Individual Assignment Report

Jinghan Wang z5286124

Abstract

Language models have indeed made several tasks easier than ever. You can now type “Write a poem about kangaroos in the style of William Wordsworth” in the ChatGPT tool, and it generates exactly that for you. How easy is that! I wonder what challenges a probabilistic poetry generator might face. Ah, implementing one will help me understand attention in Transformer much better!

---

1. Part I

1.1. Preprocessing

According to the requirements, after loading the data only the text part from the second section is extracted and tokenized. Since all digits must be removed during tokenization, the process first calls

`re.sub(r'\d+', '', text)`

. Then, because the word pattern is designed to match words composed of letters and allows for one or more occurrences of either curly or straight single quotes within the word, the regex pattern

`r"['']?[A-Za-z]+(?:['’][A-Za-z]+)*['’]?"`

is used. Additionally, to capture symbols such as question marks and exclamation points, or words with only one letter that are ignored by the regular expression above, use

`[^\\sa-zA-Z]`

to capture.

1.2. Bigrams Model Implementation

All matched words are extracted and sorted. For every pair of words in the vocabulary, a count is initialized to 1. The code then iterates through each sentence’s bigrams and updates the corresponding counts. Finally, normalization is performed to convert these counts into probabilities, thereby building a language model that utilizes Laplace smoothing.

1.3. Prediction

This section of the code generates poetry. It splits the input sentence into four fixed lines based on the specified number of words per line to ensure there are no errors due to insufficient input words. For each line, all words except the last one are generated by predicting based on the previous word. When generating the final word of each line, if the rhyme scheme (ABAB or AABB) applies, the last word from the corresponding previous line is used to obtain candidate rhyming words, and then a word matching one of these candidates is selected from the predictions; if no match is found, a regular prediction is used instead. Finally, the four lines are printed in order with appropriate spacing between words.

2. Part II

2.1. Output

Below is the output text extracted from the program screenshot 1

```
-s 5 -w "and when they" -n 10 -r ABAB
step 1
and when they philip sheaf splitting was shaped vacant already
lane wastes prizes trust breast latmian drum moses noon abominations
delight experunce unspoken slaying clang sports carved peace burial steady
thrill de delay worship throne farr who’s moods remembrance generations

-s 5 -w "and when they" -n 10 -r AABB
step 1
and when they philip sheaf splitting was shaped vacant already
lane wastes prizes trust breast latmian drum moses noon steady
delight experunce unspoken slaying clang sports carved peace burial ’em
thrill de delay worship throne farr who’s moods remembrance i’m

-s 5 -w "and when large" -n 10 -r ABAB
step 1
and when large perpetual sheaf splitting was shaped vacant already
lane wastes prizes trust breast latmian drum moses noon abominations
delight experunce unspoken slaying clang sports carved peace burial steady
thrill de delay worship throne farr who’s moods remembrance generations

-s 5 -w "and when part" -n 10 -r ABAB
step 1
and when part perish sheaf splitting was shaped vacant already
lane wastes prizes trust breast latmian drum moses noon abominations
delight experunce unspoken slaying clang sports carved peace burial steady
thrill de delay worship throne farr who’s moods remembrance generations

-s 3 -w "and when part" -n 6 -r ABAB
step 1
and when part doubtfle most gracious
overleaning pick bat about sweetest embraces
done your latter sweep leaves port
cherub’s plume thoughts merciful she paces

-s 20 -w "and when part" -n 6 -r ABAB
step 1
and when part turned rich sleeping
turn embraces poet turn throne none
coming ho yon bores foam weighed
journeys daylight flue two fray run
```

```
(py312) → ass1 git:(master) × python boilerplate.py -s 5 -w "and when they" -n 10 -r ABAB
step 1
and when they philip sheaf splitting was shaped vacant already
lane wastes prizes trust breast latmian drum moses noon abominations
delight experunce unspoken slaying clang sports carved peace burial steady
thrill de delay worship throne farr who's moods remembrance generations
(py312) → ass1 git:(master) × python boilerplate.py -s 5 -w "and when they" -n 10 -r AABB
step 1
and when they philip sheaf splitting was shaped vacant already
lane wastes prizes trust breast latmian drum moses noon steady
delight experunce unspoken slaying clang sports carved peace burial 'em
thrill de delay worship throne farr who's moods remembrance i'm
(py312) → ass1 git:(master) × python boilerplate.py -s 5 -w "and when large" -n 10 -r ABAB
step 1
and when large perpetual sheaf splitting was shaped vacant already
lane wastes prizes trust breast latmian drum moses noon abominations
delight experunce unspoken slaying clang sports carved peace burial steady
thrill de delay worship throne farr who's moods remembrance generations
(py312) → ass1 git:(master) × python boilerplate.py -s 5 -w "and when part" -n 10 -r ABAB
step 1
and when part perish sheaf splitting was shaped vacant already
lane wastes prizes trust breast latmian drum moses noon abominations
delight experunce unspoken slaying clang sports carved peace burial steady
thrill de delay worship throne farr who's moods remembrance generations
(py312) → ass1 git:(master) × python boilerplate.py -s 3 -w "and when part" -n 6 -r ABAB
step 1
and when part doubtfle most gracious
overleaning pick bat about sweetest embraces
done your latter sweep leaves port
cherub's plume thoughts merciful she paces
(py312) → ass1 git:(master) × python boilerplate.py -s 20 -w "and when part" -n 6 -r ABAB
step 1
and when part turned rich sleeping
turn embraces poet turn throne none
coming ho yon bores foam weighed
journeys daylight flue two fray run
```

