# Language Models and the $N$-gram Model

(*Part 1 of Modelling Textual Data Series*)

Rey R. Cuenca*

May 2022

**Abstract**

This paper gives a quick introduction to the language models and the $n$-gram model.

## 1 Motivation: Modelling Language Intuition

Any person with the some basic fluency of the English language would find that sentence (A) "sounds more grammatical" than sentence (B).

(A) *The food is already served to the customers.*

(B) *The food served is already to the customers.*

How do they do this? It would be great to mathematized this kind of "grammatical intuition". Doing so would help us build machines (computers) to automate the process of checking the grammaticality of any given sentence and lessen both burden and inaccuracies when doing it manually specially in situations where they are provided in volumes.

In this paper, we introduce the concept of *language models* (LMs) and present one, belonging to this family, called the *$n$-gram* model.

## 2 Notations and Representations

Recall that our goal is to provide a mathematical representation, a *mathematical model* to be exact, of the "grammatical intuition" discussed in the previous section.

*Department of Mathematics and Statistics, MSU-Iligan Institute of Technology

Since we are to translate such abstract idea to mathematical terms, we need to introduce some mathematical notations. First is the representation of sentences.

| Notation | Description |
|---|---|
| $w_i$ | tokens (e.g. words, phrases,...) |
| $\mathbf{w} = (w_1, w_2, \ldots, w_n)$ | sentences |

**Example 2.1.** Consider the following sentence:

```
The dog is barking.
```

We could write this as a vector

$$\mathbf{w} = (\texttt{The}, \texttt{dog}, \texttt{is}, \texttt{barking})$$

where

| $i$ | $w_i$ |
|---|---|
| 1 | `The` |
| 2 | `dog` |
| 3 | `is` |
| 4 | `barking` |

Notice, we did not include the period "." symbol. However, depending on the usage case, we may include this as part one of the tokens in sentences or represent it with another notation like in the following example.

**Example 2.2.** In some cases, it is useful to introduce some auxiliary notations representing the starting and ending of a given sentence like the following.

```
<s> The dog is barking </s>
```

Writing this in a vector form, we have

$$\mathbf{w} = (\texttt{<s>}, \texttt{The}, \texttt{dog}, \texttt{is}, \texttt{barking}, \texttt{</s>})$$

where

| $i$ | $w_i$ |
|---|---|
| 1 | `<s>` |
| 2 | `The` |
| 3 | `dog` |

| $i$ | $w_i$ |
|---|---|
| 4 | `is` |
| 5 | `barking` |
| 6 | `</s>` |

# 3 The Langauge Model

The "grammatical intuition" can essentially be viewed in two ways:

1. *Given a sentence, identify whether it's grammatical or not.*
2. *Given two sentences, identify which is more grammatically sound.*

The mathematical formulation for the first one is equivalent to finding a function $f$ such that given a sentence $\mathbf{w}$,

$$f(\mathbf{w}) = \begin{cases} 1 & \mathbf{w} \text{ is grammatical} \\ 0 & \text{otherwise} \end{cases} \tag{1}$$

Function $f$ here is an example of what we call a *classifier*. The term comes from the goal of identifying whether an input sentence is grammatical or not using $f$. Knowing how to construct $f$ is discussed in the second part of this tutorial.

The second one can be regarded as finding a function $P(\cdot)$ that gives a numerical score to a given sentence $\mathbf{w}$ of its "degree of grammaticality". That is, the higher the score $P(\mathbf{w})$ the more grammatical it is. Thus, given two sentences $\mathbf{w}_1$ and $\mathbf{w}_2$, we say that $\mathbf{w}_1$ is more grammatical than $\mathbf{w}_2$ if and only if

$$P(\mathbf{w}_1) > P(\mathbf{w}_2). \tag{2}$$

If we stipulate some further conditions that $P$ is a probability function, that is,

(1) $P(\mathbf{w}) \geq 0$ for all $\mathbf{w}$ and

(2) $\int P(\mathbf{w})\mathrm{d}\mathbf{w} = 1$

then we call $P$ a *language model*. The question now is how do we construct $P$.

