First-two-char Input Method with N-gram Model and Transformer-based Language Model

Issa Sugiura
Graduate School of Informatics
Kyoto University
sugiura.issa.63n@st.kyoto-u.ac.jp

Abstract

Input method is a method to input text to a computer. In this paper, we focus on first-two-char input method (e.g. Input: "fi ", "tw", "ch", "in", "me" -> Output: "first", "two", "character", "input", "method"). We compare the performance of the N-gram model and the Transformer-based language model for this task. In the experiment, we use the Shakespeare dataset as the training data and the test data. The results show that the 2-gram model achieves the best performance on the test data. The inference time of the N-gram model is much faster than the Transformer-based language model. Therefore, we conclude that the N-gram model is more suitable for this task. Code is available at https://github.com/speed1313/input-method

1 Introduction

Input methods are techniques used to enter text into a computer. Humans typically use keyboards or touchscreens for this purpose. However, typing all characters individually can be time-consuming. To address this issue, two-character input methods (e.g., Input: "fi", "tw", "ch", "in", "me" -> Output: "first", "two", "character", "input", "method") can be used. To implement input methods, language models are used. In this context, the model predicts the corresponding word based on the previous and current characters. The N-gram model is simple and widely used for this task. However, there are limitations to the N-gram model. For instance, it cannot capture long-range dependencies. To overcome this limitation, the Transformer model [1] can be used. The Transformer is a neural network model that can capture long-range dependencies using the attention mechanism. The Transformer-based language model is widely used for natural language generation, such as in GPT [2]. The Transformer-based language model can be used for input method in a sentence-to sentence translation format. For example, [3] utilizes ChatGPT for Kana-Kanji translation, a Japanese input method that converts kana (phonetic characters) into kanji (logographic characters). However, the Transformer-based language model can also be used for input method in a sequence-to-sequence translation format. In this paper, We apply the Transformer-based language model for two-character input methods in a sequence-to-sequence translation format. In our experiment, we compare the performance of two models; the N-gram model and the Transformer-based language model. To evaluate the performance of these models, we use the Shakespeare dataset[4]. The results show that the N-gram model achieves the best performance on the test data. Additionally, the inference time of the N-gram model is significantly faster than that of the Transformer-based language model.

2 Problem Statement

Input: A sequence of two characters $(c_1, c_2, ..., c_n)$

- Example: ("fi ", "tw", "ch", "in", "me")

Preprint.

- if the length of W_i is more than two, $c_i = W_i[:2]$, else, $c_i = W_i + \text{" "}$

Output: Corresponding words $(W_1, W_2, ..., W_n)$

- Example: ("fi", "two", "character", "input", "method")

Restrictions:

- We ignore non-alphabet characters. (e.g. "." is stripped)
- We ignore case sensitivity. (e.g. "A" is considered as "a")
- We can't use future information. (e.g. we can't use c_{i+1} to predict W_i)

3 Method

3.1 Tokenization

In this problem, the input is a sequence of two characters and the output is a sequence of corresponding words.

Therefore, we use two different tokenizer (TwoCharTokenizer and WordTokenizer) for the input and output, respectively.

TwoCharTokenizer The vocabulary of the TwoCharTokenizer is the set of all two characters.

```
vocab = {"a ", "b ", ..., "z ", "aa", "ab", ..., "zz"}
```

The TwoCharTokenizer converts the elements of the vocab to the corresponding tokens (e.g. "a " -> 0, "b " -> 1, ..., "aa" -> 26, "ab" -> 27, ..., "zz" -> 702)

WordTokenizer The vocabulary of the WordTokenizer is the set of all words in the dataset. Therefore, the vocabulary size depends on the dataset. The WordTokenizer converts the elements of the vocab to the corresponding tokens (e.g. "a" -> 0, "two" -> 32, "character" -> 443, "input" -> 34, "method" -> 55, ...)

3.2 N-gram Model

N-gram model is a statistical language model. It predicts the probability of the next word given the previous words. The probability is calculated by the following formula.

$$P(w_n|w_1, w_2, ..., w_{n-1}) = \frac{count(w_1, w_2, ..., w_n)}{count(w_1, w_2, ..., w_{n-1})}$$

where w_i is the i-th word in the sequence.

However, in this problem, we want to predict the corresponding word given the previous and current two characters. Therefore, we modify the formula as follows.

$$P(W_n|c_1, c_2, ..., c_n) = \frac{count(c_1, c_2, ..., c_n = W_n[:2])}{count(c_1, c_2, ..., c_n)}$$

where c_i is the i-th two characters in the sequence.

Prediction To predict the corresponding word, we calculate the probability of all candidate words and select the most probable one.

$$\hat{W_n} = \arg\max_{W_n} P(W_n|c_1, c_2, ..., c_n)$$

When the n is large, any n-gram may not be found in the training data. In this case, the probability is always zero. To avoid this problem, we use the following formula (also known as backoff [5]) when the probability is zero.

$$\hat{W_n} = \arg\max_{W_n} P(W_n|c_2, ..., c_n)$$

That is, we ignore the first two characters and predict the corresponding word based on the last n-1 characters.

3.3 Transformer-based Language Model

Transformer-based language model is a neural network model that predicts the probability of the next word given the previous words [2]. The probability is calculated by the following formula.

$$P(w_n|w_1, w_2, \ldots, w_{n-1}) = \frac{\exp(f(w_1, w_2, \ldots, w_{n-1})_{w_n})}{\sum_{w'} \exp(f(w_1, w_2, \ldots, w_{n-1})_{w'})}$$

where f is a neural network model.

However, in this problem, we want to predict the corresponding word given the previous and current two characters. Therefore, we modify the formula as follows.

$$P(W_n|c_1, c_2, ..., c_n) = \frac{\exp(f(c_1, c_2, ..., c_n)_{W_n})}{\sum_{W_i} \exp(f(c_1, c_2, ..., c_n)_{W_i})}$$

In this case, $f \in \mathbb{R}^{d_{in}} \rightarrow \mathbb{R}^{d_{out}}$, where d_{in} is the dimension of the input (the vocab size of the TwoCharTokenizer) and d_{out} is the vocabulary size of the candidate words (the vocab size of the WordTokenizer).

3.4 Model Architecture

Figure 1 shows the overview of the Transformer-based language model for first-two-char input method. This model takes the previous and current two characters as input and predicts the candidate token of the current two characters' token. We use NanoLM[1] as the decoder only transformer block.

4 Experiments

In this experiment, we compare the performance of the N-gram model and the Transformer-based language model.

4.1 Dataset

In our experiment, we use the Shakespeare dataset as the training data and the test data.

Shakespeare[2] This dataset is a txt file of the Shakespeare's works.

- Data size: 5.4MB
- Token (Word) size: 928017
- TwoCharTokenizer's vocab size: 26 + 26 * 26 = 702
- WordTokenizer's vocab size: 23685

We split the dataset into the training data and the test data with a ratio of 9:1.

Hyperparameters for Transformer-based Language Model To prepare the dataset for the Transformer-based language model, we choose block size (context length) $n = 4$ and the stride $s = 1$. By choosing the stride $s = 1$, we can use all the data in the training data. This is the same condition as the N-gram model. Hyperparameters of the Transformer-based language model are shown in Table 1.

4.2 Results

[1] `https://optax.readthedocs.io/en/latest/_collections/examples/nanolm.html`

[2] `https://ocw.mit.edu/ans7870/6/6.006/s08/lecturenotes/files/t8.shakespeare.txt`

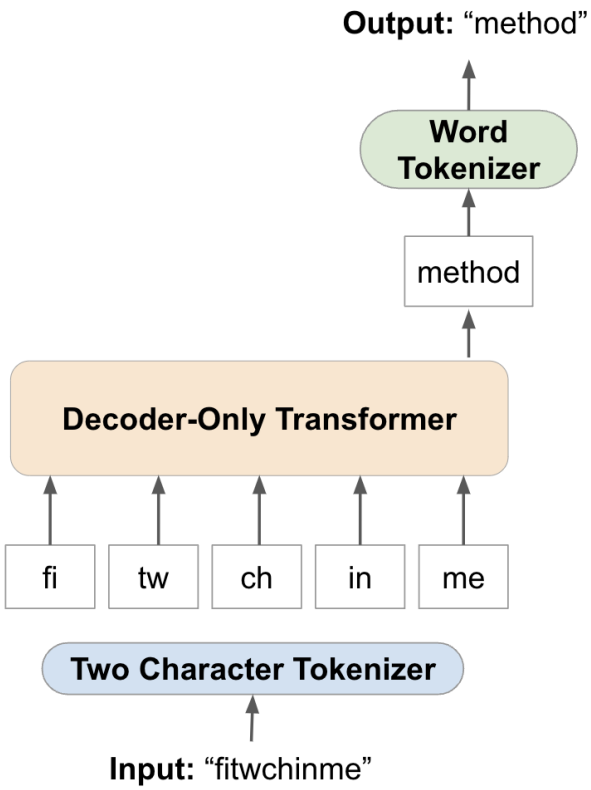


Figure 1: Transformer-based language model for first-two-char input method.

Table 1: Hyperparameters of the Transformer-based language model.

Hyperparameter	Value
Batch size	256
Training steps	5000
Learning rate	0.001
dropout rate	0.1
Number of layers	8
Embedding size	256
Head size	32
Number of heads	8

N-gram model Table 2 shows the accuracy of the N-gram model on the training data and the test data. The 2-gram model achieves the best performance on the test data. The accuracy of the N-gram model increases as the N increases. However, the test accuracy decreases when the N is 3 or more. This indicates that the model is overfitting to the training data when the N is 3 or more.

Transformer-based language model Table 3 shows the accuracy of the Transformer-based language model on the training data and the test data. When the block size is 16, the model achieves the best performance on the test data. test accuracy increases as the block size increases. However, when the block size is 32 or more, the test accuracy decreases. In contrast, the training accuracy increases as the block size increases. This indicates that the model is overfitting to the training data when the block size is 32 or more. Figure 2 shows the loss dynamics of the Transformer-based language model on the training data and the test data. We can see that the more the block size, the more the model overfits to the training data.

Table 2: Accuracy of the N-gram model.

N	Train Accuracy	Test Accuracy
1	0.472	0.473
2	0.549	0.522
3	0.765	0.505
4	0.965	0.499
5	0.997	0.499
6	1.000	0.499
7	1.000	0.499
8	1.000	0.499
9	1.000	0.499
10	1.000	0.499

Table 3: Accuracy of the Transformer-based language model.

Block Size	Train Accuracy	Test Accuracy
1	0.459	0.466
2	0.484	0.486
3	0.496	0.492
4	0.498	0.495
8	0.526	0.513
16	0.570	0.518
32	0.671	0.515
64	0.807	0.503
128	0.907	0.490

4.3 Training Time

The training time for the 10-gram model is approximately 7.0 seconds on a MacBook Air M3 with 16 GB of RAM. Training time of the Transformer-based language model on a single A100 GPU is shown in Table 4.

4.4 Inference Time

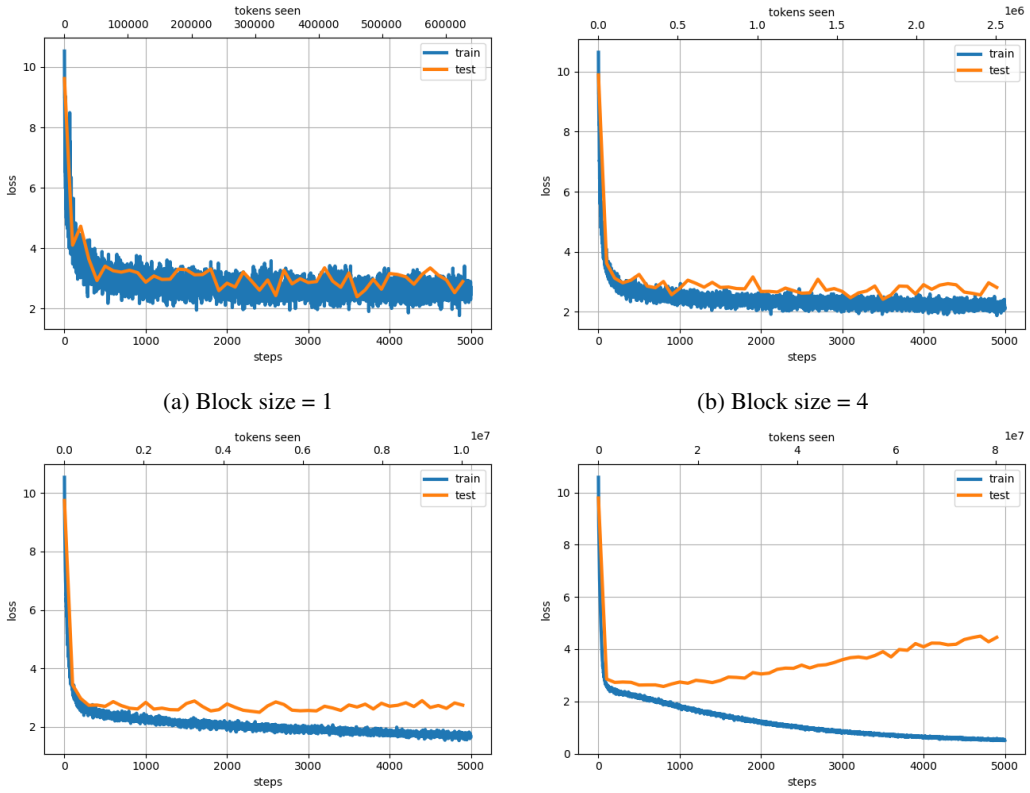
We measure the inference time of the N-gram model and the Transformer-based language model. The total inference time for each model when the n-gram or block size is 4 on the test data is shown in Table 5. We can see that the inference time of the N-gram model is much faster than the Transformer-based language model.

5 Conclusion

In this paper, we compare the performance of the N-gram model and the Transformer-based language model for the first-two-char input method task. We use the Shakespeare dataset as the training data and the test data. The results show that the 2-gram model achieves the best performance on the test data. The inference time of the N-gram model is much faster than the Transformer-based language model. We conclude that first-two-char input method is simpler than the next token prediction task, and the N-gram model is more suitable for this task.

Table 4: Training time of the Transformer-based language model.

Block Size	Training Time
4	0.5 minutes
32	1.5 minutes
128	5.0 minutes



(a) Block size = 1

(b) Block size = 4

(c) Block size = 16

(d) Block size = 128

Figure 2: Loss dynamics of the Transformer-based language model on the training data and the test data.

Table 5: Total inference time of the N-gram model and the Transformer-based language model on the test data (n-gram or block size = 4).

Model	Inference Time
N-gram model	0.11 seconds
Transformer-based language model	8.3 seconds

Limitations

In this paper, we use the Shakespeare dataset as the training data and the test data. The dataset is relatively small. As scaling laws suggest, the performance of the Transformer-based language model may improve with a larger dataset [6]. Therefore, we need to evaluate the performance of the Transformer-based language model on a larger dataset in the future.

References

[1] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[2] Alec Radford, Karthik Narasimhan, Tim Salimans, Ilya Sutskever, et al. Improving language understanding by generative pre-training. 2018.

[3] Miwa. Let's try kana-kanji conversion with chatgpt and gpt-4. May 2024.

[4] Project Gutenberg. t8.shakespeare.txt: The complete works of william shakespeare.

[5] S. Katz. Estimation of probabilities from sparse data for the language model component of a speech recognizer. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 35(3):400–401, 1987.

[6] Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361*, 2020.