# 语言模型和表示学习

CS2916 大语言模型

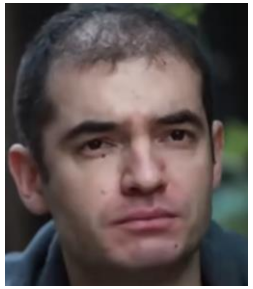
饮水思源 爱国荣校

https://plms.ai/teaching/index.html

# 课程内容

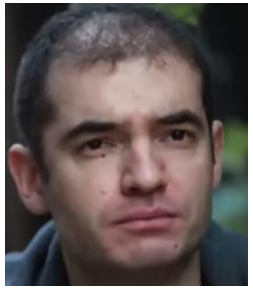
- 语言模型
  - 理解语言模型基本概念和评估方法
  - 了解语言模型常见应用
  - 掌握基于统计、和基于神经网络的学习方法
- 表示学习
  - 了解词表示的学习概念和意义
  - 掌握基于神经网络的词表示学习方法
    - word2vec的基本原理
  - 了解不同词表示学习方法的差异
  - 了解句子表示的学习概念和意义
  - 掌握基于神经网络的句子表示学习方法
- 预训练
  - 了解预训练的基本内容和价值

# “The Next-token prediction is enough for AGI”



Ilya Sutskever
OpenAI CSO

# "The Next-token prediction is enough for AGI"



Ilya Sutskever
OpenAI CSO

Predicting the next token well means that **you understand the underlying reality that led to the creation of that token**.

It's the statistics but what is statistics? In order to understand those statistics to compress them, you need to **understand what is it about the world that creates those statistics**

# 语言模型

李明回家的路上被小狗咬了

小狗回家的路上被李明咬了

# 语言模型

$\boxed{\text{李明回家的路上被小狗咬了}}$ **更容易发生**

小狗回家的路上被李明咬了

# 语言模型

今天天气太热，我回家路上吃了__

包子

烤红薯

冰淇淋

# 语言模型

今天天气太热，我回家路上吃了__

包子

烤红薯

冰淇淋

**更容易发生**

# 语言模型

李明回家的路上被小狗咬了

小狗回家的路上被李明咬了

**P(李明回家的路上被小狗咬了) > P(小狗回家的路上被李明咬了)**

# 语言模型

今天天气太热，我回家路上吃了__

包子

烤红薯

冰淇淋

**更容易发生**

**P(冰淇淋|今天天气太热，我回家路上吃了) > P(烤红薯|今天天气太热，我回家路上吃了)**

# 语言模型

- 给每一个句子都赋予其出现的概率。以$w_1, w_2, \dots w_n$这n个词组成的句子为例，它出现的概率为

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

来衡量句子符合自然语言的语法和语义规则的置信度

# 语言模型

- 给每一个句子都赋予其出现的概率。以$w_{1,} w_{2,} \ldots w_{n,}$这n个词组成的句子为例，它出现的概率为

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

- 语言模型的基本功能
  - **判别：判断一段文本是否符合一种语言的语法和语义规则**

# 语言模型

- 给每一个句子都赋予其出现的概率。以$w_1, w_2, \dots w_n,$ 这n个词组成的句子为例，它出现的概率为

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

- 语言模型的基本功能
  - 判别：判断一段文本是否符合一种语言的语法和语义规则

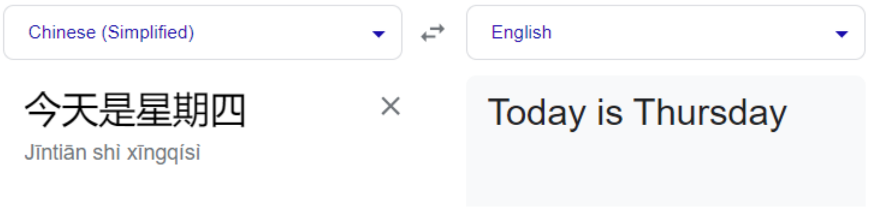
| | |
|---|---|
| 李明回家的路上被小狗咬了 **概率更大** | She went to Shanghai yesterday **概率更大** |
| 小狗回家的路上被李明咬了 | She go to Shanghai yesterday |

# 语言模型

- 给每一个句子都赋予其出现的概率。以$w_1, w_2, \ldots w_n$ 这n个词组成的句子为例，它出现的概率为

$$P(w_1 w_2 \ldots w_n) = P(w_1)P(w_2|w_1)\ldots P(w_n|w_1 w_2 \ldots w_{n-1})$$

- 语言模型的基本功能
  - 判别：判断一段文本是否符合一种语言的语法和语义规则
  - **生成：生成符合一种语言语法或语义规则的文本**



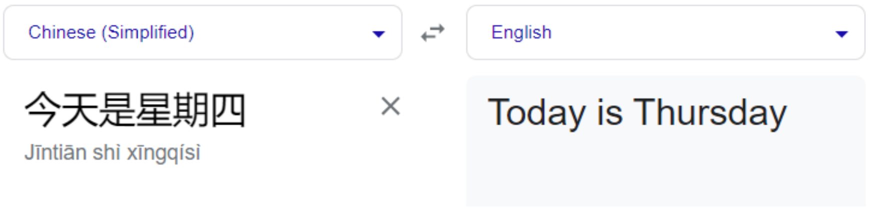

机器翻译

# 语言模型

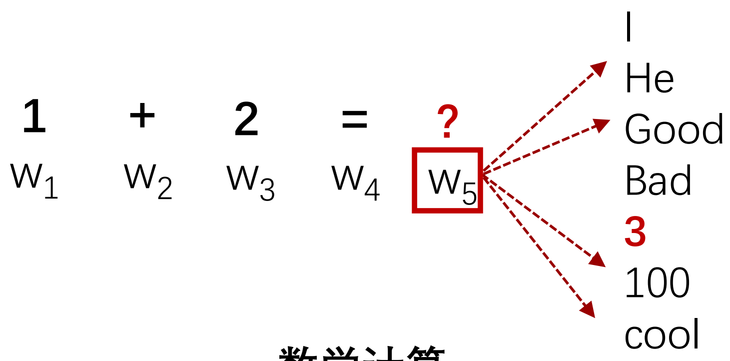
- 给每一个句子都赋予其出现的概率。以$w_1, w_2, \dots w_n$这n个词组成的句子为例，它出现的概率为

$$P(w_1w_2...w_n) = P(w_1)P(w_2|w_1)...P(w_n|w_1w_2...w_{n-1})$$

- 语言模型的基本功能
  - 判别：判断一段文本是否符合一种语言的语法和语义规则
  - **生成：生成符合一种语言语法或语义规则的文本**




机器翻译




数学计算

# 语言模型

- 给每一个句子都赋予其出现的概率。以$w_1, w_2, \ldots w_n$这n个词组成的句子为例，它出现的概率为

$$P(w_1w_2...w_n) = P(w_1)P(w_2|w_1)...P(w_n|w_1w_2...w_{n-1})$$

- 语言模型的基本功能
  - 判别：判断一段文本是否符合一种语言的语法和语义规则
  - 生成：生成符合一种语言语法或语义规则的文本
  - **任务：信息压缩、构建世界模型**

# 如何学习一个语言模型?

$s = w_1 \cdots w_{\mathrm{T}}$ $\longrightarrow$ $\boxed{f(w_1 \cdots w_T | \theta)}$ $\longrightarrow$ $P(w_1 w_2 \ldots w_n)$

# 方法1：统计世界上所有的句子

$$s = w_1 \cdots w_{\mathrm{T}} \longrightarrow \boxed{f(w_1 \cdots w_T | \theta)} \longrightarrow P(w_1 w_2 \ldots w_n)$$

$$D = \{S_i\}_{i=1}^{\boxed{N}}$$

所有可能组合的句子可能

$$f(\cdot) = \frac{\mathrm{count}(s)}{N}$$

# 方法1：统计世界上所有的句子

$$s = w_1 \cdots w_{\mathrm{T}} \longrightarrow \boxed{f(w_1 \cdots w_T | \theta)} \longrightarrow P(w_1 w_2 \ldots w_n)$$

$$D = \{S_i\}_{i=1}^{\boxed{N}}$$

所有可能组合的句子可能

$$f(\cdot) = \frac{\mathrm{count}(s)}{N}$$

**不可计算！**

# 方法2：统计世界上所有的单词和其组合

$$
\begin{aligned}
P(S=w_{1:T}) &= P(w_1,\cdots,w_T) \\
&= P(w_1)P(w_2|w_1)P(w_3|w_1w_2)\cdots P(w_T|w_{1:(T-1)}) \\
&= P(w_1)\prod_{t=2}^{T} P(w_t|w_{1:(t-1)}) \\
&= \prod_{t=1}^{T} P(w_t|w_{1:(t-1)}),
\end{aligned}
$$

