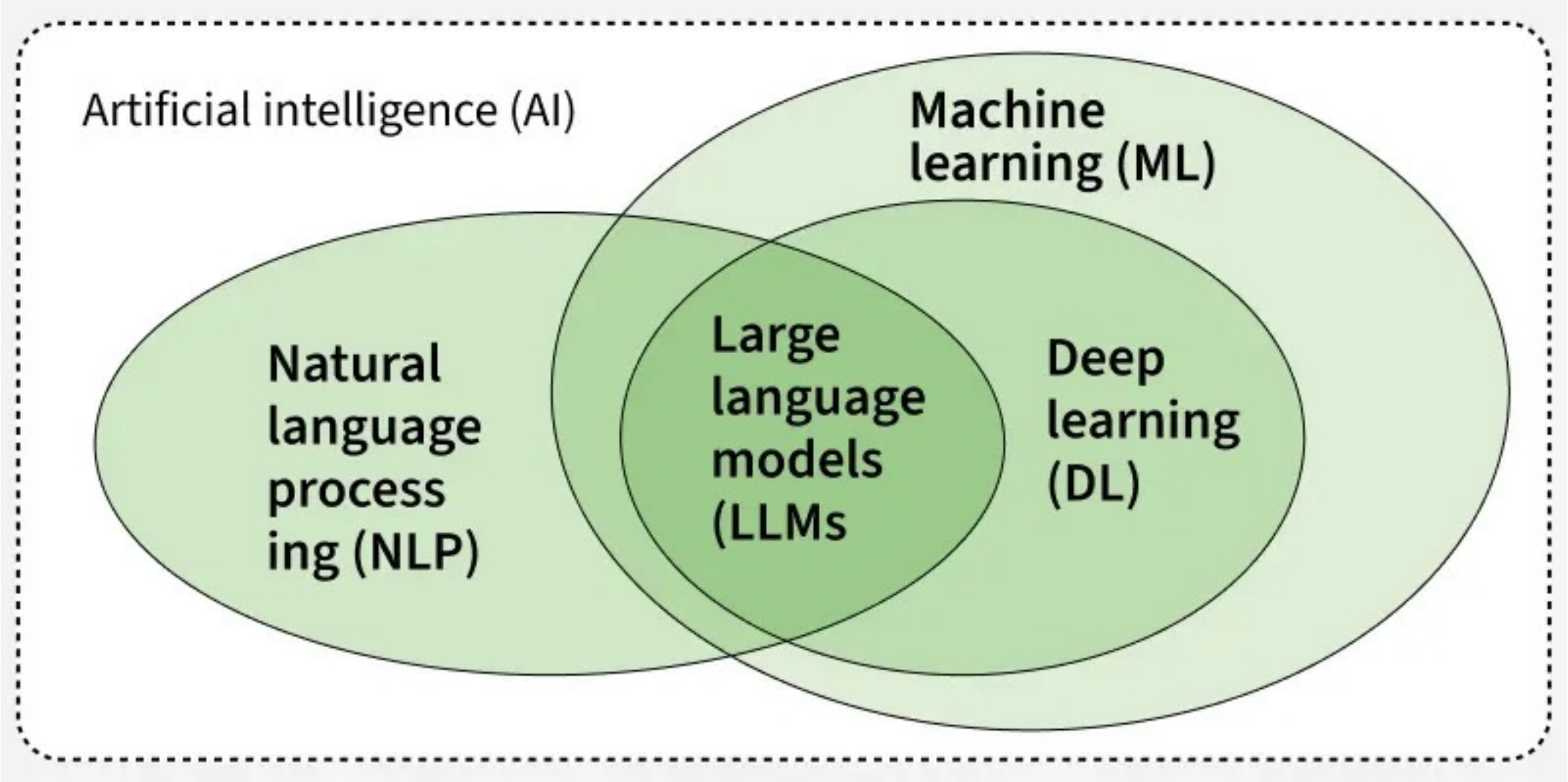
第八周 自然语言处理

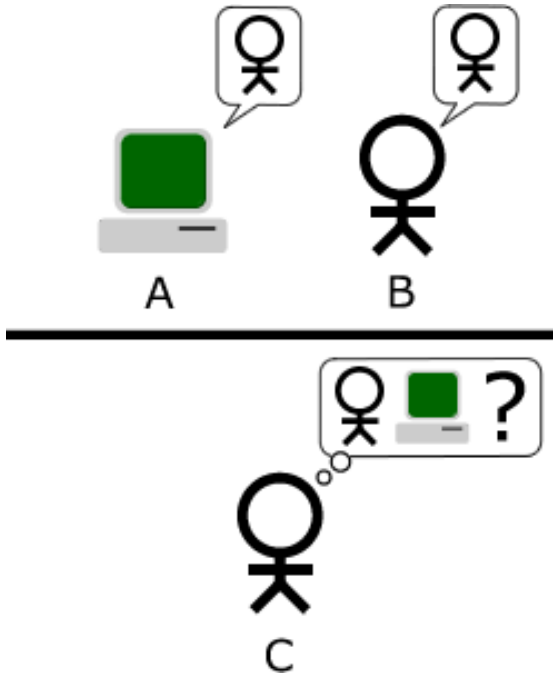
李泽榉，复旦大学 生物医学工程与技术创新学院

目录

自然语言处理是一个独立学科



图灵测试——屏幕后和你聊天的是人类还是计算机?



模仿游戏

机器能思考吗?

提问: 给我写一首歌颂秋天的诗歌吧?

回答 : 别开我玩笑了，我哪有这个能耐啊。

提问: 34,957 加上 70,764等于多少?

回答: (停顿了大概10秒) 105,621.

艾伦·麦席森·图灵

(1912 - 1954)

