is also around 3 to 10)

What to Report:

- Your BERT related training hyperparameters.
- Performance Metrics

```
# Path to your log file
log_file_path = "../src/sst5_bert_output_24546558.log"

# Open and read the log file
with open(log_file_path, 'r') as file:
    lines = file.readlines()

# Filter lines containing keywords like 'Epoch', 'accuracy', 'f1',
'precision', 'recall'
filtered_lines = []
for line in lines:
    if ("epoch" in line) or ("accuracy" in line) or ("f1" in line) or
("precision" in line) or ("recall" in line):
        filtered_lines.append(line.strip())

# Print the filtered lines
print("\n".join(filtered_lines))
```

```
{'loss': 1.5706, 'grad_norm': 5.626635551452637, 'learning_rate':
1.963295880149813e-05, 'epoch': 0.09}
{'loss': 1.378, 'grad_norm': 9.561117172241211, 'learning_rate':
1.9258426966292136e-05, 'epoch': 0.19}
{'loss': 1.264, 'grad_norm': 8.776640892028809, 'learning_rate':
1.888389513108614e-05, 'epoch': 0.28}
{'loss': 1.2212, 'grad_norm': 12.716158866882324, 'learning_rate':
1.8509363295880153e-05, 'epoch': 0.37}
{'loss': 1.1867, 'grad_norm': 13.83592414855957, 'learning_rate':
1.813483146067416e-05, 'epoch': 0.47}
{'loss': 1.1719, 'grad_norm': 7.0720534324646, 'learning_rate':
1.7760299625468167e-05, 'epoch': 0.56}
{'loss': 1.1541, 'grad_norm': 11.458871841430664, 'learning_rate':
1.7385767790262175e-05, 'epoch': 0.66}
{'loss': 1.1952, 'grad_norm': 11.151716232299805, 'learning_rate':
1.701123595505618e-05, 'epoch': 0.75}
{'loss': 1.1173, 'grad_norm': 8.932943344116211, 'learning_rate':
1.663670411985019e-05, 'epoch': 0.84}
{'loss': 1.1607, 'grad_norm': 11.366340637207031, 'learning_rate':
1.6262172284644194e-05, 'epoch': 0.94}
```

```
{'loss': 1.1214, 'grad_norm': 13.686240196228027, 'learning_rate':
1.5887640449438206e-05, 'epoch': 1.03}
{'loss': 0.9299, 'grad_norm': 16.394588470458984, 'learning_rate':
1.551310861423221e-05, 'epoch': 1.12}
{'loss': 0.9009, 'grad_norm': 18.83949089050293, 'learning_rate':
1.5138576779026219e-05, 'epoch': 1.22}
{'loss': 0.8915, 'grad_norm': 7.689004421234131, 'learning_rate':
1.4764044943820226e-05, 'epoch': 1.31}
{'loss': 0.9349, 'grad_norm': 17.080909729003906, 'learning_rate':
1.4389513108614232e-05, 'epoch': 1.4}
{'loss': 0.9202, 'grad_norm': 25.845863342285156, 'learning_rate':
1.4014981273408241e-05, 'epoch': 1.5}
{'loss': 0.9142, 'grad_norm': 15.816347122192383, 'learning_rate':
1.3640449438202248e-05, 'epoch': 1.59}
{'loss': 0.9558, 'grad_norm': 12.840490341186523, 'learning_rate':
1.3265917602996256e-05, 'epoch': 1.69}
{'loss': 0.9323, 'grad_norm': 18.783967971801758, 'learning_rate':
1.2891385767790263e-05, 'epoch': 1.78}
{'loss': 0.9156, 'grad_norm': 10.621670722961426, 'learning_rate':
1.251685393258427e-05, 'epoch': 1.87}
{'loss': 0.8788, 'grad_norm': 12.913186073303223, 'learning_rate':
1.2142322097378278e-05, 'epoch': 1.97}
{'loss': 0.8142, 'grad_norm': 15.748014450073242, 'learning_rate':
1.1767790262172285e-05, 'epoch': 2.06}
{'loss': 0.7456, 'grad_norm': 16.365236282348633, 'learning_rate':
1.1393258426966293e-05, 'epoch': 2.15}
{'loss': 0.6692, 'grad_norm': 15.960987091064453, 'learning_rate':
1.101872659176603e-05, 'epoch': 2.25}
{'loss': 0.6716, 'grad_norm': 9.67022705078125, 'learning_rate':
1.0644194756554307e-05, 'epoch': 2.34}
{'loss': 0.7073, 'grad_norm': 14.097466468811035, 'learning_rate':
1.0269662921348315e-05, 'epoch': 2.43}
{'loss': 0.744, 'grad_norm': 21.10797882080078, 'learning_rate':
9.895131086142323e-06, 'epoch': 2.53}
{'loss': 0.6643, 'grad_norm': 11.315186500549316, 'learning_rate':
9.52059925093633e-06, 'epoch': 2.62}
{'loss': 0.6632, 'grad_norm': 24.325084686279297, 'learning_rate':
9.146067415730337e-06, 'epoch': 2.72}
{'loss': 0.6798, 'grad_norm': 15.628936767578125, 'learning_rate':
8.771535580524345e-06, 'epoch': 2.81}
{'loss': 0.6676, 'grad_norm': 16.24275016784668, 'learning_rate':
8.397003745318352e-06, 'epoch': 2.9}
{'loss': 0.6472, 'grad_norm': 21.186817169189453, 'learning_rate':
8.02247191011236e-06, 'epoch': 3.0}
{'loss': 0.4493, 'grad_norm': 15.13821029663086, 'learning_rate':
7.647940074906369e-06, 'epoch': 3.09}
{'loss': 0.4884, 'grad_norm': 19.540966033935547, 'learning_rate':
7.273408239700375e-06, 'epoch': 3.18}
{'loss': 0.4854, 'grad_norm': 16.71976661682129, 'learning_rate':
```

```
6.898876404494382e-06, 'epoch': 3.28}
{'loss': 0.4911, 'grad_norm': 17.714141845703125, 'learning_rate':
6.52434456928839e-06, 'epoch': 3.37}
{'loss': 0.5225, 'grad_norm': 10.025727272033691, 'learning_rate':
6.1498127340823975e-06, 'epoch': 3.46}
{'loss': 0.484, 'grad_norm': 18.158191680908203, 'learning_rate':
5.775280898876405e-06, 'epoch': 3.56}
{'loss': 0.4808, 'grad_norm': 16.839183807373047, 'learning_rate':
5.400749063670413e-06, 'epoch': 3.65}
{'loss': 0.4887, 'grad_norm': 23.925931930541992, 'learning_rate':
5.026217228464419e-06, 'epoch': 3.75}
{'loss': 0.501, 'grad_norm': 22.637502670288086, 'learning_rate':
4.651685393258427e-06, 'epoch': 3.84}
{'loss': 0.4756, 'grad_norm': 29.462804794311523, 'learning_rate':
4.2771535580524345e-06, 'epoch': 3.93}
{'loss': 0.4364, 'grad_norm': 13.025620460510254, 'learning_rate':
3.902621722846442e-06, 'epoch': 4.03}
{'loss': 0.3535, 'grad_norm': 20.39722442626953, 'learning_rate':
3.5280898876404497e-06, 'epoch': 4.12}
{'loss': 0.3878, 'grad_norm': 20.80815887451172, 'learning_rate':
3.1535580524344572e-06, 'epoch': 4.21}
{'loss': 0.3449, 'grad_norm': 12.114013671875, 'learning_rate':
2.779026217228465e-06, 'epoch': 4.31}
{'loss': 0.3717, 'grad_norm': 22.281856536865234, 'learning_rate':
2.404494382022472e-06, 'epoch': 4.4}
{'loss': 0.3533, 'grad_norm': 18.32479476928711, 'learning_rate':
2.0299625468164795e-06, 'epoch': 4.49}
{'loss': 0.3091, 'grad_norm': 19.425079345703125, 'learning_rate':
1.6554307116104871e-06, 'epoch': 4.59}
{'loss': 0.3182, 'grad_norm': 15.570513725280762, 'learning_rate':
1.2808988764044945e-06, 'epoch': 4.68}
{'loss': 0.3747, 'grad_norm': 21.145214080810547, 'learning_rate':
9.06367041198502e-07, 'epoch': 4.78}
{'loss': 0.3671, 'grad_norm': 7.242622375488281, 'learning_rate':
5.318352059925094e-07, 'epoch': 4.87}
{'loss': 0.3402, 'grad_norm': 23.21162223815918, 'learning_rate':
1.5730337078651687e-07, 'epoch': 4.96}
{'train_runtime': 836.5072, 'train_samples_per_second': 51.069,
'train_steps_per_second': 3.192, 'train_loss': 0.7368625278330028,
'epoch': 5.0}
Evaluation Metrics: {'accuracy': 0.5180995475113123, 'precision':
0.5263131346323328, 'recall': 0.5180995475113123, 'f1':
0.520377337495899}
```