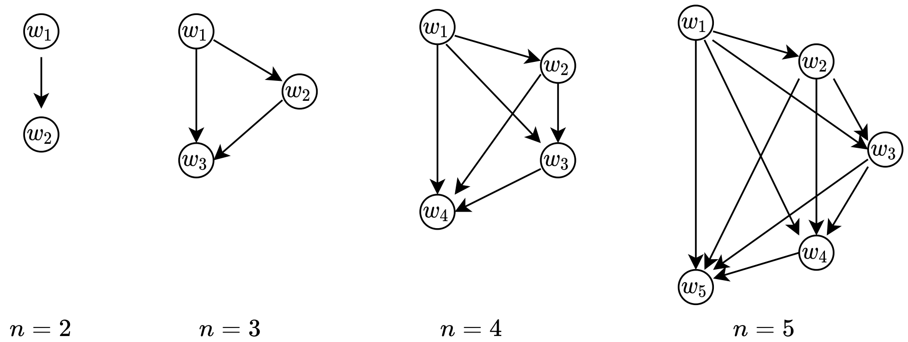



**Figure 1:** Probabilistic graphical presentation of $P(\mathbf{w})$ for $n = 2, 3, 4, 5$ number of tokens.

# 4 $N$-gram Language Model

Since the function $P$ in a language model is defined as a probability function, sentences w are treated as random vectors. Thus, using the product rule of probabilities

$$\begin{aligned} P(\mathbf{w}) &= P(w_1, w_2, \ldots, w_n) \\ &= P(w_1)P(w_2|w_1)P(w_3|w_1, w_2) \cdots P(w_n|w_1, w_2, \ldots, w_{n-1}). \end{aligned}$$

If we let $\mathbf{w}_{1:k} := (w_1, \ldots, w_k)$, then we can write more compactly the above product as

$$P(\mathbf{w}) = \prod_{k=1}^{n} P(w_k|\mathbf{w}_{1:k-1}) \tag{3}$$

where $(w_r|\mathbf{w}_{u:v}) = w_r$ if $u > v$.

The question of interest now is "*How do we estimate the probabilities?*". The simplest way to do this is to resort to *counts*. Let's first add some additional notations in our arsenal.

| Notation | Description |
|---|---|
| $C(w_a)$ | number of times the token $w_a$ occurs in the corpus |
| $C(w_a w_b)$ | number of times the sequence of tokens $w_a w_b$ occurs in the corpus |
| $C(\mathbf{w}_{1:k})$ | number of times the sequence of tokens $w_{1:k}$ occurs in the corpus |

**Example 4.1.** Consider the brown corpus data set available for download in Kaggle (Brown Corpus Data, 2018). The full code is presented in Appendix A.

```r
tokens_dt
```

```
#> Source: local data table [1,208,536 x 3]
#> Call:   `_DT2`
#>
#>       id label    token
#>    <int> <chr>    <chr>
#> 1      1 religion <s>
#> 2      1 religion furthermore
#> 3      1 religion ,
#> 4      1 religion as
#> 5      1 religion an
#> 6      1 religion encouragement
#> # ... with 1,208,530 more rows
#>
#> # Use as.data.table()/as.data.frame()/as_tibble() to access results
```

Now let us consider counting the token $w = (\texttt{in})$.

```r
tokens_dt %>%
  filter(token == "in") %>%
  count(token) %>%
  as_tibble
```

```
#> # A tibble: 1 x 2
#>   token     n
#>   <chr> <int>
#> 1 in    21337
```

Thus, $C(\texttt{in}) = 21,337$.

