HOMEWORK 0
INTRO[1]

CS 678 ADVANCED NATURAL LANGUAGE PROCESSING (SPRING 2023)
https://nlp.cs.gmu.edu/course/cs678-spring23/

OUT: January 23, 2023
DUE: February 2, 2023

Based on an assignment created by Mohit Iyyer.

Your name: Abhishek Jallawaram

Your GID: G01373042

IMPORTANT: The homework is accompanied by a python notebook. After copying this notebook to your Google Drive or One 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. Alternatively, you can upload the completed .ipynb file along with your completed .pdf report to Gradescope.
If you fail to do this, you will receive no credit for this homework!

Your Notebook solution:
https://drive.google.com/file/d/1IZNXPJJ2NtJLFaF2ybrEoHpF2LP6Cap1/view?usp=share_li

Graded Questions: 100 points
Bonus Questions: 20 points
Total Points Available: 120/100

Additional Notes:

- Upload the whole report PDF (including all pages, also if you're scanning it).
- 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.

[1]Compiled on Friday 3rd February, 2023 at 17:38

Instructions for Specific Problem Types

For "Select One" questions, please fill in the appropriate bubble completely:

Select One: Who is teaching this course?

- ● Antonios Anastasopoulos and Sina Ahmadi
- ○ Marie Curie
- ○ Noam Chomsky

If you need to change your answer, you may cross out the previous answer and bubble in the new answer:

Select One: Who is teaching this course?

- ● Antonios Anastasopoulos and Sina Ahmadi
- ○ Marie Curie
- ~~●~~ Noam Chomsky

For "Select all that apply" questions, please fill in all appropriate squares completely:

Select all that apply: Which are scientists?

- ■ Stephen Hawking
- ■ Albert Einstein
- □ Isaac Newton
- □ None of the above

For questions where you must fill in a blank, please make sure your final answer is fully included in the given space. You may cross out answers or parts of answers, but the final answer must still be within the given space.

Fill in the blank: What is the course number?

$\boxed{678}$ $\boxed{\cancel{876}\ 678}$

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` in the accompanying python notebook.

1 Course Policies [6 pts]

In this section, you will work through a number of problems covering prerequisite and introductory material.The first subsection covers common course policy questions.

This section covers important course policies that every student should know and understand. These questions MUST be finished in order for the whole homework to be considered for grading.

1. (1 point) How many grace days do you have in total for all homework? Can you combine grace days with late days to extend a homework submission deadline by more than 4 days?

   Select one:

   ○ As many as I want; Of course!

   ● 4; No

   ○ 5; Yes

   ○ 8; Yes

2. (2 points) Seeking help from other students in understanding course materials needed to solve homework problems is ALLOWED under which of the follow conditions?

   Select all that apply:

   ■ Any written notes are taken on an impermanent surface (e.g. whiteboard, chalkboard) and discarded before writing up one's solution alone.

   ■ Learning is facilitated not circumvented; i.e., the purpose of seeking help is to learn and understand the problem instead of merely getting an answer

   ■ Help both given and received is reported in collaboration questions in the homework

   □ The student updates their collaborative questions even if it is after submitting their own assignment

   □ None of the above

3. (1 point) Which of the following is (are) strictly forbidden in solving and submitting homework?

   Select all that apply:

   - ☐ Searching on the internet for solutions or sample codes
   - ■ Consulting people outside this class who have seen or solved the problem before
   - ■ Turning in someone else's homework
   - ■ Using anyone else's, or allowing other classmates to use your computer or Gradescope account in connection with this course
   - ☐ None of the above

4. (1 point) If you solved your assignment completely on your own, you can skip the collaboration questions at the end of each homework.

   Select one:

   - ○ True
   - [✓ ● False]

5. (1 point) Assume a difficult situation arises in the middle of the semester (e.g. medical, personal etc.) that might prevent you from submitting assignments on time or working as well as you would like. What should you do? Select all that apply

   - ■ Talk to the course staff early so they can point you to the available resources on campus and make necessary arrangements
   - ☐ Do not speak to the course staff, try to finish the class, reach out to the course staff in the end of the semester explaining your special situation
   - ■ Reach out to your academic advisor so that they are aware of the situation
   - ☐ None of the above

2 Probability Basics [24 pts]

This and the following section use the paired python notebook. Below, provide the answers to the questions as generated by the completed codeblocks in the similarly marked questions.