概率公式

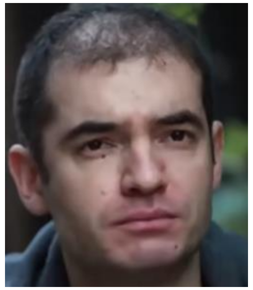
句子中每个词$w_t$ 在给定前面词序列$w_{1:(t-1)}$ 时的条件概率

# 方法2：统计世界上所有的单词和其组合

$$
\begin{aligned}
P(S=w_{1:T}) &= P(w_1,\cdots,w_T) \\
&= P(w_1)P(w_2|w_1)P(w_3|w_1w_2)\cdots P(w_T|w_{1:(T-1)}) \\
&= P(w_1)\prod_{t=2}^{T} P(w_t|w_{1:(t-1)}) \\
&= \prod_{t=1}^{T} P(w_t|w_{1:(t-1)}),
\end{aligned}
$$

概率公式

句子中每个词$w_t$ 在给定前面词序列$w_{1:(t-1)}$ 时的条件概率



Ilya Sutskever
OpenAI CSO

The Next-token prediction is enough for AGI

# 方法2：统计世界上所有的单词和其组合

$$
\begin{aligned}
P(S = w_{1:T}) &= P(w_1, \cdots, w_T) \\
&= P(w_1)P(w_2|w_1)P(w_3|w_1 w_2)\cdots P(w_T|w_{1:(T-1)}) \\
&= P(w_1)\prod_{t=2}^{T} P(w_t|w_{1:(t-1)}) \\
&= \prod_{t=1}^{T} P(w_t|w_{1:(t-1)}),
\end{aligned}
$$

概率公式

句子中每个词$w_t$ 在给定前面词序列$w_{1:(t-1)}$ 时的条件概率

$$\boxed{P(w_t|w_{1:(t-1)}) = P(w_t|w_{(t-n+1):(t-1)})}$$

马尔可夫假设：一个词的概率只依赖于其前面的n－1个词

**N-gram模型**

# 统计语言模型

- Unigram Language Model

$$P(w_1, \ldots, w_T) = \prod_{t=1}^{T} P(w_t)$$

- Bigram Language Model

$$P(w_1, \ldots, w_T) = \prod_{t=1}^{T} P(w_t | w_{t-1})$$

- Trigram Language Model

$$P(w_1, \ldots, w_T) = \prod_{t=1}^{T} P(w_t | w_{t-2}, w_{t-1})$$

- N-gram Language Model

$$P(w_1, \ldots, w_T) = \prod_{t=1}^{T} P(w_t | w_{t-n+1}, \ldots, w_{t-1})$$

# 统计语言模型

- Unigram Language Model

如何进行参数估计?

$$P(w_1,\dots,w_T)=\prod_{t=1}^{T}P(w_t)=\frac{\mathrm{count}(w)}{\sum_{w\in v}\mathrm{count}(w)}$$

- Bigram Language Model

$$P(w_1,\dots,w_T)=\prod_{t=1}^{T}P(w_t|w_{t-1})=\frac{\mathrm{count}(w,w_{t-1})}{\sum_{w\in v}\mathrm{count}(w,w_{t-1})}$$

- Trigram Language Model

$$P(w_1,\dots,w_T)=\prod_{t=1}^{T}P(w_t|w_{t-2},w_{t-1})=\frac{\mathrm{count}(w,w_{t-1},w_{t-2})}{\sum_{w\in v}\mathrm{count}(w,w_{t-1,}w_{t-2,})}$$

- N-gram Language Model

$$P(w_1,\dots,w_T)=\prod_{t=1}^{T}P(w_t|c)=\frac{\mathrm{count}(w,c)}{\sum_{w\in v}\mathrm{count}(w,c)}$$

# 语言模型的评价方法

- **外部评价指标**：看看是否可以提高下游任务的性能
- **内部评价指标**：测试模型在一个特定文本上的所有句子的联合概率

**困惑度**　（Perplexity）

# 困惑度

- 可以用来衡量一个分布的不确定性

分布 $p(x) = P(X = x), x \in \mathcal{X}$

困惑度 $2^{H(P)} = 2^{-\sum_{x \in \mathcal{X}} p(x) \log_2 p(x)}$

- 给定测试集合
  - N个句子: $s_1, \cdots, s_N$
  - 每个句子是独立抽取的: $s_i = w_1^{(i)}, \cdots, w_{n_i}^{(i)}$
  - 用语言模型对每个句子计算概率: $p(s_i)$

$$
\begin{aligned}
\mathcal{PPL}_M &= 2^{-\frac{1}{T}\sum_{i=1}^{N} \log p(s_i)} \\
&= 2^{-\frac{1}{T}\sum_{i=1}^{N}\sum_{j=1}^{n_i} \log p(w_j^{(i)} | w_{(j-n+1):(j-1)}^{(i)})} \\
&= \left(\prod_{i=1}^{N}\prod_{j=1}^{n_i} p(w_j^{(i)} | w_{(j-n+1):(j-1)}^{(i)})\right)^{-1/T}
\end{aligned}
$$

困惑度为每个词条件概率的几何平均数的倒数。句子概率越大，困惑度越小，语言模型越好

# 例子: 实现一个Bigram语言模型

**语料库**

The swift fox jumps over the lazy dog.
The swift river flows under the ancient bridge.
The swift breeze cools the warm summer evening.

**如何实现?**

**语言模型**

$$P(w_1, \ldots, w_T) = \frac{\text{count}(w, w_{t-1})}{\sum_{w \in \mathcal{v}} \text{count}(w, w_{t-1})}$$

# 例子: 实现一个Bigram语言模型

**语料库**

The swift fox jumps over the lazy dog.
The swift river flows under the ancient bridge.
The swift breeze cools the warm summer evening.

**语言模型**

$$P(w_1, \dots, w_T) = \frac{\text{count}(w, w_{t-1})}{\sum_{w \in v} \text{count}(w, w_{t-1})}$$

**如何实现?**

统计语言模型 – "数数"(Counting)

# 例子: 实现一个Bigram语言模型

## 语料库

The swift fox jumps over the lazy dog.
The swift river flows under the ancient bridge.
The swift breeze cools the warm summer evening.

## 语言模型

$$P(w_1, \dots, w_T) = \frac{\mathrm{count}(w, w_{t-1})}{\sum_{w \in v} \mathrm{count}(w, w_{t-1})}$$

```python
from collections import defaultdict
import numpy as np

# Define sentences
sentences = [
    "The swift fox jumps over the lazy dog.",
    "The swift river flows under the ancient bridge.",
    "The swift breeze cools the warm summer evening."
]

# Preprocess sentences: tokenize and add start (<s>) and end (</s>) tokens
preprocessed_sentences = [["<s>"] + sentence.lower().replace(".", "").split() + ["</s>"] for

# Initialize bigram model
bigram_model = defaultdict(lambda: defaultdict(lambda: 0))

# Populate the bigram model with counts
for sentence in preprocessed_sentences:
    for w1, w2 in zip(sentence[:-1], sentence[1:]):
        bigram_model[w1][w2] += 1

# Convert counts to probabilities
for w1 in bigram_model:
    total_count = float(sum(bigram_model[w1].values()))
    for w2 in bigram_model[w1]:
        bigram_model[w1][w2] /= total_count

# Display the bigram probabilities
bigram_model_probs = {w1: dict(w2) for w1, w2 in bigram_model.items()}
bigram_model_probs
```

# 例子: 实现一个Bigram语言模型

### 语料库

The swift fox jumps over the lazy dog.
The swift river flows under the ancient bridge.
The swift breeze cools the warm summer evening.

### 语言模型

$$P(w_1, \dots, w_T) = \frac{\text{count}(w, w_{t-1})}{\sum_{w \in v} \text{count}(w, w_{t-1})}$$

```python
from collections import defaultdict
import numpy as np

# Define sentences
sentences = [
    "The swift fox jumps over the lazy dog.",
    "The swift river flows under the ancient bridge.",
    "The swift breeze co
]

# Preprocess sentences: tokenize and add start (<s>) and end (</s>) tokens
preprocessed_sentences = [["<s>"] + sentence.lower().replace(".", "").split() + ["</s>"] for

# Initialize bigram model
bigram_model = defaultdict(lambda: defaultdict(lambda: 0))

# Populate the bigram model with counts
for sentence in preprocessed_sentences:
    for w1, w2 in zip(sentence[:-1], sentence[1:]):
        bigram_model[w1][w2] += 1

# Convert counts to probabilities
for w1 in bigram_model:
    total_count = float(sum(bigram_model[w1].values()))
    for w2 in bigram_model[w1]:
        bigram_model[w1][w2] /= total_count

# Display the bigram probabilities
bigram_model_probs = {w1: dict(w2) for w1, w2 in bigram_model.items()}
bigram_model_probs
```

添加首位结束符

# 例子: 实现一个Bigram语言模型

### 语料库

The swift fox jumps over the lazy dog.
The swift river flows under the ancient bridge.
The swift breeze cools the warm summer evening.