Part 3: LLM-based Exam Agent For NLP (20')

Agentic AI is a popular word nowadays. People in CS or Non-CS uses LLM models to build all kinds of Agents with no-code or low code fashion. Please read a Nvidia's introduction about Agentic AI. https://blogs.nvidia.com/blog/what-is-agentic-ai/

In this assignment, we focus on the ngram question we used in version A of our mid-term(), let us build a simple agent to solve this via LangChain.

Requirements:

- API version of Open AI: The agent requires you to obtain your OPENAI_API_KEY by binding your bank account to OpenAI. It is Pay-as-you-go serveice, it will be a few dollars for this assignment.
- ChatGPT-free version: You need to test ChatGPT's capability for the questions. Students can claim 2 months of ChatGPT Plus for free. Get access to premium features when you verify you're a student in the US or Canada. https://chatgpt.com/students

Section 3.1. The standard way you use ChatGPT Everyday! (chatgpt.com)

By the time of this assignment, the default model of ChatGPT is GPT-4o (GPT-4-omini). The model is capable of performing a wide range of tasks and using tools. Hence, you can ask it answer the above question 3 and 4. You will find out the default ChatGPT(GPT-4o) works great!

For question-3,I have used chatgpt-4o version and i just copy pasted the question, ie

My response,Given the following corpus:

```
<s> some are fast </s>
<s> and some are slow </s>
<s> some are high </s>
<s> and some are low </s>
```

Treat `<s>` and `</s>` as tokens.(i.e include them in vocabulary and counts) What is the conditional probability of seeing 'some' given that we just saw 'and' (P(some | and))? Assume bigram and no smoothing."

And below is the chatgpt-4o response,

Alright, let's walk through this carefully.

The corpus is:

```
<s> some are fast </s>
<s> and some are slow </s>
<s> some are high </s>
<s> and some are low </s>
```

We are asked: **What is P(some | and)** using a **bigram model** and **no smoothing**.

Steps:

1. **Count how many times** the bigram ("and", "some") appears.
2. **Count how many times** "and" appears as the first word of any bigram.

Then: $$P(\text{some} \mid \text{and}) = \frac{\text{Count(and, some)}}{\text{Count(and)}}$$

First, tokenize everything (include `<s>` and `</s>`):

The tokenized sentences are:

- <s> some are fast </s>
- <s> and some are slow </s>
- <s> some are high </s>
- <s> and some are low </s>

Now list the bigrams:

- (<s>, some), (some, are), (are, fast), (fast, </s>)
- (<s>, and), (and, some), (some, are), (are, slow), (slow, </s>)
- (<s>, some), (some, are), (are, high), (high, </s>)
- (<s>, and), (and, some), (some, are), (are, low), (low, </s>)

Now count:

- **Count(and, some):** How many times does (and, some) occur?
  → From second and fourth sentence, (and, some) appears **2 times**.

- **Count(and):** How many times does "and" appear as the first word in a bigram?
  → "and" is followed by "some" both times → **2 times** total.

(Notice: "and" is never followed by anything else.)

Thus: $$P(\text{some} \mid \text{and}) = \frac{2}{2} = 1$$

□ Final Answer:1

For question-4,I have used chatgpt-4o version and i just copy pasted the question, ie

My response, Given the earlier corpus, and a bigram language model (with add-1 smoothing) what is the probability that will be assigned to the following sentence? Give your answer in detailed steps with fractions, no worries for the final numerical result.

`<s>` some are good `</s>`

Below is the chatgpt-4o version response,

Alright, let's take this step-by-step carefully and clearly.

We are given:

- Corpus:

```
<s> some are fast </s>
<s> and some are slow </s>
<s> some are high </s>
<s> and some are low </s>
```

- Task: Compute the **bigram probability with add-1 smoothing** for:

```
<s> some are good </s>
```

Step 1: Find the Vocabulary Size (V)

Vocabulary includes **all unique tokens** (including `<s>`, `</s>`, and any other words we see).

From the corpus:

- `<s>`, `</s>`, `some`, `are`, `fast`, `and`, `slow`, `high`, `low`

Thus: [V = 9]

Step 2: Get Bigram Counts

From earlier tokenization:

- (`<s>`, some): 2 times
- (`<s>`, and): 2 times
- (some, are): 4 times
- (and, some): 2 times
- (are, fast): 1 time

- (are, slow): 1 time
- (are, high): 1 time
- (are, low): 1 time
- (fast, </s>): 1 time
- (slow, </s>): 1 time
- (high, </s>): 1 time
- (low, </s>): 1 time

Important:

- "good" **never appeared** in the training data.
- So any bigram involving "good" will have a count of **0**.

Step 3: Break the Sentence into Bigrams

Sentence: <s> some are good </s>

Bigrams:

- (<s>, some)
- (some, are)
- (are, good)
- (good, </s>)

We need the smoothed probability of each bigram and then **multiply** them together.

Step 4: Formula for Add-1 Smoothed Bigram Probability

Add-1 smoothed bigram probability is:

[P(w_i \mid w_{i-1}) = \frac{\text{Count}(w_{i-1}, w_i) + 1}{\text{Count}(w_{i-1}) + V}]

Where:

- (\text{Count}(w_{i-1}, w_i)) is the number of times the bigram appears
- (\text{Count}(w_{i-1})) is the number of times (w_{i-1}) appears as the first word in **any** bigram
- (V = 9)

Step 5: Compute Each Bigram Probability

Bigram 1: (<s>, some)

- Count(<s>, some) = 2
- Count(<s>) = 4 (because <s> appears at the start of each sentence, 4 times total)

Thus:

[P(\text{some} \mid \text{<s>}) = \frac{2+1}{4+9} = \frac{3}{13}]

Bigram 2: (some, are)

- Count(some, are) = 4
- Count(some) = 4 (since "some" is always followed by "are")

Thus:

[P(\text{are} \mid \text{some}) = \frac{4+1}{4+9} = \frac{5}{13}]

Bigram 3: (are, good)

- Count(are, good) = 0 (never appeared)
- Count(are) = 4 (because "are" is followed by "fast", "slow", "high", and "low" once each — 4 total)

Thus:

[P(\text{good} \mid \text{are}) = \frac{0+1}{4+9} = \frac{1}{13}]

Bigram 4: (good, </s>)

- Count(good, </s>) = 0 (because "good" was never seen)
- Count(good) = 0 (no appearances of "good" in training)

Thus:

[P(\text{</s>} \mid \text{good}) = \frac{0+1}{0+9} = \frac{1}{9}]

(Important: when the word itself wasn't seen before, its count is 0, but vocabulary size is still 9.)