Now let us try to count the token sequence $\mathbf{w} = (\texttt{in}, \texttt{the})$.

```r
tokens_dt %>%
  mutate(token2 = lead(token, default = "")) %>%
  filter(token == "in" & token2 == "the") %>%
  count(token, token2) %>%
  as_tibble -> count_in_the

count_in_the
```

```
#> # A tibble: 1 x 3
#>   token token2     n
#>   <chr> <chr>  <int>
#> 1 in    the     6025
```

Thus, $C(\texttt{in}) = 6,025$.

One can verify using similar code the following counts:

$$C(\texttt{of}, \texttt{the}) = 9,717$$
$$C(\texttt{to}, \texttt{the}) = 3,484$$
$$C(\texttt{to}, \texttt{be}) = 1,718$$

With this $C$-notation we can estimate[1] the probabilities as

$$P(w_k|\mathbf{w}_{1:(k-1)}) = \frac{P(w_k, \mathbf{w}_{1:(k-1)})}{P(\mathbf{w}_{1:(k-1)})} = \frac{P(\mathbf{w}_{1:k})}{P(\mathbf{w}_{1:(k-1)})} = \frac{C(\mathbf{w}_{1:k})}{C(\mathbf{w}_{1:(k-1)})} \tag{4}$$

However, it would be impractical to apply this formula because almost any long stretch of token sequence rarely occur in a given corpus, i.e., it would result to zero counts. For example, from the given brown corpus above, one can show (See Appendix B) the following count:

$$C(\texttt{in}, \texttt{the}, \texttt{garden}, \texttt{just}, \texttt{outside}, \texttt{city}) = 0$$

With this practical restriction, instead of computing probabilities from past $(n-1)$ token sequences we simply approximate it with the $N$ token neighbors to the left, i.e.,

$$P(w_n|\mathbf{w}_{1:(n-1)}) \approx P(w_n|\mathbf{w}_{(n-N+1):(n-1)}).$$

This simplifies the model given by Eq (3) into

$$P(\mathbf{w}) = \prod_{k=1}^{n} P(w_k|\mathbf{w}_{(k-N+1):(k-1)}) \tag{5}$$

where $(w_r|\mathbf{w}_{u:v}) = w_r$ if $u > v$.

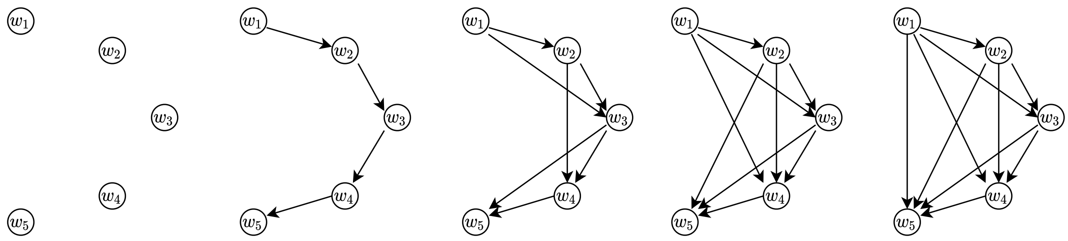
Equation (5) is called the $N$-*gram model*.

- When $N = 1$, we have a *unigram* and Eq. (5) simplifies to

$$P(\mathbf{w}) = \prod_{k=1}^{n} P(w_k) \tag{6}$$

  noting that $P(w_k|\mathbf{w}_{k:(k-1)}) = P(w_k)$.

[1] See SNLP p.33.




$n = 5$, $N = 1$, unigram | $n = 5$, $N = 2$, bigram | $n = 5$, $N = 3$, trigram | $n = 5$, $N = 4$, quadgram | $n = 5$, $N = 5$, pentagram

**Figure 2:** Probabilistic graphical presentation of an $N$-gram model with $n = 5$ tokens and for $N = 1, 2, 3, 4, 5$.

- When $N = 2$, we have a *bigram* and Eq. (5) simplifies to

$$P(\mathbf{w}) = \prod_{k=1}^{n} P(w_k | w_{k-1}) \tag{7}$$

  noting that $P(w_k | \mathbf{w}_{(k-1):(k-1)}) = P(w_k | w_{k-1})$.

