Natural Language Processing
自然語言處理

黃瀚萱
Department of Computer Science
National Chengchi University
2020 Fall

Lesson 4

Language Modeling

Schedule

Date	Topic
9/16	Introduction
9/23	Linguistic Essentials
9/30	Collocation
10/7	**Language Model**
10/14	Word Sense Disambiguation
10/21	~~**NLP and Cybersecurity**~~
10/28	~~**Text Classification**~~
11/4	POS Tagging
11/11	Midterm Exam

Schedule

Date	Topic
9/16	Introduction
9/23	Linguistic Essentials
9/30	Collocation
10/7	**Language Model**
10/14	Word Sense Disambiguation
10/21	Text Classification
10/28	**Invited Talk: Tutorial for NLP and Cybersecurity (Term Project)**
11/4	POS Tagging
11/11	Midterm Exam

Schedule

Date	Topic
11/18	Chinese Word Segmentation
11/25	Word Embeddings
12/2	Neural Networks for NLP
12/9	Parsing
12/16	Discourse Analysis
12/23	**Invited Talk**
12/30	Final Project Presentation I
1/6	Final Project Presentation II
1/13	Final Exam

Agenda

- Language modeling
    - Chain rule
- Markov assumption
    - N-gram Models
- Estimation of N-gram probabilities
    - Maximum likelihood estimation
- Neural network based language modeling
- Smoothing
    - Modeling unseen words
- Evaluation
- Text classification with language modeling

Language Modeling

- Language modeling is a probability distribution over sequences of words.
  - About to *model* the relationship between a word/character with its **contextual information** in a *language*.
- Context (上下文、脈絡)
  - The information that are related to something and that help you to understand it.
  - The **words** that come just before and after a word or sentence and that help you understand its meaning
    - The surrounding words
    - In particular, the **preceding** words

Probabilistic Language Modeling

- Measure the probability of a piece of text (a sequence of words):

$$P(W) = P(w_1, w_2, w_3, ..., w_n)$$

- Prediction of the next word

$$P(w_{n+1}) = \arg\max_{w \in V} P(w | w_1, w_2, w_3, ..., w_n)$$

- Only the sequential relationship is taken into account.
  - Lack of the syntactic information from the tree-like structure.

Traditional Applications

- Optical character recognition
  - Improving the accuracy for the uncertain recognitions.
- Speech recognition
  - Resolving the ambiguity
  - *P*(“I saw a fan”) vs *P*(“eyes awe of an”)

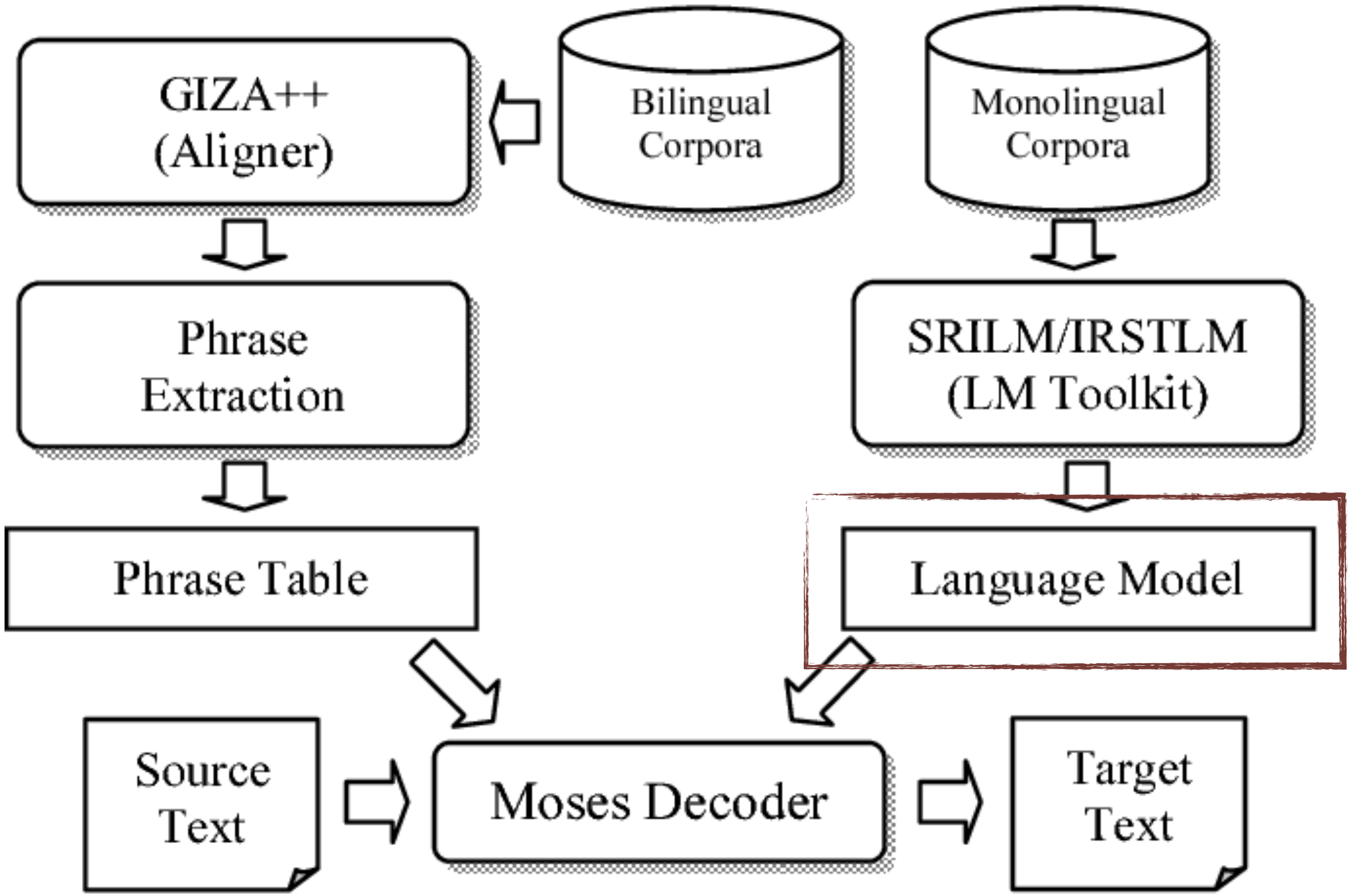
Traditional Applications

- Machine translation
  - Generating a smoother translation
  - 他是一個好人 => P(“he is a nice person”) vs P(“he is a good person”)
- Grammatical error correction
  - He is in the bus
  - He is at the bus
  - He is on the bus

Applications

- Spell correction
  - *You should have followed my ~~advise~~.*
  - P(my advice) > P(my advise)
- Speech recognition
  - P(I saw a van) > P(eyes awe of an)

Language Model in Machine Translator



How to Compute P(S)

- How to compute the joint probabilities of the following to sentences?
  - *P*(“The dog ate my cake”)
  - *P*(“The cake ate my dog”)

Chain Rule

- Conditional probabilities and joint probabilities

$$P(A|B) = \frac{P(A,B)}{P(B)}$$

$$P(A,B) = P(B)P(A|B)$$

- More than two variables

$$P(A,B,C,D) = P(A)P(B|A)P(C|A,B)P(D|A,B,C)$$

- *P*(“The dog ate my cake”) = *P*(The) *P*(dog | The) *P*(ate | The dog) *P*(my | The dog ate) *P*(cake | The dog ate my)

Chain Rule

- In general, the probability of a sequence $x_1, x_2, x_3, \ldots, x_n$ can be computed as

-

$$P(x_1, x_2, x_3, ..., x_n)$$
$$= P(x_1)P(x_2|x_1)P(x_3|x_1, x_2)...P(x_n|x_1, x_2, ..., x_{n-1})$$
$$= \prod_{i=1}^{n} P(w_i|w_1, w_2, ..., w_{i-1})$$

Estimation of the Probabilities

- *P*(“The dog ate my cake”) = *P*(The) *P*(dog | The) *P*(ate | The dog) *P*(my | The dog ate) *P*(cake | The dog ate my)

$$P(A|B) = \frac{P(A,B)}{P(B)}$$

$$P(\text{dog}|\text{The}) = \frac{P(\text{The dog})}{P(\text{The})}$$

$$P(\text{The dog}) = \frac{C(\text{The dog})}{N}$$

$$P(\text{The}) = \frac{C(\text{The})}{N}$$

Counted from a corpus

Sparsity of Long Sequence

- *P*(dog | The very small and very special cake ate my)
- Practically, there are too many possible long sentences.
  - No or only rare data in the corpus
  - Leading to a zero count

$$
\begin{aligned}
&P(\text{The very small and very special cake ate my}) \\
&= \frac{C(\text{The very small and very special cake ate my})}{N} \\
&\approx 0
\end{aligned}
$$