6. (10 points) Let's begin with a quick probability review. In the task of language modeling, we're interested in computing the **joint** probability of some text. Say we have a sentence s with n words $(w_1, w_2, w_3, \ldots, w_n)$ and we want to compute the joint probability $P(w_1, w_2, w_3, \ldots, w_n)$. Assume we are given a model that produces the conditional probability of the next word in a sentence given all preceding words: $P(w_i|w_1, w_2, \ldots, w_{i-1})$. How can we use this model to compute the joint probability of sentence s?

We can use the conditional probability of the next word given all preceding words to compute the joint probability of a sentence by using the chain rule of probability. The chain rule states that the joint probability of a sequence of events can be written as the product of their individual conditional probabilities.

In the case of a sentence with n words, we can compute the joint probability of the sentence using the following formula:

$P(w_1, w_2, w_3, \ldots, w_n)$= $P(w_1)$ * $P(w_2|w_1)$ *$P(w_3|w_1, w_2)$ * . . . * $P(w_n|w_1, w_2, w_3, ..., w_{n-1})$

Where $P(w_1, w_2, w_3, \ldots, w_{i-1})$ is the conditional probability of the next word in the sentence given all preceding words and $P(w_1)$ is the probability of the first word in the sentence.

7. (10 points) Why would we ever want to compute the joint probability of a sentence? Provide two different reasons why this probability might be useful to solve an NLP task.

   1. Language Model Evaluation: The joint probability of a sentence can be used to assess the proficiency of a language model. A language model is designed to predict the next word in a sentence based on the occurence of its previous words. The joint probability of the sentence can used as a reference indicator of how accurately the model can predict the sentence. The higher the joint probability, the more probable the sentence is based on the model, thus indicating a more accurate prediction and a better performing model.
   2. Sentence Generation: The joint probability of a sentence can also be employed in producing new sentences that resemble a specified sentence. By determining the joint probability of a sentence, we can determine the probable words that will come after a particular sequence of words, and utilize these probabilities to generate new sentences that resemble the original sentence.

8. (4 points) Here is a simple way to build a language model: for any prefix $w_1, w_2, \ldots, w_{i-1}$, retrieve all occurrences of that prefix in some huge text corpus (such as the Common Crawl[2]) and keep count of the word w_i that follows each occurrence. We can then use this to estimate the conditional probability $P(w_i|w_1, w_2, \ldots, w_{i-1})$ for any prefix. Explain why this method is completely impractical!

   1. The Common Crawl corpus is huge and contains billions of words. Retrieving all occurrences of a given prefix in the corpus and counting the following words would require a significant amount of computational resources and storage space.
   2. The method is based on counting occurrences on a particular text corpus, it's not generalizable to other texts and will perform poorly on unseen data or new data.
   3. The number of occurrences of a particular prefix in the corpus may be very low, making it difficult to estimate the conditional probability $P(w_1, w_2, w_3, \ldots, w_{i-1})$ accurately.

[2] https://https://commoncrawl.org/

3 Language Modeling [15 pts]

9. (5 points) Let's switch over to coding! Coding cell 1 in Section 3 in the notebook contains the opening paragraph of Daphne du Maurier's novel "Rebecca". Write some code in this cell to compute the number of unique word **types** and total word **tokens** in this paragraph. Use a whitespace tokenizer to separate words (i.e., split the string on white space using Python's `split()` function).

Number of types

76 & 74
(including punctuation & excluding punctuation)

Number of tokens

111 & 100
(including punctuation & excluding punctuation)

10. (5 points) Now let's look at the most frequently used word types in this paragraph. Write some code in code cell 2 to print out the ten most frequently-occurring types. We have initialized a Counter object that you should use for this purpose. In general, Counters are very useful for text processing in Python.

Top-10 words and counts with punctuation

Word	Count	Word	Count
i	6	it	3
.	6	a	3
the	6	and	3
,	5	my	3
to	4	again	2

Top-10 words and counts without punctuation

Word	Count	Word	Count
i	6	and	3
the	6	my	3
to	4	again	2
it	3	that	2
a	3	was	2

11. (5 points) What do you notice about these words and their linguistic functions (i.e., parts-of-speech)? These words are known as "stopwords" in NLP and are often removed from the text before any computational modeling is done. Why do you think that is?