- When $N = 3$, we have a *trigram* and Eq. (5) simplifies to

$$P(\mathbf{w}) = \prod_{k=1}^{n} P(w_k | w_{k-2} w_{k-1}) \tag{8}$$

  noting that $P(w_k | \mathbf{w}_{(k-2):(k-1)}) = P(w_k | w_{k-2} w_{k-1})$.

We can now simplify the probability provided in Eq. (4):

- When $N = 1$,

$$P(w_k) = \frac{C(w_k)}{\sum_{k=1}^{n} C(w_k)} \tag{9}$$

- When $N = 2$,

$$P(w_k | w_{k-1}) = \frac{C(w_{k-1} w_k)}{C(w_{k-1})} \tag{10}$$

- When $N = 3$,

$$P(w_k | w_{k-2} w_{k-1}) = \frac{C(w_{k-2} w_{k-1} w_k)}{C(w_{k-2} w_{k-1})} \tag{11}$$

**Example 4.2** (Brown Corpus Unigram)**.** Let us construct a unigram model from Brown corpus `tokens_dt`.

```
tokens_dt %>%
  count(token) %>%
  arrange(desc(n)) %>%
  ungroup %>%
  mutate(unigram_prob = n/sum(n)) %>%
  rename(unigram = token) -> unigram_dt

unigram_dt %>%
  as_tibble
```

```
#> # A tibble: 49,780 x 3
#>    unigram     n unigram_prob
#>    <chr>   <int>        <dbl>
#>  1 the     69971      0.0579
#>  2 ,       58306      0.0482
#>  3 </s>    57340      0.0474
#>  4 <s>     57340      0.0474
#>  5 of      36412      0.0301
#>  6 and     28870      0.0239
#>  7 to      26158      0.0216
#>  8 a       23196      0.0192
#>  9 in      21337      0.0177
#> 10 that    10594      0.00877
#> # ... with 49,770 more rows
```

**Example 4.3** (Brown Corpus Bigram)**.** Let us construct a bigram model from Brown corpus `tokens_dt`.

```
tokens_dt %>%
  mutate(w2 = lead(token, default = "")) %>%
  filter(token!="</s>" & w2!="<s>") %>%
  count(token, w2) %>%
  arrange(desc(n)) %>%
  ungroup %>%
  group_by(token) %>%
  mutate(bigram_prob = n/sum(n)) %>%
  ungroup %>%
  mutate(bigram = str_c(token,"_",w2)) %>%
  select(bigram,n,bigram_prob) -> bigram_dt

bigram_dt %>%
  as_tibble
```

```
#> # A tibble: 430,392 x 3
```

```
#>    bigram      n bigram_prob
#>    <chr>    <int>       <dbl>
#>  1 of_the   9717       0.267
#>  2 <s>_the  6857       0.120
#>  3 ,_and    6300       0.108
#>  4 in_the   6025       0.282
#>  5 <s>_``   4168       0.0727
#>  6 ,_the    3783       0.0649
#>  7 to_the   3484       0.133
#>  8 <s>_he   2983       0.0520
#>  9 on_the   2466       0.366
#> 10 and_the  2247       0.0778
#> # ... with 430,382 more rows
```

**Example 4.4** (Brown Corpus Trigram)**.** Let us construct a trigram model from Brown corpus `tokens_dt`.

```
tokens_dt %>%
  mutate(w2 = lead(token, default = "")) %>%
  mutate(w3 = lead(w2, default = "")) %>%
  filter((token!="</s>" & w2!="<s>") & (w2!="</s>" & w3!="<s>")) %>%
  count(token, w2, w3)  %>%
  arrange(desc(n)) %>%
  ungroup %>%
  group_by(token, w2) %>%
  mutate(trigram_prob = n/sum(n)) %>%
  ungroup %>%
  mutate(trigram = str_c(token,"_",w2,"_",w3)) %>%
  select(trigram,n,trigram_prob) -> trigram_dt

trigram_dt %>%
  as_tibble
```

```
#> # A tibble: 842,641 x 3
#>    trigram                n trigram_prob
#>    <chr>              <int>        <dbl>
#>  1 ,_and_the            656        0.104
#>  2 <s>_it_is            586        0.287
#>  3 <s>_it_was           562        0.275
#>  4 <s>_``_i             435        0.104
#>  5 <s>_in_the           434        0.248
#>  6 one_of_the           403        0.574
#>  7 <s>_he_was           363        0.122
#>  8 ''_,_he              346        0.170
#>  9 the_united_states    336        0.855
```

```
#> 10 ,_however_,          321        0.879
#> # ... with 842,631 more rows
```