Markov Assumption

- To group similar long sequences by only considering the prior **local** context.
  - The last few words

$$
\begin{aligned}
&P(\text{dog}|\text{The very small and very special cake ate my}) \\
&= \frac{P(\text{The very small and very special cake ate my dog})}{P(\text{The very small and very special cake ate my})} \\
&\approx \frac{P(\text{very special cake ate my dog})}{P(\text{special cake ate my})} \\
&\approx \frac{P(\text{ate my dog})}{P(\text{ate my})}
\end{aligned}
$$

Markov Assumption

- The conditional probability of a long sequence can be approximated by using a shorter context (n-grams)

$$P(w_i|w_1, w_2, w_3, ..., w_{i-1}) \approx P(w_i|w_{i-k}, w_{i-k+1}, ..., w_{i-1})$$

- The probability of a sequence can be computed by the conditional n-gram probabilities of its components.

$$P(w_1, w_2, w_3, ..., w_n)$$
$$\approx \prod_i P(w_i|w_{i-k}, w_{i-k+1}, ..., w_{i-1})$$

The Simplest Language Model: Unigram Model

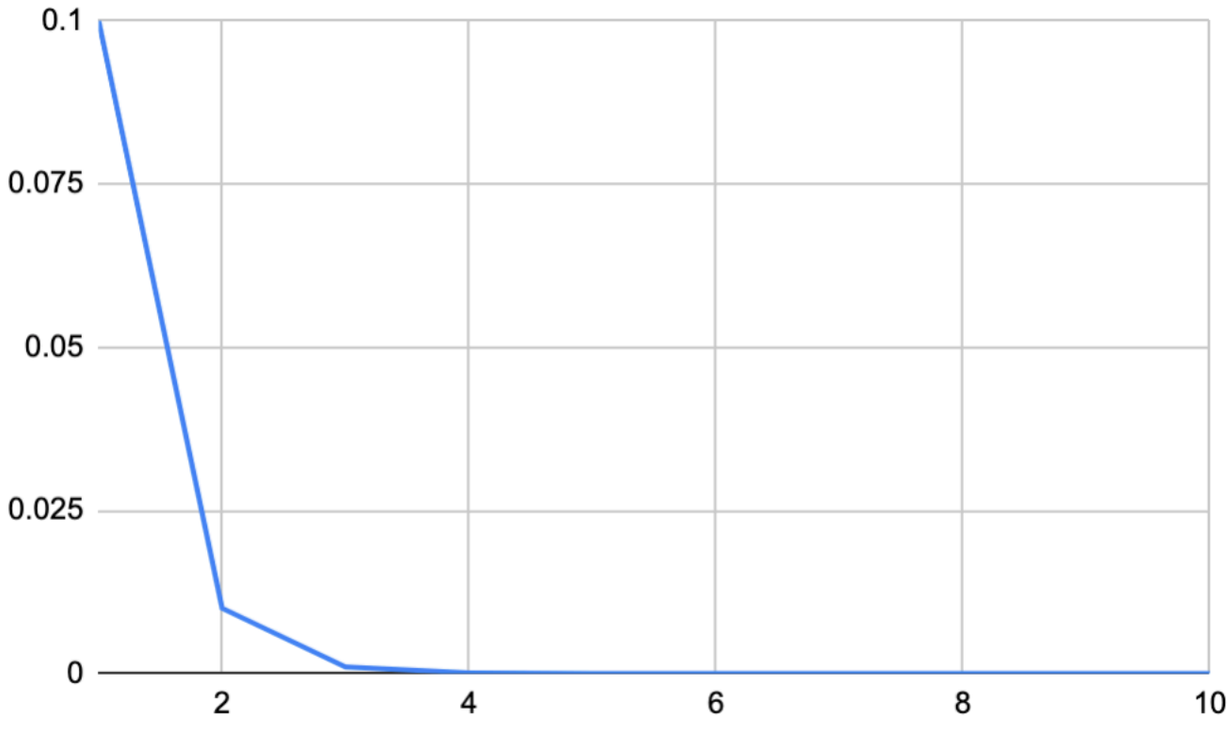
- Compute the probability of a sequence with unigram language model:

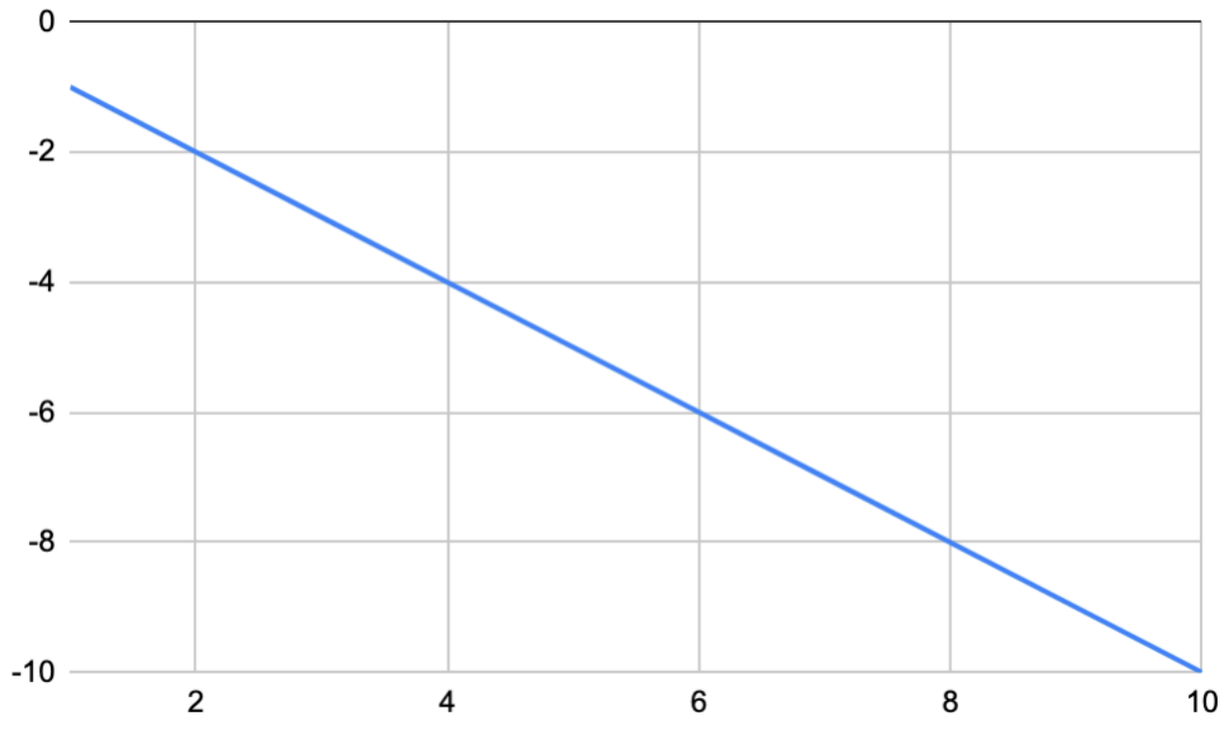
$$P(w_1, w_2, w_3, ..., w_n) \approx \prod_{i=1}^{n} P(w_i)$$

- The product of the probabilities of multiple items may lead to underflow.
  - Summation of log probabilities are used in practice.

$$\log P(w_1, w_2, w_3, ..., w_n) \approx \sum_{i=1}^{n} \log P(w_i)$$

i	$P(x_i)$	$P(x_1)P(x_2)$, …, $P(x_i)$	log $P(x_i)$	$P(x_1)+P(x_2)+$, …, $+P(x_i)$
1	0.1	0.1	-1	-1
2	0.1	0.01	-1	-2
3	0.1	0.001	-1	-3
4	0.1	0.0001	-1	-4
5	0.1	0.00001	-1	-5
6	0.1	0.000001	-1	-6
7	0.1	0.0000001	-1	-7
8	0.1	0.00000001	-1	-8
9	0.1	0.000000001	-1	-9
10	0.1	0.0000000001	-1	-10





Log Probabilities

- Computation in the logarithm domain
  - Reduce the risk of underflow
    - 0.000012 * 0.000005 * 0.00000046 * 0.0008 … ~ 0
  - Addition is faster than multiplication

$$\log(p_1 \times p_2 \times p_3 \times ... \times p_n) = \log p_1 + \log p_2 + \log p_3 + ... + \log p_n$$

- Derive the probability:

$$p = 2^{\log_2 p}$$

Results of the Unigram Model

- The information of word order is completely missing
  - P(the dog ate my cake) = P(my cake ate the dog) = 5.02147142632189 7e-19

Word	P
the	5.400743E-02
dog	6.028288E-05
ate	1.377894E-05
my	9.998347E-04
cake	1.119539E-05