### 语言模型

$$P(w_1, \ldots, w_T) = \frac{\text{count}(w, w_{t-1})}{\sum_{w \in v} \text{count}(w, w_{t-1})}$$

```python
from collections import defaultdict
import numpy as np

# Define sentences
sentences = [
    "The swift fox jumps over the lazy dog.",
    "The swift river flows under the ancient bridge.",
    "The swift breeze cools the warm
]

# Preprocess sentences: tokenize and add start (<s>) and end (</s>) tokens
preprocessed_sentences = [["<s>"] + sentence.lower().replace(".", "").split() + ["</s>"] for

# Initialize bigram model
bigram_model = defaultdict(lambda: defaultdict(lambda: 0))

# Populate the bigram model with counts
for sentence in preprocessed_sentences:
    for w1, w2 in zip(sentence[:-1], sentence[1:]):
        bigram_model[w1][w2] += 1

# Convert counts to probabilities
for w1 in bigram_model:
    total_count = float(sum(bigram_model[w1].values()))
    for w2 in bigram_model[w1]:
        bigram_model[w1][w2] /= total_count

# Display the bigram probabilities
bigram_model_probs = {w1: dict(w2) for w1, w2 in bigram_model.items()}
bigram_model_probs
```

数据预处理

# 例子: 实现一个Bigram语言模型

**语料库**

The swift fox jumps over the lazy dog.
The swift river flows under the ancient bridge.
The swift breeze cools the warm summer evening.

**语言模型**

$$P(w_1, \ldots, w_T) = \frac{\mathrm{count}(w, w_{t-1})}{\sum_{w \in v} \mathrm{count}(w, w_{t-1})}$$

**统计量**

count(the swift) = 3

…

| Previous Word | Next Word | Probability |
|---|---|---|
| <s> | the | 1.000 |
| ancient | bridge | 1.000 |
| breeze | cools | 1.000 |
| bridge | </s> | 1.000 |
| cools | the | 1.000 |
| dog | </s> | 1.000 |
| evening | </s> | 1.000 |
| flows | under | 1.000 |
| fox | jumps | 1.000 |
| jumps | over | 1.000 |
| lazy | dog | 1.000 |
| over | the | 1.000 |
| river | flows | 1.000 |
| summer | evening | 1.000 |
| swift | breeze | 0.333 |
| swift | fox | 0.333 |
| swift | river | 0.333 |
| the | ancient | 0.167 |
| the | lazy | 0.167 |
| the | swift | 0.500 |
| the | warm | 0.167 |
| under | the | 1.000 |
| warm | summer | 1.000 |

# 例子: 实现一个Bigram语言模型

### 语料库

The swift fox jumps over the lazy dog.
The swift river flows under the ancient bridge.
The swift breeze cools the warm summer evening.

### 语言模型

$$P(w_1, \ldots, w_T) = \frac{\text{count}(w, w_{t-1})}{\sum_{w \in \mathcal{v}} \text{count}(w, w_{t-1})}$$

### 参数估计

P(swift|the) = 3/6
P(over|jumps) = 1

| Previous Word | Next Word | Probability |
|---|---|---|
| <s> | the | 1.000 |
| ancient | bridge | 1.000 |
| breeze | cools | 1.000 |
| bridge | </s> | 1.000 |
| cools | the | 1.000 |
| dog | </s> | 1.000 |
| evening | </s> | 1.000 |
| flows | under | 1.000 |
| fox | jumps | 1.000 |
| jumps | over | 1.000 |
| lazy | dog | 1.000 |
| over | the | 1.000 |
| river | flows | 1.000 |
| summer | evening | 1.000 |
| swift | breeze | 0.333 |
| swift | fox | 0.333 |
| swift | river | 0.333 |
| the | ancient | 0.167 |
| the | lazy | 0.167 |
| the | swift | 0.500 |
| the | warm | 0.167 |
| under | the | 1.000 |
| warm | summer | 1.000 |

# 例子: 实现一个Bigram语言模型

### 语料库

The swift fox jumps over the lazy dog.
The swift river flows under the ancient bridge.
The swift breeze cools the warm summer evening.

### 语言模型

$$P(w_1, \dots, w_T) = \frac{\text{count}(w, w_{t-1})}{\sum_{w \in \nu} \text{count}(w, w_{t-1})}$$

p(The swift fox)?

| Previous Word | Next Word | Probability |
|---|---|---|
| <s> | the | 1.000 |
| ancient | bridge | 1.000 |
| breeze | cools | 1.000 |
| bridge | </s> | 1.000 |
| cools | the | 1.000 |
| dog | </s> | 1.000 |
| evening | </s> | 1.000 |
| flows | under | 1.000 |
| fox | jumps | 1.000 |
| jumps | over | 1.000 |
| lazy | dog | 1.000 |
| over | the | 1.000 |
| river | flows | 1.000 |
| summer | evening | 1.000 |
| swift | breeze | 0.333 |
| swift | fox | 0.333 |
| swift | river | 0.333 |
| the | ancient | 0.167 |
| the | lazy | 0.167 |
| the | swift | 0.500 |
| the | warm | 0.167 |
| under | the | 1.000 |
| warm | summer | 1.000 |

# 例子: 实现一个Bigram语言模型

### 语料库

The swift fox jumps over the lazy dog.
The swift river flows under the ancient bridge.
The swift breeze cools the warm summer evening.

### 语言模型

$$P(w_1, \ldots, w_T) = \frac{\text{count}(w, w_{t-1})}{\sum_{w \in \mathcal{V}} \text{count}(w, w_{t-1})}$$

p(The swift fox) = P(the|<s>) P(swift|the) P(fox|swift)

| Previous Word | Next Word | Probability |
|---|---|---|
| <s> | the | 1.000 |
| ancient | bridge | 1.000 |
| breeze | cools | 1.000 |
| bridge | </s> | 1.000 |
| cools | the | 1.000 |
| dog | </s> | 1.000 |
| evening | </s> | 1.000 |
| flows | under | 1.000 |
| fox | jumps | 1.000 |
| jumps | over | 1.000 |
| lazy | dog | 1.000 |
| over | the | 1.000 |
| river | flows | 1.000 |
| summer | evening | 1.000 |
| swift | breeze | 0.333 |
| swift | fox | 0.333 |
| swift | river | 0.333 |
| the | ancient | 0.167 |
| the | lazy | 0.167 |
| the | swift | 0.500 |
| the | warm | 0.167 |
| under | the | 1.000 |
| warm | summer | 1.000 |

# 例子: 实现一个Bigram语言模型

### 语料库

The swift fox jumps over the lazy dog.
The swift river flows under the ancient bridge.
The swift breeze cools the warm summer evening.

### 语言模型

$$P(w_1, \dots, w_T) = \frac{\text{count}(w, w_{t-1})}{\sum_{w \in \mathcal{v}} \text{count}(w, w_{t-1})}$$

p(The swift fox) = P(the|<s>) P(swift|the) P(fox|swift)

= 1.0×0.5×0.3333=0.1667

| Previous Word | Next Word | Probability |
|---|---|---|
| <s> | the | 1.000 |
| ancient | bridge | 1.000 |
| breeze | cools | 1.000 |
| bridge | </s> | 1.000 |
| cools | the | 1.000 |
| dog | </s> | 1.000 |
| evening | </s> | 1.000 |
| flows | under | 1.000 |
| fox | jumps | 1.000 |
| jumps | over | 1.000 |
| lazy | dog | 1.000 |
| over | the | 1.000 |
| river | flows | 1.000 |
| summer | evening | 1.000 |
| swift | breeze | 0.333 |
| swift | fox | 0.333 |
| swift | river | 0.333 |
| the | ancient | 0.167 |
| the | lazy | 0.167 |
| the | swift | 0.500 |
| the | warm | 0.167 |
| under | the | 1.000 |
| warm | summer | 1.000 |

# 例子: 实现一个Bigram语言模型

## 语料库

The swift fox jumps over the lazy dog.
The swift river flows under the ancient bridge.
The swift breeze cools the warm summer evening.