Step 6: Multiply the probabilities

The overall probability of the sentence is:

[P(\text{<s> some are good </s>}) = P(\text{some} \mid \text{<s>}) \times P(\text{are} \mid \text{some}) \times P(\text{good} \mid \text{are}) \times P(\text{</s>} \mid \text{good})]

Substituting:

[= \frac{3}{13} \times \frac{5}{13} \times \frac{1}{13} \times \frac{1}{9}]

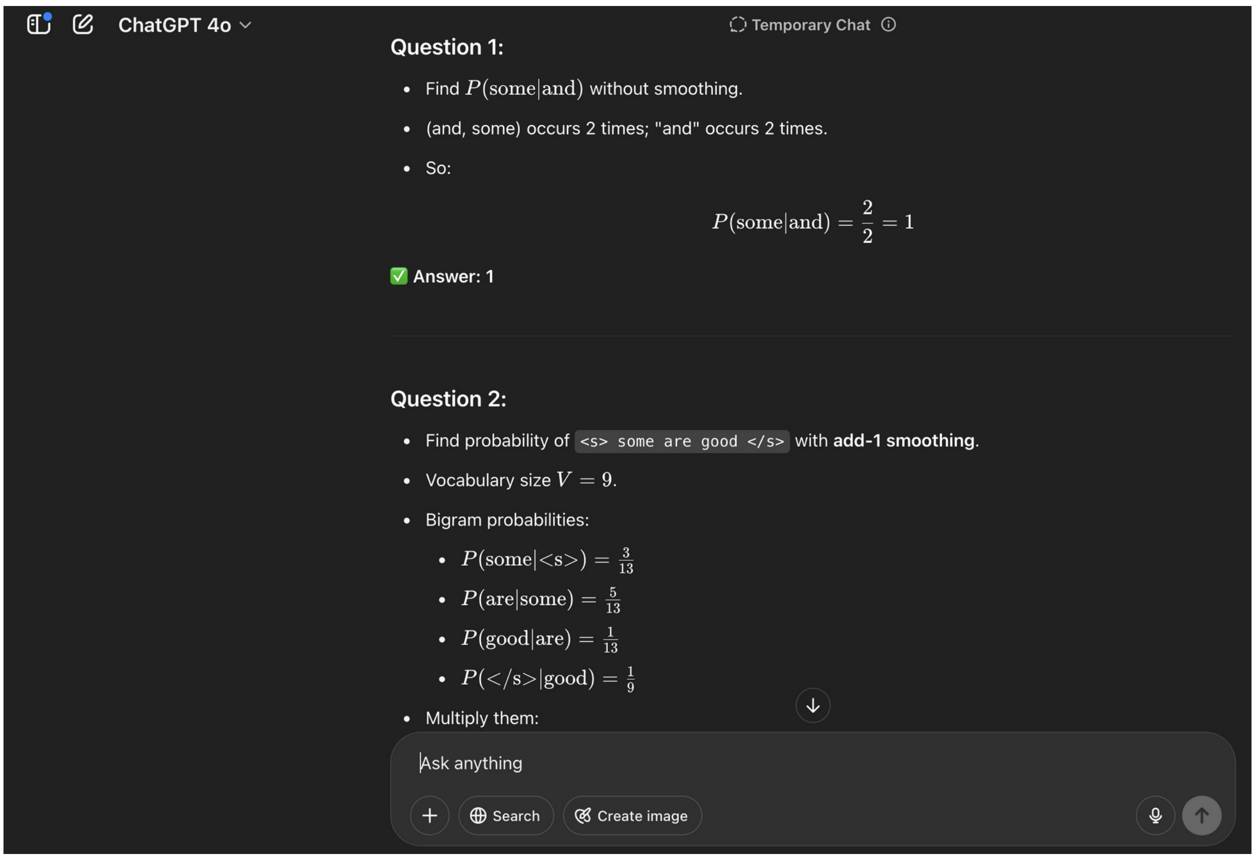
Final Answer (in detailed fraction form):

[\boxed{\frac{3}{13} \times \frac{5}{13} \times \frac{1}{13} \times \frac{1}{9}}]

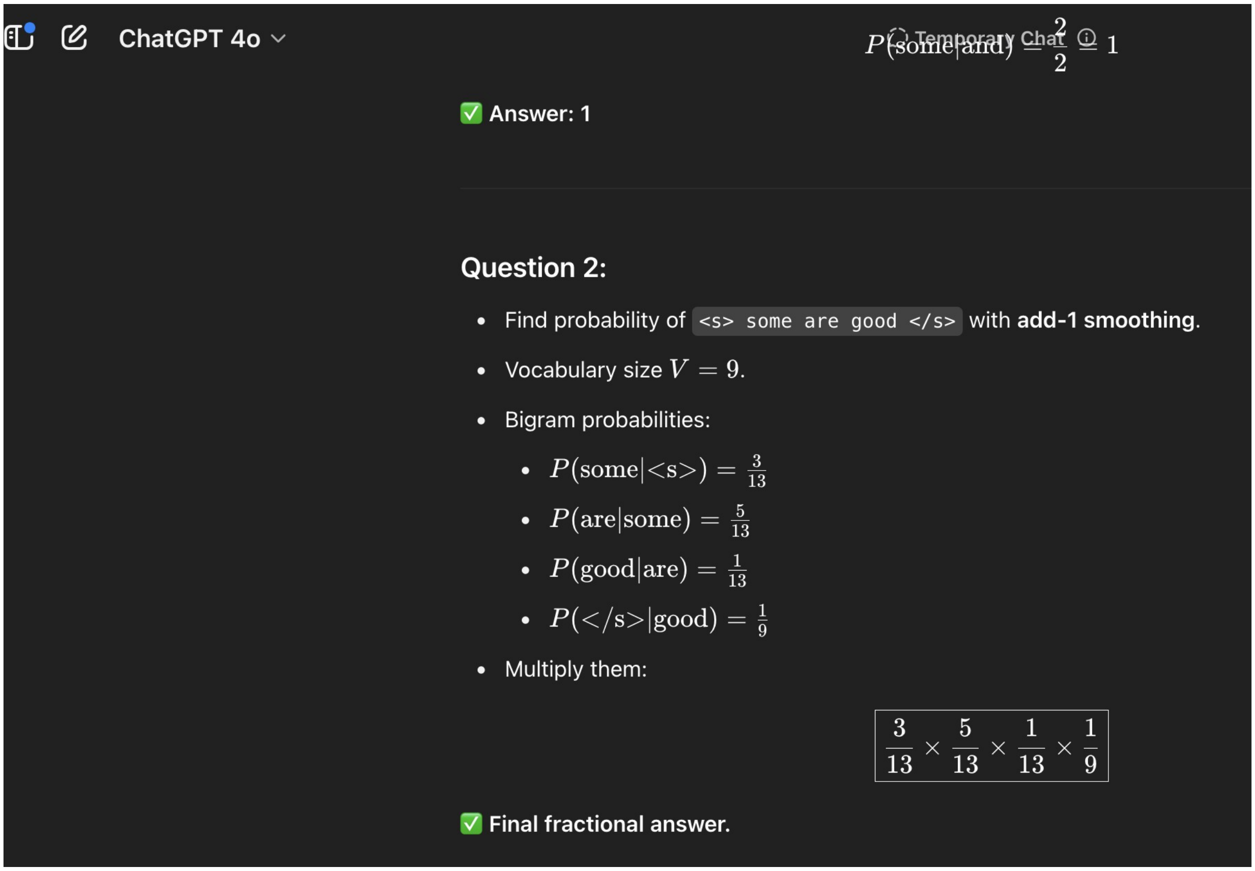
Would you like me to also simplify this a little further just for curiosity? (like multiply the fractions)? □

Below images are the summary of both questions how chatgpt solved

```python
from IPython.display import Image, display

# Example: showing a screenshot named "my_screenshot.png" in your data folder
display(Image(filename="../data/output01.png"))
```




```python
from IPython.display import Image, display

# Example: showing a screenshot named "my_screenshot.png" in your data folder
display(Image(filename="../data/output02.png"))
```



Section 3.2. API and PlayGround: The Researcher's Way Using ChatGPT

Now let's test some previous LLMs from OpenAI, https://platform.openai.com/playground/prompts?models=gpt-4o When we test the GPT-3.5-turbo(That is the earlier version of ChatGPT). We found that the previous model could not finish the whole solution.