Word	P
my	9.998347E-04
cake	1.119539E-05
ate	1.377894E-05
the	5.400743E-02
dog	6.028288E-05

Bigram Model

- The state of the previous word is taken into account.

$$P(w_1, w_2, w_3, ..., w_n) \approx \prod_{i=1}^{n} P(w_i|w_{i-1})$$

- The product of the probabilities of multiple items may lead to underflow.
  - Summation of log probabilities are used in practice.

$$\log P(w_1, w_2, w_3, ..., w_n) \approx \sum_{i=1}^{n} \log P(w_i|w_{i-1})$$

Results of the Bigram Model

- P(the election was conducted) > P(was election the conducted)

2.12×10^{-15} 2.01×10^{-20}

w_1	w_2	P
<s>	the	2.769008E-03
the	election	1.768108E-04
election	was	7.126314E-05
was	conducted	6.075796E-05

w_1	w_2	P
<s>	was	8.818498E-06
was	election	1.518949E-05
election	the	1.781578E-05
the	conducted	8.419564E-06

N-gram Models

- Bigram model can be further extended to trigram model, four-gram, five-gram model, and so on.
  - *k*-gram model

$$P(w_1, w_2, w_3, ..., w_n) \approx \prod_{i=1}^{n} P(w_i | w_{i-k+1}, w_{i-k+2}, ..., w_{i-1})$$

- The product of the probabilities of multiple items may lead to underflow.
  - Summation of log probabilities are used in practice.

$$\log P(w_1, w_2, w_3, ..., w_n) \approx \log \sum_{i=1}^{n} \log P(w_i | w_{i-k+1}, w_{i-k+2}, ..., w_{i-1})$$

Shortage of N-gram Models

- For small N, the long distance dependencies are missing.
  - I would like to **take** a wide range of linguistic features **into account**.
- For larger N, the empirical data can be very sparse.

Estimation of N-gram Probabilities

- Estimating the parameters for a model *M* from a dataset *D*
  - Maximizing the likelihood of the dataset *D* given the model *M*
- The word “dog” appears 6,312 times in a dataset containing 1,200,000 words
  - MLE: 6,312 / 1,200,000 = 0.00526
  - A word in other dataset from the same distribution will the word “dog”.
  - The estimation may be very bad for some other datasets.

Maximum Likelihood Estimation for N-gram Models

- Bigram

$$P(w_i|w_{i-1}) = \frac{C(w_{i-1}, w_i)}{C(w_{i-1})}$$

- Trigram

$$P(w_i|w_{i-2}, w_{i-1}) = \frac{C(w_{i-2}, w_{i-1}, w_i)}{C(w_{i-2}, w_{i-1})}$$

Example

- A special symbol <s> to denote the beginning of a sentence.

$$P(w_i|w_{i-1}) = \frac{C(w_{i-1}, w_i)}{C(w_{i-1})}$$

$$P(\text{The dog ate my cake})$$
$$= P(\text{The}|<\text{s}>)P(\text{dog}|\text{The})P(\text{ate}|\text{dog})P(\text{my}|\text{ate})P(\text{cake}|\text{my})$$
$$= \frac{C(<\text{s}>\text{ The})}{C(<\text{s}>)}\frac{C(\text{The dog})}{C(\text{The})}\frac{C(\text{dog ate})}{C(\text{dog})}\frac{C(\text{ate my})}{C(\text{ate})}\frac{C(\text{my cake})}{C(\text{my})}$$

Neural Language Models

- Instead of the N-gram model, recurrent neural network-based and transformer-based model can also be used for language modeling, and do much better!

- The major drawback of the N-gram model
  - Hard to model the long term relationships among words due to the high sparsity.

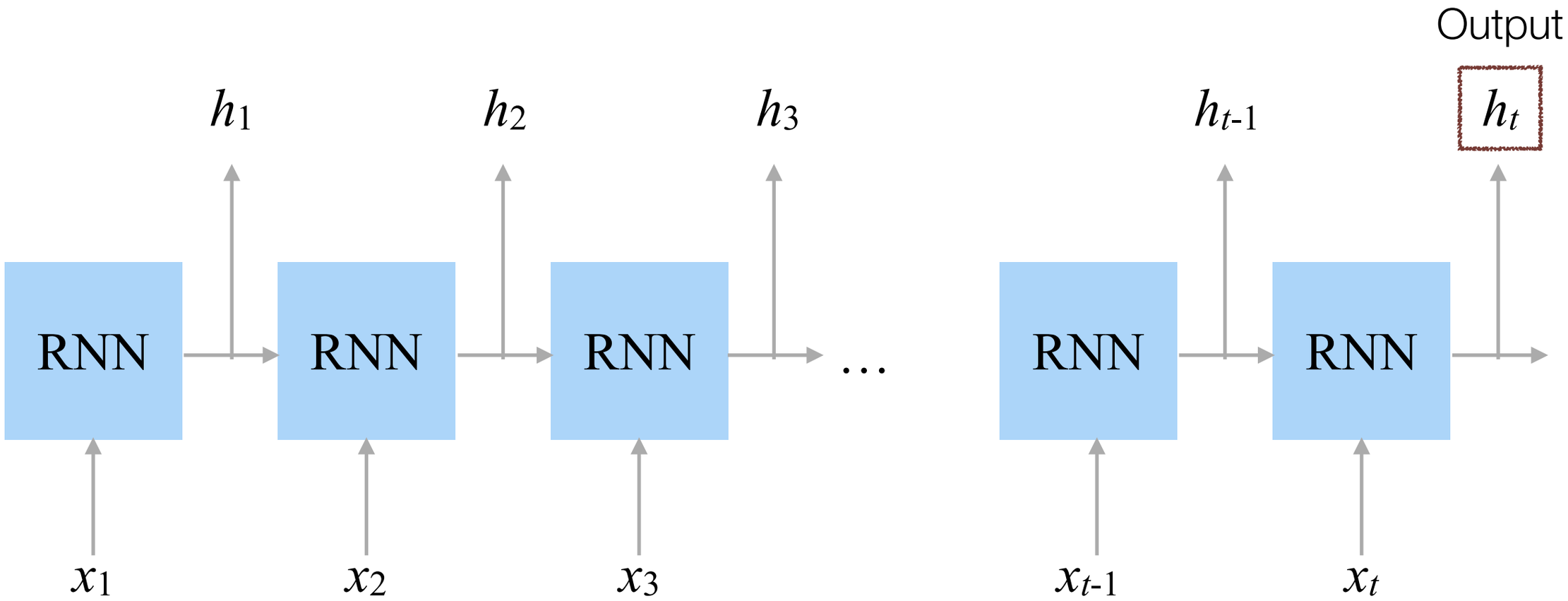
- The NN-based language model can easily handle the longer term relationships even the naive RNN.

Advantages of Neural Language Models

- Able to capture longer dependency
- Reducing the sparsity
- Semantic information captured in word embeddings