A. M. Turing (1950) Computing Machinery and Intelligence. Mind 49: 433-460.

自然语言处理任务

论文写作　　百科写作

报告生成

新闻写作

自动作诗

语言生成

专利写作

语言理解　　自动摘要　　自动问答　　机器翻译　　对话系统

阅读理解

文本分类

自然语言处理的应用

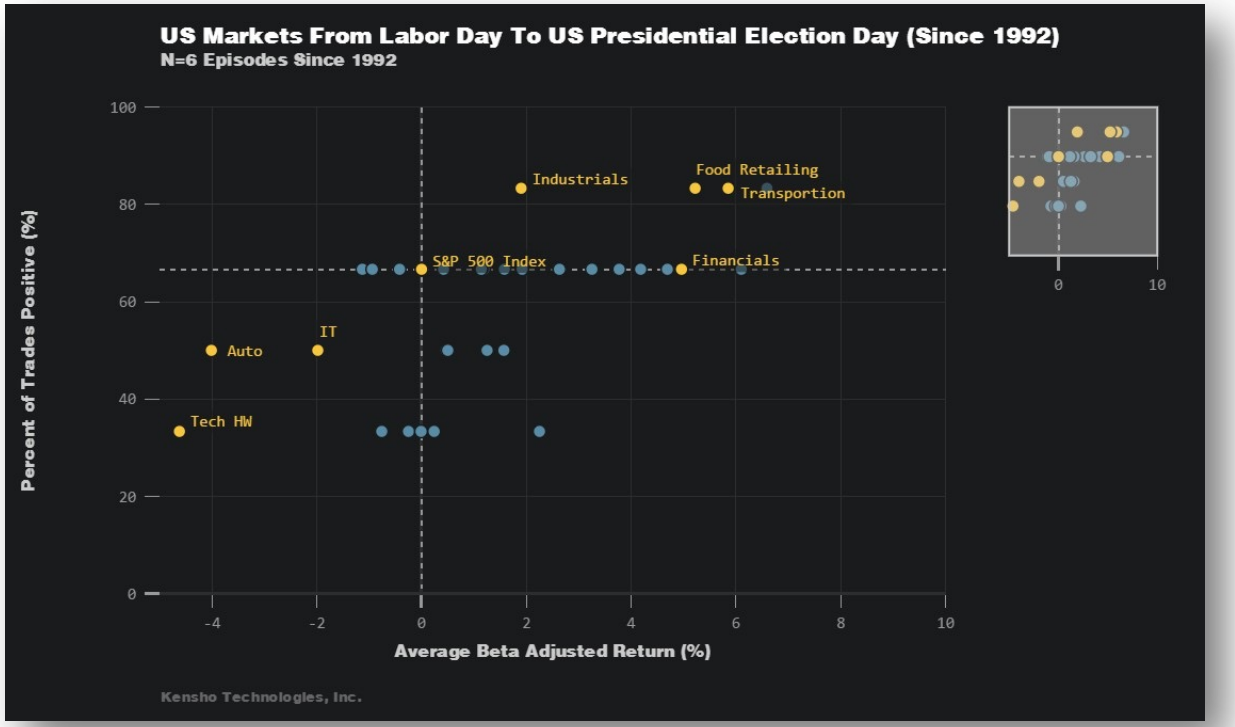
谷歌翻译每天服务10亿次

机器翻译—语言无障碍

1960　　1990 – 2000　　2015　　Future

1960　　2015　**语言翻译的岗位会消失吗?**

Kensho：比阿尔法狗还残暴的华尔街之狼



Kensho：1992年起，总统竞选阶段行业情况概览

■ Kensho如何影响传统金融

Kensho的软件Warren主要能实现两种功能：寻找事件和资产之间的相关性及对于其价格的影响，以及基于这些事件对资产未来价格走势做预测。

■ 如何用好Kensho这类智能金融产品?

——“你仍然需要问正确的问题”

Warren只能做到变量延展，但却无法替用户去“逻辑推理”事件可能的影响因素。

——“相关性不代表因果性”

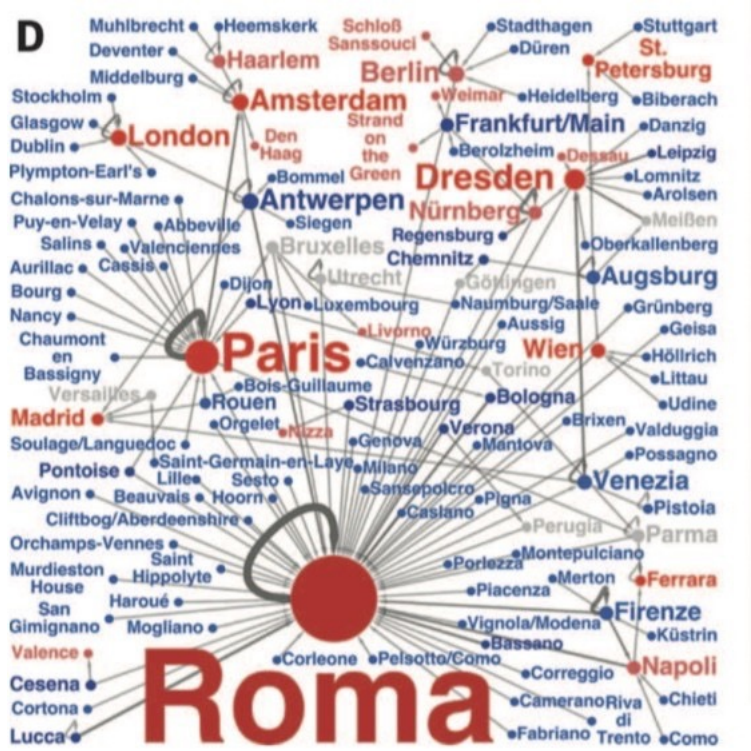
当影响资产的相关因素越来越多的时候，如何识别事件背后的相关性和因果性就变得更加困难。