## 语言模型

$$P(w_1, \dots, w_T) = \frac{\text{count}(w, w_{t-1})}{\sum_{w \in \nu} \text{count}(w, w_{t-1})}$$

P(I love this movie) = ?

| Previous Word | Next Word | Probability |
|---|---|---|
| <s> | the | 1.000 |
| ancient | bridge | 1.000 |
| breeze | cools | 1.000 |
| bridge | </s> | 1.000 |
| cools | the | 1.000 |
| dog | </s> | 1.000 |
| evening | </s> | 1.000 |
| flows | under | 1.000 |
| fox | jumps | 1.000 |
| jumps | over | 1.000 |
| lazy | dog | 1.000 |
| over | the | 1.000 |
| river | flows | 1.000 |
| summer | evening | 1.000 |
| swift | breeze | 0.333 |
| swift | fox | 0.333 |
| swift | river | 0.333 |
| the | ancient | 0.167 |
| the | lazy | 0.167 |
| the | swift | 0.500 |
| the | warm | 0.167 |
| under | the | 1.000 |
| warm | summer | 1.000 |

# 例子: 实现一个Bigram语言模型

### 语料库

The swift fox jumps over the lazy dog.
The swift river flows under the ancient bridge.
The swift breeze cools the warm summer evening.

### 语言模型

$$P(w_1, \ldots, w_T) = \frac{\text{count}(w, w_{t-1})}{\sum_{w \in v} \text{count}(w, w_{t-1})}$$

P(I love this movie) = P(I|<s>)… = 0

| Previous Word | Next Word | Probability |
|---|---|---|
| <s> | the | 1.000 |
| ancient | bridge | 1.000 |
| breeze | cools | 1.000 |
| bridge | </s> | 1.000 |
| cools | the | 1.000 |
| dog | </s> | 1.000 |
| evening | </s> | 1.000 |
| flows | under | 1.000 |
| fox | jumps | 1.000 |
| jumps | over | 1.000 |
| lazy | dog | 1.000 |
| over | the | 1.000 |
| river | flows | 1.000 |
| summer | evening | 1.000 |
| swift | breeze | 0.333 |
| swift | fox | 0.333 |
| swift | river | 0.333 |
| the | ancient | 0.167 |
| the | lazy | 0.167 |
| the | swift | 0.500 |
| the | warm | 0.167 |
| under | the | 1.000 |
| warm | summer | 1.000 |

# 基于n-gram的语言模型的问题

- 频率估计可靠性依赖于语料大小
- 包含没有见过的ngram的句子的概率为0

# 如何处理未登录词

- 构建词表的时候把低频词替换成<UNK>
- 计算的时候如果遇到未登录词当作<UNK>处理

# 如何处理未登录词

- 构建词表的时候把低频词替换成<UNK>
- 计算的时候如果遇到未登录词当作<UNK>处理

未登录词（Out-Of-Vocabulary words, 简称OOV），是指在自然语言处理（NLP）、信息检索、计算机词汇学等领域中，那些没有出现在现有词汇数据库、字典或索引中的词语

如何估计未登录词的频率

# 语言模型的平滑

**平滑技术**：是增加低频词的频率，而降低高频词的频率

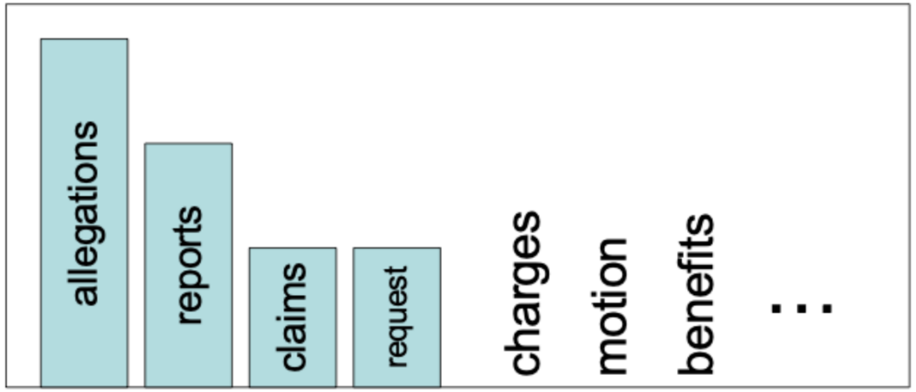
$$\boxed{P(w_t|w_{1:(t-1)}) = P(w_t|w_{(t-n+1):(t-1)})} = \frac{\mathbf{count}(w_{(t-n+1):t})}{\mathbf{count}(w_{(t-n+1):(t-1)})}$$

$$= \frac{\text{count}(w_{(t-n+1):t}) + \delta}{\text{count}(w_{(t-n+1):(t-1)}) + \delta|\mathcal{V}|}$$

# 语言模型的平滑

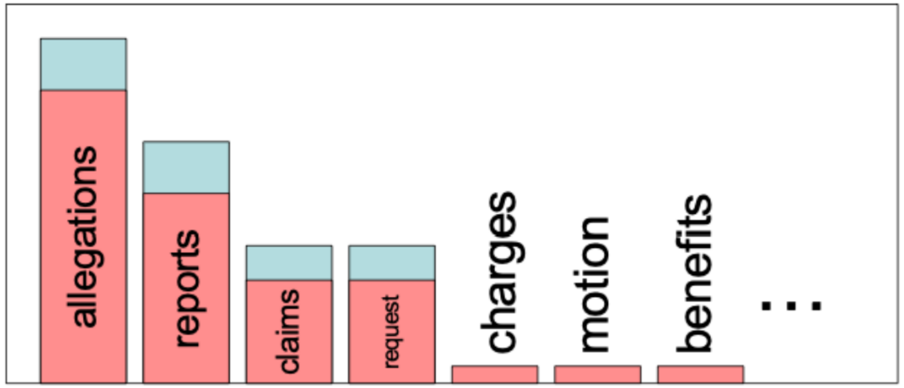
P(w | denied the)
3 allegations
2 reports
1 claims
1 request

7 total




P(w | denied the)
2.5 allegations
1.5 reports
0.5 claims
0.5 request
2 other

7 total




图来自于：Dan Klein and John DeNero

# 神经网络语言模型

- 给每一个句子（$S$）都赋予其出现的概率。以$w_1, w_2, \dots w_T$,这T个词组成的句子为例，它出现的概率为

$$
\begin{aligned}
P(S = w_{1:T}) &= P(w_1, \cdots, w_T) \\
&= P(w_1)P(w_2|w_1)P(w_3|w_1w_2)\cdots P(w_T|w_{1:(T-1)}) \\
&= P(w_1)\prod_{t=2}^{T} P(w_t|w_{1:(t-1)}) \\
&= \prod_{t=1}^{T} \boxed{P(w_t|w_{1:(t-1)})},
\end{aligned}
$$

句子中每个词$w_t$ 在给定前面词序列$w_{1:(t-1)}$ 时的条件概率，如何变成一个分类问题？

# 神经网络语言模型

- 给每一个句子（*S*）都赋予其出现的概率。以$w_{1,} w_{2,} \ldots w_{T,}$这T个词组成的句子为例，它出现的概率为

$$
\begin{aligned}
P(S = w_{1:T}) &= P(w_1, \cdots, w_T) \\
&= P(w_1)P(w_2|w_1)P(w_3|w_1 w_2)\cdots P(w_T|w_{1:(T-1)}) \\
&= P(w_1)\prod_{t=2}^{T} P(w_t|w_{1:(t-1)}) \\
&= \prod_{t=1}^{T} \boxed{P(w_t|w_{1:(t-1)})},
\end{aligned}
$$

**这个问题可以转换为一个类别数为|V|的多类分类问题**

$$P_\theta(v_k|w_{1:(t-1)})$$ 词汇表V 中的每个词 $v_k(1 \leqslant k \leqslant |V|$ 出现的概率

$$= f_k(w_{1:(t-1)}, \theta),$$

# 神经网络语言模型

- 给每一个句子（$S$）都赋予其出现的概率。以$w_1, w_2, \ldots w_T$, 这T个词组成的句子为例，它出现的概率为

$$
\begin{aligned}
P(S = w_{1:T}) &= P(w_1, \cdots, w_T) \\
&= P(w_1)P(w_2|w_1)P(w_3|w_1w_2)\cdots P(w_T|w_{1:(T-1)}) \\
&= P(w_1)\prod_{t=2}^{T} P(w_t|w_{1:(t-1)}) \\
&= \prod_{t=1}^{T} \boxed{P(w_t|w_{1:(t-1)})},
\end{aligned}
$$