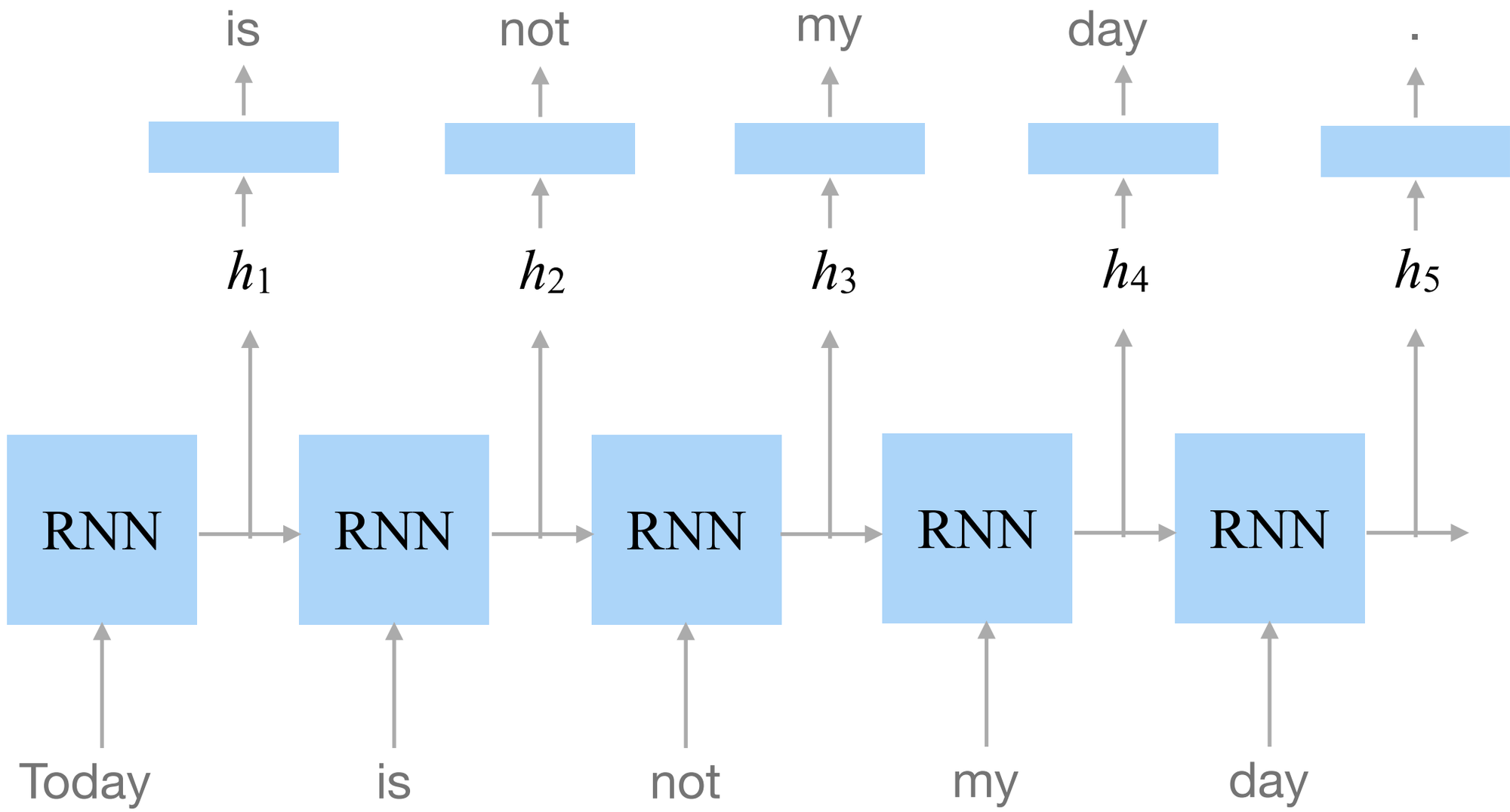
Recurrent Neural Network

- The output of step t-1 is passed to next step
- Unlike HMM, the output of step t is determined by not only the information of x_t and y_{t-1}, but also the information from 1, 2, 3, …, t-2
- No Markov assumption

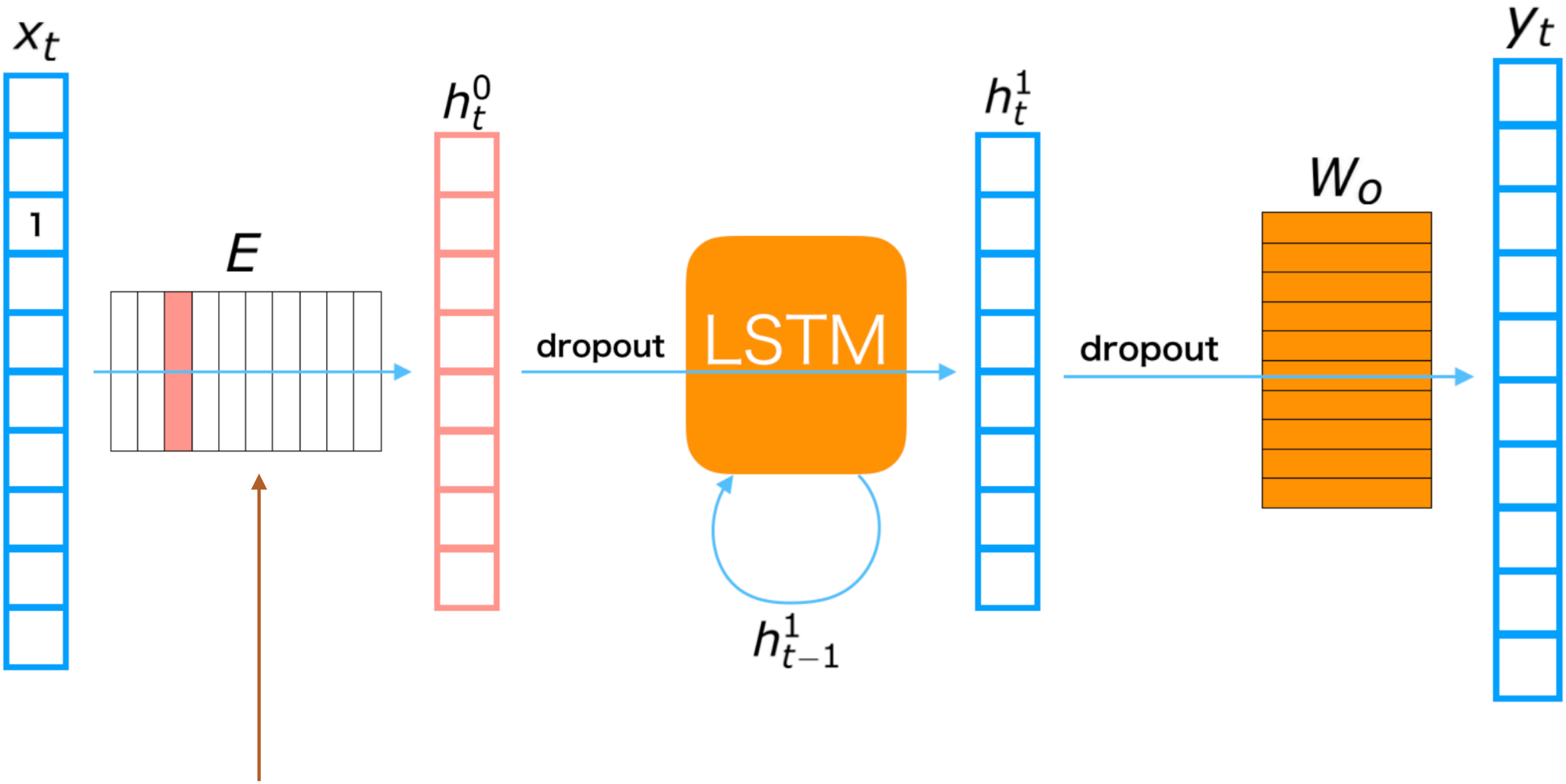


RNN for Language Modeling

- Given the w_1, w_2, ..., $w_{t\text{-}1}$, the model is asked to predict the w_t.