Stopwords are most commonly used words in a language that do not contribute much meaning to the text, such as "the," "an," "and," etc in the English language. The reason for removing stopwords is that they frequently appear in text and can dominate the statistical information that NLP models are trained on. This can lead to over-representation of stopwords in the model's feature space and negatively impact its performance on NLP tasks. Therefore, removing stopwords from text before computational modeling can help reduce the size of the feature space and make it more informative for NLP tasks. This can improve the performance of NLP models and make them more effective at solving various NLP problems.

4 Neural Language Modeling [35 pts]

12. (10 points) In *neural* language models, we represent words with low-dimensional vectors also called *embeddings*. We use these embeddings to compute a vector representation $\boldsymbol{x}$ of a given prefix, and then predict the probability of the next word conditioned on $\boldsymbol{x}$. In code cell 3, we use **PyTorch**[3], a machine learning framework, to explore this setup. You are provided embeddings for the prefix "Alice talked to"; your job is to combine them into a single vector representation $\boldsymbol{x}$ using *element-wise vector addition.*[4]

    TIP: if you're finding the PyTorch coding problems difficult, you may want to run through this 60 minutes blitz tutorial: `https://pytorch.org/tutorials/beginner/deep_learning_60min_blitz.html`.

    Copy below the resulting vectors, as produced by the `print()` statement at the end of the cell.

    Prefix Vector

-0.1770	-2.3993	-0.4721	2.6568	2.7157	-0.1408	-1.8421	-3.6277	2.2783	1.1165

13. (5 points) Modern language models do not use element-wise addition to combine the different word embeddings in the prefix into a single representation (a process called *composition*). What is a major issue with element-wise functions that makes them unsuitable for use as composition functions?

    Element-wise addition, are unsuitable for use as composition functions in modern language models because they do not take into account the relationships between the different word embeddings in the prefix. This means that they are unable to capture the meaning of the entire prefix as a single semantic representation. Element-wise addition would simply add the individual word embeddings together, regardless of their order, relationships, or context. This does not account for the fact that the meaning of a sentence can change based on the ordering of its words, or that the meaning of a word can change based on the context in which it is used.

14. (10 points) One very important function in neural language models (and for basically every task we'll look at this semester) is the **softmax**[5], which is defined over an n-dimensional vector $< x_1, x_2, \ldots, x_n >$ as $\text{softmax}(x_i) = \frac{e^{x_i}}{\sum_{1\le j\le n} e^{x_j}}$. Let's say we have our prefix representation $\boldsymbol{x}$ from before. We can use the softmax function, along with a linear projection using a matrix W, to go from $\boldsymbol{x}$ to a probability distribution p over the next word: $p = \text{softmax}(W\boldsymbol{x})$. You can explore this in code cell 4 in the notebook.

    Copy below the resulting probability distribution over the vocabulary, as produced by the `print()` statement at the end of code cell 4:

    Probability Distribution

0.2316	0.0035	0.1040	0.2055	0.4555

15. (8 points) So far, we have looked at just a single prefix ("Alice talked to"). In practice, it is common for us to compute many prefixes in one computation, as this enables us to take advantage of GPU parallelism and also obtain better gradient approximations (we'll talk more about the latter point later). This is called **batching**, where each prefix is an example in a larger batch. Here, you'll redo the

[3] `https://pytorch.org`

[4] You might find this helpful: `https://ml-cheatsheet.readthedocs.io/en/latest/linear_algebra.html#elementwise-operations`.

[5] `https://pytorch.org/docs/master/nn.functional.html#softmax`

computations from the previous questions/cells, but instead of having one prefix, you'll have a batch of two prefixes. The final output of this cell should be a 2×5 matrix that contains two probability distributions, one for each prefix. **NOTE: YOU WILL LOSE POINTS IF YOU USE ANY LOOPS IN YOUR ANSWER!** Your code should be completely vectorized (a few large computations is faster than many smaller ones).

