Text Generation

So.... we have seen AI is capable of generating text in last week lecture. The model used is very sophisticated (GPT3 model by openai) and it's not an easy task to replicate the feat. Instead, we are exploring to build a simple language model in this homework, namely, the **N-Gram model**. First of all, some key words to understand:

1. Language Model -- A model that assign probabilities to sequences of words.
2. N-gram -- A sequence of *n* words. For example, 1-gram (unigram), 2-gram (bigram), 3-gram (trigram), ... Feel free to quickly Google to learn more about N-gram.

make_ngram() and pad()

First, let's implement a function to generate ngrams. Take a look on how you could implement it using a sliding window technique: https://www.researchgate.net/publication/354427958/figure/fig1/AS:1065576361369600@1631064610082/Extraction-of-n-grams-with-a-sliding-window-approach-n-3.png (https://www.researchgate.net/publication/354427958/figure/fig1/AS:1065576361369600@1631064610082/Extraction-of-n-grams-with-a-sliding-window-approach-n-3.png)

In [1]:
```
def make_ngrams(list_of_words, n):
    '''
    For example, list_of_words = ['I', 'am', 'happy']
    make_ngrams(list_of_words, 1) returns [['I'], ['am'], ['happy']]
    make_ngrams(list_of_words, 2) returns [['I', 'am'], ['am', 'happy']]
    make_ngrams(list_of_words, 3) returns [['I', 'am', 'happy']]
    '''
    tokens = [token for token in list_of_words if token != ""]
    ngrams = zip(*[tokens[i:] for i in range(n)])
    return [" ".join(ngram).split() for ngram in ngrams]
```

In [2]:
```
# Check your work
input = "I am happy".split(" ")
assert make_ngrams(input, 1) == [['I'], ['am'], ['happy']]
assert make_ngrams(input, 2) == [['I', 'am'], ['am', 'happy']]
assert make_ngrams(input, 3) == [['I', 'am', 'happy']]
```

In practice, we usually "pad" the input sentences with starting and ending tokens (think of it as special "words"), according to the *n* used in n-grams. For example, we would pad the sentence "I am happy" with ~~and <s> tokens so the input to make_ngrams() looks like ['', 'I', 'am', 'happy', '<\s>']~~

In [3]:
```
def pad(list_of_words, n):
    '''
    For example, list_of_words = ['I', 'am', 'happy']
    pad(list_of_words, n =1) does nothing and return ['I', 'am', 'happy']
    pad(list_of_words, n =2) returns ['<s>', 'I', 'am', 'happy', '<\s>']
    pad(list_of_words, n =3) returns ['<s>', '<s>', 'I', 'am', 'happy', '<\s>', '<\s>']
    '''
    for i in range(n-1):
        list_of_words= ['<s>']+ list_of_words+ ['</s>']
    return list_of_words
```

In [4]:
```
# Check your work
input = "I am happy".split(" ")
assert pad(input, 1) == ['I', 'am', 'happy']
assert pad(input, 2) == ['<s>', 'I', 'am', 'happy', '</s>']
assert pad(input, 3) == ['<s>', '<s>', 'I', 'am', 'happy', '</s>', '</s>']
```

Now, try pad() a sentence first, and then make_ngrams() on the padded sentence.

In [5]:
```
NGRAM = 3
input = "I am happy".split(" ")
padded_input = pad(input, NGRAM)
expected = [
    ['<s>', '<s>', 'I'],
    ['<s>', 'I', 'am'],
    ['I', 'am', 'happy'],
    ['am', 'happy', '</s>'],
    ['happy', '</s>', '</s>']
]
assert make_ngrams(padded_input, NGRAM) == expected

input = "I am sad".split(" ")
padded_input = pad(input, NGRAM)
expected = [
    ['<s>', '<s>', 'I'],
    ['<s>', 'I', 'am'],
    ['I', 'am', 'sad'],
    ['am', 'sad', '</s>'],
    ['sad', '</s>', '</s>']
]
assert make_ngrams(padded_input, NGRAM) == expected
```

Big Picture

Now that we understand padding and N-grams, how can we use it to "Generate" text? Here is what we are going to do:

1. Grab some text data, pad it, and generate n-grams
2. Records the n-grams counts in a Python dictionary.
3. Use that dictionary to sample and generate text.