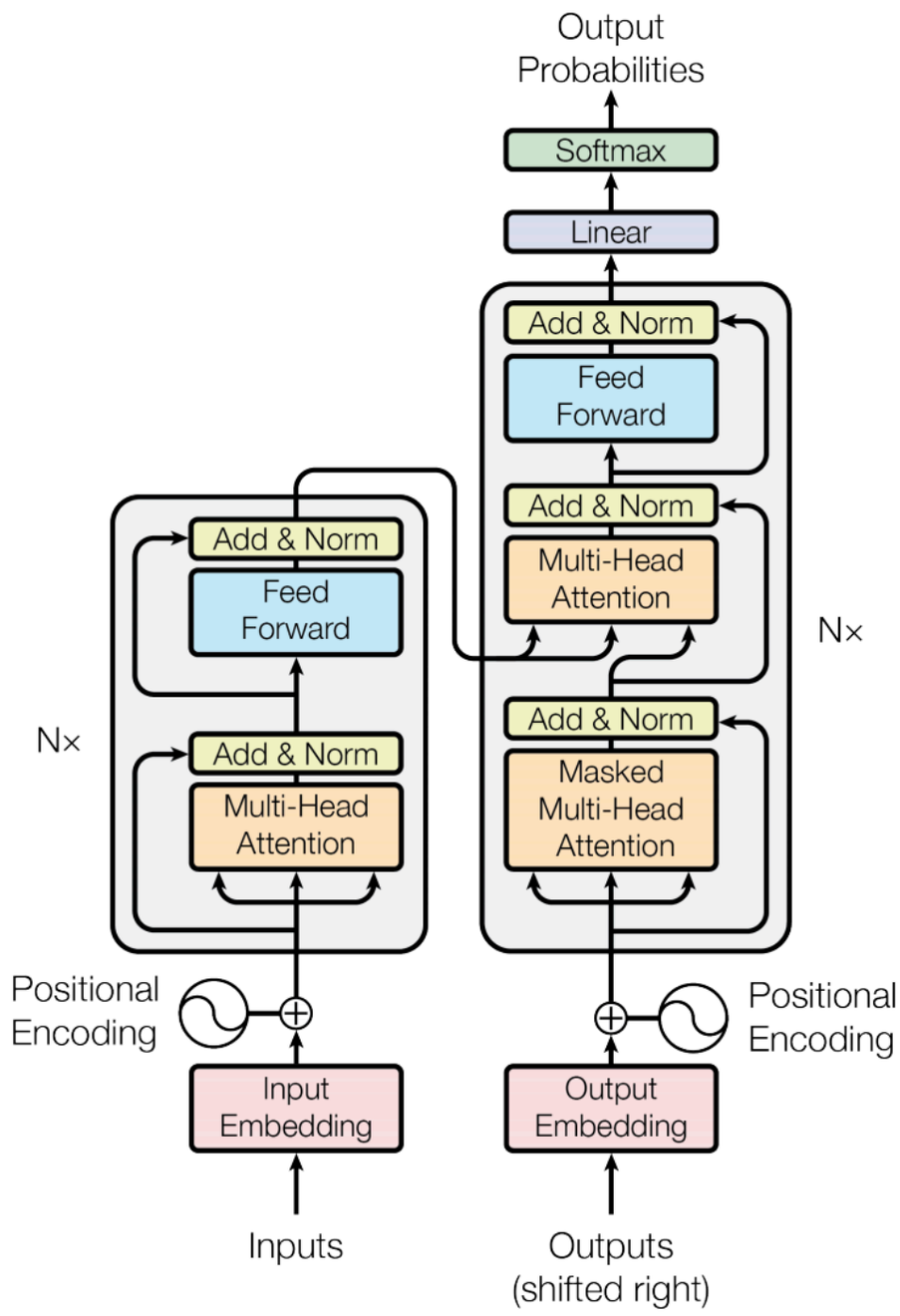


RNN Language Model



Language Models based on Transformer

- Transformer is a type of neural network architectures that is the mainstream model for NLP.
- It generally outperforms CNN/ RNN families due to its ability in modeling the long distance dependency, which is very important in textual information.
- Explored as the backbone of modern language models.



GPT: Language Model based on Transformer

- GPT extends the ability of language modeling to generate more meaningful text

Python is a programming language that lets you work more quickly and integrate your systems more effectively. You can learn to use Python and see almost immediate gains in productivity and lower maintenance costs. Python runs on Windows, Linux/Unix, Mac OS X, and has been ported to the Java and .NET virtual machines. Python is free to use, even for commercial products, because of its OSI-approved open source license. Python's C-style syntax will be familiar to anyone who has used similar languages, but Python is very clear and readable. Python's elegant syntax and dynamic typing, together with its interpreted nature, make it an ideal language for scripting

https://www.twilio.com/blog/ultimate-guide-openai-gpt-3-language-model

Generation with GPT-3

- Prompt:
  - *We're releasing an API for accessing new AI models developed by OpenAI. Unlike most AI systems which are designed for one use-case, the API today provides a general-purpose "text in, text out" interface, allowing users to try it on virtually any English language task. You can now request access in order to integrate the API into your product, develop an entirely new application, or help us explore the strengths and limits of this technology.*
- Generation
  - **The road to making AI safe and useful is long and challenging, but with the support of the developer community we expect to get there much faster than working alone.**

LSTM vs GPT

LSTM

段譽和王語嫣面面相對，呼吸可聞，雖身處污泥，心中卻充滿了喜樂之情，誰也沒想到要爬出井去。兩人同時慢慢的伸手出來，四手相握，心意相通。

南海鱷神哈哈一笑，說道：「這裏斜行走了出去，肚中空蕩蕩地，頗為心驚。只聽他說「不出」，沒想到這三步來的變招，居然心下惴惴，伸掌在膝頭褲子、套心甚是指尖，那人可隨便運勁，將那本事大喝一聲。虛竹不去救司眾的臉上，全身酒水淋漓，密密熱炭的在那裏水來。游坦之乘著一名弟子，當時只覺馬鞭擊了過，說道：「我這一曲可就不錯。」李延宗道：「書上的事情卻到了大哥？我這長老賊的眼光之親，便是穿新衣衫，手背陳長老後頸微微一紅，突然氣往上衝，腦袋上內咬下來的兵刃掉在地下，額角撞了個圓圈之中，就算不出南來三俠的衝到，將兩柄飛刀到了，猛聽得蕭峯對敵傾覆，有的伸手叫好，喝道：「你怎知我受苦受寒毒的折磨你，我早生了。」一面哭笑，一個女子聲音笑道：「我嫁了王姑娘，我愛你做花。」奔向出去，見四具屍體匆匆忙逼？踏了下去，大聲急衝揚手，將他抱在身上。阿紫柔聲安慰，向蕭峯道：「孩兒，懂得甚麼？四大惡人齊去應敵。」蕭峯道：「我送了給外人，是否衝劍出仔細說罷。眾位姑娘，嗯，在下姑蘇燕子塢慕容先生仙逝。」蕭峯低沉著聲音陰神看了一眼，心下雪亮，見蕭峯背負布袋，兀自未曾出外了。穿過一個少年深的肌膚，一口都喝到一聲：「啊喲！」發足上湧，怒氣漸漸鬆軟 ...

GPT

段譽和王語嫣面面相對，呼吸可聞，雖身處污泥，心中卻充滿了喜樂之情，誰也沒想到要爬出井去。兩人同時慢慢的伸手出來，四手相握，心意相通。

過了良久，鳩摩智嘆了口氣，說道：「咱們一面看去，不過得你爺兒倆的事，那就誰也不會大意了。」段譽道：「我不會再來見你。」鳩摩智道：「你既不見我，我也不會見我。我只求你一件事，以便遵命。」 段譽道：「你這就奇了。我是甚麼？」鳩摩智道：「我是這樣說，你總是要見到我的。」段譽胸口一酸，心想：「要我幹麼？你一求，我便幹，就幹麼？」說道：「要我答允你，我是永遠不離開你的啦。」 王語嫣坐起身來，月光之下，只見鳩摩智的衣袖在月光下，向她微微掃視，她的容貌秀麗，美麗的臉龐上帶著幾分嬌艷，不禁心中一動：「段公子，你的心早就一定要過來了。」 段譽胸口一熱，頭暈紅了，卻不想要答應他的話，卻說不出口。王語嫣道：「段公子，咱們是再度相會，你這一生一世，決不見外了。世上有許多事情，你一輩子寂寞，豈不是好？」 慕容復哼了一聲，道：「你這人當真不要？你這人雖然清俏，心中卻也不快活。我跟你說，你是不是真的待你好色之人，是不是？」 王語嫣抬起頭來，淺淺一笑，道：「我要是見到你這等容貌，從來沒跟我談起，更沒想到這件事。你剛才在過湖裏，為了你，要跟我談談，你卻不許？」 段譽胸口一熱，淚水便欲奪眶而出，哽咽道：「姑娘，我實是不願意 ...

https://leemeng.tw/gpt2-language-model-generate-chinese-jing-yong-novels.html

Grammatical Error Correction with GPT-3

- Input
  - *Poor English: Thank you for picking me as your designer. I'd appreciate it.*
    Corrected English:
- Output
  - **Thank you for picking me as your designer. I appreciate it.**

The Applications of Modern Language Models

- GPT-3 is confirmed effective for a wide range of NLP tasks, not limited to next word prediction or generation
  - Grammatical error correction
  - Text classification
  - Translation
  - Question answering
  - Chatbot
  - ...
- Many NLP tasks can be seen as the tasks of sentence completion.

C Program Generation with an LSTM LM

```
/* Rung state control macro.  */
struct (cpp_reader *pfile, const char *IT_And, to_line));
      true: dir->name);
    }
  else if (token->type == CPP_HEFION_UNS'   }
      *pp_cre to bat wreap that %s", any sef GNADE->name);
  else if (dir == NULL)
  close = read_filens (pfile, fragian, but, struct dir_hash_entry () *hname, *)
   {
     for (if (coult == 0)
  _cpp_backup_tokens (pfile, CPP_W__PPRIG))
   cpp_error_with_line (pfile, CPP_W_TRADITIONAL, ,   XCENSION (pfile, CPP_W_BEARNING,
        int angle_brackets)
   {
     file->st.nt_dir = pfile->directive->name;
    cpp_token *tok;
const char *space, const char *, directives->ld_call]->file_hash_entry (pfile, true);
```

https://github.com/meenavyas/Misc/blob/master/codeGeneration.py

Code Generation with GPT-3

```
// Here are the 2 description:code pairs used to give GPT-3
some context for how to provide a response

// sample 1
description: a red button that says stop
code: <button style={{color: 'white', backgroundColor:
'red'}}>Stop</button>

//sample 2
description: a blue box that contains 3 yellow circles with
red borders
code: <div style={{backgroundColor: 'blue', padding: 20}}><div
style={{backgroundColor: 'yellow', border: '5px solid red',
borderRadius: '50%', padding: 20, width: 100, height: 100}}>
</div><div style={{backgroundColor: 'yellow', borderWidth: 1,
border: '5px solid red', borderRadius: '50%', padding: 20,
width: 100, height: 100}}></div><div style={{backgroundColor:
'yellow', border: '5px solid red', borderRadius: '50%',
padding: 20, width: 100, height: 100}}></div></div>
```

carbon
carbon.now.sh

https://twitter.com/sharifshameem/status/1282676454690451457

Smoothing

Data Sparsity

- For the ngram w_1, w_2, …, w_n not occurring in the training corpus
  - $C(w_1, w_2, \ldots, w_n) = 0$
  - $P(w_1, w_2, \ldots, w_n) = 0$
- We cannot measure the probabilities of unseen words