自然语言处理的应用

QUANTITATIVE SOCIAL SCIENCE

A network framework of cultural history

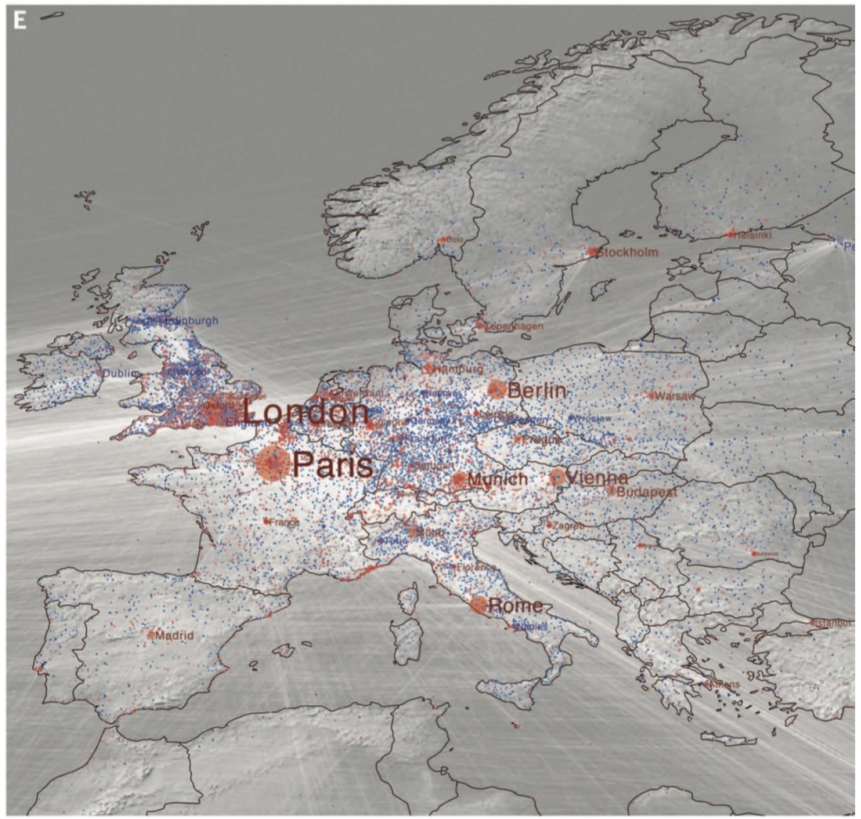
Maximilian Schich,[1,2,3*] Chaoming Song,[4] Yong-Yeol Ahn,[5] Alexander Mirsky,[2] Mauro Martino,[3] Albert-László Barabási,[3,6,7] Dirk Helbing[2]