Provide your solution by copying the code in code cell 5 below (it currently includes the provided code):

Probability Distribution

```
# for this problem, we'll just copy our old prefix over three times
# to form a batch. in practice, each example in the batch
# would be different.
batch_indices = torch.cat(2 * [indices]).reshape((2, 3))
batch_embs = embeddings(batch_indices)
#print(batch_embs)
print('batch embedding tensor size: ', batch_embs.size())

# now, follow the same procedure as before:
# step 1: compose each example's embeddings into a
# single representation using element-wise addition.
# HINT: check out the "dim" argument of the torch.sum function!
x = torch.sum(batch_embs, axis=1)
#print(x.size())

random_seed = 1
torch.manual_seed(random_seed)
W = torch.rand(10, 5)

# step 2: project each composed representation into
# a 5-d space using matrix W
# step 3: use the softmax function to obtain a 2x5 matrix
# with the probability distributions.
# Please store this probability matrix in the "batch_probs" variable.
batch_probs = torch.rand(2,5)
matrix_prod = torch.matmul(x,W)
batch_probs = torch.nn.functional.softmax(matrix_prod,dim = -1)

### DO NOT MODIFY THE BELOW LINE
print("batch probability distributions:", batch_probs)

### DO NOT MODIFY THE BELOW LINE
print("batch probability distributions:", batch_probs)
```

16. (2 points) Also copy below the resulting batch probability distribution over the vocabulary, as produced by the `print()` statement at the end of code cell 5:

Batch Probability Distribution

0.2316	0.0035	0.1040	0.2055	0.4555
0.2316	0.0035	0.1040	0.2055	0.4555

5 NLP Paper Reading [20 pts]

17. (20 points) Choose one paper from ACL 2022[6] that you find interesting. A good way to do this is by scanning the titles and abstracts; there are hundreds of papers so take your time before selecting one! Then, write a summary in your own words of the paper you chose. Your summary should answer the following questions: what is its motivation? Why should anyone care about it? Were there things in the paper that you didn't understand at all? What were they? Fill out the below answers, and make sure to write 2-4 paragraphs for the summary to receive full credit!

Paper Summary

Title of paper *DIBIMT: A Novel Benchmark for Measuring Word Sense Disambiguation Biases in Machine Translation.*

Authors *Niccolò Campolungo, Federico Martelli, Francesco Saina, Roberto Navigli*

Conference name ACL 2022

Paper URL https://aclanthology.org/2022.acl-long.298/

"DIBIMT: A Novel Benchmark for Measuring Word Sense Disambiguation Biases in Machine Translation" is a research paper that proposes a benchmark for evaluating the performance of machine translation systems with respect to word sense disambiguation. Word sense disambiguation is the task of identifying the correct sense of a word in context, which is critical for effective translation of natural language material. The authors strongly argue that existing machine translation system benchmarks do not effectively depict the complexity of the word sense disambiguation task and propose a novel standard, DIBIMT, to address this problem.

The proposed DIBIMT benchmark involves a complete evaluation of machine translation systems on a large-scale dataset, employing both intrinsic and extrinsic evaluation metrics. The intrinsic evaluation metrics assess the accuracy of machine translation systems' translations, whereas the extrinsic evaluation metrics assess the influence of the translations on the end-task. The authors demonstrate that DIBIMT provides a more accurate and thorough evaluation of machine translation systems than existing benchmarks since it incorporates for word sense disambiguation biases that might effect translation accuracy.

The authors describe the research results of DiBiMT experiments on various machine translation systems, indicating that the benchmark provides useful insights into the strengths and drawbacks of these systems in terms of word sense disambiguation. The findings illustrate the difficulties that machine translation systems encounter when dealing with word sense disambiguation and offer areas for development. The authors argue that their proposed benchmark can be used to improve machine translation systems that can handle word sense disambiguation more effectively, resulting in more accurate translations.

In conclusion, *"DIBIMT: A Novel Benchmark for Measuring Word Sense Disambiguation Biases in Machine Translation"* is an important contribution to the field of NLP, as it provides a comprehensive benchmark for evaluating the performance of machine translation systems in handling word sense disambiguation.The proposed benchmark provides important insights into the strengths and weaknesses of current machine translation systems, and can be used to guide the development of improved systems. The paper is a valuable resource for anyone interested in improving the accuracy of machine translation systems.

[6] https://aclanthology.org/events/acl-2022/

6 [Bonus] Understanding Waama [10 pts]

In the left column below appear sentences in Waama, or Yoabu, a Gur language of Benin spoken by roughly 50,000 people. These sentences appear in the writing system of the language. You do not need to know how the writing system's letters are pronounced to solve this problem. In the right column below, the translations of these sentences in English appear in a scrambled order.

