Homework 0, CS685 Spring 2023

This is due on **February 17th, 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/1028HBIatC9std_7qcQ1jpAhErG4Uxn4l?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 so that you can run all cells almost instantly. If it is taking longer than that, you have made a mistake in your code.
- 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!

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. We may 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. Then you'll be sure it's actually right. 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.

Question 1.1 (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?

- By Bayes Rule of probability, $P(A, B)$ = $P(A|B) * P(B)$

  > This implies, $P(A, B, C)$ = $P(C|B, A) * P(B|A) * P(A)$

Hence one can use chain rule for a sequence of words to estimate the joint probability of a given sequence by multiplying together their conditional probabilities.

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

> Here we begin with the base case $P(w1)$ from the model and use the given model to getting the probability of next word given we have previous words. Finally, we multiply all such probabilites from the model to compute the final joint probability of the given sentence s.

> Thus, the joint probability of the given sentence s is $P(s) = \prod_{i=1}^{n} P(w_i|w_1, w_2, w_3, \ldots, w_{i-1})$

Question 1.2 (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.

Write your answer here! Please keep it brief (i.e., 2-3 sentences). evaluating coherence of sentence

1. Joint probability helps to decide what sequence of words would make more contextual sense in predictive modelling of a language as it uses conditional probability- one can compute the most probable next word in a sentence. . For example, "Students open books" will have a much higher joint probability(likelihood) of occurrence than "Student open lamps." Thus joint probability usage has some practical applications, such as auto-complete on mobile keyboards.
2. Some combinations of words or sentences are more probable than others which helps in tasks such as machine translation or grammar correction, where the system may have to choose the most likely combination of words.
3. It is also helpful to identify texts in an extremely noisy environment and helps to identify ones that best fit in. e.g., "I saw a van" would have a better probability than "Eyes awe of an" in speech recognition tasks where we have to string together phonemes to identify words to make a coherent sentence.

Question 1.3 (5 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) and keep count of the word w_i that follows each occurrence. I 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!

Write your answer here! Please keep it brief (i.e., 2-3 sentences).

This method is completely impractical because:

1. New sentences are created all the time, and there are chances some sentences have no instances on even large text corpus such as the Web. Even though we have a vast corpus, we can always get a word that was not in the vocabulary by replacing one of the words in a given sentence resulting in a zero count in the training set and, thus, zero joint probability overall for that sentence. So, we can only create a corpus with some of the possible sentences. It is computationally inefficient to calculate joint probabilities from large corpus.
2. The bigger the n-gram, the more computation and space are required. It gets problematic when the sequence of words is large. For example, "Sachin Tendulkar is master of playing Cricket." In this case, to compute the probability of this sentence, we will have to find the sum of counts of all the possible combination sequences from these 7 words apart from calculating the count of occurrence of the given sentence in the corpus, which results in massive computations (this grows exponentially with the increasing length of the sentence) Hence scaling up is an issue.

Question 2.1 (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

# YOUR CODE HERE! POPULATE THE types AND tokens VARIABLES WITH THE CORRECT VALUES!
tokenList = paragraph.split() #string splits on whitespace by default
types = len(set(tokenList))
tokens = len(tokenList)

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

```
Number of word types: 76, number of word tokens:100
```

Question 2.2 (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(tokenList)

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

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

Question 2.3 (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?

In linguistic functions (i.e. parts of speech), certain words also known as "stopwords" perform the role of determiners, preposition, pronouns or conjunctions. Such "stopwords" in NLP are often removed from text before any computational modeling is done because

- While performing any computational modeling like bag of words, such high frequency words are so generic they they do not impart much information and can be ignored without having any effect on the final model.
- Moreover, removal of such words helps in reducing dimensionality and thus less time to process amount of text.

> One must be careful with the removal of such "stopwords" when capturing context is involved. *E.g. Removal of Negation (never, not, no etc.) may completely alter the context of an entire sentence.*

Question 3.1 (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)
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".

### YOUR CODE HERE!
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, -1.8421, -3.6277,
         2.2783,  1.1165], grad_fn=<SumBackward1>)
```

Question 3.2 (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?

Write your answer here! Please keep it brief (i.e., 2-3 sentences).

The major issue with element-wise functions that makes them unsuitable for use as composition functions:

Composition Functions (like using element-wise addition to combine different word embeddings on a prefix to form a single representation) do not consider the order of representation of the words and spatial information. Even if the words were shuffled in a random order, the final output embedding after composition would be the same, which might not be very helpful.

There could be a different combination of words with entirely different meanings, which, when averaging, might result in the same final output embeddings. However, this problem is not much prominent in high-dimensional embedding space. For example, in a hypothetical scenario, the average of (5, 0, -5) is the same as the average of (0, 0, 0) in a single-dimensional space where (x, y, z) is the single-dimensional vector (Represented by integers, this represents any three words respectively)

Although simpler compositional models are more computationally efficient, they are not very expressive.

Question 3.3 (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: $p = \text{softmax}(W^T \boldsymbol{x})$. 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:
```

```
W = torch.rand(10, 5)

# 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
projectMatrix = torch.matmul(W.T,x)
probs = torch.nn.functional.softmax(projectMatrix)

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