**这个问题可以转换为一个类别数为|V|的多类分类问题**

$P_\theta(v_k|w_{1:(t-1)})$ 词汇表V 中的每个词 $v_k(1 \leqslant k \leqslant |V|$ 出现的概率

$= f_k(w_{1:(t-1)}, \theta),$

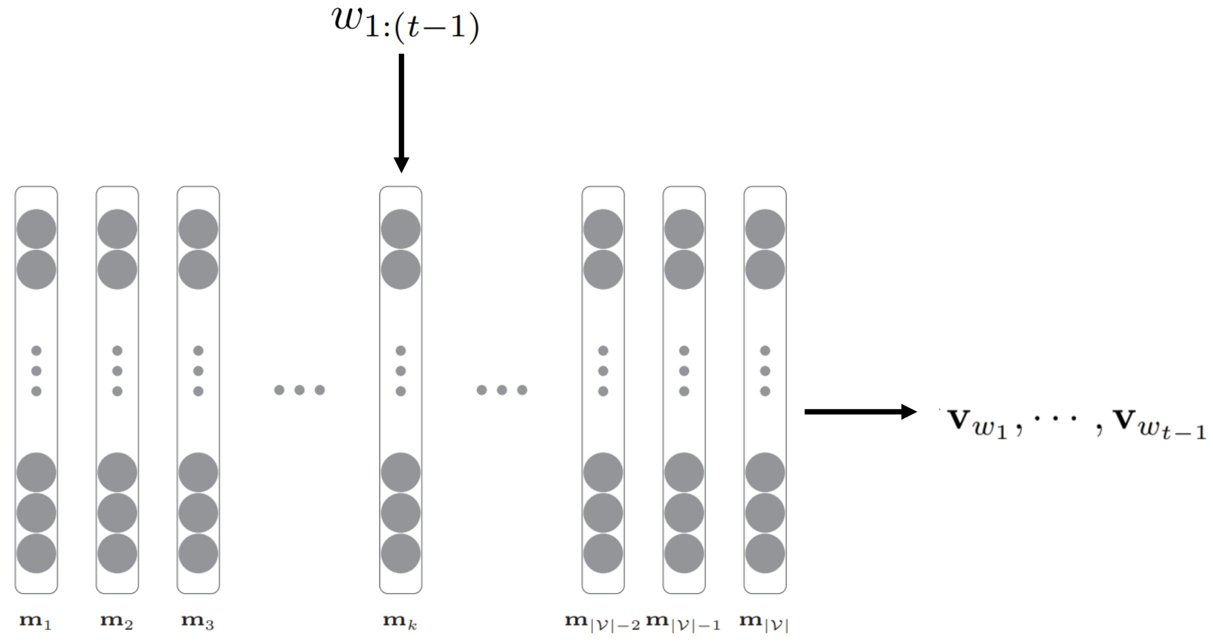
可以用不同分类器来估计语言模型的条件概率

# 神经网络语言模型

- 输入层
  - 功能：将离散的单词 $w_{1:(t-1)}$ 转化成向量表示
  - **词嵌入矩阵**：存储词表大小多个向量（$\mathbf{M} \in \mathbb{R}^{d_1 \times |\mathcal{V}|}$）
  - 查表： $\mathbf{v}_{w_i} = \mathbf{M}\mathbf{e}_k =_k$

# 神经网络语言模型

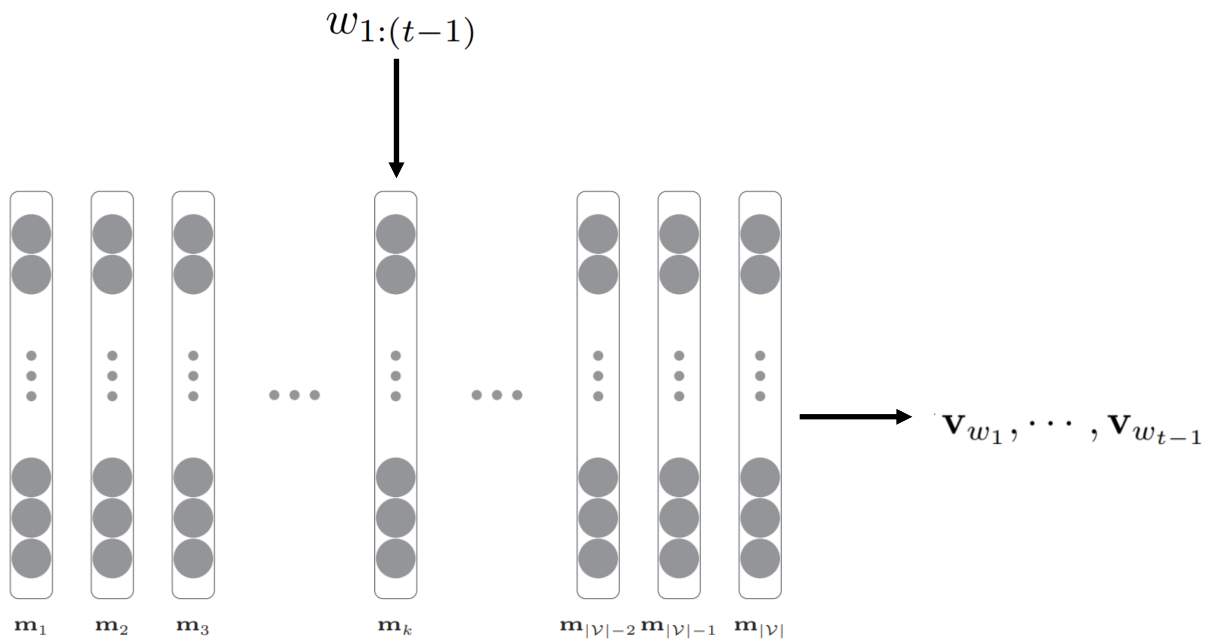
- 输入层
  - 功能：将离散的单词 $w_{1:(t-1)}$ 转化成向量表示
  - 词嵌入矩阵：存储词表大小多个向量（$\mathbf{M} \in \mathbb{R}^{d_1 \times |\mathcal{V}|}$）
  - 查表： $\mathbf{v}_{w_i} = \mathbf{M}\mathbf{e}_k =_k$



# 神经网络语言模型

## 输入层

- 功能：将离散的单词 $w_{1:(t-1)}$ 转化成向量表示
- 词嵌入矩阵：存储词表大小多个向量（$\mathbf{M} \in \mathbb{R}^{d_1 \times |\mathcal{V}|}$）
- 查表： $\mathbf{v}_{w_i} = \mathbf{M}\mathbf{e}_k =_k$




I love the movie very much!

| I | good | hit | do | book | | July |
|---|---|---|---|---|---|---|
| 1 | 2 | 1 | 3 | 3 | | 0 |
| 0 | 1 | 0 | 0 | 2 | | 0 |
| 0 | 0 | 1 | 1 | -1 | ... | 3 |
| 0 | 3 | 0 | 0 | 0 | | 1 |
| 0 | 1 | 1 | 0 | 3 | | 1 |

| | | | | | |
|---|---|---|---|---|---|
| I | 1 | 0 | 0 | 0 | 0 |
| love | 2 | 1 | 1 | 2 | 1 |
| the | 1 | 1 | 2 | 2 | 0 |
| movie | 2 | 2 | 1 | 0 | 0 |
| very | 2 | 1 | 2 | 1 | 1 |
| much | 2 | 2 | 1 | 0 | 0 |
| ! | 2 | 1 | 2 | 1 | 1 |

# 神经网络语言模型

- 隐藏层
  - 功能：总结输入的词向量为历史信息向量
  - 输入：$\mathbf{v}_{w_1}, \cdots, \mathbf{v}_{w_{t-1}}$
  - 输出：$\mathbf{h}_t$

# 神经网络语言模型

- 隐藏层
  - 功能：总结输入的词向量为历史信息向量
  - 输入：$\mathbf{v}_{w_1}, \cdots, \mathbf{v}_{w_{t-1}}$
  - 输出：$\mathbf{h}_t$

