

1 Initial Implementation

The following output was produced using `./randsent grammar 10`.

```
is it true that every sandwich under every floor kissed the sandwich ?
is it true that the pickle under the pickle on a delicious pickled president pickled the
    president under the pickle ?
the president on the president in every fine floor under the sandwich in every president
    under a chief of staff ate the floor under a chief of staff on the pickle on a floor in
    every chief of staff with the perplexed president on the president .
a chief of staff pickled the sandwich in a floor on the sandwich with the president with the
    fine floor in every sandwich in a floor on a pickle on every sandwich with every sandwich
    under the pickle under a sandwich on the president under a pickled sandwich in the
    delicious floor on the pickled pickled pickle under the chief of staff in every president
    on the perplexed pickle on the pickle in every president on every floor on a sandwich in
    the chief of staff in every pickle in the pickle with a floor on the sandwich with every
    floor on a president with a chief of staff with a floor on every floor in every chief of
    staff in a sandwich on every floor on the president on the president under the chief of
    staff in every pickle under every chief of staff with every pickle with every pickle on
    every president on the pickle .
a pickle on the sandwich with every sandwich on the chief of staff under the chief of staff
    on a chief of staff under every pickle with every president in a pickle under every pickle
    on a chief of staff with the pickle under every pickle on a chief of staff under every
    delicious sandwich under a pickle with every president under the president in the sandwich
    in every pickle in the sandwich under the pickled chief of staff with every sandwich under
    the floor with a delicious floor with a pickle in a pickled chief of staff under every
    floor under every sandwich in every floor under every chief of staff under every chief of
    staff in every sandwich with the chief of staff with the president under the pickle under
    every pickle with every pickle in the floor in every president with the president with a
    pickled sandwich on a president under a floor in the fine floor in a chief of staff under
    a chief of staff under every floor in a perplexed fine floor with a president with the
    president under a president in the delicious delicious floor on every pickle on the pickle
    under a fine pickle with the chief of staff with every chief of staff on the sandwich in
    every sandwich on every chief of staff in the sandwich under a pickle in every delicious
    perplexed sandwich with the floor with a perplexed chief of staff with the president on
    the floor on the pickle in a pickle under every floor ate the chief of staff !
every delicious president wanted the fine pickle .
is it true that the pickle wanted the sandwich ?
a floor wanted a president !
every pickle with every chief of staff kissed a delicious fine president with a president .
a chief of staff with a floor pickled the perplexed perplexed fine sandwich .
```

(Note: the indentation, or “tabs”, seen in this LaTeX document were not present in the generation of these sentences. They are there to help distinguish each sentence from the other.)

2 Initial Discussion

(a) The rule

$$\text{NP} \rightarrow \text{NP PP}$$

is responsible for all these long sentences.

To explain why, let's first name the nonterminals that are not preterminals "true-nonterminals" because they do not turn into terminals in one step. Mostly, a preterminal (except "Noun") will eventually turn into one word in the final sentence, while a true-nonterminal will always turn into more than one word. Generally, true-nonterminals have more potential to generate long sentences.

Looking at all the rules that turn true-nonterminals into other symbols, we can find two rules turn a true-nonterminal to more than one true-nonterminals. they are

(i) $\text{S} \rightarrow \text{NP VP}$,

(ii) $\text{NP} \rightarrow \text{NP PP}$.

However, rule (i) is the only rule that turns **S** into other symbols—there are no other possible options **S** is given. (And with our grammar, we are *always* given **S** right after **ROOT**.)

When given **NP**, it's a bit different. We have the possibility of creating terminals (i.e. turning **NP** into **Det Noun**). However, if we use rule (ii) when **NP** is given, then **NP** becomes **NP PP**. Because **PP** can only turn into **Prep NP**, we are then left with an expression containing *two* **NPs**: **NP Prep NP**.

Rule (ii) creates something sort of like an explosion: one **NP** could give us two, and two **NPs** could give us even more. And this is why we generate so many long sentences!

(b) The intuition here is that we have six rules—all of equal probability—that turn the symbol **Noun** into other symbol(s), with only one of those rules generating an **Adj**. (That rule is **Noun** \rightarrow **Adj Noun**).

