1. **In your own words in about half a page, describe what language models are and why they are appealing to certain applications. [4 points]**

A language model is a prediction model that uses some context (local – previous output(s)/global – entire sentence bidirectionally) to generate the next output token. The mechanism could be statistical in nature where the model's output is dependent on the frequency distribution of word occurrences of the document it was trained on; or it could be neural in nature where a deep neural network is trained using a large corpus of words to adjust the corresponding weights and biases. Either way, the language model learns the patterns so it assigns probabilities to its vocabulary and the word corresponding to the highest probability is the output.

These language models are appealing to Seq to Seq applications like Machine Translation, Text auto complete, voice to text because of their potential for reducing repetitive work that humans do in their day to day life like writing emails or generating transcripts. The big pinpoint for many language models is the long-term dependency problem where as the length of a document increases, it becomes difficult for the model to retain earlier context. This problem is more commonly referred to as the vanishing gradient problem.

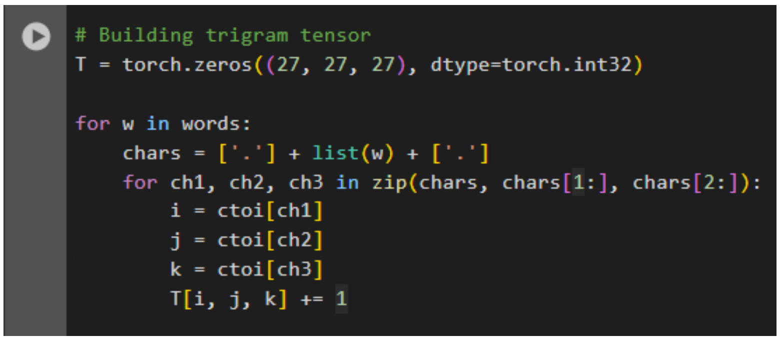
2. **In the class we saw how to construct bigram model for text prediction. Do the same with a trigram model. Here, you will take two characters to predict the next character. Provide your qualitative assessment of any differences you see between bigram-based generation and trigram-based generation. Include appropriate parts of your code segments to show the key elements of constructing the trigram model. [6 points]**

My thought process before writing code:

This is a trigram model that will use previous two characters to predict the next one. I will need to use a 27 x 27 x27 tensor. The tensor T[i][j][k] will count the frequency of character k after the combination of characters i and j. Since this model is again statistical in nature (just like bigram), the words generated will not make much sense. However, there is local context of two words. So, the generated words will be relatively more like the training corpus as compared to what they were from bigram model. I also want to highlight the concept of multinomial distribution here. This multinomial distribution will lead to high-probability characters to be more likely to appear, but there will be some randomness (poorly analogous to exploitation and exploration in reinforcement learning, just that it's statistical in nature).

I also think that since the training corpus (.txt file is the same) but the tensor complexity has increased in terms of dimensions, there will be more sequences for which we will not have any value in the text. In other words, sparsity will be more in a trigram model. One way to address it is to have a larger training dataset to produce more reliable probabilities.

I have tweaked the code from the bi-gram model to create the results. Below are some snippets:

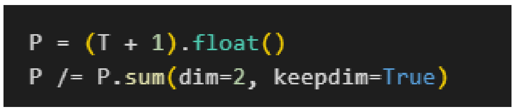


```
# Building trigram tensor
T = torch.zeros((27, 27, 27), dtype=torch.int32)

for w in words:
    chars = ['.'] + list(w) + ['.']
    for ch1, ch2, ch3 in zip(chars, chars[1:], chars[2:]):
        i = ctoi[ch1]
        j = ctoi[ch2]
        k = ctoi[ch3]
        T[i, j, k] += 1
```

While generating the output, I got error: "RuntimeError: probability tensor contains either inf, nan or element < 0."

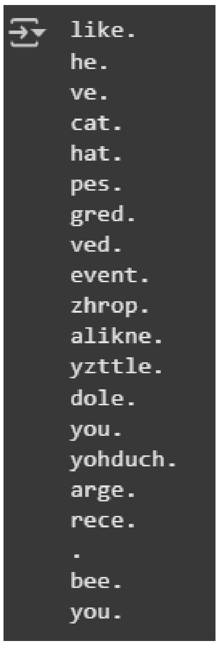
This was due the sparsity problem, as some values in the tensor were NaN. So, I used simple Laplace smoothing by adding 1 before normalizing.



```
P = (T + 1).float()
P /= P.sum(dim=2, keepdim=True)
```

Output words were gibberish, but more fluent and human like compared to bi-gram model.

Key learning: As the complexity increases bi-gram to tri-gram models, we need more data (and computation) to address sparsity issue. Consequently, learning improves that we get more realistic results.



```
like.
he.
ve.
cat.
hat.
pes.
gred.
ved.
event.
zhrop.
alikne.
yzttle.
dole.
you.
yohduch.
arge.
rece.
.
bee.
you.
```