My Key Observations for Section 3.2:

Tested on platform.openai.com/playground/prompts using both:GPT-3.5-turbo and GPT-4o (latest model)

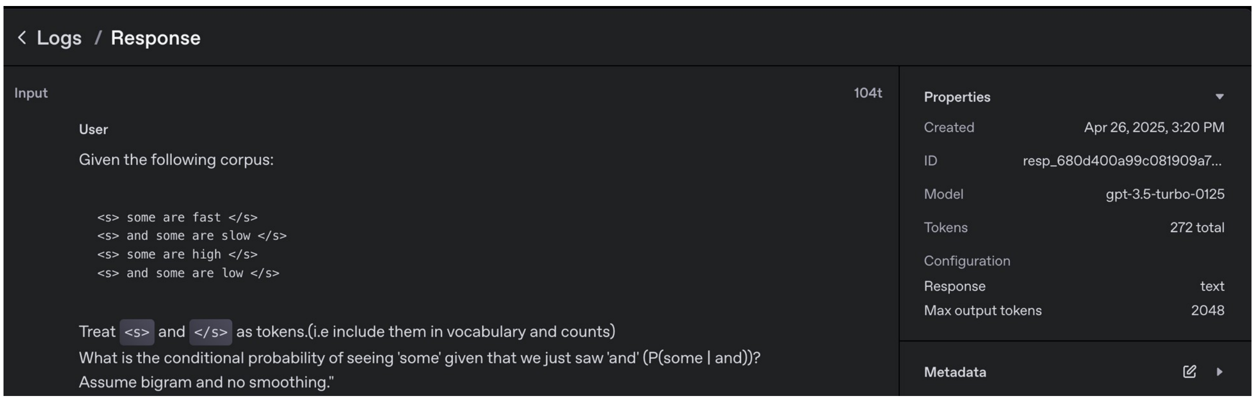
Observations with GPT-3.5-turbo (Playground):

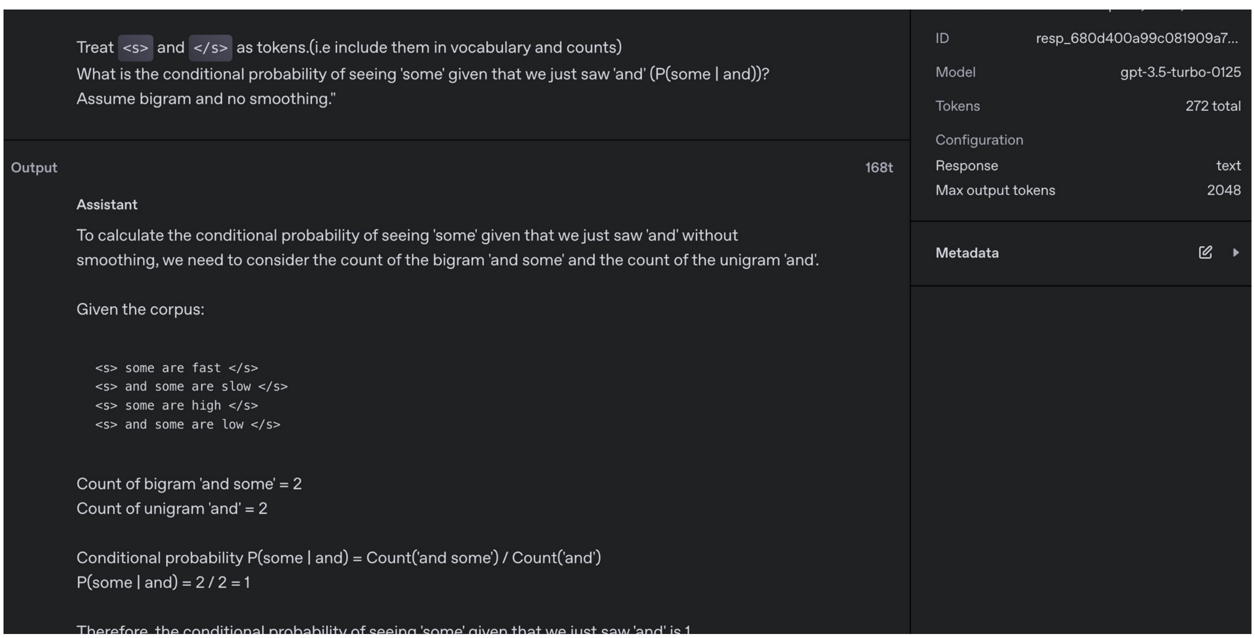
First Attempt (Q3 + Q4 together):

▯ Question 3 (Bigram P(some | and)): Correct solution, complete and accurate.

 Question 4 (Bigram sentence probability with Add-1 smoothing): Correct solution, complete and accurate.

```python
from IPython.display import Image, display

image_list = ["../data/output03.png", "../data/output04.png",
"../data/output05.png", "../data/output06.png",
"../data/output07.png",]

for img_path in image_list:
    display(Image(filename=img_path))
```





< Logs / Response

Input — Previous Responses — 337t

User

Given the earlier corpus, and a bigram language model (with add-1 smoothing) what is the probability that will be assigned to the following sentence? Give your answer in detailed steps with fractions, no worries for the final numerical result.

`<s>` some are good `</s>`

Output — 596t

Assistant

To calculate the probability of the sentence `<s> some are good </s>` using a bigram language model with add-1 smoothing, we need to calculate the smoothed probabilities of each bigram in the sentence and then multiply them together.

Given the corpus:

```
<s> some are fast </s>
<s> and some are slow </s>
<s> some are high </s>
<s> and some are low </s>
```

We will first calculate the smoothed probabilities of the bigrams in the sentence `<s> some are good </s>`.

1. Probability of `<s> some`:

Properties

Created	Apr 26, 2025, 3:27 PM
ID	resp_680d41a9d2648190a9a...
Model	gpt-3.5-turbo-0125
Tokens	933 total
Configuration	
Response	text
Max output tokens	2048

Metadata

No metadata entries

< Logs / Response

1. Probability of `<s> some`:
   Count(`<s> some`) = 2
   Count(`<s>`) = 4
   Probability = (Count(`<s> some`) + 1) / (Count(`<s>`) + V)
   V is the size of the vocabulary, and in this case, includes `<s>`, `</s>`, `some`, `and`, `are`, `fast`, `high`, `low` and `good`.
   Therefore, the probability of `<s> some` = (2 + 1) / (4 + 9)
2. Probability of `some are`:
   Count(`some are`) = 4
   Count(`some`) = 4
   Probability = (Count(`some are`) + 1) / (Count(`some`) + V)
   Therefore, the probability of `some are` = (4 + 1) / (4 + 9)
3. Probability of `are good`:
   Count(`are good`) = 0 (This bigram is not present in the corpus)
   Count(`are`) = 4
   Probability = (Count(`are good`) + 1) / (Count(`are`) + V)
   Therefore, the probability of `are good` = (0 + 1) / (4 + 9)
4. Probability of `good </s>`:
   Count(`good </s>`) = 0 (This bigram is not present in the corpus)
   Count(`good`) = 0 (The unigram 'good' doesn't appear in the corpus)
   Probability = (Count(`good </s>`) + 1) / (Count(`good`) + V)
   Therefore, the probability of `good </s>` = (0 + 1) / (0 + 9)

To find the probability of the entire sentence `<s> some are good </s>`, we multiply the probabilities of each consecutive bigram calculated above.