Zeros

Corpus for Training	Real World Data
… denied the allegations	… denied the offer
… denied the reports	… denied the loan
… denied the claims	
… denied the request	

$$P(\text{offer}|\text{denied the}) = 0$$

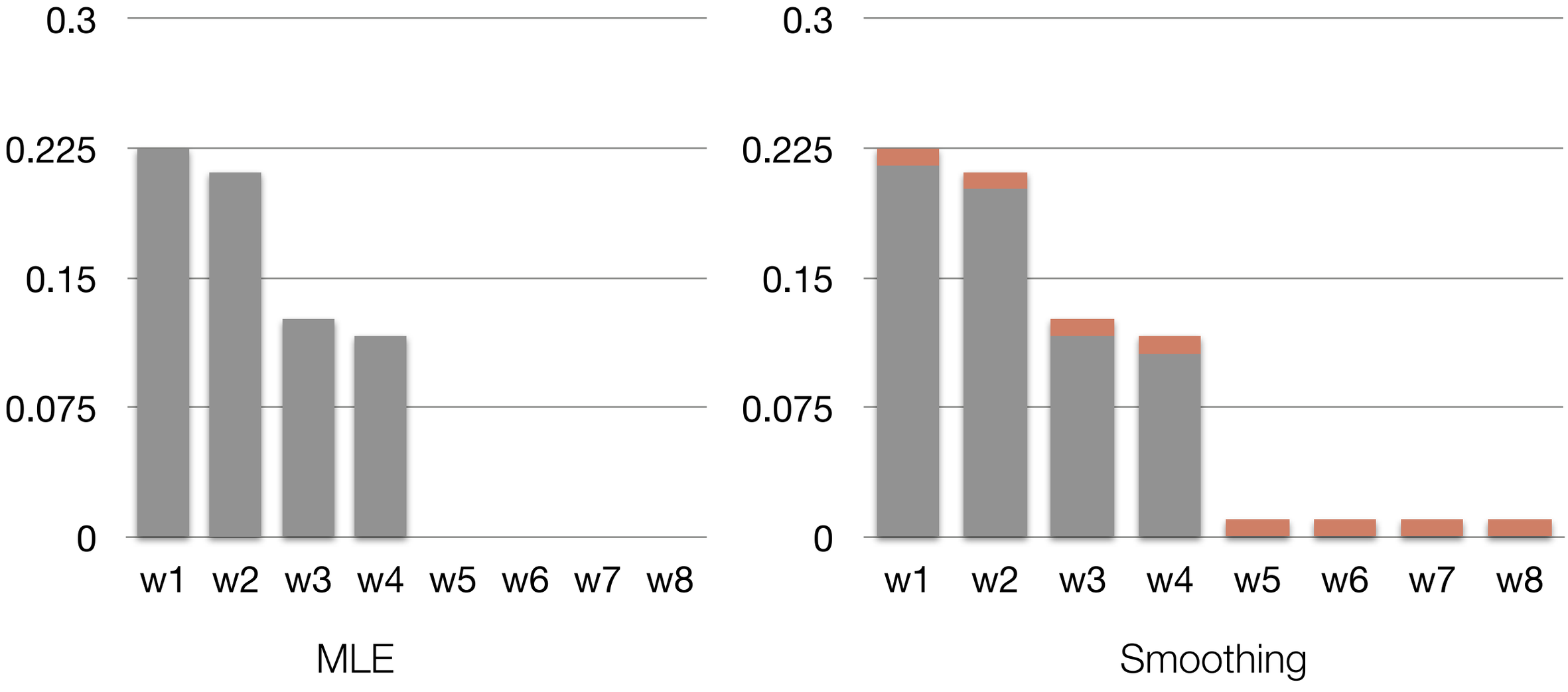
$$P(w_1, w_2, w_3, ..., w_n)$$
$$= \prod_{i=1}^{n} P(w_i|w_{i-2}, w_{i-1})$$
$$= 0 \text{ if } \exists j, 1 \leq j \leq n, P(w_j|w_{j-2}, w_{j-1}) = 0$$

Zero Probability of N-grams

- The MLE of the N-gram is zero
  - The n-gram w_1, w_2, w_3, …, w_n is unseen in the corpus
  - $C(w_1, w_2, w_3, \ldots, w_n) = 0$
- We cannot estimate the probability of a sequence that contains an unseen n-gram.

Smoothing

- Reserve small amount of probabilities from the seen words for the unseen words.
- Pretend the model has saw each word once.



Add-One (Laplace) Estimation

- Just add one to all the counts
- Original MLE:

$$P_{MLE}(w_i|w_{i-1}) = \frac{C(w_{i-1}, w_i)}{C(w_{i-1})}$$

- Add-1 Estimation:

$$P_{ADD-1}(w_i|w_{i-1}) = \frac{C(w_{i-1}, w_i) + 1}{C(w_{i-1}) + |V|}$$

Number of possible bigrams

$$P_{ADD-1}(w_i|w_{i-1}) = \frac{1}{C(w_{i-1}) + |V|} \text{ if } C(w_{i-1}, w_i) = 0$$

For unseen ngrams

Add-One (Laplace) Estimation

- Just add one to all the counts
- Original MLE:

$$P_{MLE}(w_i|w_{i-1}) = \frac{C(w_{i-1}, w_i)}{C(w_{i-1})}$$

- Add-1 Estimation:

$$P_{ADD-1}(w_i|w_{i-1}) = \frac{C(w_{i-1}, w_i) + 1}{C(w_{i-1}) + |V|}$$

Number of all unigrams

$$P_{ADD-1}(w_i|w_{i-1}) = \frac{1}{C(w_{i-1}) + |V|} \text{ if } C(w_{i-1}, w_i) = 0$$

For unseen ngrams

Add-One Smoothing

$$P_{ADD-1}(w_i|w_{i-1}) = \frac{C(w_{i-1}, w_i) + 1}{C(w_{i-1}) + |V|}$$

$$P_{ADD-1}(w_i|w_{i-1}) = \frac{1}{C(w_{i-1}) + |V|} \text{ if } C(w_{i-1}, w_i) = 0$$

Actual occurrences of the ngram in data

$$P_{ADD-1}(x_1, x_2, ..., x_n) = \frac{C(x_1, x_2, ..., x_n) + 1}{N + B}$$

All occurrences of ngrams in data

All possible ngrams $|V|^n$

Overestimation

- A corpus of 44 x 10^6 words (N)
- 400K unique words (unigrams)
  - B = Possible bigrams $|V|^2$ = 1.6 x 10^{11}
  - B >> N

$$P_{ADD-1}(x_1, x_2, ..., x_n) = \frac{C(x_1, x_2, ..., x_n) + 1}{N + B}$$

Bound by the corpus size (44 x 10^6)

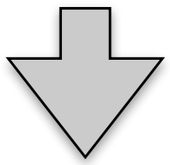
Overestimation

- Add-one estimation reserves too much probability for possible unseen ngrams.
- In some cases, 99.9% of the probability mass has actually been give to bigrams.
- Reducing the probability estimates of more frequent events.

Additive Smoothing

- Adding one to unseen case is too much
  - Adding a smaller number instead of one

$$P_{ADD-1}(x_1, x_2, ..., x_n) = \frac{C(x_1, x_2, ..., x_n) + 1}{N + B}$$



$$P_{ADD}(x_1, x_2, ..., x_n) = \frac{C(x_1, x_2, ..., x_n) + \lambda}{N + \lambda B}$$

Additive Smoothing

- Or Lidstone smoothing

$$P_{ADD-1}(w_i|w_{i-1}) = \frac{C(w_{i-1}, w_i) + 1}{C(w_{i-1}) + |V|}$$

$$P_{ADD}(w_i|w_{i-1}) = \frac{C(w_{i-1}, w_i) + \lambda}{C(w_{i-1}) + \lambda|V|}$$

$$P_{ADD}(w_i|w_{i-1}) = \frac{\lambda}{C(w_{i-1}) + \lambda|V|} \text{ if } C(w_{i-1}, w_i) = 0$$

2 in practice

Estimate the Number of Unseen Ngrams

$$P_{ADD-1}(x_1, x_2, ..., x_n) = \frac{C(x_1, x_2, ..., x_n) + 1}{N + B}$$

$$P_{ADD}(x_1, x_2, ..., x_n) = \frac{C(x_1, x_2, ..., x_n) + \lambda}{N + \lambda B}$$

- B = $|V|^n$ is unreality since not all permutations of alphabets will be used in real scenario.
  - ts, cdt, dgkt, bfyhh, …
- Can we have a better approach to the estimation of the size of unseen ngrams?

