# Motivating the Backoff N-Gram

Colan F. Biemer

## 1 A Brief Introduction to N-Grams

Imagine we are given a document—in the literature, this is referred to as a corpus—and want to use it to generate text. [1] Perhaps the simplest approach is to count the occurrence of each word in the document and use the counts as weights for picking a new word. To put this into perspective, imagine we have the following counts from a document:

| Word | Count |
|---|---|
| the | 120 |
| war | 32 |
| was | 100 |
| horrible | 30 |
| ... | ... |

To calculate the probability of each word being generated, we use its count over the sum of all counts—see equation below. The equation assumes the existence of a count function that will go to the table and return the number of times the word was in the input document. If the word was not seen, the count function will return 0. The equation also uses the notation $j \in \mathbb{R}^n$ to represent that there exists $n$ unique words in the input document. The sum in the denominator will get the word count of the document.

$$P[\text{word}_i] = \frac{\text{count}(\text{word}_i)}{\sum_{j \in \mathbb{R}^n} \text{count}(\text{word}_j)}$$

With this, we can choose the next generated word with two strategies. The first strategy is to select the word with the highest probability. This isn't ideal since the same word would be generated every time. The second strategy is to use the counts for a weighted choice. If we look at our dataset above, the word "was" is much more likely to occur than "war" or "horrible" but less likely to occur than "the". When we use the weighted output strategy, the output is supposed [2] to fit the input's distribution.

The model we've detailed is called a uni-gram or a 1-gram. What then is a 2-gram? Otherwise known as a bi-gram, this model uses a prior to determine what output is possible given the prior. In mathematical terms, this would look like $P[\text{word}_i \mid \text{word}_j]$. This asks, what is the probability of $\text{word}_i$ given the prior of $\text{word}_j$? If this isn't clear, let's instead look at it through the lens of a data structure. In the table below, you can see that the prior word links to a table very similar for the uni-gram. In fact, one way to implement this is to have every prior link to a uni-gram. This approach extends for a 3-gram, 4-gram, and beyond. No matter the size of the prior, it will always link to a uni-gram with the number of occurrences for each possible output.

| Prior | Uni-Gram | |
|---|---|---|
| | Word | Count |
| the | war | 20 |
| | horrible | 31 |
| ... | ... | |

| Prior | Uni-Gram | |
|---|---|---|
| | Word | Count |
| [the, war] | horrible | 3 |
| | was | 19 |
| ... | ... | |

Table 1: Left-hand-side table shows an example of a bi-gram where the prior is "the" and the uni-gram attached. Right-hand-side is an example of a tri-gram where the prior is composed of "the" and "war", in that order; order matters.

[1] This document only looks at generation using n-grams, but there is more that they can be used for such as playing rock-paper-scissors or predicting the likelihood that someone plagiarized.

[2] I use the word "supposed" here since it is random. There is no guarantee the distribution will be exactly met, only that it is likely.

If we were to generate a word using one of these n-grams, where $n > 1$, we would need a starting prior. In the bi-gram case in table 1, we could start generating a new sentence starting with the word "the". In the example, we can see that 20 out of 51 times, "war" will be generated. Otherwise "horrible" will be generated. Say that we generate the more likely word "horrible". Our sentence is now "The horrible." If we wish to generate another word, we would go back to our table with "horrible" as our prior. This process can continue until we reach the desired number of words, a word is found that ends the sentence, or it could go on infinitely or until a prior has no known output. This last case is rare when dealing with a bi-gram—though you could easily create a contrived dataset to see the problem yourself—but the problem becomes more prevalent as $n$ increases.

## 2 The Backoff N-Gram

The backoff n-gram is a hierarchical n-gram; which is to say that it is made up of many n-grams. A backoff n-gram given the size of six will be made up of six n-grams: a six-gram, a five-gram, a four-gram, a three-gram, a two-gram, and a uni-gram. Though, some work has looked at not including every value less than $n$ [1].

Now imagine a scenario where we give our backoff 6-gram the prior, "the war was horrible but". For our hypothetical example, this prior was not seen in the input training dataset. If we use a standard 6-gram only, we have an unseen prior and our model fails. If we use a backoff n-gram, though, we can resolve the problem.

The reason why the word "backoff" is in the name of the model is because it is the strategy used when an unknown prior is received. In our example, we have a prior we haven't seen and we handle it by reducing the prior and backing off to the model's five-gram. The prior is now "war was horrible but". If our five-gram has seen this prior, then we can generate a word using it. If not, we reduce the prior to "was horrible but" and try with our four-gram. We continue this process of backing off to the next n-gram until a valid prior is found or until we've reached a prior of "". At this point, we now use our uni-gram, which doesn't require any prior, to generate.

The primary problem that the backoff n-gram and any hierarchical n-gram [2] aims to handle is an unknown prior. With a hierarchical approach, an n-gram is guaranteed to be able to continue outputting words without a problem.

## References


[1] S. Snodgrass and S. Ontanón, "Controllable procedural content generation via constrained multi-dimensional markov chain sampling." in *IJCAI*, 2016, pp. 780–786.

[2] D. Jurafsky, *Speech & language processing.* Pearson Education India, 2000.