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.

```
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.

```
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], grad
```

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>)
```

Colab paid products - Cancel contracts here

✓ 0s completed at 5:39 PM