**Example 4.5** (Sentence Probability)**.** Under unigram, bigram and trigram models, compute the probability of the sentence

```
<s> I want to eat food </s>
```

Note first that we need to normalize the sentence, e.g., convert all to lowercase letters. Thus, we have $\mathbf{w} = (\texttt{<s>}, \texttt{i}, \texttt{want}, \texttt{to}, \texttt{eat}, \texttt{food}, \texttt{</s>})$.

- Under the unigram model:

$$P(\mathbf{w}) = \prod_{k=1}^{6} P(w_k) = P(\texttt{<s>})P(\texttt{i})P(\texttt{want})P(\texttt{to})P(\texttt{eat})P(\texttt{food})P(\texttt{</s>}).$$

```r
sen <- "<s> I want to eat food </s>" %>%
  str_split("\\s+") %>%
  unlist %>%
  str_to_lower(.)

unigram_dt %>%
  filter(unigram %in% sen) %>%
  as_tibble -> pu_sen

pu_sen
```

```
#> # A tibble: 7 x 3
#>   unigram     n unigram_prob
#>   <chr>   <int>        <dbl>
#> 1 </s>    57340    0.0474
#> 2 <s>     57340    0.0474
#> 3 to      26158    0.0216
#> 4 i        5165    0.00427
#> 5 want      328    0.000271
#> 6 food      147    0.000122
#> 7 eat        61    0.0000505
```

Thus,

$$\begin{aligned}
&P(\texttt{<s> I want to eat food </s>}) \\
&\quad = P(\texttt{<s>})P(\texttt{i})P(\texttt{want})P(\texttt{to})P(\texttt{eat})P(\texttt{food})P(\texttt{</s>}) \\
&\quad = (0.0474)(0.0043)(0.0003)(0.0216)(0.0001)(0.0001)(0.0474) \\
&\quad = 3.4697174 \times 10^{-19} \\
&\quad \approx 0
\end{aligned}$$

The last expression just shows that under the unigram model, finding the sentence `<s> I want to eat food </s>` from this corpus is almost impossible.

- Using the bigram model:

$$
\begin{aligned}
P(\mathbf{w}) &= \prod_{k=1}^{6} P(w_k|w_{k-1}) \\
&= P(\texttt{<s>})P(\texttt{i}|\texttt{<s>})P(\texttt{want}|\texttt{i})P(\texttt{to}|\texttt{want}) \\
&\quad P(\texttt{eat}|\texttt{to})P(\texttt{food}|\texttt{eat})P(\texttt{</s>}|\texttt{food}) \\
&= \frac{C(\texttt{<s>})}{N_T} \cdot \frac{C(\texttt{<s>},\texttt{i})}{C(\texttt{<s>})} \cdot \frac{C(\texttt{i},\texttt{want})}{C(\texttt{i})} \cdot \frac{C(\texttt{want},\texttt{to})}{C(\texttt{want})} \\
&\quad \frac{C(\texttt{to},\texttt{eat})}{C(\texttt{to})} \cdot \frac{C(\texttt{eat},\texttt{food})}{C(\texttt{eat})} \cdot \frac{C(\texttt{food},\texttt{</s>})}{C(\texttt{food})} \\
&= (0.0474458)(0.02397976979421)(0.010648596321394) \\
&\quad (0.496951219512195)(0.000879272115605169)(0.122448979591837) \\
&= 6.4822787 \times 10^{-10}
\end{aligned}
$$