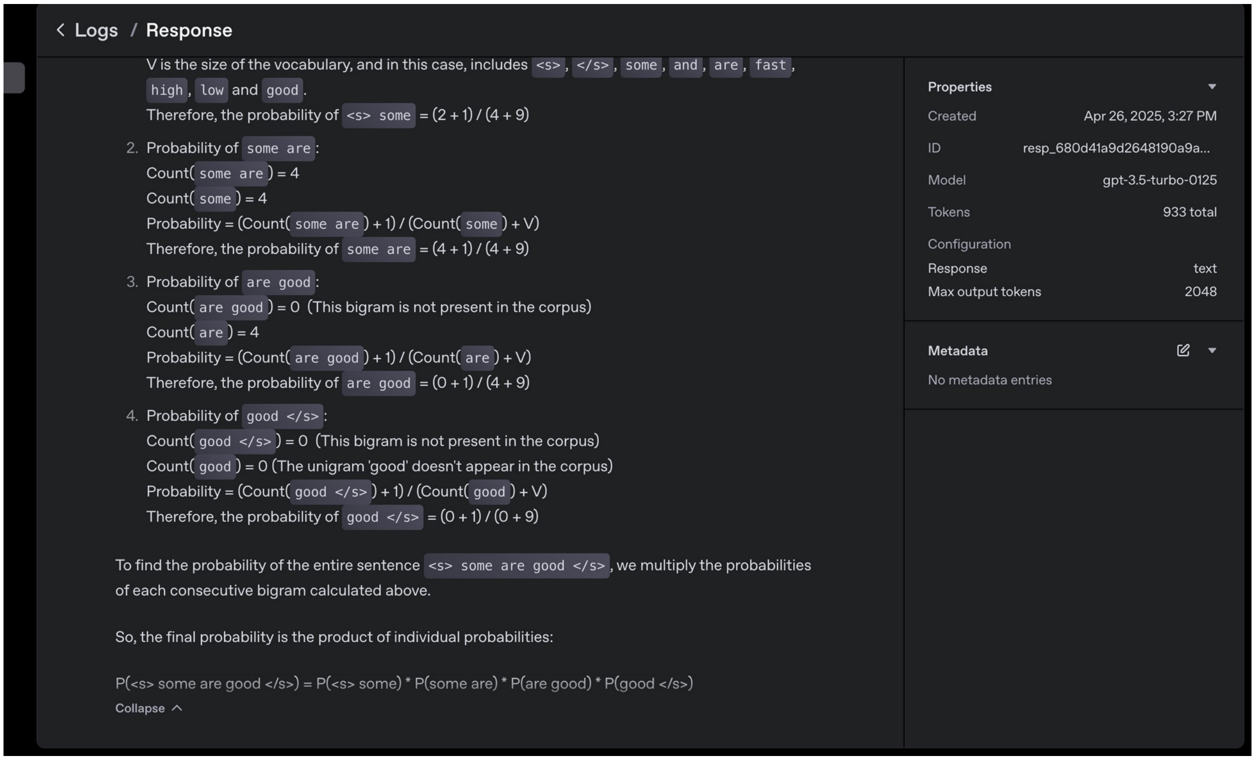
Collapse

Properties

Created	Apr 26, 2025, 3:27 PM
ID	resp_680d41a9d2648190a9a...
Model	gpt-3.5-turbo-0125
Tokens	933 total
Configuration	
Response	text
Max output tokens	2048

Metadata

No metadata entries

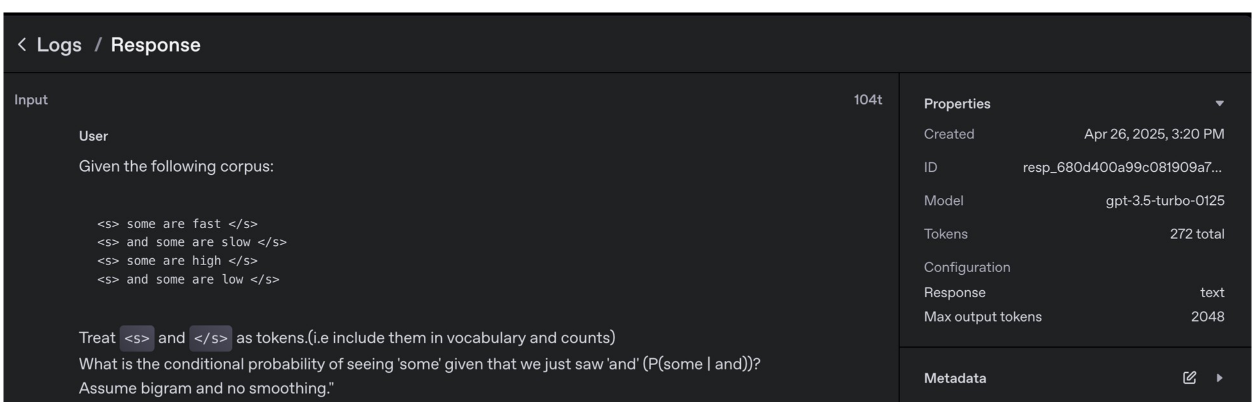


Second Attempt (Q3 + Q4 again):

Question 3: Correct, complete answer without issues.

Question 4: Failed to finish the solution fully at one go — stopped midway, incomplete response.

```python
from IPython.display import Image, display

image_list = ["../data/output03.png", "../data/output04.png",
"../data/output08.png", "../data/output09.png"]

for img_path in image_list:
    display(Image(filename=img_path))
```



Treat `<s>` and `</s>` as tokens.(i.e include them in vocabulary and counts)
What is the conditional probability of seeing 'some' given that we just saw 'and' (P(some | and))?
Assume bigram and no smoothing."

ID	resp_680d400a99c081909a7...
Model	gpt-3.5-turbo-0125
Tokens	272 total
Configuration	
Response	text
Max output tokens	2048

Metadata

Output 168t

Assistant

To calculate the conditional probability of seeing 'some' given that we just saw 'and' without smoothing, we need to consider the count of the bigram 'and some' and the count of the unigram 'and'.

Given the corpus:

```
<s> some are fast </s>
<s> and some are slow </s>
<s> some are high </s>
<s> and some are low </s>
```

Count of bigram 'and some' = 2
Count of unigram 'and' = 2

Conditional probability P(some | and) = Count('and some') / Count('and')
P(some | and) = 2 / 2 = 1

Therefore, the conditional probability of seeing 'some' given that we just saw 'and' is 1

Logs / Response

Input ^ Previous Responses 410t

User

Given the earlier corpus, and a bigram language model (with add-1 smoothing) what is the probability that will be assigned to the following sentence? Give your answer in detailed steps with fractions, no worries for the final numerical result.
<s> some are good </s>

Properties

Created	Apr 26, 2025, 4:35 PM
ID	resp_680d51b0534c8190abb...
Model	gpt-3.5-turbo-0125
Tokens	908 total
Configuration	
Response	text
Max output tokens	2048

Metadata

No metadata entries

Output 498t

Assistant

To calculate the probability assigned to the sentence "<s> some are good </s>" using a bigram language model with add-1 smoothing based on the given corpus, we need to consider the occurrence counts of the relevant bigrams and unigrams in the corpus.