Now let's do some calculations. Suppose there are n **Nouns**. Because the probability of any rule being applied is uniformly random, we can choose a rule for **Noun** uniformly randomly.

After one "step" of the rule is applied to all n **Nouns** in the sentence, we will have an expected $\frac{5n}{6}$ words and $\frac{n}{6}$ **Adj Nouns**. After the second step, we have an expected $\frac{5n}{6^2}$ **Adj Nouns** and $\frac{n}{6^2}$ **Adj Adj Nouns**.

In fact, we can generalize the expected number of consecutive **Adjs**:

$$\begin{aligned} \mathbf{E}[\text{Adj}] &= \frac{n}{6^2} + \frac{n}{6^3} + \frac{n}{6^4} + \cdots \\ &= \sum_{i=2}^{\infty} \frac{n}{6^i} \\ &= \left(\sum_{i=1}^{\infty} \frac{n}{6^i} \right) - \frac{n}{6} \\ &= \frac{n}{5} - \frac{n}{6} \\ &= \frac{n}{30} \end{aligned}$$

That is, the expected number of consecutive **Adjs** appearing next to each other is only one-thirtieth of the number of **Nouns** that appear in the sentence.

(c) This has been implemented in `Grammar.py`.

- (d) To reduce the number of long sentences, we have to reduce the possibility that a nonterminal changes into another nonterminal. To increase the number of adjectives, we can simply increase the probability that a **Noun** changes into a **Adj Noun**.

In our grammar, we changed the rules:

- (a) $1 \text{ NP} \rightarrow \text{NP PP}$ into $0.2 \text{ NP} \rightarrow \text{NP PP}$,
(b) $1 \text{ Noun} \rightarrow \text{Adj Noun}$ into $3 \text{ Noun} \rightarrow \text{Adj Noun}$.

Change (i) solves the problem of long sentences, in that it reduces the likelihood of a single nonterminal turning into multiple nonterminals.

Change (ii) solves the problem of too few adjectives, in that it increases the likelihood of adjective generation.

- (e) The changes made for **grammar2** are:

```
2    ROOT  S .
#    ...
0.3  Verb  pickled
0.5  Det   every
0.5  Noun  pickle
0.5  Noun  chief of staff
1.5  Adj   fine
1.5  Prep  on
1.5  Prep  in
```

The reasoning for these changes are:

- a sentence ending with a period is more common than either a sentence ending with an exclamation point or a sentence ending with a question mark,
- the verb “picked” is not frequently used in the English language,
- the frequencies are picked relative to the frequencies of other words of the same type.

Ten sentences generated with the new grammar are pasted below:

```
the chief of staff wanted a pickle .
the president pickled a president .
the floor wanted every president on the sandwich on the sandwich !
is it true that the president wanted every pickle ?
a delicious pickled pickle wanted the sandwich !
a chief of staff ate the sandwich .
the pickled floor under the president with the pickled floor wanted the fine fine floor !
a perplexed president under the fine pickled perplexed delicious sandwich ate the floor
    with the pickle with the floor .
a chief of staff understood a floor !
a floor kissed a perplexed perplexed sandwich .
```

(Note: the indentation, or “tabs”, seen in this LaTeX document were not present in the generation of these sentences. They are there to help distinguish each sentence from the other.)

3 Modifications

4 Trees, Pretty Prints, Brackets

5 Alternate Derivations

(a)

(b)

6 Parse

- (a) The parser does not always recover the original derivation that was “intended” by **randsent**. In fact, we can give an example of a sentence for which **parse** “misunderstood” and found an alternative derivation.