where $P(\texttt{<s>})$ is computed as follows

```r
unigram_dt %>%
  filter(unigram == "<s>") %>%
  as_tibble
```

```
#> # A tibble: 1 x 3
#>   unigram     n unigram_prob
#>   <chr>    <int>        <dbl>
#> 1 <s>      57340       0.0474
```

while $P(w_k|w_{k-1})$ are computed as follows.

```r
bigram_dt %>%
  filter(bigram %in% c("<s>_i","i_want","want_to",
                       "to_eat","eat_food","food_</s>")) %>%
  as_tibble
```

```
#> # A tibble: 5 x 3
#>   bigram        n bigram_prob
#>   <chr>     <int>       <dbl>
#> 1 <s>_i      1375    0.0240
#> 2 want_to     163    0.497
#> 3 i_want       55    0.0106
#> 4 to_eat       23    0.000879
#> 5 food_</s>    18    0.122
```

Observe that the probability $(6.4822787 \times 10^{-10})$ under the bigram model is quite higher by a factor of $1.868244 \times 10^{9}$ compared to that computed under the unigram.

- Using the trigram model:

$$
\begin{aligned}
P(\mathbf{w}) &= \prod_{k=1}^{6} P(w_k|w_{k-1}) \\
&= P(\texttt{<s>})P(\texttt{i}|\texttt{<s>})P(\texttt{want}|\texttt{<s>},\texttt{i})P(\texttt{to}|\texttt{i},\texttt{want}) \\
&\qquad P(\texttt{eat}|\texttt{want},\texttt{to})P(\texttt{food}|\texttt{to},\texttt{eat})P(\texttt{</s>}|\texttt{eat},\texttt{food}).
\end{aligned}
$$

However, there's a problem. We don't have counts for the trigrams `want_to_eat`, `to_eat_food`, and `eat_food_</s>`. As shown by the following code:

```
trigram_dt %>%
  filter(trigram %in% c("want_to_eat",
                        "to_eat_food",
                        "eat_food_</s>")) %>%
  as_tibble
```

```
#> # A tibble: 0 x 3
#> # ... with 3 variables: trigram <chr>, n <int>, trigram_prob <dbl>
```

This is a typical example of the practical restrictions that the longer the token sequence that rare it is to find the corpus. Computing probabilities is such cases will be answered in the lesson about *smoothing*.

**Example 4.6.** Suppose we want to know what is the most likely next word of the token sequence

$$\texttt{on} \quad \texttt{the} \quad \underline{\quad ? \quad}$$

This problem is equivalent to finding $w^*$ where

$$w^* = \arg\max_w P(w|\texttt{on},\texttt{the})$$

Using the estimates for the trigram:

$$w^* = \arg\max_w \frac{C(\texttt{on},\ \texttt{the},\ w)}{C(\texttt{on},\ \texttt{the})} = \arg\max_w C(\texttt{on},\ \texttt{the},\ w)$$

where the second equality follows since the denominator does not depend on $w$.

Implementing the counting R, we have

```r
trigram_dt %>%
  filter(str_detect(trigram, "^on_the_")) %>%
  arrange(desc(trigram_prob)) -> prob_on_the

prob_on_the %>%
  as_tibble
```

```
#> # A tibble: 1,262 x 3
#>    trigram            n trigram_prob
#>    <chr>          <int>        <dbl>
#>  1 on_the_other      99      0.0401
#>  2 on_the_basis      57      0.0231
#>  3 on_the_floor      28      0.0114
#>  4 on_the_ground     27      0.0109
#>  5 on_the_part       26      0.0105
#>  6 on_the_same       22      0.00892
#>  7 on_the_side       22      0.00892
#>  8 on_the_way        21      0.00852
#>  9 on_the_road       20      0.00811
#> 10 on_the_first      19      0.00770
#> # ... with 1,252 more rows
```

From the results above, we find that the most likely word that follows is $w^* =$ `other` with $C(\texttt{on}, \texttt{the}, w^*) = 99$ or, in terms of probability,

$$P(w^*|\texttt{on}, \texttt{the}) = 0.040146.$$

# 5 Generating Random Sentences

With the given $N$-gram model $P$, we can generate, i.e. sample, sequence of tokens $\mathbf{w}$.

