Programming for Computational Linguistics 2018/2019

Final Project Checkpoint 4 — `p_next`

These checkpoint documents provide a detailed recommendation for how to proceed with the project. They will break the requirements into small, well-defined steps. It is not necessary to follow this plan — you can choose to implement the requirements however makes the most sense to you. However, if you are unsure of how to proceed, this should provide you a good starting point.

Now that we can train our language model, we need to allow it to make predictions. The method `p_next` will help us with that. Remember, this method should take as input a sequence of tokens, and should output a probability distribution for the next token in the text.

Before we write `p_next`, let's write a minor function that will help us. This function will be called `normalize(word_counts)`. It could be a method in the `LanguageModel` class, but it could also be a function outside the class. As input, it should take a dict `word_counts`. The keys of this dict are words, and the values of this dict are numbers, representing how many times that word occurred. These are like the inner dicts in the instance variable `counts`. This function should return a new dict. The keys of this new dict should be words, just like the input. The values of this new dict should be probabilities. The probability for each word should be proportional to its count, and all probabilities should add up to 1. Write this method. Be careful that you don't change the input dict — just return a new one.

Now we can write the method `p_next`. Remember, the input to `p_next` is a list of arbitrarily many tokens. Your method `p_next` should extract the final $n-1$ tokens from this (after padding appropriately), look those tokens up in `counts` to get `word_counts`, use `normalize` to turn those counts into a probability distribution, and return that distribution. Write this method. If the final $n-1$ tokens do not occur in counts, you should still return a valid probability distribution, but you can generate this however you like.