For example, let say our data simply consists of two sentences:

```
[
  ["I", "am", "happy"],
  ["I", "am", "sad"]
]
```

and we use `NGRAM = 3`. Then, we can records the counts of n-grams like:

{n-1}gram	word	Count
<s> <s>	I	2
<s> I	am	2
I am	happy	1
I am	sad	1
am happy		1
am sad		1
happy </s>		1
sad </s>		1

We essentially take every n-grams, split them into `(n-1)-grams` and a `word`, and store them into a dictionary. If the same n-grams appear again, we increment the count of seeing such a combination. The following combinations has `Count` of 2 because they appear in both sentences.

{n-1}gram	word	Count
<s> <s>	I	2
<s> I	am	2

Later on, when we want to generate sentence, we can use the `Count` to guide us.
For example, given `[I, am]` as the `(n-1)-grams`, we want to "generate" the next word. The following shows that the words `happy` and `sad` are equally probable, so the probability of generating `happy` is 0.5 and the same goes to `sad`

{n-1}gram	word	Count
I , am	happy	1
I , am	sad	1

Assuming that our data instead is

```
[
  ["I", "am", "happy"],
  ["I", "am", "happy"],
  ["I", "am", "sad"]
]
```

The dictionary would look like:

{n-1}gram	word	Count
I , am	happy	2
I , am	sad	1

so it's more likely to generate `happy` as the next word. We will revisit this later.

Acquiring data

Read the Shakespeare Play data from here (https://raw.githubusercontent.com/TheanLim/cs6120/main/shakespeare_plays.txt). The text is "cleaned" such that all words are in lower case, and all punctuations are stripped.

In [6]:
```
# Read and store the text into a variable named `data`
# `data` should be a list of list, where each inner list represents a line of the text
import requests

link = "https://raw.githubusercontent.com/TheanLim/cs6120/main/shakespeare_plays.txt"
raw_text= requests.get(link).text
data = raw_text.split('\n')[:-1]

assert len(data) == 29959
```

In [7]:
```
# Split each sentence by white spaces
data = [sentence.split(" ") for sentence in data]
assert len(data) == 29959
```

Pad the data and create n-grams

In [8]:
```
NGRAM = 5
# Create Ngrams for each sentence in data
## TODO
data_ngrams= [make_ngrams(pad(sentence, NGRAM), NGRAM) for sentence in data]
assert len(data_ngrams[0][0]) == NGRAM
```

Records the n-grams counts in a Python dictionary

While I previously represented the dictionary as a table:

{n-1}gram	word	Count
<s>	I	2
<s> I	am	2
I , am	happy	1
I , am	sad	1
am , happy		1
am , sad		1
happy ,		1
sad ,		1

it's more natural to represent it using Python Nested Dictionary

```
ngram_dict =
{
  '<s>,<s>': {'I': 2},
  '<s>,I': {'am': 2},
  'I,am': {'happy': 1, 'sad': 1},
  'am,happy': {'</s>': 1},
  'am,sad': {'</s>': 1},
  'happy,</s>': {'</s>': 1},
  'sad,</s>': {'</s>': 1}
 }
```

In [9]:
```
from collections import defaultdict
ngram_dict = defaultdict(lambda: defaultdict(int))
NGRAM = 5

for list_of_words in data_ngrams:
    for ngram in list_of_words:
        prefix= ",".join(ngram[:NGRAM- 1])
        subfix= ngram[-1]
        ngram_dict[prefix][subfix]+= 1
```

Generate Sentences

Here is the exciting part - Text Generation. Our strategy here is to generate one word at a time, until we see the ending token `</s>`.

- We use (n-1)-grams to get a list of possible `next_words`, as well as its associated `Counts`.
- We then sample and pick one word from the list of `next_words`, using those `Counts` as the probability (aka weights). Take a look at `random.choices(population, weights)`
- Since we padded sentences to start with `<s>`, we should also use that as the start in our sentences generation. In a `NGRAM = 3` example, we want to start with two `<s>`.

Note:
The `ngram_dict` will always have a key made up of `NGRAM-1` `<s>`. If `NGRAM = 3`, then `ngram_dict[<s>,<s>]` will always return you something.