Held Out Estimation

- Empirical estimation
- Split corpus into training set and held out set.
- See how often ngrams that appear *c* times in the training set tend to turn up in the held out set.

Held Out Estimator

$$C_T(w_1, ..., w_n) = \text{Count of } w1, ..., w_n \text{ in the training set}$$
$$C_H(w_1, ..., w_n) = \text{Count of } w1, ..., w_n \text{ in the held out set}$$

Total number of times that all n-grams that appear *c* times in the training set in the held out set

$$T_c = \sum_{\{w_1,...,w_n : C_T(w1,...,w_n)=c\}} C_H(w_1, ..., w_n)$$

$$C_{HE}(w_1, ..., w_n) = \frac{T_c}{n_c} \text{ for } C(w_1, ..., w_n) = c$$

$$P_{HE}(w_1, ..., w_n) = \frac{C_{HE}(w_1, ..., w_n)}{N} = \frac{Tc}{n_c N} \text{ for } C(w_1, ..., w_n) = c$$

Adjusted count and probability for n-grams that appear *c* times in the training set

Good-Turing Estimation

- If we know the ratio of size of n-grams occurring twice to the size of n-grams occurring once, and we can predict the size of unseen n-grams.
    - Estimating the frequency of the n-gram appears *c* times with the frequency of the n-gram appears *c* + 1 times.
- Reallocate the probability mass of n-grams occurring *c* + 1 times in the training corpus to the n-grams occurring exact *c* times.
    - Reallocate the probability mass of n-grams occurring once to the unseen n-grams.

Adjusted Counts

c : Occurrences of n-grams occurring exactly c times

$c_0 = 0, c_1 = 1, c_2 = 2, ...$

n_c : Number of n-grams occurring exactly c times in the corpus

n_1 : Number of n-grams occurring once

n_2 : Number of n-grams occurring twice

n_0 : Number of n-grams unseen in the corpus (theoretically)

Number of all possible n-grams theoretically

$$n_0 = B - \sum_{c=1}^{\infty} n_c \text{ where } B = |V|^k \text{ and } k \text{ is the order of n-grams}$$

Number of distinct n-grams in the corpus

Adjusted Counts

c : Occurrences of n-grams occurring exactly c times

$c_0 = 0, c_1 = 1, c_2 = 2, ...$

n_c : Number of n-grams occurring exactly c times in the corpus

n_1 : Number of n-grams occurring once

n_2 : Number of n-grams occurring twice

n_0 : Number of n-grams unseen in the corpus (theoretically)

$$n_0 = B - \sum_{c=1}^{\infty} n_c \text{ where } B = |V|^k \text{ and } k \text{ is the order of n-grams}$$

$$c^\star = (c+1)\frac{n_{c+1}}{n_c}$$

Ratio of size of ngrams occurring c+1 times to the size of ngrams occurring c times.

Good-Turing Smoothing

- Smoothed **occurrences** of an n-gram appears *c* times

$$c^\star = (c+1)\frac{n_{c+1}}{n_c}$$

- Smoothed **probability** of the n-gram appears *c* times

$$P_{GT}(x_c) = \frac{c^\star}{N} = \frac{c^\star}{\sum_{c=1}^{\infty} c \times n_c}$$

The n-grams occurring c times in the corpus

Total n-grams in corpus

- Given a corpus
  - N = 22M (total tokens)
  - $|V|$ = 273,266 (unique words)
  - B = $|V|^2$ = 74,674,306,756

Theoretical number of n-grams **Observed number of n-grams in a corpus**

c	Empirical	Add-1	Good Turing	n_c	T_c
0	0.000027	0.000295	0.000027	74,671,100,000	2,019,187
1	0.448	0.000589	0.396	2,018,046	903,206
2	1.25	0.000884	1.26	449,721	564,153
3	2.24	0.00188	2.24	188,933	424,015
4	3.24	0.00147	3.24	105,668	341,099
5	4.21	0.00177	4.22	68,379	287,776
6	5.23	0.00206	5.19	48,190	251,951

Additive Smoothing vs Good Turing Smoothing

- Additive smoothing
  - Leaving too many mass for unseen n-grams
  - Some n-grams are impossible in real data
    - "size of unseen n-grams" vs. "n-grams unseen of size"
- Good Turing smoothing
  - Estimating the mass of unseen n-grams by using the information from the data

How Good of a Language Model

Evaluation of N-gram Models: Extrinsic Evaluation

- Employ the model *M* into an application system such as a machine translator.
    - Evaluate the performance of the system with *M*.
- Employ the baseline model *B* into the same application system.
    - Evaluate the performance of the system with *B*.
- Compare the performances of the system with these two models.

Extrinsic Evaluation （外在驗證）

- Pros
  - Target oriented
  - Evaluate the performance of the target application.
- Cons
  - Time consuming
  - Have to integrate the experimented language model into the target system.

Extrinsic Evaluation of a Language Model

- Machine translation
  - Building a machine translation system that employs a language model as a component.
  - Evaluating the **translation performances** of machine translators with different language models used.
- News classification
  - Building a news classifier based on a language model.
  - Evaluating the classification performances of the classifiers with different language models used.

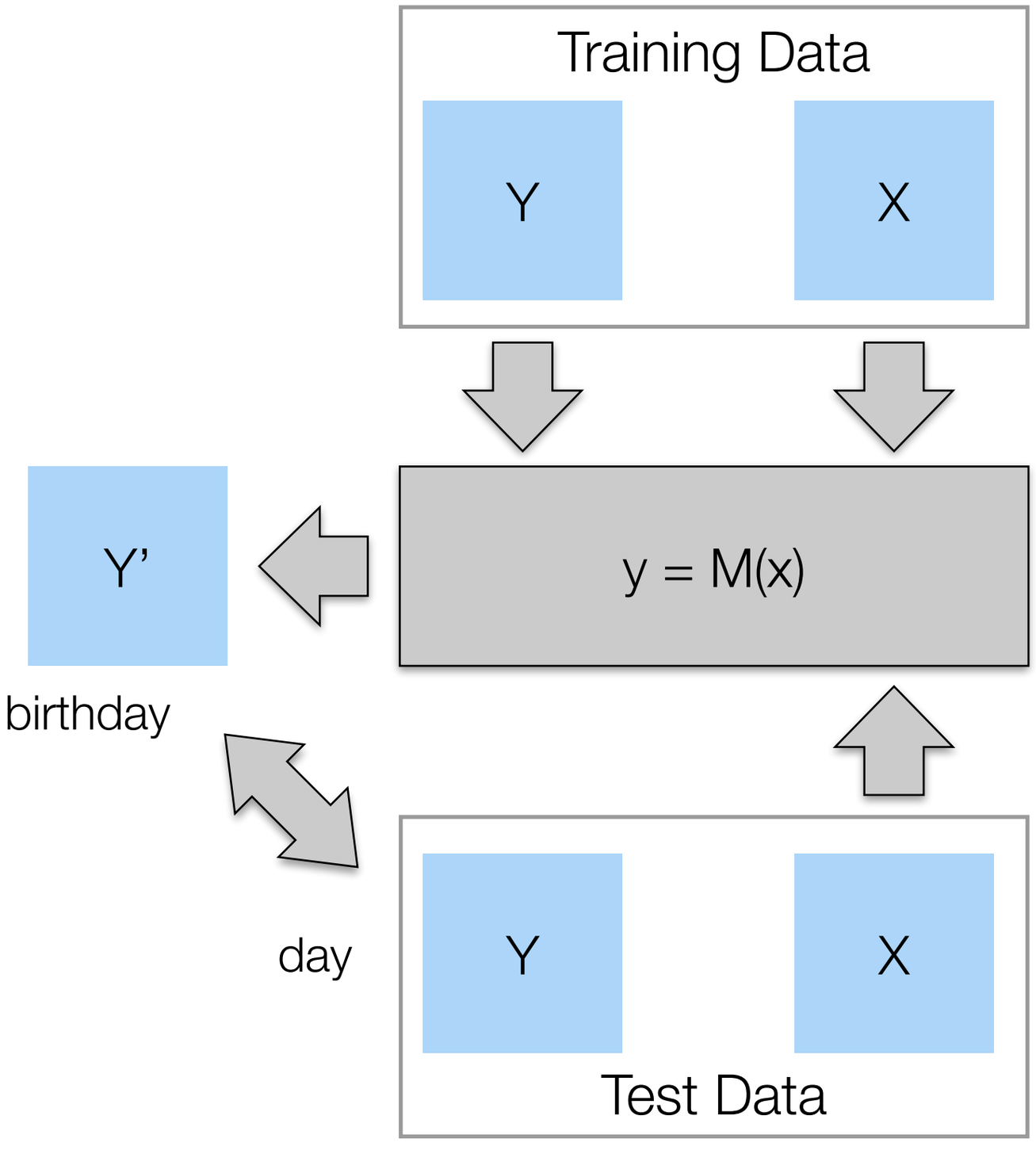
Intrinsic Evaluation (內在驗證)

- Reserve a part of data as test data
- Given a test sequence, how can the model predict the next word?
  - The dog ate my ____ (cake | cat | mouse)
  - I live with my ___ (sister | coffee | elephant)
- The performance can be directly measured by **perplexity**.