如何实现？

# 神经网络语言模型

- 隐藏层
  - 功能：总结输入的词向量为历史信息向量
  - 输入： $\mathbf{v}_{w_1}, \cdots, \mathbf{v}_{w_{t-1}}$
  - 输出： $\mathbf{h}_t$
  - 网络类型：
    - Bag-of-words

$$\mathbf{h}_t = \sum_{1}^{t-1} C_i \mathbf{v}_{w_i},$$

# 神经网络语言模型

- 隐藏层
  - 功能：总结输入的词向量为历史信息向量
  - 输入： $\mathbf{v}_{w_1}, \cdots, \mathbf{v}_{w_{t-1}}$
  - 输出： $\mathbf{h}_t$
  - 网络类型：
    - Bag-of-words
    - 前馈神经网络:

$$\mathbf{x}_t = \mathbf{v}_{w_{t-n+1}} \oplus \cdots \oplus \mathbf{v}_{w_{t-1}}$$

$$\mathbf{h}_t = \tanh(W\mathbf{x}_t + \mathbf{c})$$

# 神经网路语言模型

- 隐藏层
  - 功能：总结输入的词向量为历史信息向量
  - 输入： $\mathbf{v}_{w_1}, \cdots, \mathbf{v}_{w_{t-1}}$
  - 输出： $\mathbf{h}_t$
  - 网络类型:
    - Bag-of-words
    - 前馈神经网络
    - 循环神经网络

$$\mathbf{h}_t = \tanh(U\mathbf{h}_{t-1} + W\mathbf{v}_{w_{t-1}} + \mathbf{c}),$$

# 神经网络语言模型

- 输出层
  - 大小：输出层大小为|V|
  - 输入：历史信息表示向量 $\mathbf{h}_t \in \mathbb{R}^{d_2}$
  - 输出：词表大小的概率分布 $\mathbf{y}_t \in \mathbb{R}^{|\mathcal{V}|}$

# 神经网络语言模型

- 输出层
  - 大小：输出层大小为|V|
  - 输入：历史信息表示向量 $\mathbf{h}_t \in \mathbb{R}^{d_2}$
  - 输出：词表大小的概率分布 $\mathbf{y}_t \in \mathbb{R}^{|\mathcal{V}|}$

$$\mathbf{y}_t = \mathrm{softmax}(\mathbf{O}\mathbf{h}_t + \mathbf{b}),$$

输出词嵌入矩阵

$$\begin{aligned} P_\theta(v_k|h_t) &= [\mathbf{y}_t]_k \\ &= \mathrm{softmax}(s(v_k, h_t; \theta)) \\ &= \frac{\exp(s(v_k, h_t; \theta))}{\sum_{j=1}^{|\mathcal{V}|} \exp(s(v_j, h_t; \theta))}, \end{aligned}$$

其中：

$$s(v_k, h; \theta) = \mathbf{o}_k^{\mathrm{T}} \mathbf{h} + b_k$$

# 神经网络语言模型

- 输出层
  - 大小：输出层大小为|V|
  - 输入：历史信息表示向量 $\mathbf{h}_t \in \mathbb{R}^{d_2}$
  - 输出：词表大小的概率分布 $\mathbf{y}_t \in \mathbb{R}^{|\mathcal{V}|}$

$$\mathbf{y}_t = \text{softmax}(\mathbf{O}\mathbf{h}_t + \mathbf{b}),$$

输出词嵌入矩阵

- 输入层
  - 功能：将离散的单词 $w_{1:(t-1)}$ 转化成向量表示
  - 词嵌入矩阵：存储词表大小多个向量（$\mathbf{M} \in \mathbb{R}^{d_1 \times |\mathcal{V}|}$）
  - 查表：$\mathbf{v}_{w_i} = \mathbf{M}\mathbf{e}_k =_k$

$$\begin{aligned} P_\theta(v_k|h_t) &= [\mathbf{y}_t]_k \\ &= \text{softmax}(s(v_k, h_t; \theta)) \\ &= \frac{\exp(s(v_k, h_t; \theta))}{\sum_{j=1}^{|\mathcal{V}|} \exp(s(v_j, h_t; \theta))}, \end{aligned}$$

其中：

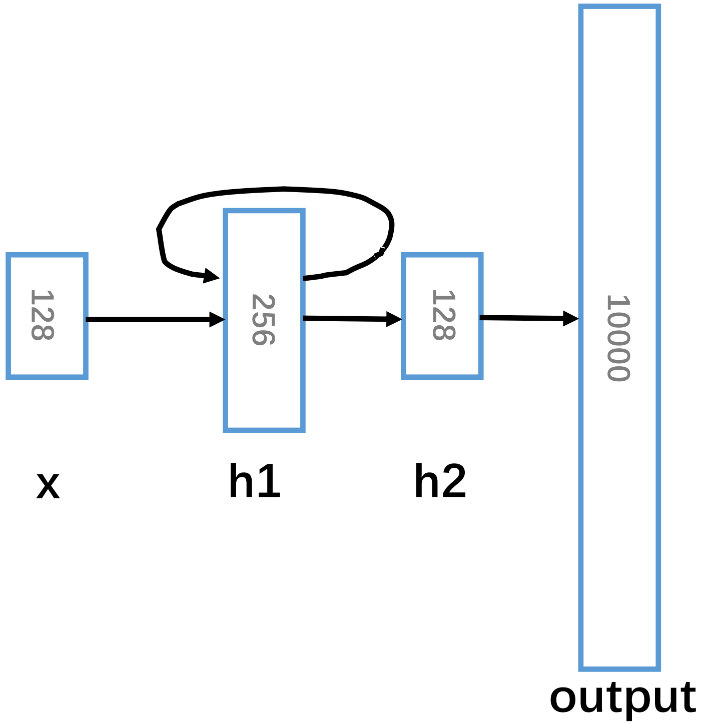
$$s(v_k, h; \theta) = \mathbf{o}_k^{\mathrm{T}}\mathbf{h} + b_k$$

# 样例实现：基于循环神经网络语言模型

- 1个输入层、1个RNN层、一个全连接层和输出层
- 假设词表大小为10000

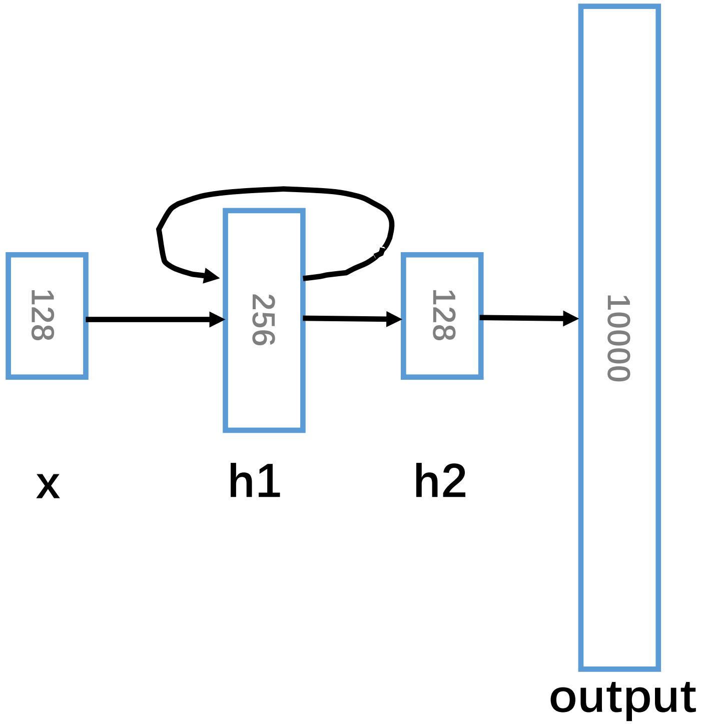
# 样例实现：基于循环神经网络语言模型

- 1个输入层、1个RNN层、一个全连接层和输出层
- 假设词表大小为10000



# 样例实现：基于循环神经网络语言模型

- 1个输入层、1个RNN层、一个全连接层和输出层
- 假设词表大小为10000



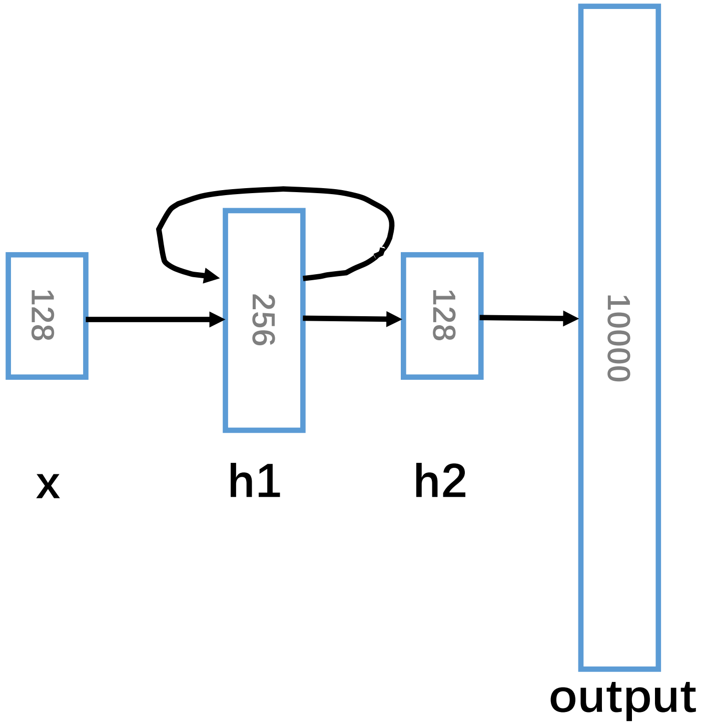

```python
vocab_size = 10000  # 假设我们有10000个唯一的单词
embedding_dim = 128
hidden_dim1 = 256
hidden_dim2 = 128
```

```python
class ThreeLayerNNLM(nn.Module):
    def __init__(self, vocab_size, embedding_dim, hidden_dim1, hidden_dim2):
        super(ThreeLayerNNLM, self).__init__()
        self.embedding = nn.Embedding(vocab_size, embedding_dim)
        self.rnn = nn.RNN(embedding_dim, hidden_dim1, batch_first=True)
        self.hidden2 = nn.Linear(hidden_dim1, hidden_dim2)
        self.output = nn.Linear(hidden_dim2, vocab_size)

    def forward(self, x):
        embeds = self.embedding(x)
        out, _ = self.rnn(embeds)
        out = torch.relu(self.hidden2(out[:, -1, :])) # 取RNN最后时刻的输出
        out = self.output(out)
        return out
```