## 5.1 Unigram Sentence Generation

For unigram models, we use the following algorithm to generate a sentence:

1. Set $w_1 =$ `<s>`.
2. Generate $U \sim \mathsf{Unif}(0, 1)$.
3. If $\sum_{k=1}^{i-1} P(w_k) < U \leq \sum_{k=1}^{i} P(w_k)$, then set $w = w_i$ as the next token in the sequence.
4. While $w_i \neq$ `</s>`, repeat steps 2 and 3.

Let us now try illustrating this algorithm using the unigram model obtained from the Brown Corpus above. Since a assign $w_1 =$ <s> with probability 1, we need calibrate the probabilites as follows

```
unigram_dt %>%
  filter(unigram!="<s>") %>%
  mutate(unigram_prob = n/sum(n)) -> n_unigram_dt

n_unigram_dt %>%
  as_tibble
```

```
#> # A tibble: 49,779 x 3
#>    unigram     n unigram_prob
#>    <chr>   <int>        <dbl>
#>  1 the     69971       0.0608
#>  2 ,       58306       0.0506
#>  3 </s>    57340       0.0498
#>  4 of      36412       0.0316
#>  5 and     28870       0.0251
#>  6 to      26158       0.0227
#>  7 a       23196       0.0201
#>  8 in      21337       0.0185
#>  9 that    10594       0.00920
#> 10 is      10109       0.00878
#> # ... with 49,769 more rows
```

Next, we need to form new columns for $\sum_{k=1}^{i-1} P(w_k)$ and $\sum_{k=1}^{i} P(w_k)$.

```
n_unigram_dt %>%
  filter(unigram != "<s>") %>%
  mutate(P = cumsum(unigram_prob)) %>%
  mutate(LP = lag(P, default = 0)) -> Punigram
```

We now initialize our sentence vector with $w_1 =$ <s> and start the loop as stated in Steps 2 to 4:

```
set.seed(11)
sen_gen <- "<s>"
repeat {
  U <- runif(1)
  Punigram %>%
    filter(LP < U & U <= P) %>%
    pull(unigram) -> w
  if (w != "</s>"){
```

```
    sen_gen <- str_c(sen_gen," ",w)
    print(sen_gen)
  } else {
    sen_gen <- str_c(sen_gen," ",w)
    break
  }
}
```

```
#> [1] "<s> in"
#> [1] "<s> in the"
#> [1] "<s> in the first"
#> [1] "<s> in the first the"
#> [1] "<s> in the first the ,"
#> [1] "<s> in the first the , nonmetallic"
#> [1] "<s> in the first the , nonmetallic ,"
#> [1] "<s> in the first the , nonmetallic , is"
#> [1] "<s> in the first the , nonmetallic , is fellows"
```

```
sen_gen
```

```
#> [1] "<s> in the first the , nonmetallic , is fellows </s>"
```