In [10]:
```
import random
def generate_word(prompt, ngram_dict):
    '''
    Given a prompt, sample and return a next word from ngram_dict
    The prompt should be made up of NGRAM-1 words
    and should be a legitimate key of ngram_dict
    '''
    population = ngram_dict[prompt]
    keys= list(population.keys())
    weights= []
    for key in keys:
        weights.append(population[key])
    if keys and len(keys)> 0:
        next_word = random.choices(keys, weights)
        return next_word
    return ["</s>"]

def generate_sentence(n, ngram_dict, NGRAM):
    '''
    Generate a sentence with a maximum of `n` words.

    Note:
    - Do not output <s> or </s>. The output sentence should look like a normal sentence
    - You should call generate_word() from this function, and stop when the return word is </s>
    '''
    res = []
    cur = ["<s>"]* (NGRAM- 1)
#     prompt= ",".join(cur)
#     next_word = generate_word( prompt, ngram_dict)
#     cur= cur[1:]+ next_word
#     print (cur)
    while n> 0:
        prompt= ",".join(cur)
        next_word = generate_word( prompt, ngram_dict)
        if next_word == "</s>":
            break
        cur= cur[1:]+ next_word
        res+= next_word

        n-= 1
#     return " ".join(res)
    return " ".join([word for word in res if word not in ["<s>", "</s>"]])
```

In [11]:
```
# Generate 20 sentences with a maximum 20 words each
for i in range(20):
    print(generate_sentence(20, ngram_dict, NGRAM))
    print()
```

```
find his way without his eyes for out o doors he went without their help and to the last bended

macbeth give him tending he brings great news exit messenger the raven himself is hoarse that croaks the fatal entrance

give her good watch i pray you exit horatio o this is the poison of deep grief it springs all

very distant time stood as it were in

i cannot make true wars i ll frame convenient peace now good aufidius were you in my stead would you

you sir desire my man s abode where i did leave him with his forces and do expect him here

greater he shall not be if he serve god we ll serve him too and be his fellow so revolt

hand and with a thought seven of the eleven i paid prince o monstrous eleven buckram men grown out of

cast your election on him sicinius say you chose him more after our commandment than as guided by your own

immortal title to your crown king richard we thank you both yet one but flatters us as well appeareth by

parolles a filthy officer he is in those suggestions for the young earl beware of them diana their promises enticements

first a very excellent good conceited thing after a wonderful sweet air with admirable rich words to it and then

pardon from himself though you and all the kings of christendom are led so grossly by this

here i lay and thus i bore my point four rogues in buckram let drive at me prince what four

disdaining me and throwing favours on the low posthumus slanders so her judgment that what s else rare is chok

done bora we ll wait upon your lordship exeunt scene 3 pomfret castle enter sir richard ratcliff with halberds carrying

dangerous third watchman ay but give me worship and quietness i like it better than dangerous honour if warwick knew

worm cleopatra get thee hence farewell clown i wish you all joy of the worm sets down the basket cleopatra

this thou shalt not choose but go do not deny beshrew his soul for me he started one poor heart

laugh but in that maid s company but indeed she is given too much to allicholy and musing but for
```

Reference

1. https://web.stanford.edu/~jurafsky/slp3/3.pdf (https://web.stanford.edu/~jurafsky/slp3/3.pdf)

Sentiment Analysis

In [12]:
```
!pip install nltk
# Install a pip package in the current Jupyter kernel
import sys
!{sys.executable} -m pip install nltk
```

```
Requirement already satisfied: nltk in c:\python310\lib\site-packages (3.7)
Requirement already satisfied: joblib in c:\python310\lib\site-packages (from nltk) (1.2.0)
Requirement already satisfied: tqdm in c:\python310\lib\site-packages (from nltk) (4.64.1)
Requirement already satisfied: click in c:\python310\lib\site-packages (from nltk) (8.1.3)
Requirement already satisfied: regex>=2021.8.3 in c:\python310\lib\site-packages (from nltk) (2022.10.31)
Requirement already satisfied: colorama in c:\python310\lib\site-packages (from click->nltk) (0.4.6)

WARNING: Ignoring invalid distribution -ip (c:\python310\lib\site-packages)
WARNING: Ignoring invalid distribution -ip (c:\python310\lib\site-packages)
WARNING: Ignoring invalid distribution -ip (c:\python310\lib\site-packages)
WARNING: Ignoring invalid distribution -ip (c:\python310\lib\site-packages)
WARNING: Ignoring invalid distribution -ip (c:\python310\lib\site-packages)
WARNING: Ignoring invalid distribution -ip (c:\python310\lib\site-packages)

[notice] A new release of pip available: 22.2.2 -> 22.3.1
[notice] To update, run: python.exe -m pip install --upgrade pip

Requirement already satisfied: nltk in c:\users\tongz\appdata\local\programs\python\python38-32\lib\site-packages (3.7)
Requirement already satisfied: regex>=2021.8.3 in c:\users\tongz\appdata\local\programs\python\python38-32\lib\site-packages (from nltk) (2022.10.31)
Requirement already satisfied: click in c:\users\tongz\appdata\local\programs\python\python38-32\lib\site-packages (from nltk) (8.1.3)
Requirement already satisfied: joblib in c:\users\tongz\appdata\local\programs\python\python38-32\lib\site-packages (from nltk) (1.1.0)
Requirement already satisfied: tqdm in c:\users\tongz\appdata\local\programs\python\python38-32\lib\site-packages (from nltk) (4.64.1)
Requirement already satisfied: colorama; platform_system == "Windows" in c:\users\tongz\appdata\local\programs\python\python38-32\lib\site-packages (from click->nltk) (0.4.3)

WARNING: You are using pip version 20.0.2; however, version 22.3.1 is available.
You should consider upgrading via the 'c:\users\tongz\appdata\local\programs\python\python38-32\python.exe -m pip install --upgrade pip' command.
```