# 样例实现：基于循环神经网络语言模型

- 1个输入层、1个RNN层、一个全连接层和输出层
- 假设词表大小为10000




```python
model = ThreeLayerNNLM(vocab_size, embedding_dim, hidden_dim1, hidden_dim2)
loss_function = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)
```

```python
# 假设X_train和y_train已经准备好，分别表示输入的单词序列和目标单词
for epoch in range(num_epochs):
    for batch in batch_loader(X_train, y_train, batch_size):
        inputs, targets = batch
        model.zero_grad()
        outputs = model(inputs)
        loss = loss_function(outputs, targets)
        loss.backward()
        optimizer.step()
```

# 论文赏析：A Neural Probabilistic Language Model

## A Neural Probabilistic Language Model

**Yoshua Bengio** BENGIOY@IRO.UMONTREAL.CA
**Réjean Ducharme** DUCHARME@IRO.UMONTREAL.CA
**Pascal Vincent** VINCENTP@IRO.UMONTREAL.CA
**Christian Jauvin** JAUVINC@IRO.UMONTREAL.CA
*Département d'Informatique et Recherche Opérationnelle*
*Centre de Recherche Mathématiques*
*Université de Montréal, Montréal, Québec, Canada*

关注作者、团队

关注机构

A **neural probabilistic language model**

Y Bengio, R Ducharme… - Advances in **neural** …, 2000 - proceedings.neurips.cc

A goal of statistical **language modeling** is to learn the joint **probability** function of sequences of words. This is intrinsically difficult because of the curse of dimensionality: we propose to …

Save Cite Cited by 11082 Related articles All 63 versions

影响力

# 论文赏析：A Neural Probabilistic Language Model

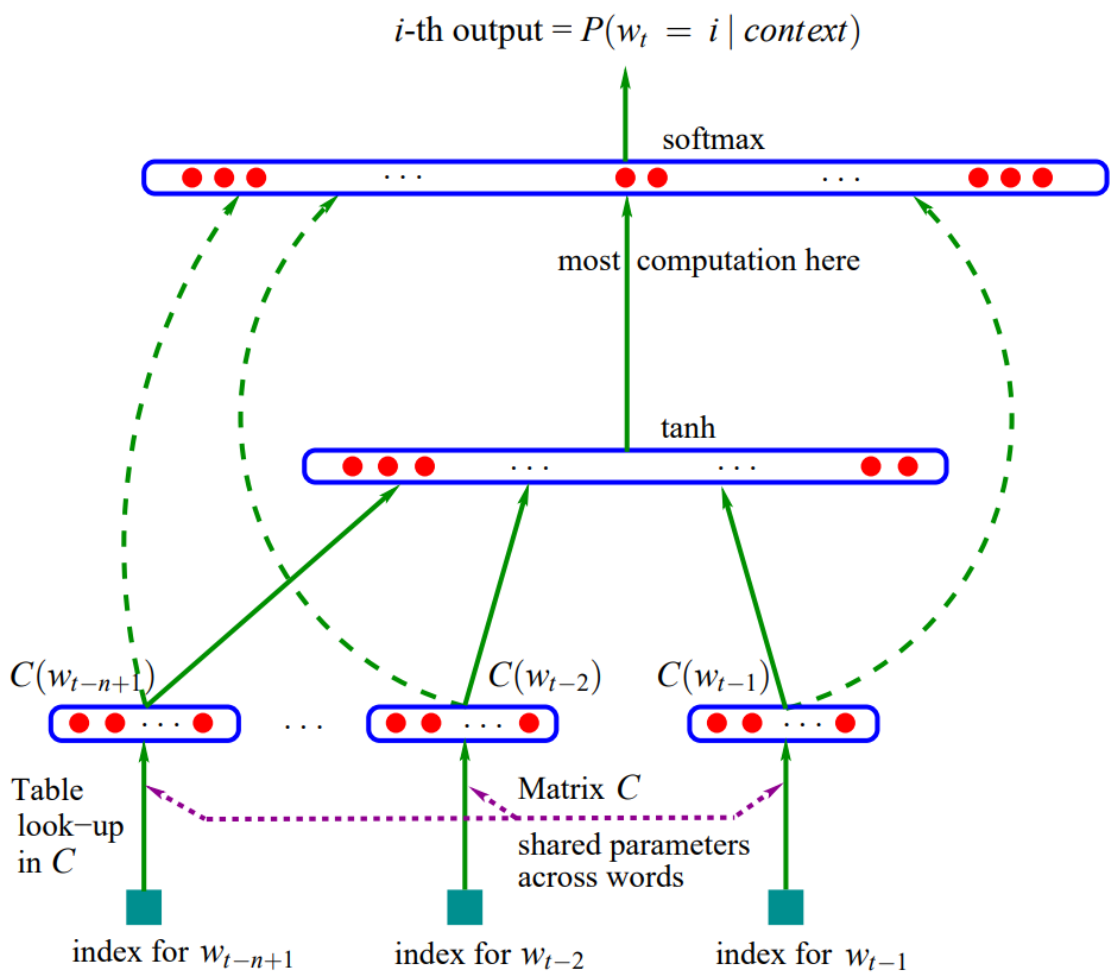



Figure 1: Neural architecture: $f(i, w_{t-1}, \cdots, w_{t-n+1}) = g(i, C(w_{t-1}), \cdots, C(w_{t-n+1}))$ where $g$ is the neural network and $C(i)$ is the $i$-th word feature vector.

# 论文赏析：A Neural Probabilistic Language Model

**未来十多年来遇到的难题！**

$$
\begin{aligned}
P_\theta(v_k|h_t) &= [\mathbf{y}_t]_k \\
&= \mathrm{softmax}(s(v_k, h_t; \theta)) \\
&= \frac{\exp(s(v_k, h_t; \theta))}{\sum_{j=1}^{|\mathcal{V}|} \exp(s(v_j, h_t; \theta))},
\end{aligned}
$$