Winckelmann Corpus, 18世纪人物

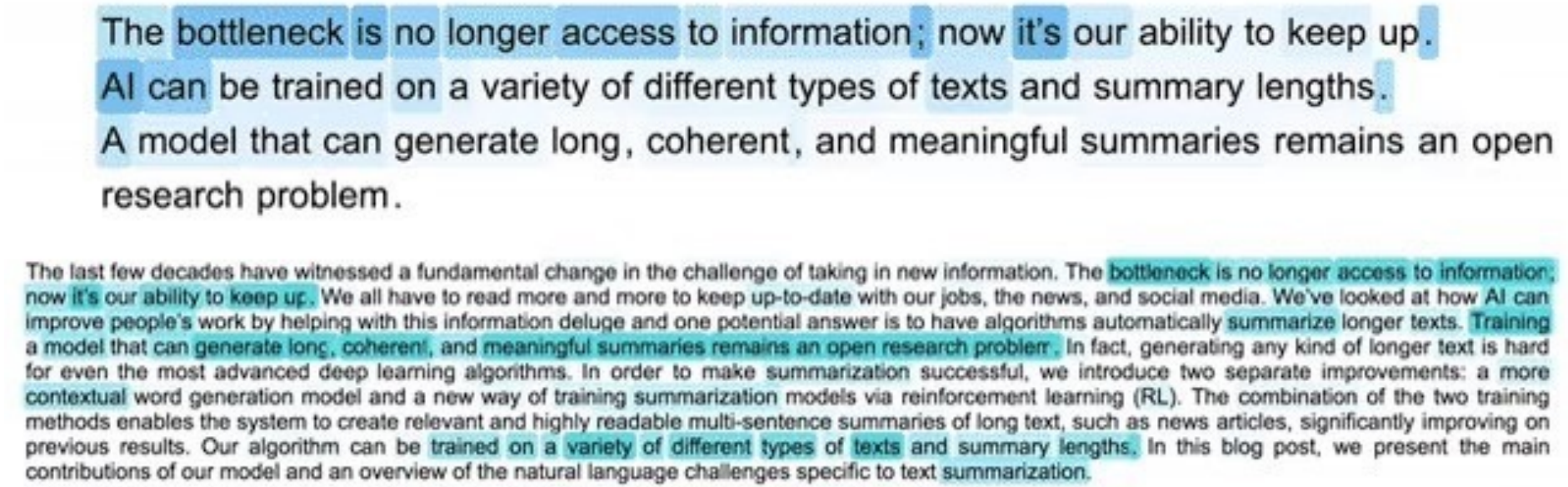
名 人

出 生 地 点 －〉死 亡 地 点



Freebase

生成式文本摘要

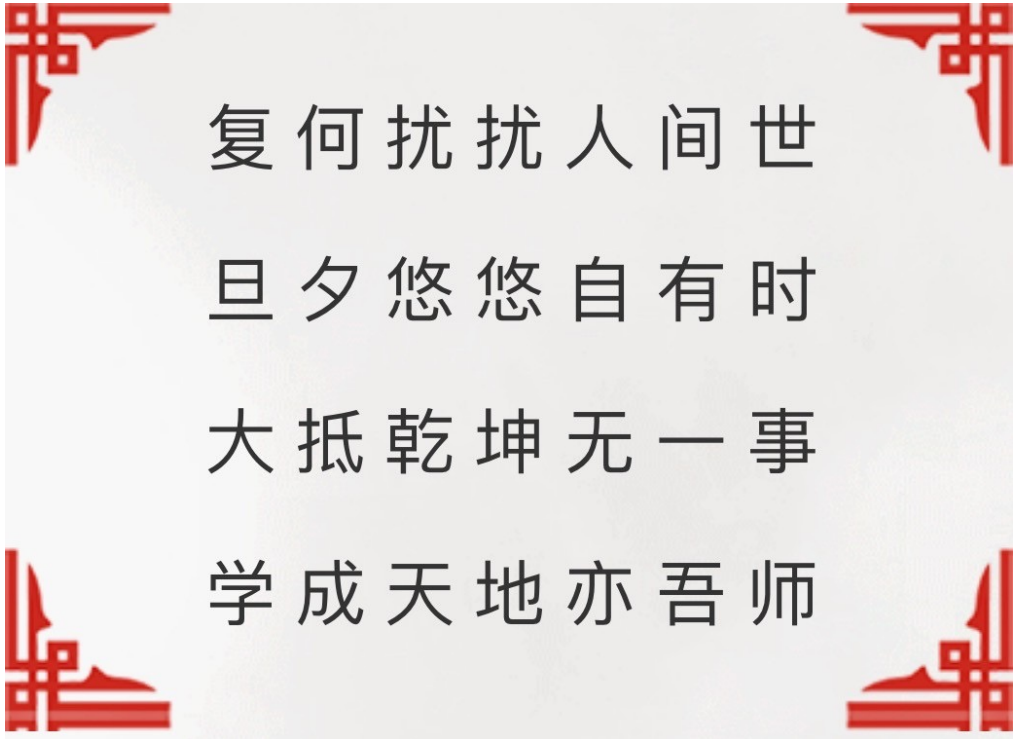


Paulus, Romain, Caiming Xiong, and Richard Socher. "A deep reinforced model for abstractive summarization." 2017

https://tryolabs.com/blog/2017/12/12/deep-learning-for-nlp-advancements-and-trends-in-2017/

自动写诗

http://jiuge.thunlp.org/



字幕生成

Show and Tell: A Neural Image Caption Generator, NIPS 2014

看图讲故事

Story #1: The brother and sister were ready for the first day of school. They were excited to go to their first day and meet new friends. They told their mom how happy they were. They said they were going to make a lot of new friends . Then they got up and got ready to get in the car .

Story #2: The brother did not want to talk to his sister. The siblings made up. They started to talk and smile. Their parents showed up. They were happy to see them

No Metrics Are Perfect: Adversarial Reward Learning or Visual Storytelling, ACL 2018