1. *Cando dɛbite kpi, o ǹ faa o suka.*	A. The tree fell in the forest.
2. *Tando dori.*	B. A car passed by earlier.
3. *N pe saaki ti yete.*	C. I went to my friend's house.
4. *Bika kɔɔsi kɔɔka.*	D. The child fell.
5. *Soosada kaate.*	E. Marie lost the money, but she found it.
6. *Suka kpi.*	F. It rained.
7. *Ba kaate tiibu band.*	G. My hen went to Yooto's house.
8. *N yeentire n daaso.*	H. My wife swept our house.
9. *Bisu yɔkɔɔti.*	I. The children had fun.
10. *Tiibu dori puŋa mii.*	J. Tchando's neighbor died, and he inherited his car.
11. *N taka n daaso yete.*	K. They gathered under the tree.
12. *Maari dikitifa pei, o ǹ fa piisi.*	L. I hurt my friend.
13. *Suka miiki pɔmpɔmma.*	M. The soldiers assembled.
14. *Bika dori.*	N. The car broke down.
15. *N kɔɔka taka Yooto yete.*	O. The child sold the hen.

18. (4 points) Match each Waama sentence to their English translation:

Waama Translations

Waama sent:	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
English sent:	J	F	H	O	M	N	K	L	I	A	C	E	B	D	G

19. (2 points) One of the verbs you encountered in the sentences in the left column above would be used by speakers of this language in all 4 sentences below, each of which is translated into English. Which of the verbs you encountered above is that verb?

N tokore ***kpi.***	'My shirt is torn.'
Yaama ***kpi.***	'The matter is settled.'
O beere ***kpi.***	'He lost his fame.'
Yima ***kpi.***	'The water is frozen.'

The verb is

Verb

kpi

20. (3 points) Translate the following English sentences into Waama:
(a) The children gathered under the house.

Translation 1

Bisu kaate yete band.

(b) I sold my car.

Translation 2

N kɔɔsi n suka.

(c) Her friend played in the rain.

Translation 3

O daaso yɔkɔɔti tando mii.

21. (3 points) Translate the following Waama sentences into English:

(a) Maari daaso fa faa.

Translation 4

Maari's friend inherited it.

(b) N susu kpi.

Translation 5

My car broke down.

(c) Ba kɔɔsi ti kɔɔsu.

Translation 6

They sold our hens.

Collaboration Questions Please answer the following:

1. Did you receive any help whatsoever from anyone in solving this assignment?
   Yes /No.
   No
   - If you answered 'yes', give full details: ____________________
   - (e.g. "Jane Doe explained to me what is asked in Question 3.4")
2. Did you give any help whatsoever to anyone in solving this assignment?
   Yes / No.
   No
   - If you answered 'yes', give full details: ____________________
   - (e.g. "I pointed Joe Smith to section 2.3 since he didn't know how to proceed with Question 2")
3. Did you find or come across code that implements any part of this assignment ?
   Yes / No.
   No
   (See below policy on "found code")
   - If you answered 'yes', give full details: ____________________
   - (book & page, URL & location within the page, etc.).

Homework 0, CS678 Spring 2023

This is due on January 30th, 2022, to be submitted via Gradescope as a PDF (File>Print>Save as PDF). 100 points total.

IMPORTANT: After copying this notebook to a Google Drive or One 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: paste your link here

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`.
- This assignment is designed so that you can run all cells almost instantly. If it is taking longer than that, you have made a mistake in your code.
- Note that there are more questions in the PDF than the ones present in this notebook (which only includes the ones requiring code).

How to submit this problem set:

- After filling in the missing code, provide all the answers in LaTeX template released with the assignment. Once you are finished with this notebook, share it on Blackboard. You can also generate a PDF via (File -> Print -> Save as PDF) and upload it to Gradescope as supplementary material.

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.