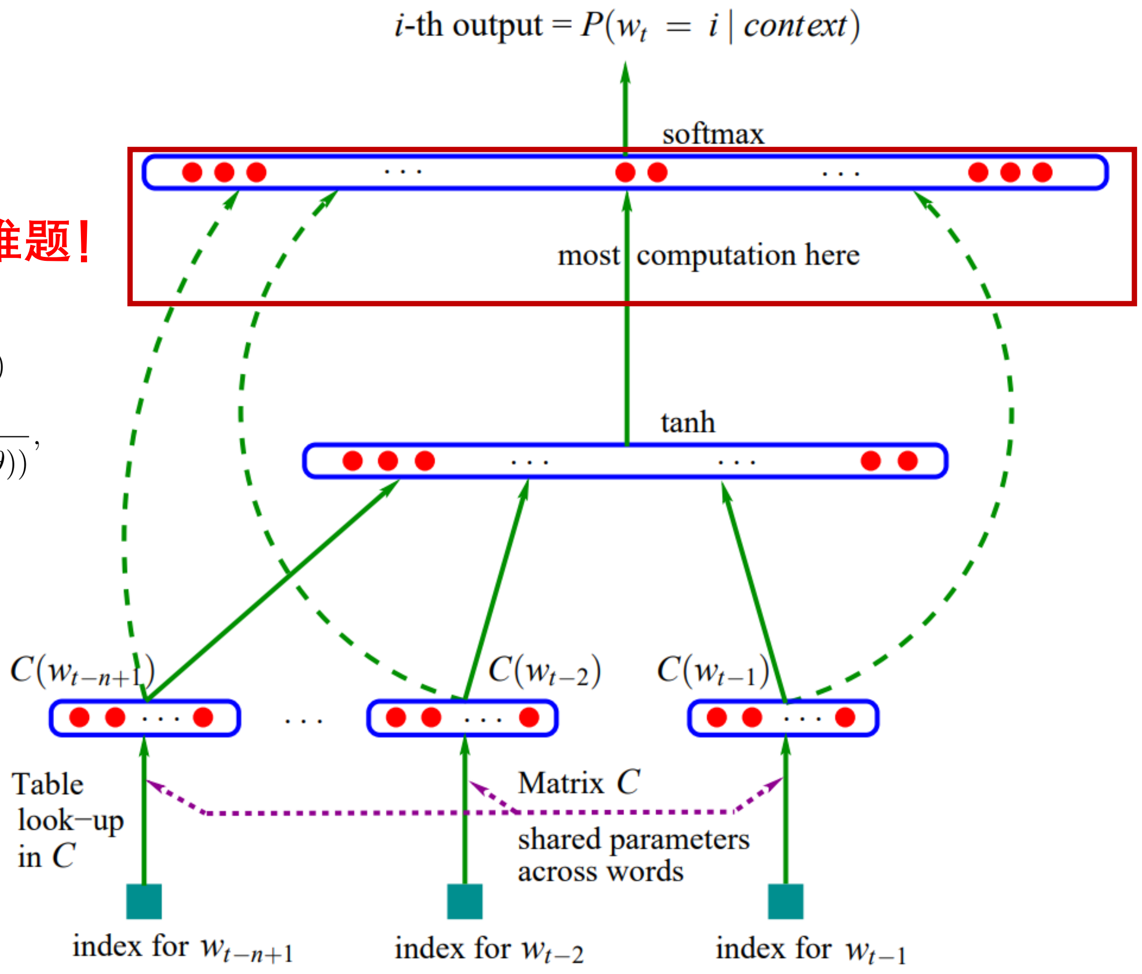



Figure 1: Neural architecture: $f(i, w_{t-1}, \cdots, w_{t-n+1}) = g(i, C(w_{t-1}), \cdots, C(w_{t-n+1}))$ where $g$ is the neural network and $C(i)$ is the $i$-th word feature vector.

# 论文赏析：A Neural Probabilistic Language Model

**包含了哪些的大胆想法：**

- 即使输出层很大，仍然用NN进行建模
- 使用FFN架构建模
- 词表示层、提出词表示学习任务
- 解决变长问题
- 做出来效果了

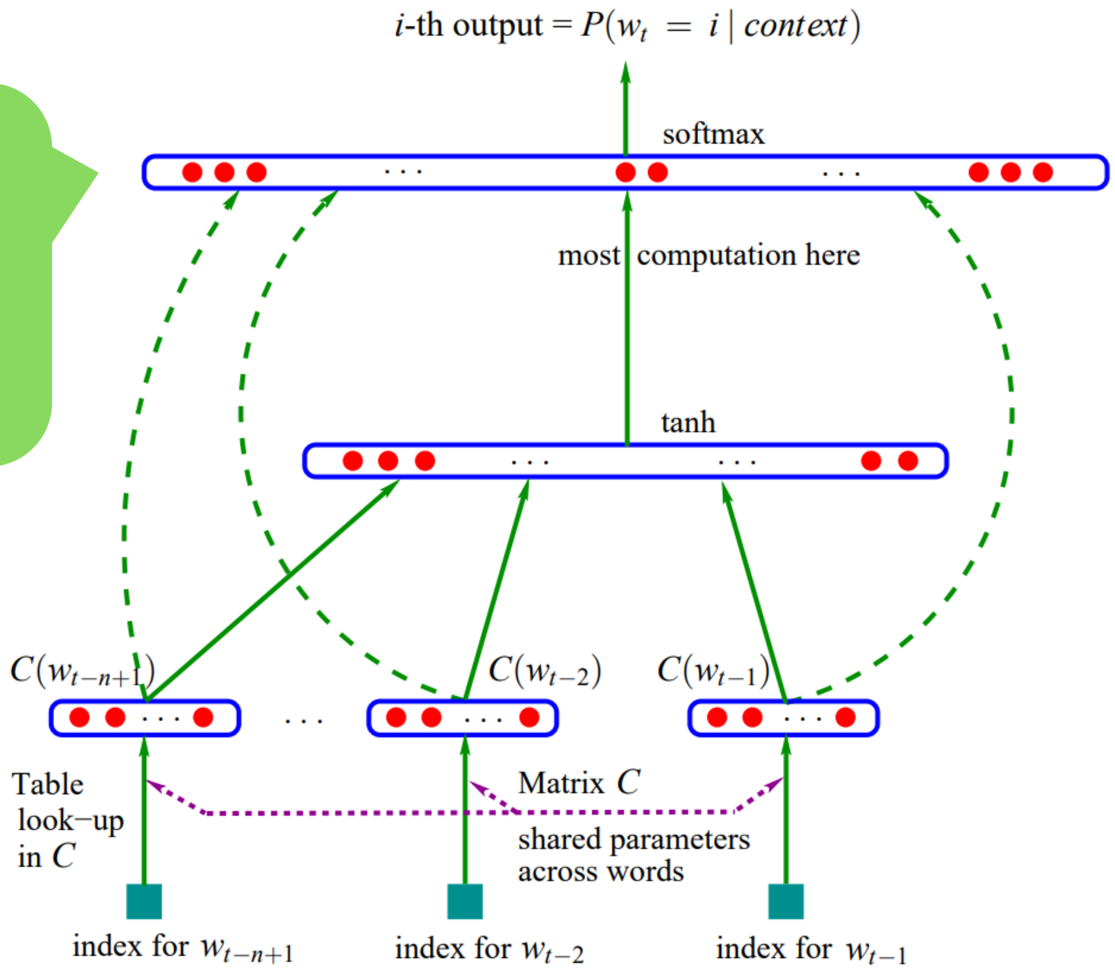



Figure 1: Neural architecture: $f(i, w_{t-1}, \cdots, w_{t-n+1}) = g(i, C(w_{t-1}), \cdots, C(w_{t-n+1}))$ where $g$ is the neural network and $C(i)$ is the $i$-th word feature vector.

# 论文赏析：A Neural Probabilistic Language Model

1. associate with each word in the vocabulary a distributed *word feature vector* (a real-valued vector in $\mathbb{R}^m$),
2. express the joint *probability function* of word sequences in terms of the feature vectors of these words in the sequence, and
3. learn simultaneously the *word feature vectors* and the parameters of that *probability function*.

# 论文赏析：A Neural Probabilistic Language Model

1. Decomposing the network in sub-networks, for example using a clustering of the words. Training many smaller networks should be easier and faster.

2. Representing the conditional probability with a tree structure where a neural network is applied at each node, and each node represents the probability of a word class given the context and the leaves represent the probability of words given the context. This type of representation has the potential to reduce computation time by a factor $|V|/\log|V|$ (see Bengio, 2002).

3. Propagating gradients only from a subset of the output words. It could be the words that are conditionally most likely (based on a faster model such as a trigram, see Schwenk and Gauvain, 2002, for an application of this idea), or it could be a subset of the words for which the trigram has been found to perform poorly. If the language model is coupled to a speech recognizer, then only the scores (unnormalized probabilities) of the acoustically ambiguous words need to be computed. See also Bengio and Senécal (2003) for a new accelerated training method using importance sampling to select the words.

# 论文赏析：A Neural Probabilistic Language Model

4. Introducing a-priori knowledge. Several forms of such knowledge could be introduced, such as: semantic information (e.g., from WordNet, see Fellbaum, 1998), low-level grammatical information (e.g., using parts-of-speech), and high-level grammatical information, e.g., coupling the model to a stochastic grammar, as suggested in Bengio (2002). The effect of longer term context could be captured by introducing more structure and parameter sharing in the neural network, e.g. using time-delay or recurrent neural networks. In such a multi-layered network the computation that has been performed for small groups of consecutive words does not need to be redone when the network input window is shifted. Similarly, one could use a recurrent network to capture potentially even longer term information about the subject of the text.

5. Interpreting (and possibly using) the word feature representation learned by the neural network. A simple first step would start with $m = 2$ features, which can be more easily displayed. We believe that more meaningful representations will require large training corpora, especially for larger values of $m$.

6. Polysemous words are probably not well served by the model presented here, which assigns to each word a single point in a continuous semantic space. We are investigating extensions of this model in which each word is associated with multiple points in that space, each associated with the different senses of the word.