精准广告投放

诸城广潍长城哈弗 4S 店

广告

新次元 激擎座驾

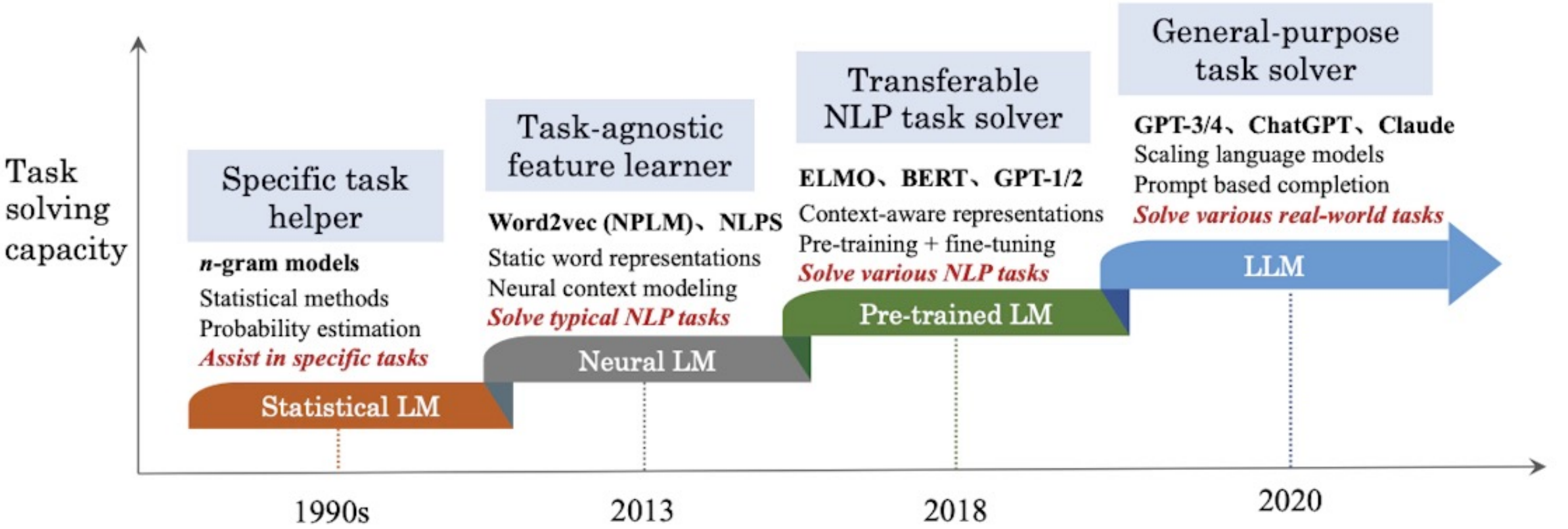
哈弗赤兔新次元座驾，10.98 万起。买哈弗到诸城广潍！

限时活动　潮派　智能网联

潍坊 · 诸城广潍长城哈弗 4s 店

大语言模型-NLP不存在了？

- 从GPT-3时代开始，纯微调单个中小型模型来处理独立NLP任务的范式，在绝大多数场景下已无法与LLM的通用能力匹敌

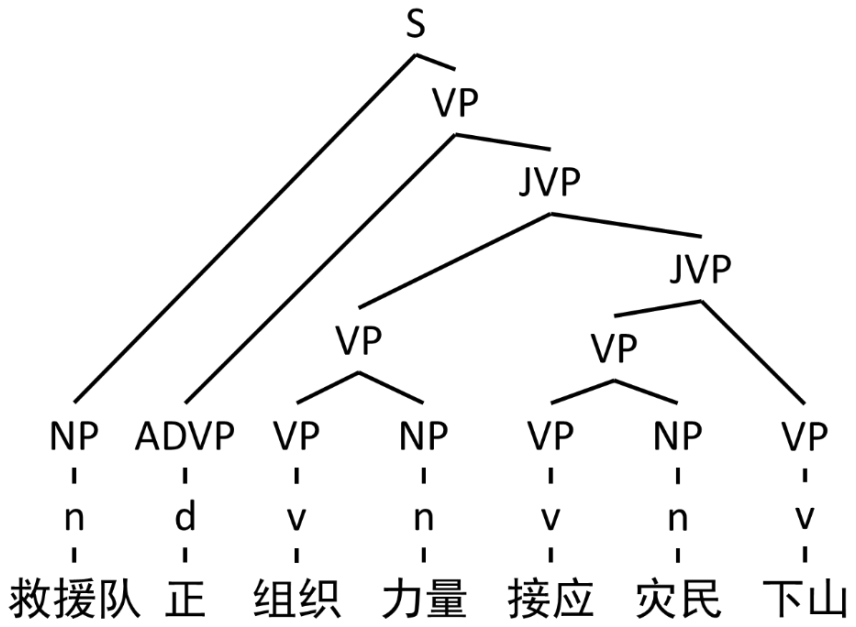


目录

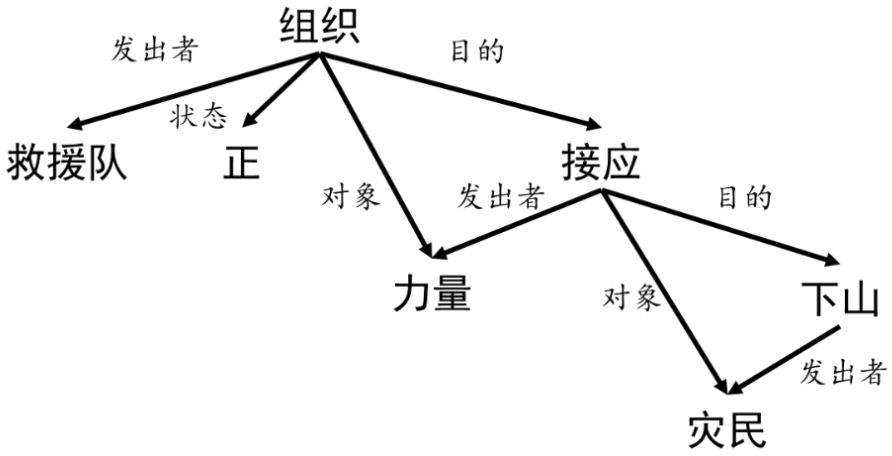
自然语言理解

输入： 救援队正组织力量接应灾民下山

输出：



句法结构



语义结构

自然语言理解的本质是结构预测

《自然语言处理》
刘知远

自然语言理解复杂程度

中文6763个常用字（GB2312），新闻文档平均句子长度20+

6763^{20} = 4.00e+76

6763^{40} = 1.61e+153

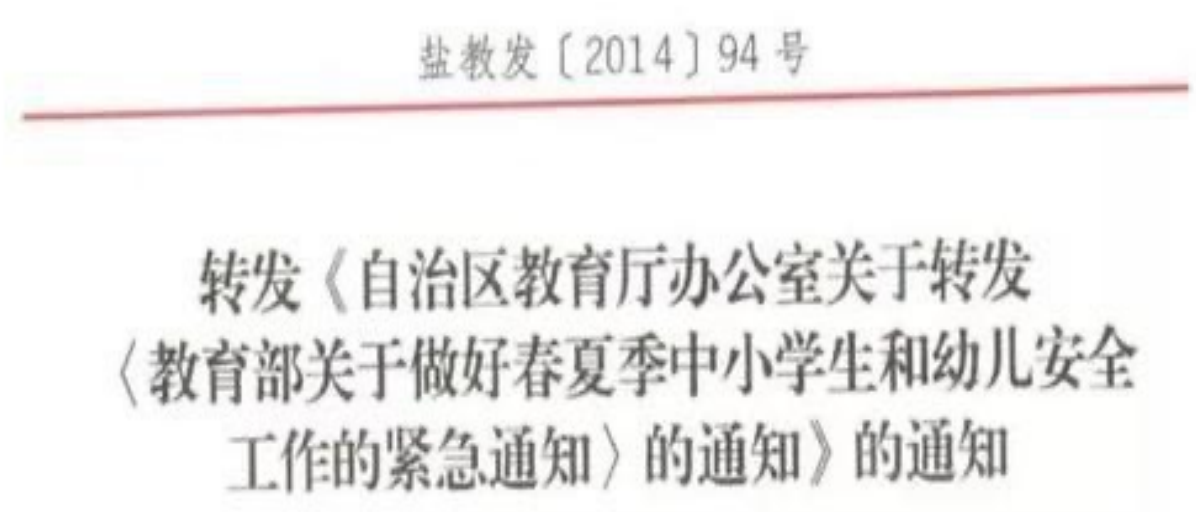
利用句法语义等知识**约束**求解

特点	例句
口语化	亲，请点赞哦！
缩略语	中石油
中英混合	我今天很Happy