In [12]:
```
import nltk, random
from nltk.corpus import movie_reviews
import random
nltk.download("movie_reviews")
```

```
[nltk_data] Downloading package movie_reviews to
[nltk_data]     C:\Users\tongz\AppData\Roaming\nltk_data...
[nltk_data]   Package movie_reviews is already up-to-date!
```

Out[12]: True

In [13]:
```
documents = [(list(movie_reviews.words(fileid)), category)
             for category in movie_reviews.categories()
             for fileid in movie_reviews.fileids(category)]
# Each document is a tuple of (list_of_words, str_of_pos_or_neg_label)
```

In [14]:
```
from collections import Counter
count_pos_neg = Counter([label for (words, label) in documents])
print(count_pos_neg)
```

```
Counter({'neg': 1000, 'pos': 1000})
```

Split data into train and test

Make sure that the positive and negative labels are balanced in the train and test data

In [15]:
```
from sklearn.model_selection import train_test_split
## TODO
train, test = train_test_split(documents, test_size=0.15, random_state= 32)

print(Counter([label for (words, label) in train]))
print(Counter([label for (words, label) in test]))
```

```
Counter({'neg': 855, 'pos': 845})
Counter({'pos': 155, 'neg': 145})
```

In [16]:
```
# Split data into X and Y
## TODO
X_train = [ele[0] for ele in train]
X_test = [ele[0] for ele in test]
y_train = [ele[1] for ele in train]
y_test = [ele[1] for ele in test]
```

Bag of Words

We are going to use Bag of Words as "Features". Take a look at this image (https://miro.medium.com/max/1400/1*hLvya7MXjsSc3NS2SoLMEg.webp)

1. Create a vocabulary of words. Only keep a word if it appears more than 10 times.
2. Use each word in the vocabulary as a "feature". We record whet how many times a word appear within a review

In [17]:
```
MIN_COUNT = 10
## TODO  Create a vocab.  Only keep a word if it appears more than MIN_COUNT times
cnt= defaultdict(int)
vocab= {}
for words in [ele[0] for ele in documents]:
    for word in words:
        cnt[word]+= 1
idx= 0
for (key, val) in cnt.items():
    if val> MIN_COUNT:
        vocab[key]= idx
        idx+= 1
print (len(vocab))
```

```
8817
```

In [18]:
```
from collections import Counter
def create_features(review, vocab):
    '''
    Create and return a list of features given a movie review

    '''
    features = [0] * len (vocab)
    for word in review:
        if word in vocab:
            features[vocab[word]] += 1
    assert len(features) == len (vocab)
    return [int( f) for f in features]
```

In [19]:
```
# Create features for each movie review
X_train_features = [create_features(review, vocab) for review in X_train]
X_test_features = [create_features(review, vocab) for review in X_test]
```

In [20]:
```
# print (X_train_features[0])
```

Fit a Logistic Regression Classifier

In [25]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
clf = LogisticRegression(random_state=64, solver='lbfgs', max_iter=1000).fit(X_train_features, y_train)
y_pred = clf.predict(X_test_features)
```

In [29]:
```
print ("Accuracy:", accuracy_score(y_test, y_pred))
```

```
Accuracy: 0.84
```

How does your model perform on the test data?

I tried several different sets of parameters and was able to get stable results If we want to go further, we can normalize the FEATURES.

In [ ]: