Ch 03. N-gram Language Models

Hanseul Kim

January 2025

1 Summary

1.1 N-gram

The **n-gram** language modeling uses previous $n-1$ words(tokens) to predict the next word.
for example,

- **unigram** predicts just single word probability $p(w_i)$.
- **bigram** predicts the next word based on a previous word $p(w_i|w_{i-1})$.
- **trigram** predicts the next word based on two previous words $p(w_i|w_{i-1}, w_{i-2})$

Using n-gram modeling, we can use the chain rule of probability to **approximate** the probability of a given sentence by using **Markov assumption**.

$$P(w_1 \dots w_n) = P(w_1)P(w_2|w_1)P(w_3|w_{1:2})\dots P(w_n|w_{1:n-1})$$

$$= \prod_{k=1}^{b} P(w_k|w_{1:k-1})$$

$$P(w_n|w_{1:n-1}) \approx P(w_n|W_{n-N+1:n+1})$$

$$P(w_{1:n}) \approx \prod_{k-1}^{n} P(w_k|w_{k-1})$$

* It is also possible to use long context n-grams. see ∞-gram

1.2 Maximum likelihood estimation using frequency

One of the simplest kinds of language model can be made by estimating maximum likelihood of the next word based on its count in the training data.
For example in bigram model,

$$P(w_n|w_{n-1}) = \frac{C(w_{n-1}w_n)}{\sum_w C(w_{n-1}w)}$$

And when predicting the next word, we can pick the w_n with the highest $p(w_n|w_{n-1})$. In practice, dealing with just probability value might cause underflow when dealing with long contexts. So, in language modeling probabilities are stored as **log probabilities**. Which is beneficial because we can change the multiplication operation to simple addition.

$$p_1 \times p_2 = \exp(\log p_1 + \log p_2)$$

1.3 Evaluation

There is two kinds of evaluation methods. One is **extrinsic evaluation** which is done to evaluate final performance (based on each tasks) of the language model which is very expensive to do. Another method is to use **intrinsic evaluation** which just uses the probability outputs of the language model as a performance metric. Note that a higher intrinsic evaluation does not necessarily means higher extrinsic evaluation.

1.3.1 Perplexity

The most used intrinsic evaluation method is the **perplexity** which is defined,

$$\begin{aligned} \text{perplexity}(W) &= P(w_1 w_2 \dots w_n)^{-\frac{1}{N}} \\ &= \sqrt[N]{\frac{1}{P(w_1 w_2 \dots w_N)}} = \sqrt[N]{\prod_{i=1}^{N} \frac{1}{P(w_i|w_1 \dots w_{i-1})}} \end{aligned}$$

Note that **the lower the perplexity of a model on the data, the better the model.**
Example bigram perplexity is defined,

$$\text{perplexity}(W) = \sqrt[N]{\prod_{i=1}^{N} \frac{1}{P(w_i|w_{i-1})}}$$

1.4 Smoothing, Interpolation, and Backoff

There is a big problem in this kinds of n-gram modeling. When there is no such n-gram sequence in the training data, output probability is assigned 0, which leads to both over-fitting to the training dataset and impossible inference in unseen sequences.

1.4.1 Smoothing

One of the simplest solution is to smooth the probability which is just adding one to all of the words count. This method is called **Laplace smoothing**

$$P_{\text{Laplace}}(w_i) = \frac{c_i + 1}{N + V}$$

Or other ways looking at this is using **adjusted count**

$$c_i^* = (c_i + 1)\frac{N}{N+V}$$

$$P_i = \frac{c_i^*}{N}$$

Which can be seen as **discounting** the count by discount ration d_i

$$d_i = \frac{c_i^*}{c_i}$$

Example Laplace discount on bigram model is,

$$P_{\text{Laplace}}(w_n|w_{n-1}) = \frac{C(w_{n-1}w_n)+1}{\sum_w (C(w_n w)+1)} = \frac{C(w_{n-1}w_n)+1}{C(w_{n-1})+V}$$

This kind of smoothing can be generalized into **add-k smoothing**

$$P^*_{\text{Add-k}}(w_n|w_{n-1}) = \frac{C(w_{n-1}w_n)+k}{C(w_{n-1})+kV}$$

1.4.2 Interpolation

One ways to calculate the sentence probability in on seen data is to used hierarchy n-gram models, which is called **interpolation**.

$$\begin{aligned}\hat{P}(w_n|w_{n-2}w_{n-1}) &= \lambda_1 P(w_n) \\ &+\lambda_2 P(w_n|w_{n-1}) \\ &+\lambda_3 P(w_n|w_{n-2}w_{n-1}) \\ &(\lambda_1+\lambda_2+\lambda_3 = 1)\end{aligned}$$

or with learnable (or tuned) parameters,

$$\begin{aligned}\hat{P}(w_n|w_{n-2}w_{n-1}) &= \lambda_1(w_{n-2:n-1})P(w_n) \\ &+\lambda_2(w_{n-2:n-1})P(w_n|w_{n-1}) \\ &+\lambda_3(w_{n-2:n-1})P(w_n|w_{n-2}w_{n-1})\end{aligned}$$

1.4.3 Backoff

Another method is to use simpler model when the inference is impossible, which is called **backoff**. To adjust the different scale of the probability models, there should be a discount factor. However, for the simplest non-discounted model, we called it **stupid backoff** which is defined,

$$S(w_i|w_{i-N+1:i-1}) = \begin{cases} \frac{\text{count}(w_{i-N+1:i})}{\text{count}(w_{i-N+1:i-1})} & \text{if count}(w_{i-N+1:i}) > 0 \\ \lambda S(w_i|w_{i-N+2:i-1}) & \text{otherwise} \end{cases}$$

$\lambda = 0.4$ worked well from Brants et al. (2007)

2 Exercises

3.1

a) Write out the equation for trigram probability estimation.

$$P(w_n|w_{n-1}w_{n-2}) = \frac{C(w_{n-2}w_{n-1}w_n)}{C(w_{n-1}w_{n-2})}$$

b) Write out all the non-zero trigram probabilities for "I am Sam" corpus on page 35.

<s> I am Sam </s>
<s> Sam I am </s>
<s> I do not like green eggs and ham </s>

$$\begin{array}{ccc} P(\text{am} \mid <\text{s}>, \text{I}) = 0.5 & P(\text{Sam} \mid \text{I}, \text{am}) = 0.5 & P(</\text{s}> \mid \text{am}, \text{Sam}) = 1 \\ P(\text{I}| <\text{s}>, \text{Sam}) = 1 & P(\text{am}|\text{Sam}, \text{I}) = 1 & P(</\text{s}> |\text{I}, \text{am}) = 0.5 \\ P(\text{not}|\text{I}, \text{do}) = 1 & P(\text{like}|\text{do}, \text{not}) = 1 & P(\text{green}|\text{not}, \text{like}) = 1 \\ P(\text{egg}|\text{like}, \text{green}) = 1 & P(\text{and}|\text{green}, \text{egg}) = 1 & P(\text{ham}|\text{egg}, \text{and}) = 1 \\ P(</\text{s}> |\text{and}, \text{ham}) = 1 & & \end{array}$$

3.2

a) Calculate the probability of the sentence *i want chinese food.* Give two probabilities, one using Fig 3.2 and the useful probabilities just below it on page 37.
* Fig 3.2 usage (page 37)

$$P(\text{i want chinese food}) = P(\text{want}|\text{I})P(\text{chinese}|\text{want})P(\text{food}|\text{chinese})$$

$$= 0.33 * 0.0065 * 0.52 = 0.0011154$$

b) and another using the add-1 smoothed table in Fig 3.7. Assume the additional add-1 smoothed probabilities $P(\text{i}| <\text{s}>) = 0.19$ and $P(</\text{s}> |\text{food}) = 0.40$ * Fig 3.7 usage

$$P(\text{i want chinese food}) = P(\text{want}|\text{I})P(\text{chinese}|\text{want})P(\text{food}|\text{chinese})$$

$$= 0.21 * 0.0029 * 0.052 = 0.000031668$$

Which is less biased to training data. Note that both needs to be multiplied $P(\text{i}| <\text{s}>)P(</\text{s}> |\text{food}) = 0.19 * 0.40 = 0.076$ for start/end token consideration.

3.3

Which of the two probabilities you computed in the previous exercise is higher, unsmoothed or smoothed? Explain why.
Like I said before, unsmoothed is higher due to less overfitting to training dataset.

3.4

We are given the following corpus, modified from the one in the chapter:

<s> I am Sam </s>
<s> Sam I am </s>
<s> I am Sam </s>
<s> I do not like green eggs and Sam </s>

Using a bigram language model with add-one smoothing, what is $P(\text{Sam}|\text{am})$?

$$P(\text{Sam}|\text{am}) = \frac{C(\text{am Sam}) + 1}{\sum_w C(\text{am } w) + V}$$

$$V = 9$$

$$= \frac{2+1}{3+9} = 0.25$$

3.5

Suppose we didn't use the end-symbol $< /\text{s} >$. Train an unsmoothed bigram grammer on the following training corpus without using the end-symbol $< /\text{s} >$:

<s> a b
<s> b b
<s> b a
<s> b a

Demonstrate that your bigram model does not assign a single probability distribution across all sentence lengths by showing that the sum of the probability of the four possible 2 word sentences over the alphabet {a, b} is 1.0, and the sum of the probability of all possible 3 word sentences over the alphabet {a,b} is also 1.0

$$P(\text{a}|<\text{s}>) = 0.5 \quad P(\text{b}|\text{a}) = 0.5 \quad P(\text{b}|<\text{s}>) = 0.5$$
$$P(\text{b}|\text{b}) = 0.5 \quad P(\text{a}|\text{b}) = 0.5 \quad P(\text{a}|\text{a}) = 0.5$$

$$\sum_{w_1 \in \{a,b\}} \sum_{w_2 \in \{a,b\}} P(w_1 w_2) = 0.5 * 0.5 * 4 = 1$$

$$\sum_{w_1 \in \{a,b\}} \sum_{w_2 \in \{a,b\}} \sum_{w_3 \in \{a,b\}} P(w_1 w_2 w_3) = 0.5 * 0.5 * 0.5 * 2^3 = 1$$

3.6

Suppose we train a trigram language model with add-one smoothing on a given corpus. The corpus contains V word types. Express a formula for estimating $P(w_3|w_1, w_2)$, where w_3 is a word which follows the bigram (w_1, w_2), in terms of various n-gram counts and V

$$P(w_3|w_1, w_2) = \frac{C(w_1, w_2, w_3) + 1}{\sum_{w \in V} C(w_1, w_2, w) + V}$$

3.7

We are given the following corpus, modified from the one in the chapter:

<s> I am Sam </s>
<s> Sam I am </s>
<s> I am Sam </s>
<s> I do not like green eggs and Sam </s>

If we used linear interpolation smoothing between a maximum-likelihood bigram model and maximum-likelihood unigram model with $\lambda_1 = 0.5$ and $\lambda_2 = 0.5$, what is $P(\text{Sam}|\text{am})$ Include $< \text{s} >$ and $< /\text{s} >$ tokens in the count.

$$P(\text{Sam}|\text{am}) = \lambda_1 P(\text{Sam}) + \lambda_2 P(\text{Sam}|\text{am})$$

$$V = 11$$

$$= 0.5 * \frac{3+1}{20+11} + 0.5 * \frac{2+1}{3+14} = \frac{161}{1054}$$

3.8

Write a program to compute unsmooth unigrams and bigrams.
codes at here

3.9

Run your n-gram program on two different small corpora of your choice. Now compare the statistics of two corpora. What are the differences in the most common unigrams between the two? How about interesting differences in bi-gram?
codes at here

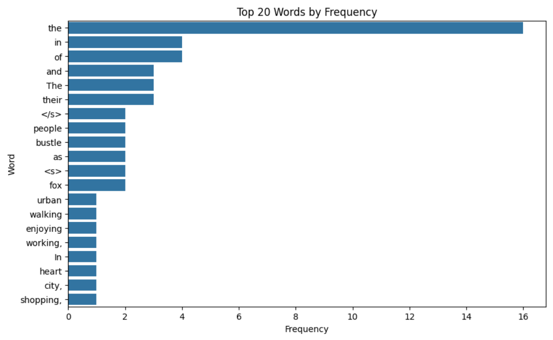


Figure 1: Word frequency of unigram poet model

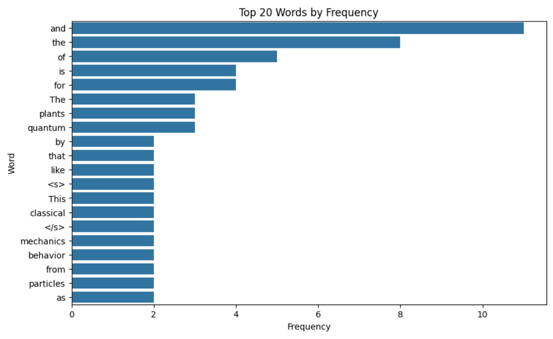


Figure 2: Word frequency of unigram science model

There seems to be similar top 1 to 5 words which are commonly used in english language. Other words seems to be dependent on context.

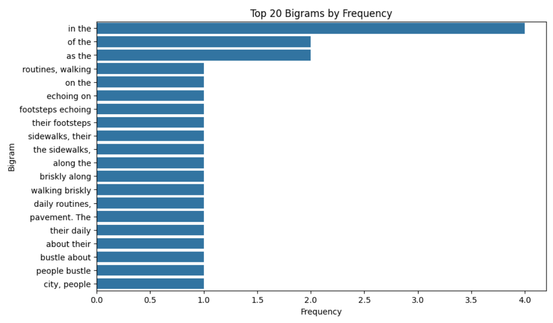


Figure 3: Words frequency of bigram poet model

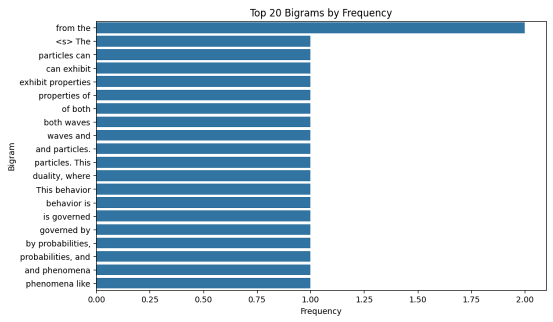


Figure 4: Words frequency of bigram science model

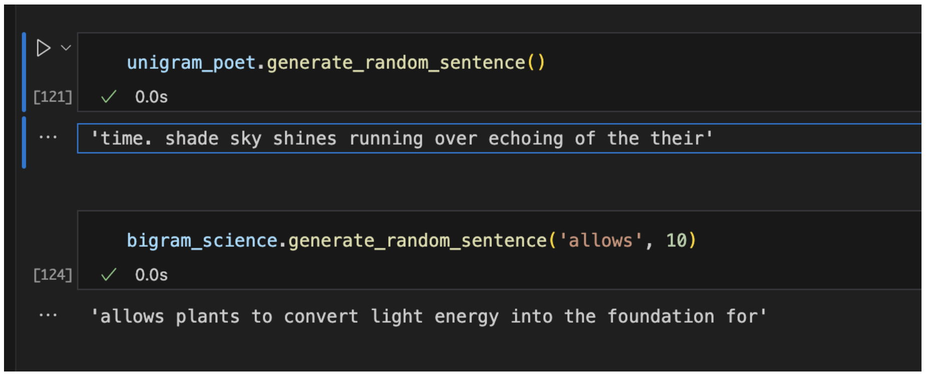
Bigram models seem to have less common words frequency, which means that it is biased(overfitted) to training dataset.

3.10

Add an option to your program to generate random sentences.
codes at here

3.11

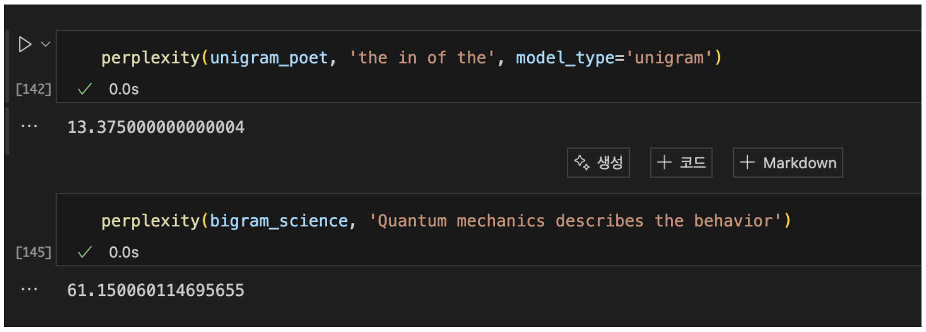
Add an option to your program to compute the perplexity of a test set
. codes at here



```
[121]
unigram_poet.generate_random_sentence()
✓ 0.0s
'time. shade sky shines running over echoing of the their'
```

```
[124]
bigram_science.generate_random_sentence('allows', 10)
✓ 0.0s
'allows plants to convert light energy into the foundation for'
```

Figure 5: Random sentences generated with unigram and bigram models



```
[142]
perplexity(unigram_poet, 'the in of the', model_type='unigram')
✓ 0.0s
13.375000000000004
```

```
[145]
perplexity(bigram_science, 'Quantum mechanics describes the behavior')
✓ 0.0s
61.150060114695655
```

Figure 6: perplexity

3.12

You are given a training set of 100 numbers that consists of 91 zeros and 1 each of the other digits 1-9. Now we see the following test set: 0 0 0 0 0 3 0 0 0 0. What is the unigram perplexity?

$$\text{perplexity}(W) = P(w_1 w_2 \dots w_n)^{-\frac{1}{N}} = \sqrt[N]{\frac{1}{P(w_1 w_2 \dots w_N)}} = \sqrt[N]{\prod_{i=1}^{N} \frac{1}{P(w_i | w_1 \dots w_{i-1})}}$$

$$= \left(\frac{1}{\frac{91}{100}^9 \frac{1}{100}}\right)^{\frac{1}{10}} = 1.73$$