Evaluation

- We split the dataset into two isolated parts:
  - Training data
    - Used to train the language model (or other kinds of classifiers)
  - Test data
    - The trained model predicts the results given the test data
    - Compare the predictions made by the model with the ground-truth.

Evaluation with Held Out Data



X	Y
My dog ate my	cake
My dog ate my	homework
My father is my	hero
classroom is the thrid	teacher
…	…

X	Y
Today is my	day

Unseen to M

Perplexity

- A perplexity of *k* means you are as surprised on average as you would have been if you had to guess between *k* equiprobable choices at each step.

- The lower the perplexity, the better the language model

- Training data: 38 million words from Wall Street Journal

- Test data: 1.5 million words from Wall Street Journal

N-gram Order	Unigram	Bigram	Trigram
Perplexity	962	170	109

Perplexity of N-gram

- Measuring how good a language model predicts an unseen test case.

$$Perplexity(w_1, w_2, w_3, ..., w_n)$$

$$= P(w_1, w_2, w_3, ..., w_n)^{-\frac{1}{n}}$$

Normalized by length 幾何平均

$$= \sqrt[n]{\frac{1}{P(w_1, w_2, w_3, ..., w_n)}}$$

Inverse probability of the test case

$$\approx \sqrt[n]{\prod_{i=1}^{n} \frac{1}{P(w_i|w_1, w_2, w_3, ..., w_{i-1})}}$$

Chain rule

$$\approx \sqrt[n]{\prod_{i=1}^{n} \frac{1}{P(w_i|w_{i-1})}}$$

Bigram

Author Identification for Classical Chinese Documents

- Author identification with n-grams
  - Word level
    - Classical Chinese word segmentation is required.
  - Character level
  - Phonological (音韻) level

Phonological Information

Initial 聲母	Final 介音/韻母	Tone 聲調
ㄅㄆㄇㄈ	ㄧㄨㄩ	ˉ
ㄉㄊㄋㄌ	ㄚㄛㄜㄝ	ˊ
ㄍㄎㄏ	ㄞㄟㄠㄡ	ˇ
ㄐㄑㄒ	ㄢㄣㄤㄥ	ˋ
ㄓㄔㄕㄖ	ㄦ	˙
ㄗㄘㄙ		

Phonological Systems for Chinese

- 反切
  - 通 => 透 / 東

For initial　　For final

System	Types of Initial	Types of Finals	Types of Tones
現代漢語注音	21	36	5
反切	403	1,054	
廣韻	43	203	

Complexity Reduction with Phonological Representation

- Lowest complexity: 注音
    - The smallest sets of initials, finals, and tones.

Data	Perplexity			
	Character	注音	反切	廣韻
左傳	78.27	4.34	9.83	9.35
孟子	90.71	4.94	12.36	11.35
莊子	115.01	5.02	12.80	11.97
史記	111.71	4.97	11.60	11.02
Average	98.93	4.82	11.65	10.92

Author Recognition

- 反切 and 廣韻 provides the information more accurate for the Classical Chinese documents.
- 注音 is too simplified and far away from the era.

Data	Accuracy (%)			
	Character	注音	反切	廣韻
左傳	98.53	98.53	**98.82**	98.53
孟子	70.37	62.96	70.37	**85.19**
莊子	78.76	84.96	**88.50**	86.73
史記	89.75	93.93	**96.03**	94.77
Average	84.35	85.09	88.43	**91.30**

Text Classification with Language Models

Text Classification

- Predict the opinion of a movie review
  - Positive
  - Negative

Positive Review

- films adapted from comic books have had plenty of success, whether they're about superheroes (batman, superman, spawn), or geared toward kids (casper) or the arthouse crowd (ghost world), but there's never really been a comic book like from hell before ….
- after watching "rat race" last week, i noticed my cheeks were sore and realized that, when not laughing aloud, i had held a grin for virtually all of the film's 112 minutes. … " rat race " is a riot, with terrific no-holds-barred performances from the diverse cast. see it, see it again and when the dvd comes out, buy it, because a movie this hilarious will surely have outtakes to die for.

Negative Review

- a movie like mortal kombat: annihilation works (and must be reviewed on) multiple levels. first, there's the rampant usage of randian subtext that pervades the entire movie. but occasionaly, almost as if making an ironic, self-depreciating remark, the movie tosses in clearly marxist imagery. no no. .. just kidding. had you going there for a moment, didn't i? in all seriousness however, and to be fair to the movie, it * is * necessary to provide two viewpoints : that of a movie watcher unfamiliar (or only marginally familiar) with the whole mortal kombat phenomenon, and that of a fan of the first movie and / or a fan of the games. the first movie (mortal kombat (1995)) concerned itself with a martial arts tournament that would decide the fate of earth (and it's 5 billion inhabitants).

Formulate the Problem

- We can train two language models
  - M_P: Language model for positive reviews
  - M_N: Language model for negative reviews

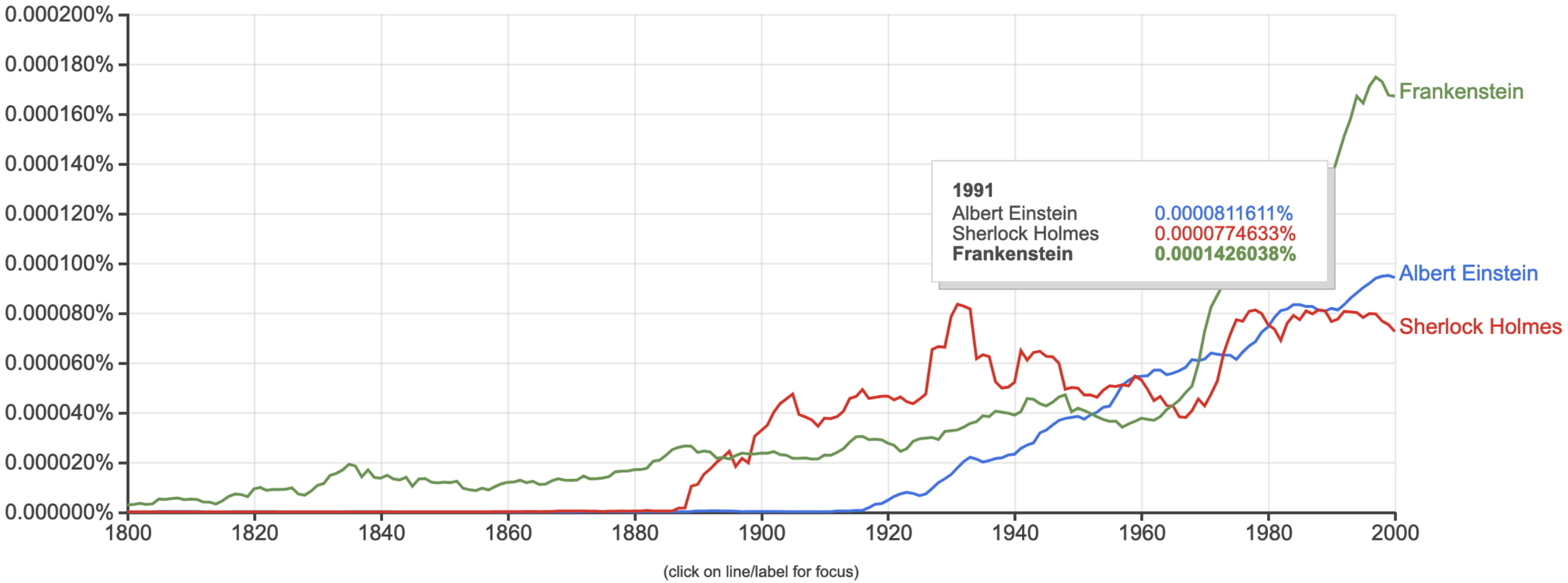
- Given a review *t*, estimate which language model is most likely to generate *t*:
  - *t* = positive if $M_P(t) > M_N(t)$
  - *t* = negative if $M_P(t) < M_N(t)$

Google N-gram Model

Google Books Ngram Viewer

Graph these comma-separated phrases: Albert Einstein,Sherlock Holmes,Frankenstein case-insensitive

between 1800 and 2000 from the corpus English with smoothing of 3. Search lots of books



https://books.google.com/ngrams

Online Demonstration

https://tinyurl.com/y2ok7cjk