Section 3

Question 10 (5 points)

Let's switch over to coding! The below coding cell contains the opening paragraph of Daphne du Maurier's novel *Rebecca*. Write some code in this cell to compute the number of unique word **types** and total word **tokens** in this paragraph (watch the lecture videos if you're confused about what these terms mean!). Use a whitespace tokenizer to separate words (i.e., split the string on white space using Python's split function). Be sure that the cell's output is visible in the PDF file you turn in on Gradescope.

```python
paragraph = '''Last night I dreamed I went to Manderley again. It seemed to me
that I was passing through the iron gates that led to the driveway.
The drive was just a narrow track now, its stony surface covered
with grass and weeds. Sometimes, when I thought I had lost it, it
would appear again, beneath a fallen tree or beyond a muddy pool
formed by the winter rains. The trees had thrown out new
low branches which stretched across my way. I came to the house
suddenly, and stood there with my heart beating fast and tears
filling my eyes.'''.lower() # lowercase normalization is often useful in NLP

types = 0
tokens = 0

paragraph = paragraph.replace(',',"  ")
paragraph = paragraph.replace('.',"  ")

#paragraph = paragraph.replace(','," , ")
#paragraph = paragraph.replace('.'," . ")

total_tokens = paragraph.split()

#t1 = set(total_tokens)

#print(t1)
#print(len(t1))

types = len(set(total_tokens))
tokens = len(total_tokens)

# YOUR CODE HERE! POPULATE THE types AND tokens VARIABLES WITH THE CORRECT VALUE


# DO NOT MODIFY THE BELOW LINE!
print('Number of word types: %d, number of word tokens:%d' % (types, tokens))
```

```
Number of word types: 74, number of word tokens:100
```

Question 11 (5 points)

Now let's look at the most frequently used word **types** in this paragraph. Write some code in the below cell to print out the ten most frequently-occurring types. We have initialized a Counter object that you should use for this purpose. In general, Counters are very useful for text processing in Python.

```python
from collections import Counter
c = Counter()

# YOUR CODE HERE! Use the counter over the above paragraph

for word in total_tokens: c[word] += 1

# DO NOT MODIFY THE BELOW LINES!
for word, count in c.most_common()[:10]:
    print(word, count)
```

```
i 6
the 6
to 4
it 3
a 3
and 3
my 3
again 2
that 2
was 2
```

Section 4

Question 13 (10 points)

In *neural* language models, we represent words with low-dimensional vectors also called *embeddings*. We use these embeddings to compute a vector representation $\boldsymbol{x}$ of a given prefix, and then predict the probability of the next word conditioned on $\boldsymbol{x}$. In the below cell, we use PyTorch, a machine learning framework, to explore this setup. We provide embeddings for the prefix "Alice talked to"; your job is to combine them into a single vector representation $\boldsymbol{x}$ using element-wise vector addition.

TIP: if you're finding the PyTorch coding problems difficult, you may want to run through the 60 minutes blitz tutorial!

```
import torch
torch.set_printoptions(sci_mode=False)
torch.manual_seed(0)

prefix = 'Alice talked to'

# spend some time understanding this code / reading relevant documentation!
# this is a toy problem with a 5 word vocabulary and 10-d embeddings
embeddings = torch.nn.Embedding(num_embeddings=5, embedding_dim=10)
# we define the vocabulary by hand below (since this is a toy problem)
vocab = {'Alice':0, 'talked':1, 'to':2, 'Bob':3, '.':4}

# we need to encode our prefix as integer indices (not words) that index
# into the embeddings matrix. the below line accomplishes this.
# note that PyTorch inputs are always Tensor objects, so we need
# to create a LongTensor out of our list of indices first.
indices = torch.LongTensor([vocab[w] for w in prefix.split()])
prefix_embs = embeddings(indices)
print('prefix embedding tensor size: ', prefix_embs.size())

# okay! we now have three embeddings corresponding to each of the three
# words in the prefix. write some code that adds them element-wise to obtain
# a representation of the prefix! store your answer in a variable named "x".

#print(prefix_embs)
### YOUR CODE HERE!
x = torch.rand(10)
#Returns the sum of all elements in the input tensor.
#dim (int or tuple of ints, optional) – the dimension or dimensions to reduce. I
x = torch.sum(prefix_embs,dim = 0)

### DO NOT MODIFY THE BELOW LINE
print('embedding sum: ', x)

```

```
prefix embedding tensor size:  torch.Size([3, 10])
embedding sum:  tensor([-0.1770, -2.3993, -0.4721,  2.6568,  2.7157, -0.1408,
         2.2783,  1.1165], grad_fn=<SumBackward1>)
```