```
probability distribution tensor([0.0718, 0.0998, 0.1331, 0.6762, 0.0191], grad_fn=<SoftmaxBackward0>)
<ipython-input-4-3d95a857da2a>:18: UserWarning: Implicit dimension choice for softmax has been deprecated. Change the call to include di
  probs = torch.nn.functional.softmax(projectMatrix)
```

Question 3.4 (15 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 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!

# 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.

embedded = torch.sum(batch_embs, dim = 1)
#W = torch.rand(10, 5)
projectMatrix  = torch.matmul(embedded, W)
batch_probs = torch.nn.functional.softmax(projectMatrix, 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([[0.0718, 0.0998, 0.1331, 0.6762, 0.0191],
        [0.0718, 0.0998, 0.1331, 0.6762, 0.0191]], grad_fn=<SoftmaxBackward0>)
```

Question 4 (20 points)

Choose one paper from EMNLP 2022 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 cell, and make sure to write 2-4 paragraphs for the summary to receive full credit!

Title of paper: Knowledge-Rich Self-Supervision for Biomedical Entity Linking

Authors: Sheng Zhang, Hao Cheng, Shikhar Vashishth, Cliff Wong, Jinfeng Xiao, Xiaodong Liu, Tristan Naumann, Jianfeng Gao, Hoifung Poon

Conference name: Conference on Empirical Methods in Natural Language Processing (2022) - Findings of the Association for Computational Linguistics: EMNLP 2022

URL: https://aclanthology.org/2022.findings-emnlp.61.pdf

Your summary:

The motivation of the paper "Knowledge-Rich Self-Supervision for Biomedical Entity Linking" in EMNLP 2022 conference is to address the challenge of biomedical entity linking, which involves identifying and linking biomedical entities mentioned in unstructured text to a knowledge base such as UMLS (Unified Medical Language System) or MeSH (Medical Subject Headings). This is a critical task in biomedical text mining, as it enables the discovery of new knowledge from the vast amount of biomedical literature.

One of the main challenges of biomedical entity linking is the lack of annotated data, which makes it difficult to train accurate models. The paper proposes a novel approach that leverages knowledge-rich self-supervision to train entity linking models in a data-efficient manner. The key idea is to use external knowledge sources, such as UMLS and MeSH, to generate synthetic training data that can be used to train the entity linking models.

The paper also proposes a novel architecture that combines a contextualized representation of the input text with a graph-based representation of the knowledge base to enable effective entity linking. The proposed approach achieves state-of-the-art performance on two benchmark datasets for biomedical entity linking, demonstrating the effectiveness of the knowledge-rich self-supervision approach. The paper has significant implications for the development of accurate and data-efficient models for biomedical entity linking, which can facilitate the discovery of new biomedical knowledge.

The paper is of significant importance for several reasons:

1. Biomedical entity linking is a critical task in biomedical text mining, as it enables the discovery of new knowledge from the vast amount of biomedical literature. Therefore, developing accurate and data-efficient models for biomedical entity linking is essential for advancing research in the field.
2. The lack of annotated data in the biomedical domain is a major challenge, which makes it difficult to train accurate models. The proposed approach in this paper leverages external knowledge sources to generate synthetic training data, enabling the training of accurate models in a data-efficient manner.
3. The proposed approach achieves state-of-the-art performance on two benchmark datasets for biomedical entity linking, demonstrating the effectiveness of the knowledge-rich self-supervision approach.
4. The paper's proposed architecture, which combines a contextualized representation of the input text with a graph-based representation of the knowledge base, has the potential to be applied to other tasks beyond biomedical entity linking.

In summary, the paper's proposed approach and architecture have significant implications for the development of accurate and data-efficient models for biomedical entity linking, which can facilitate the discovery of new biomedical knowledge.

Most questions I had was regarding the usage of KRISS framework, why the authors finalized on KRISS, what are its alternatives, and does contrastive learning perform equally good for all other alternatives. I am also unsure about cross-attention candidate re-ranking – what it means, and how its performance can be calculated.

AI Disclosure

- Did you use any AI assistance to complete this homework? If so, please also specify what AI you used.
    - *No*

(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.
    - *your response here*
- **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*

✓ 0s completed at 11:52 PM