NGram homework

- what are n-grams and how are they used to build a language model

  N-grams are sets of words used to analyze frequency and probability in a language model. For example a bigram is a pair of words (tokens) in the text and the more often the same pair occurs in a given text the more likely the two words are associated with each other. A bigram that only occurs once isn't likely to have a connection but a bigram that occurs in a significant percentage of the text is very likely to

- list a few applications where n-grams could be used

  It's really helpful for text recommendations such as emails and personal texts. N-grams recommends the next word because it sees them associated with each other a lot and there's a high likely hood "To whom" will be followed by "it may concern". One could incorporate n-grams into a text summary application to build a smart out of high probability n-grams.

- a description of how probabilities are calculated for unigrams and bigrams

  For unigrams the probability of any given word is the number of times that word appears in the training set divided by the total number of words in the training set. i.e - P(word) = word_count_in_text / all_words_in_text. Then you multiply the probability of every word together to get the probability of an entire sentence. FOR bigrams we do the same thing but instead of calculating the probability of each word we calculate the probability of every two words. This looks like: P(two, words) = (two, words)_count_in_text / all_bigrams_in_text. Again multiply probabilities together of all bigrams in a sentence gets you the probability of an entire sentence happening.

- the importance of the source text in building a language model

  Source text can make or break your language model. Training a model in a different language would produce almost 0% probability because it won't find any words that match unless the languages share similarities. Even then the language model isn't being used efficiently. For best use you should train a model on a range of source text all focused toward your goal. i.e A language model that analyzes legislative bills should be trained on the text from all bills from congress, the senate, and if you can you could include bills from state level legislative branches as well. A language model that analyzes legislative bills should NOT be trained on calculus textbooks or poetry books. Yes some similarities will be found due to the sharing of a languages but the purpose of the language model would be compromised.

- the importance of smoothing, and describe a simple approach to smoothing

  Smoothing fills in the gaps for probabilities when a model has to analyze an n-gram it has never seen / wasn't trained on. Due to us multiplying probabilities, if we encounter an n-gram that wasn't seen in training the count of that n-gram would be 0, making that probability 0, making an entire sentence have a 0% probability of matching our model simply because we never trained the model on the bigram ("cotton", "candy"). Smoothing prevents this multiply by 0 problem. A naive approach would be to add 1 to every n-gram count thus eliminating any 0 from being multiplied. Simply replacing the 0 with a 1 may still produce an extremely low, almost 0, probability. But almost 0 is not 0, and that's what saves our language model.

- describe how language models can be used for text generation, and the limitations of this approach

  Text generation can be useful for repetitive phrases but not much else. When you're a wedding planner and constantly having to explain what a gazebo is, then text generation is helpful. When you're texting your best friend what happened in your day, text generation can get in the way because a persons day can very wildly every day. Text generate is also highly limited to what it's trained on. You computer can't* (shouldn't) be coming up with new words it's never seen before and generating those. It can only choose the next word with the highest probability. Text generation also doesn't always take into account sentiment, just probabilities. The context of a sentence and whether or not it makes sense doesn't always follow probability. The model may think it's producing a coherent sentence because it choose every n-gram with the next highest probability, but the sentence as a whole may ramble on or not be tied together.

- describe how language models can be evaluated

  Does it have good probabilities, does it test good. Thinking of the probabilities you can evaluate a language model based on if it was trained right. Seeing the probabilities it goes up with for certain n-grams and determine if that is an acceptable probability or not. The probability of the sentence "I killed the goose" should not be very high in a model trained on math textbooks. Next way to evaluate would be to test the model and see if it's any good. Maybe it captured the probabilities wrong but if the model is really good at a given task you might consider keeping it around. A model trained on medical textbooks might not give you the correct answer to every med school homework question, but it can extremely accurately provide you definitions on medical terms you have forgotten. If ChatGPT doesn't give me good code examples but it does provide accurate definitions of different Python libraries then I'd give it a high evaluation for the python task and a low evaluation for the writing code task.

- give a quick introduction to Google's n-gram viewer and show an example

  Google's n-gram viewer lets you search up n-grams and see their rise and fall in probability throughout time. My example shows a few common Spanish sayings plotted alongside each other. Google even lets you search the books they used for training to get the probabilities and divides into significant time frames that usually line up with the spikes you see in the graph.