## 5.2 Bigram Sentence Generation

```
bigram_dt %>%
  separate(bigram,c("w1","w2"), sep="_", remove = FALSE) %>%
  group_by(w1) %>%
  arrange(desc(w1)) %>%
  mutate(P = cumsum(bigram_prob)) %>%
  mutate(LP= lag(P, default = 0)) %>%
  ungroup -> Pbigram

Pbigram %>%
  as_tibble
```

```
#> # A tibble: 430,392 x 7
#>    bigram              n bigram_prob w1         w2             P    LP
#>    <chr>           <int>       <dbl> <chr>      <chr>      <dbl> <dbl>
#>  1 {0,t}_,             1         1   {0,t}      ,            1     0
#>  2 zworykin_,          1         0.5 zworykin   ,            0.5   0
#>  3 zworykin_and        1         0.5 zworykin   and          1     0.5
#>  4 zwei_planeten       1         1   zwei       planeten     1     0
```

```
#>  5 zurich_,          1         0.5 zurich      ,           0.5    0
#>  6 zurich_by         1         0.5 zurich      by          1      0.5
#>  7 zurcher_of        2         1   zurcher     of          1      0
#>  8 zur_bestimmung    1         0.5 zur         bestimmung  0.5    0
#>  9 zur_khaneh        1         0.5 zur         khaneh      1      0.5
#> 10 zubkovskaya_,     1         0.5 zubkovskaya ,           0.5    0
#> # ... with 430,382 more rows
```

```
set.seed(18)
w <- sen_gen <- "<s>"
repeat {
  U <- runif(1)
  Pbigram %>%
    filter(w1==w & LP < U & U <= P) %>%
    pull(bigram) %>%
    str_split("_") %>%
    unlist -> wb

  sen_gen <- str_c(c(sen_gen, wb[2]), collapse = " ")
  print(sen_gen)
  w <- wb[2]
  if (w == "</s>"){
    break
  }
}
```

```
#> [1] "<s> whereas"
#> [1] "<s> whereas joe"
#> [1] "<s> whereas joe whippet"
#> [1] "<s> whereas joe whippet )"
#> [1] "<s> whereas joe whippet ) ,"
#> [1] "<s> whereas joe whippet ) , including"
#> [1] "<s> whereas joe whippet ) , including an"
#> [1] "<s> whereas joe whippet ) , including an ugly"
#> [1] "<s> whereas joe whippet ) , including an ugly and"
#> [1] "<s> whereas joe whippet ) , including an ugly and she"
#> [1] "<s> whereas joe whippet ) , including an ugly and she started"
#> [1] "<s> whereas joe whippet ) , including an ugly and she started building"
#> [1] "<s> whereas joe whippet ) , including an ugly and she started building </s
```

# Appendix A

The following is the R code deriving the `tokens_dt` data table.

```
suppressWarnings(
  suppressMessages(
    library(tidyverse)
    )
  )

tokens_dt <- dtplyr::lazy_dt(
  read.csv(
    file = "data/brown-corpus/brown.csv",
    header = TRUE
    )
  ) %>%
  select(tokenized_text, label) %>%
  mutate(id = seq_along(label)) %>%
  mutate(
    token = str_c("<s> ", tokenized_text, " </s>") %>%
          str_replace("[ [:punct:]]+ </s>$", " </s>") %>%
          str_to_lower %>%
          str_split("\\s+")
    )  %>%
  select(id, label, token) %>%
  as_tibble %>%
  unnest(token) %>%
  dtplyr::lazy_dt(.)
```

# Appendix B

The following code computes the count:

$$C(\texttt{in}, \texttt{the}, \texttt{garden}, \texttt{just}, \texttt{outside}, \texttt{city}) = 0$$

```
tokens_dt %>%
  mutate(token2 = lead(token, default = "")) %>%
  mutate(token3 = lead(token2, default = "")) %>%
  mutate(token4 = lead(token3, default = "")) %>%
  mutate(token5 = lead(token4, default = "")) %>%
  mutate(token6 = lead(token5, default = "")) %>%

  filter(
```

```
    token == "in"   &
    token2 == "the" &
    token3 == "garden" &
    token4 == "just" &
    token5 == "outside" &
    token6 == "city"
    ) %>%
  count(token,
        token2,
        token3,
        token4,
        token5,
        token6
        ) %>%
  as_tibble
```

# References


Brown Corpus Data, N.-D. (2018). Brown corpus of standard american english by w. N. Francis and h. Kucera 1964. *Kaggle*. Retrieved from https://www.kaggle.com/datasets/nltkdata/brown-corpus