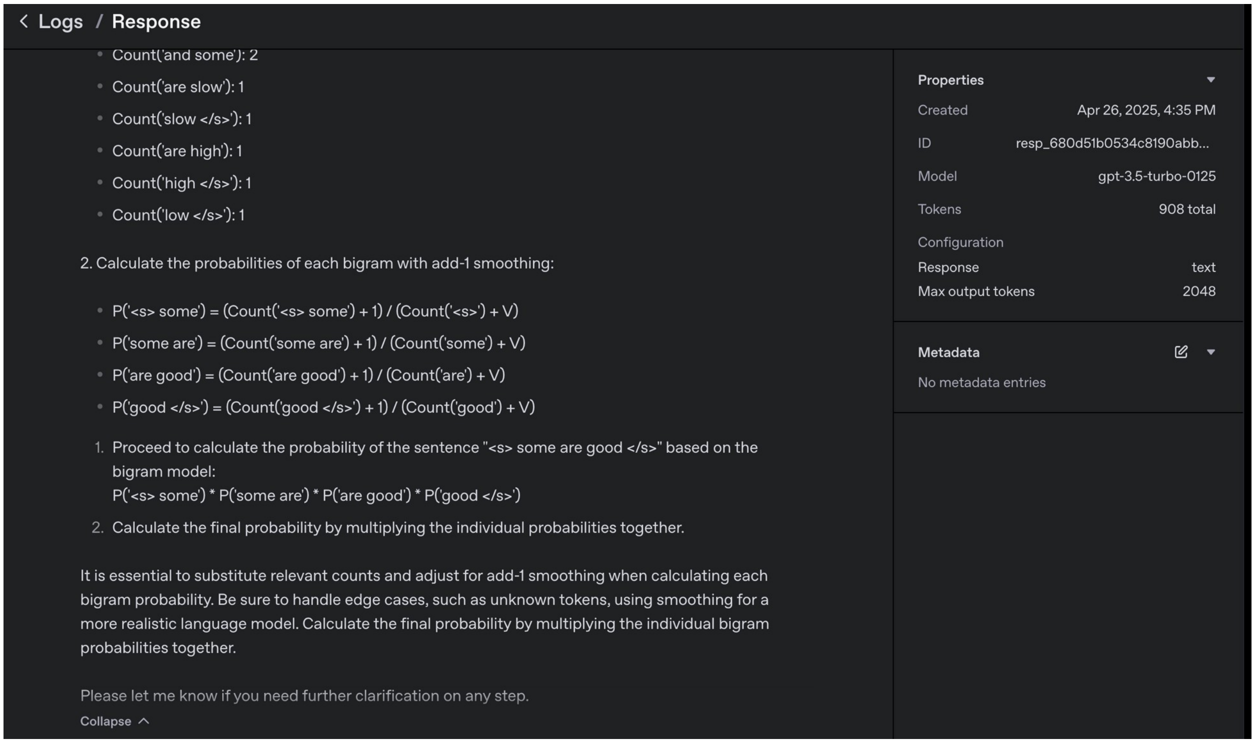
The given corpus is:
<s> some are fast </s> <s> and some are slow </s> <s> some are high </s> <s> and some are low </s>

First, we need to calculate the probabilities for each bigram based on the given counts and apply add-1 smoothing.

1. Calculate the counts of all bigrams in the corpus:

- Count('<s> some'): 3
- Count('some are'): 4
- Count('are fast'): 1
- Count('fast </s>'): 1

Collapse ^



< Logs / Response

- Count('and some'): 2
- Count('are slow'): 1
- Count('slow </s>'): 1
- Count('are high'): 1
- Count('high </s>'): 1
- Count('low </s>'): 1

2. Calculate the probabilities of each bigram with add-1 smoothing:

- P('<s> some') = (Count('<s> some') + 1) / (Count('<s>') + V)
- P('some are') = (Count('some are') + 1) / (Count('some') + V)
- P('are good') = (Count('are good') + 1) / (Count('are') + V)
- P('good </s>') = (Count('good </s>') + 1) / (Count('good') + V)

1. Proceed to calculate the probability of the sentence "<s> some are good </s>" based on the bigram model:
P('<s> some') * P('some are') * P('are good') * P('good </s>')
2. Calculate the final probability by multiplying the individual probabilities together.

It is essential to substitute relevant counts and adjust for add-1 smoothing when calculating each bigram probability. Be sure to handle edge cases, such as unknown tokens, using smoothing for a more realistic language model. Calculate the final probability by multiplying the individual bigram probabilities together.

Please let me know if you need further clarification on any step.

Collapse

Properties

Created	Apr 26, 2025, 4:35 PM
ID	resp_680d51b0534c8190abb...
Model	gpt-3.5-turbo-0125
Tokens	908 total
Configuration	
Response	text
Max output tokens	2048

Metadata

No metadata entries

Observations with GPT-4o (Playground):

First Attempt (Q3 + Q4 together):

□ Question 3 (Bigram P(some | and)): Correct solution, complete and accurate.

□ Question 4: Could produce complete steps (i.e., it didn't cut off), BUT the answer was wrong — vocabulary count errors, probability errors (like incorrect fractions or wrong vocabulary size).

```python
from IPython.display import Image, display

image_list = ["../data/output10.png", "../data/output11.png",
"../data/output12.png", "../data/output13.png"
              ,"../data/output14.png","../data/output15.png",]

for img_path in image_list:
    display(Image(filename=img_path))
```

Logs / Response

Input 102t

User

Given the following corpus:
<s> some are fast </s> <s> and some are slow </s> <s> some are high </s> <s> and some are low </s>
Treat <s> and </s> as tokens.(i.e include them in vocabulary and counts) What is the conditional probability of seeing 'some' given that we just saw 'and' (P(some | and))? Assume bigram and no smoothing."

Output 239t

Assistant

To calculate the conditional probability (P(\text{some} \mid \text{and})) using bigram probabilities, we need to determine how often the sequence "and some" occurs and how often "and" occurs as a bigram in the given corpus.

The corpus is:

- <s> some are fast </s>
- <s> and some are slow </s>
- <s> some are high </s>
- <s> and some are low </s>

Properties

Created	Apr 26, 2025, 4:00 PM
ID	resp_680d496bea448190bef...
Model	gpt-4o-2024-08-06
Tokens	341 total
Configuration	
Response	text
Max output tokens	2048

Metadata

No metadata entries

Logs / Response

bigram in the given corpus.

The corpus is:

- <s> some are fast </s>
- <s> and some are slow </s>
- <s> some are high </s>
- <s> and some are low </s>

First, list the bigrams involving "and":

- and some
- and some

Count how many times "and" appears:

- and some (2 times)

Count how many times the bigram "and some" appears:

- and some (2 times)

Now, calculate the conditional probability:
[P(\text{some} \mid \text{and}) = \frac{\text{Count(and some)}}{\text{Count(and)}} = \frac{2}{2} = 1]

So, the conditional probability (P(\text{some} \mid \text{and})) is 1.

Properties

Created	Apr 26, 2025, 4:00 PM
ID	resp_680d496bea448190bef...
Model	gpt-4o-2024-08-06
Tokens	341 total
Configuration	
Response	text
Max output tokens	2048

Metadata

No metadata entries

Logs / Response

Input

Previous Responses

404t

User

Given the earlier corpus, and a bigram language model (with add-1 smoothing) what is the probability that will be assigned to the following sentence? Give your answer in detailed steps with fractions, no worries for the final numerical result.
<s> some are good </s>

Output

879t

Assistant

To calculate the probability of the sentence "<s> some are good </s>" using a bigram model with add-1 smoothing, we need to:

1. Count the occurrences of the relevant bigrams in the corpus.
2. Count the occurrences of each unigram.
3. Calculate the smoothed probabilities.

Step 1: Count Occurrences

Corpus:

- <s> some are fast </s>
- <s> and some are slow </s>
- <s> some are high </s>
- <s> and some are low </s>

Properties

Created Apr 26, 2025, 4:00 PM

ID resp_680d498ab6948190bcf...

Model gpt-4o-2024-08-06

Tokens 1,283 total

Configuration

Response text

Max output tokens 2048

Metadata

No metadata entries

Vocabulary: <s>, </s>, some, are, fast, and, slow, high, low, good
(Vocabulary size = 10)

Unigram Counts:

- <s>: 4
- some: 4
- are: 4
- fast: 1
- and: 2
- slow: 1
- high: 1
- low: 1
- </s>: 4
- good: 0 (Note: Good is not seen in the corpus)

Bigram Counts:

- (<s>, some): 2
- (some, are): 3
- (are, fast): 1
- (are, slow): 1
- (are, high): 1

Collapse

Properties

Created Apr 26, 2025, 4:00 PM

ID resp_680d498ab6948190bcf...

Model gpt-4o-2024-08-06

Tokens 1,283 total

Configuration

Response text

Max output tokens 2048

Metadata

No metadata entries

Use add-1 smoothing (Laplace smoothing), where you add 1 to each count and increase the vocabulary size by 1 for calculations.