Figure 1. Output

2.2. Discussion

Regarding the output, first compare the results under the same seed with the same input phrase but different rhyme schemes.

```
ABAB: and when they philip sheaf splitting was
    shaped vacant already
AABB: and when they philip sheaf splitting was
    shaped vacant already

ABAB: lane wastes prizes trust breast latmian
    drum moses noon abominations
AABB: lane wastes prizes trust breast latmian
    drum moses noon steady

ABAB: delight experunce unspoken slaying clang
    sports carved peace burial steady
AABB: delight experunce unspoken slaying clang
    sports carved peace burial 'em

ABAB: thrill de delay worship throne farr who's
    moods remembrance generations
AABB: thrill de delay worship throne farr who's
    moods remembrance i'm
```

It can be observed that since the seed is the same, the initial generated content is identical until the last word of the second line. At that point, due to the AABB scheme, the requirement for the last word to rhyme with the last word of the first line causes a difference in generation. Interestingly, after that, the outputs become identical again, with only the last words of the third and fourth lines differing because of the rhyme constraints.

A similar phenomenon is observed with the same seed but different input phrases. When the rhyme scheme is the same, there may be slight differences in one or two words immediately following the input, but all subsequent words remain the same.

```
they:  and when they philip sheaf splitting was
    shaped vacant already
large: and when large perpetual sheaf splitting
    was shaped vacant already

they:  lane wastes prizes trust breast latmian
    drum moses noon abominations
large: lane wastes prizes trust breast latmian
    drum moses noon abominations

they:  delight experunce unspoken slaying clang
    sports carved peace burial steady
large: delight experunce unspoken slaying clang
    sports carved peace burial steady

they:  thrill de delay worship throne farr who's
     moods remembrance generations
large: thrill de delay worship throne farr who's
     moods remembrance generations
```

According to the code, most of the randomness in the program is produced by an algorithm that yields consistent results with the same seed. Additionally, the use of Laplace smoothing, which initializes every count to 1, means that even word pairs that rarely or never appear in the training data still have a nonzero probability, resulting in a very uniform overall probability distribution. This uniformity, coupled with the deterministic nature of the seed, leads to consistently similar generated outputs.

```
-s 20 -w "and when part" -n 6 -r ABAB

and when part doubtfle most gracious
overleaning pick bat about sweetest embraces
done your latter sweep leaves port
cherub's plume thoughts merciful she paces
```

In addition, the output shows that “gracious” and “port” do not rhyme. Since the dataset is small and contains a limited vocabulary, many common words are not included, which means that the derived rhyming words for “gracious” may not intersect with the words in the model’s vocabulary. To ensure the program runs normally and does not affect subsequent output, it does not force the use of a rhyming word that is not in the model but instead continues to randomly select one, ensuring normal operation.

3. Part III

The model in this task and the masked attention mechanism in Transformers both attempt to predict subsequent content based on preceding context. In this task, the prediction relies solely on the previous word, and masked attention exhibits similar behavior by ensuring that the current prediction is not influenced by future information.

In addition, prosody is also akin to the attention mechanism. Prosody represents the parts that require special focus in this task, liberating the program from the original fixed prediction method and directing it to generate rhythmically consistent words. Similarly, the attention mechanism filters out the key elements from a vast amount of information, enabling the model to concentrate on the most meaningful content.