Our sentence was generated via the following tree:

```
(ROOT (S (N_clause that
          (S (N_clause that
                (S (NP (NP (NP (NP (Noun_pro Sally))
                           (Conj and)
                           (NP (Det a)
                               (Noun (Adj (Adv really)
                                           (Adj fine))
                                           (Noun pickle))))
                           (Conj and)
                           (NP (NP (Det the)
                               (Noun sandwich))
                               (PP (Prep in)
                                   (NP (NP (NP (Noun_pro Xiaochen))
                                           (PP (Prep in)
                                               (NP (Noun_pro Sally))))
                                   (PP (Prep with)
                                       (NP (Det the)
                                           (Noun sandwich))))))
                           (PP (Prep on)
                               (NP (Det every)
                                   (Noun pickle))))
                           (VP (Verb wanted)
                               (NP (Noun_pro Sally))))
                (VP (VP (Verb kissed)
                        (NP (Det a)
                            (Noun (Adj (Adv really)
                                        (Adj fine))
                                        (Noun sandwich))))
                    (Conj and)
                    (VP (Verb wanted)
                        (NP (Noun_pro Xiaochen))))
            (VP (V_intran flew)))
        .)
```

(This problem is continued on the next page.)

However, **parse** found an alternate derivation via the following tree:

```
(ROOT (S (N_clause that
          (S (N_clause that
              (S (NP (NP (NP (NP (NP (Noun_pro Sally))
                          (Conj and)
                          (NP (NP (Det a)
                              (Noun (Adj (Adv really)
                                          (Adj fine))
                                          (Noun pickle)))
                              (Conj and)
                              (NP (Det the)
                                  (Noun sandwich))))
                          (PP (Prep in)
                              (NP (Noun_pro Xiaochen))))
                          (PP (Prep in)
                              (NP (Noun_pro Sally))))
                          (PP (Prep with)
                              (NP (Det the)
                                  (Noun sandwich))))
                          (PP (Prep on)
                              (NP (Det every)
                                  (Noun pickle))))
                          (VP (Verb wanted)
                              (NP (Noun_pro Sally))))
              (VP (VP (Verb kissed)
                      (NP (Det a)
                          (Noun (Adj (Adv really)
                                      (Adj fine))
                                      (Noun sandwich))))
                      (Conj and)
                      (VP (Verb wanted)
                          (NP (Noun_pro Xiaochen))))))
          (VP (V_intran flew)))
      .)
```

(Note that the two trees are not equivalent.)

The reason for this is that the “correct” tree of a specific sentence could be ambiguous, depending on the grammar that was used. That is, if there is more than one unique tree that leads to the generated sentence, the parser could guess a tree that is different from what was actually used by the generator.

In our example, the sentence was composed of several subtrees first split using the NP Conj NP rule, and then later on split using the NP PP rule. However, the parser guessed that the sentence was composed of subtrees that consecutively split using the NP PP rule, and did not use the NP Conj NP rule to split until much later on.

- (b) There are 5 ways to analyze the noun phrase

every sandwich with a pickle on the floor under the chief of staff
using the original grammar **grammar**.

Let's first define the term "modify" as follows:

Assume that the rule $NP_1 \rightarrow NP_2 PP$ exists in our grammar. Then PP modifies some $N \in \text{Noun}$ if NP_2 at some point generates N .

In this sentence:

- (i) the PP "with a pickle" can only modify the Noun "sandwich",
- (ii) the PP "on the floor" can modify the Nouns "sandwich" or "pickle",
- (iii) the PP "under the chief of staff" can modify the Nouns "sandwich", "pickle", or "floor".

By this logic, there would seem to be $1 \cdot 2 \cdot 3 = 6$ ways to parse this sentence.

However, when "on the floor" modifies "sandwich", "under the chief of staff" cannot simultaneously modify "sandwich".

Thus, there remains 5 possible ways to parse the sentence.

- (c)

- (d) (i) Why is $p(\text{best parse})$ so small? Well, we can examine the probabilities used to generate this sentence, by first taking a look at the probabilities given from **grammar**.

$$\begin{aligned} \mathbf{Pr}[\text{this sentence}] &= \mathbf{Pr}[\text{ROOT}] \cdot \mathbf{Pr}[\text{S} \mid \text{ROOT}] \cdot \mathbf{Pr}[\text{NP VP} \mid \text{S}] \cdot \mathbf{Pr}[\text{Det Noun} \mid \text{NP}] \cdot \\ &\quad \mathbf{Pr}[\text{the} \mid \text{Det}] \cdot \mathbf{Pr}[\text{president} \mid \text{Noun}] \cdot \mathbf{Pr}[\text{Verb NP} \mid \text{VP}] \cdot \\ &\quad \mathbf{Pr}[\text{ate} \mid \text{Verb}] \cdot \mathbf{Pr}[\text{Det Noun} \mid \text{NP}] \cdot \mathbf{Pr}[\text{the} \mid \text{Det}] \cdot \mathbf{Pr}[\text{sandwich} \mid \text{Noun}] \end{aligned}$$