Question 15 (10 points)

One very important function in neural language models (and for basically every task we'll look at this semester) is the softmax, which is defined over an n-dimensional vector $< x_1, x_2, \ldots, x_n >$ as $\text{softmax}(x_i) = \frac{e^{x_i}}{\sum_{1 \leq j \leq n} e^{x_j}}$. Let's say we have our prefix representation $\boldsymbol{x}$ from before. We can use the softmax function, along with a linear projection using a matrix W, to go from $\boldsymbol{x}$ to a probability distribution p over the next word: $p = \text{softmax}(W\boldsymbol{x})$.Let's explore this in the code cell below:

```
# remember, our goal is to produce a probability distribution over the
# next word, conditioned on the prefix representation x. This distribution
# is thus over the entire vocabulary (i.e., it is a 5-dimensional vector).
# take a look at the dimensionality of x, and you'll notice that it is a
# 10-dimensional vector. first, we need to **project** this representation
# down to 5-d. We'll do this using the below matrix:
# prefix1 = 'Alice talked to Bob .'
# indices1 = torch.LongTensor([vocab[w] for w in prefix1.split()])

#Generates the same random uniform distribution for W
random_seed = 1
torch.manual_seed(random_seed)
W = torch.rand(10, 5)
#print(W)


# use this matrix to project x to a 5-d space, and then
# use the softmax function to convert it to a probability distribution.
# this will involve using PyTorch to compute a matrix/vector product.
# look through the documentation if you're confused (torch.nn.functional.softmax
# please store your final probability distribution in the "probs" variable.

### YOUR CODE HERE
probs = torch.rand(5)
#print(probs)

#Matrix product of two tensors.

y = torch.matmul(x,W)

#Softmax function
#torch.nn.functional.softmax(input, dim=None, _stacklevel=3, dtype=None)
probs = torch.nn.functional.softmax(y, dim=-1)

### DO NOT MODIFY THE BELOW LINE!
print('probability distribution', probs)

```

```
probability distribution tensor([0.2316, 0.0035, 0.1040, 0.2055, 0.4555], grac
```

Questions 16 and 17 (10 points)

So far, we have looked at just a single prefix ("Alice talked to"). In practice, it is common for us to compute many prefixes in one computation, as this enables us to take advantage of GPU parallelism and also obtain better gradient approximations (we'll talk more about the latter point later). This is called *batching*, where each prefix is an example in a larger batch. Here, you'll redo the computations from the previous cells, but instead of having one prefix, you'll have a batch of two prefixes. The final output of this cell should be a 2x5 matrix that contains two probability distributions, one for each prefix. **NOTE: YOU WILL LOSE POINTS IF YOU USE ANY LOOPS IN YOUR ANSWER!** Your code should be completely vectorized (a few large computations is faster than many smaller ones).

```
# for this problem, we'll just copy our old prefix over three times
# to form a batch. in practice, each example in the batch
# would be different.
batch_indices = torch.cat(2 * [indices]).reshape((2, 3))
batch_embs = embeddings(batch_indices)
#print(batch_embs)
print('batch embedding tensor size: ', batch_embs.size())

# now, follow the same procedure as before:
# step 1: compose each example's embeddings into a
# single representation using element-wise addition.
# HINT: check out the "dim" argument of the torch.sum function!
x = torch.sum(batch_embs, axis=1)
#print(x.size())

random_seed = 1
torch.manual_seed(random_seed)
W = torch.rand(10, 5)

# step 2: project each composed representation into
# a 5-d space using matrix W
# step 3: use the softmax function to obtain a 2x5 matrix
# with the probability distributions.
# Please store this probability matrix in the "batch_probs" variable.
batch_probs = torch.rand(2,5)
matrix_prod = torch.matmul(x,W)
batch_probs = torch.nn.functional.softmax(matrix_prod,dim = -1)

### DO NOT MODIFY THE BELOW LINE
print("batch probability distributions:", batch_probs)
```

```
batch embedding tensor size:  torch.Size([2, 3, 10])
batch probability distributions: tensor([[0.2316, 0.0035, 0.1040, 0.2055, 0.45
        [0.2316, 0.0035, 0.1040, 0.2055, 0.4555]], grad_fn=<SoftmaxBackward0>
```