1. **P(some | <s>)**
   [
   P(\text{some} \mid \text{<s>}) = \frac{\text{Count}(\text{<s>, some}) + 1}{\text{Count}(\text{<s>}) + V} = \frac{2 + 1}{4 + 10} = \frac{3}{14}
   ]
2. **P(are | some)**
   [
   P(\text{are} \mid \text{some}) = \frac{\text{Count}(\text{some, are}) + 1}{\text{Count}(\text{some}) + V} = \frac{3 + 1}{4 + 10} = \frac{4}{14}
   ]
3. **P(good | are)**
   [
   P(\text{good} \mid \text{are}) = \frac{\text{Count}(\text{are, good}) + 1}{\text{Count}(\text{are}) + V} = \frac{0 + 1}{4 + 10} = \frac{1}{14}
   ]
4. **P(</s> | good)**
   [
   P(\text{</s>} \mid \text{good}) = \frac{\text{Count}(\text{good, </s>}) + 1}{\text{Count}(\text{good}) + V} = \frac{0 + 1}{0 + 10} = \frac{1}{10}
   ]

Step 3: Calculate Overall Probability

Collapse

Properties

Created Apr 26, 2025, 4:00 PM

ID resp_680d498ab6948190bcf...

Model gpt-4o-2024-08-06

Tokens 1,283 total

Configuration

Response text

Max output tokens 2048

Metadata

No metadata entries

Logs / Response

+ V} = \frac{3 + 1}{4 + 10} = \frac{4}{14}
]

3. **P(good | are)**
   [
   P(\text{good} \mid \text{are}) = \frac{\text{Count}(\text{are, good}) + 1}{\text{Count}(\text{are}) + V} = \frac{0 + 1}{4 + 10} = \frac{1}{14}
   ]
4. **P(</s> | good)**
   [
   P(\text{</s>} \mid \text{good}) = \frac{\text{Count}(\text{good, </s>}) + 1}{\text{Count}(\text{good}) + V} = \frac{0 + 1}{0 + 10} = \frac{1}{10}
   ]

Step 3: Calculate Overall Probability

Combine the smoothed probabilities:
[
P(\text{<s> some are good </s>}) = P(\text{some} \mid \text{<s>}) \times P(\text{are} \mid \text{some}) \times P(\text{good} \mid \text{are}) \times P(\text{</s>} \mid \text{good})
]

[
= \frac{3}{14} \times \frac{4}{14} \times \frac{1}{14} \times \frac{1}{10}
]

This fraction represents the probability assigned to the sentence by the bigram model with add-1 smoothing.

Collapse

Properties

Created Apr 26, 2025, 4:00 PM

ID resp_680d498ab6948190bcf...

Model gpt-4o-2024-08-06

Tokens 1,283 total

Configuration

Response text

Max output tokens 2048

Metadata

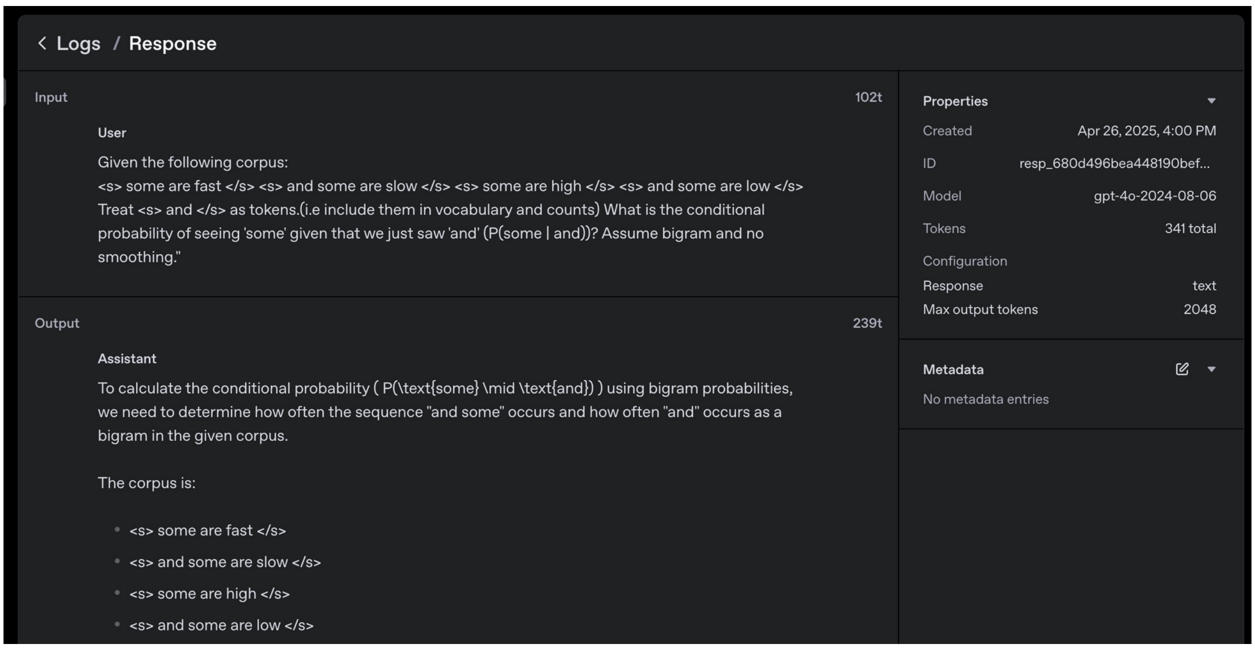
No metadata entries

Second Attempt (Q3 + Q4 together):

Question 3 (Bigram P(some | and)): Correct solution, complete and accurate.

□ Question 4: Could produce complete steps (i.e., it didn't cut off), BUT the answer was wrong — vocabulary count errors, probability errors (like incorrect fractions or wrong vocabulary size).

```
from IPython.display import Image, display

image_list = ["../data/output10.png", "../data/output11.png",
"../data/output16.png", "../data/output17.png"
              ,"../data/output18.png","../data/output19.png",]

for img_path in image_list:
    display(Image(filename=img_path))
```



Logs / Response

bigram in the given corpus.

The corpus is:

- <s> some are fast </s>
- <s> and some are slow </s>
- <s> some are high </s>
- <s> and some are low </s>

First, list the bigrams involving "and":

- and some
- and some

Count how many times "and" appears:

- and some (2 times)

Count how many times the bigram "and some" appears:

- and some (2 times)

Now, calculate the conditional probability:
[P(\text{some} \mid \text{and}) = \frac{\text{Count(and some)}}{\text{Count(and)}} = \frac{2}{2} = 1]

So, the conditional probability (P(\text{some} \mid \text{and})) is 1.

Properties

Created	Apr 26, 2025, 4:00 PM
ID	resp_680d496bea448190bef...
Model	gpt-4o-2024-08-06
Tokens	341 total
Configuration	
Response	text
Max output tokens	2048

Metadata

No metadata entries

Logs / Response

User

Given the earlier corpus, and a bigram language model (with add-1 smoothing) what is the probability that will be assigned to the following sentence? Give your answer in detailed steps with fractions, no worries for the final numerical result.
<s> some are good </s>

Output 812t

Assistant

To calculate the probability of the sentence `<s> some are good </s>` using a bigram language model with add-1 smoothing, follow these steps:

Step 1: Identify the Vocabulary

The vocabulary includes all unique tokens from the corpus:

- `<s>`, `some`, `are`, `fast`, `</s>`, `and`, `slow`, `high`, `low`

The vocabulary size (V) is 9.