Fair enough. Those are a lot of expressions multiplied together, which will probably produce a small number. But what does it equate, exactly?

$$\begin{aligned} \mathbf{Pr}[\text{ROOT}] &= 1 \\ \mathbf{Pr}[\text{S} \mid \text{ROOT}] &= 1/3 \\ \mathbf{Pr}[\text{NP VP} \mid \text{S}] &= 1 \\ \mathbf{Pr}[\text{Det Noun} \mid \text{NP}] &= 1/2 \\ \mathbf{Pr}[\text{the} \mid \text{Det}] &= 1/3 \\ \mathbf{Pr}[\text{president} \mid \text{Noun}] &= 1/6 \\ \mathbf{Pr}[\text{Verb NP} \mid \text{VP}] &= 1 \\ \mathbf{Pr}[\text{ate} \mid \text{Verb}] &= 1/5 \\ \mathbf{Pr}[\text{Det Noun} \mid \text{NP}] &= 1/2 \\ \mathbf{Pr}[\text{the} \mid \text{Det}] &= 1/3 \\ \mathbf{Pr}[\text{sandwich} \mid \text{Noun}] &= 1/6 \end{aligned}$$

$$\implies \mathbf{Pr}[\text{this sentence}] = 5.144032922 \times 10^{-5}$$

Let's call this number p , for simplicity purposes.

Using Bayes' Rule, we know that the probability that this is the best parse is equal to the

probability that this sentence occurs, multiplied with the probability that we get the best parse *conditioned* on the event of this sentence occurring. That is,

$$p(\text{best_parse}) = p(\text{best_parse}|\text{sentence}) \cdot p(\text{sentence})$$

We know that $p(\text{best_parse}|\text{sentence}) = 1$, and we just calculated p . Therefore,

$$\begin{aligned} p(\text{best_parse}) &= 1 \cdot \Pr[\text{this sentence}] \\ &= 5.144032922 \times 10^{-5} \end{aligned}$$

Now let's think about why $p(\text{sentence})$ is equivalent to $p(\text{best_parse})$.

We can again use Bayes' Rule to mathematically argue this. But we can also think intuitively. $p(\text{best_parse})$ is the probability that the given parse is the best possible parse. If we know that there is *one* best parse for the specific generated sentence, it *must* be that the probability that this parse is the best parse of *all* best parses for *all* possible sentences is equivalent to the probability of generating this sentence.

Finally, we can think about why $p(\text{best_parse}|\text{sentence}) = 1$. Using the same arguments as above, we know that there is only one possible “best” parse of the sentence. Therefore, the probability that the given parse is equivalent to the *only* best parse is 100%.

We can then check our reasoning:

```
$ ./parse -c -g grammar
the president ate the sandwich .
```

and notice that there indeed is only 1 possible parse of the given sentence.

- (ii) Looking at the possibilities from **grammar**, we see that there is actually one other parse that could exist:

```
(ROOT (S (NP (NP (Det every)
                (Noun sandwich))
            (PP (Prep with)
                (NP (NP (Det a)
                    (Noun pickle))
                    (PP (Prep on)
                        (NP (Det the)
                            (Noun floor))))))
            (VP (Verb wanted)
                (NP (Det a)
                    (Noun president))))
    .)
```

Now looking at the probabilities from **grammar** we see that the event that “on the floor” is applied to “a pickle” is equally as likely as the event that “on the floor” is applied to “every sandwich with a pickle”.

Because there are only two possible events, and their probabilities are equal, it must be that the probability of each one happening is exactly 0.5.

- (iii)
(iv)
(v)

(b)

(c)

(d)

(e)

(f)

(g)