Step 2: Calculate Bigram Probabilities with Add-1 Smoothing

For add-1 smoothing, the formula for the probability of a word (w_i) given the previous word (w_{i-1}) is:

[
P(w_i \mid w_{i-1}) = \frac{\text{Count}(w_{i-1}, w_i) + 1}{\text{Count}(w_{i-1}) + V}

Collapse

Properties

Created	Apr 26, 2025, 4:38 PM
ID	resp_680d5242363c81909a9...
Model	gpt-4o-2024-08-06
Tokens	1,272 total
Configuration	
Response	text
Max output tokens	2048

Metadata

No metadata entries

Logs / Response

Step 2: Calculate Bigram Probabilities with Add-1 Smoothing

For add-1 smoothing, the formula for the probability of a word (w_i) given the previous word (w_{i-1}) is:

[
P(w_i \mid w_{i-1}) = \frac{\text{Count}(w_{i-1}, w_i) + 1}{\text{Count}(w_{i-1}) + V}
]

Extract Relevant Counts:

- Count(`<s> some`): 2
- Count(`some are`): 2
- Count(`are fast`): 1
- Count(`are slow`): 1
- Count(`<s>`): 2
- Count(`some`): 4
- Count(`are`): 2

Step 3: Compute Probabilities

1. **P(some | <s>)**
   [
   P(\text{some} \mid <s>) = \frac{\text{Count}(<s>, \text{some}) + 1}{\text{Count}(<s>) + V} = \frac{2 + 1}{2 + 9} = \frac{3}{11}
   ]
2. P(are | some)

Collapse

Properties	
Created	Apr 26, 2025, 4:38 PM
ID	resp_680d5242363c81909a9...
Model	gpt-4o-2024-08-06
Tokens	1,272 total
Configuration	
Response	text
Max output tokens	2048

Metadata

No metadata entries

Logs / Response

Step 3: Compute Probabilities

1. **P(some | <s>)**
   [
   P(\text{some} \mid <s>) = \frac{\text{Count}(<s>, \text{some}) + 1}{\text{Count}(<s>) + V} = \frac{2 + 1}{2 + 9} = \frac{3}{11}
   ]
2. **P(are | some)**
   [
   P(\text{are} \mid \text{some}) = \frac{\text{Count}(\text{some}, \text{are}) + 1}{\text{Count}(\text{some}) + V} = \frac{2 + 1}{4 + 9} = \frac{3}{13}
   ]
3. **P(good | are)**
   [
   P(\text{good} \mid \text{are}) = \frac{\text{Count}(\text{are}, \text{good}) + 1}{\text{Count}(\text{are}) + V} = \frac{0 + 1}{2 + 9} = \frac{1}{11}
   ]
4. **P(</s> | good)**
   [
   P(</s> \mid \text{good}) = \frac{\text{Count}(\text{good}, </s>) + 1}{\text{Count}(\text{good}) + V} = \frac{0 + 1}{0 + 9} = \frac{1}{9}
   ]

Step 4: Multiply Probabilities

Finally, the probability of the sentence is the product of these probabilities:

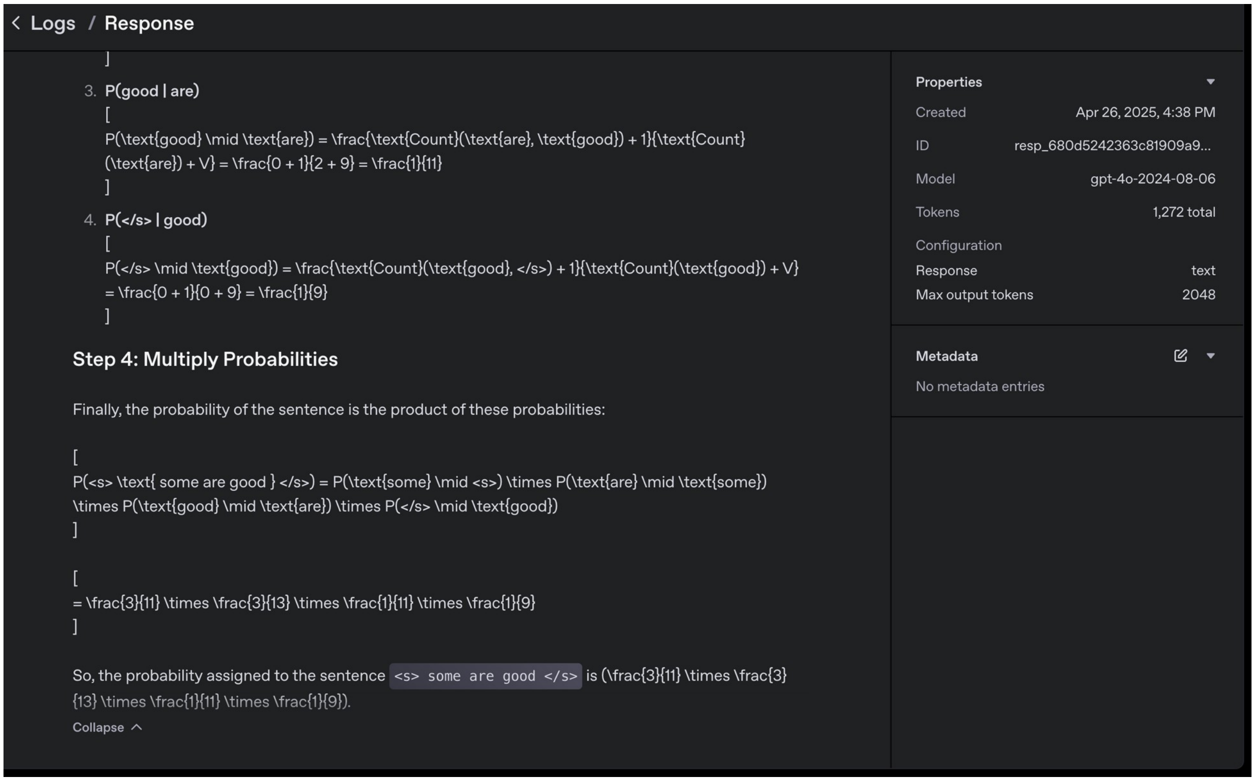
[

Collapse

Properties	
Created	Apr 26, 2025, 4:38 PM
ID	resp_680d5242363c81909a9...
Model	gpt-4o-2024-08-06
Tokens	1,272 total
Configuration	
Response	text
Max output tokens	2048

Metadata

No metadata entries



- Even after many retries, GPT-4o couldn’t consistently give the correct answer for Q4.

⚡ My Additional Insight:

Ironically, **ChatGPT.com (web app)** gave correct result (for Q4) for the first attempt,than the Playground/ API testing.

However, GPT-4o via Playground was unstable for this task for question 4.

Section 3.3. Let's Build Our Own NGram Agent with LangChain

Use the Exam_Agent_For_NLP.ipynb in the notebooks folder, build your own agent. Step 1. Test through the code, see only using more atomic tools such as "extract_text","count_occurrences", "divid", "multiply", see if "GPT-3.5-turbo" will use help. Step 2. Could you reuse the assignment 1 to fill in the unimplemented bigram-related functions, see if your agent could use the new advanced tools.

[Hint: The "GPT-3.5-turbo" is not perfect and may not stable, so please just report you find]

What to turn in:

1. The notebook itself
2. Writing your finding into the single seperate report you need to turn in.

Step 1: Testing with Atomic Tools Only (Basic Functions)

I tested the agent using only the provided basic tools:

extract_text() — for OCR extraction from the exam screenshot.

count_occurrences() — to count bigram/unigram frequencies.

divide() and multiply() — for basic arithmetic operations.

Model Used: GPT-3.5-turbo (via LangChain integration).

Observation:

The agent was able to successfully extract the text and attempt reasoning and also did the basic arithmetic operations.

Step 2: Reusing Bigram Functions from Assignment 1 (Advanced Tools)

I implemented the missing bigram-related functions:

bigram_condition_prob_in_corpus() — for conditional probability (no smoothing).

bigram_prob() — bigram probability without smoothing.

bigram_prob_smooth() — bigram probability with Add-1 smoothing.

These functions were reused and adapted from my Assignment 1 work.

After integrating these tools with the agent, the system was able to:

Correctly calculate P(some | and) for Question 3.

Attempt the smoothed probability for Question 4, though GPT-3.5-turbo still struggled at times to invoke the correct sequence of tools reliably and could not solve the question 4.