新词	累觉不爱

自然语言理解复杂程度


详情
巳月
吉尼斯世界纪录因纪录了最多吉尼斯世界纪录被吉尼斯世界纪录纪录为纪录吉尼斯世界纪录最多的吉尼斯世界纪录
2019年4月22日 下午8:04

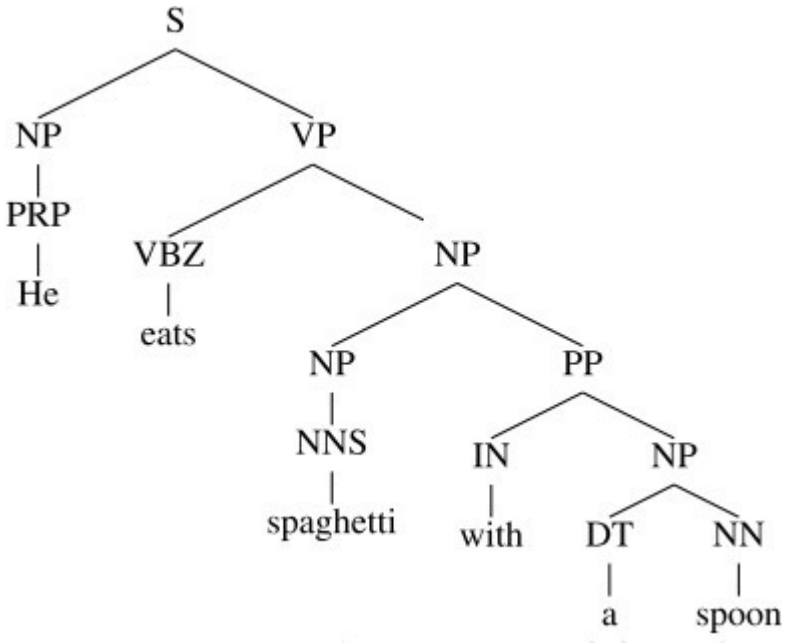
自然语言理解复杂程度

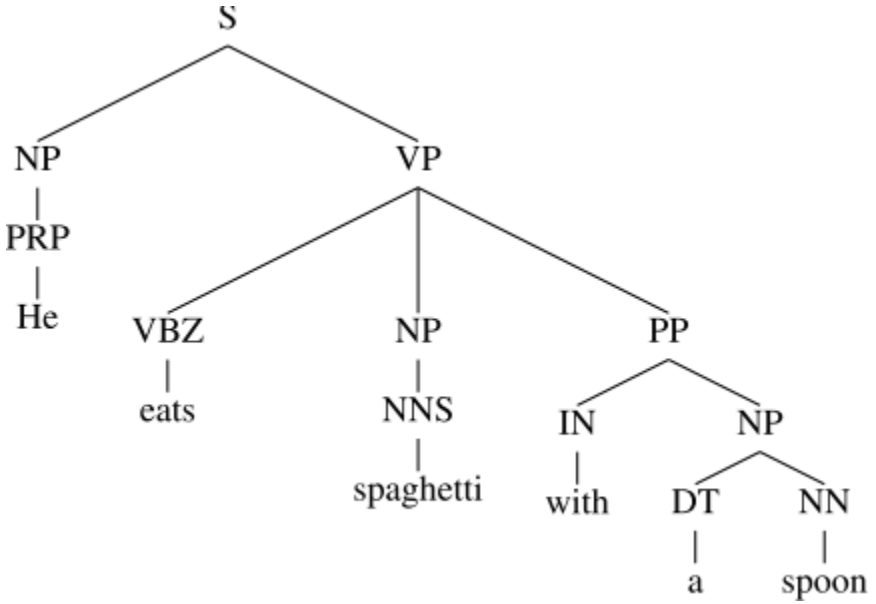


盐教发〔2014〕94号

转发《自治区教育厅办公室关于转发
〈教育部关于做好春夏季中小学生和幼儿安全
工作的紧急通知〉的通知》的通知

自然语言理解复杂程度





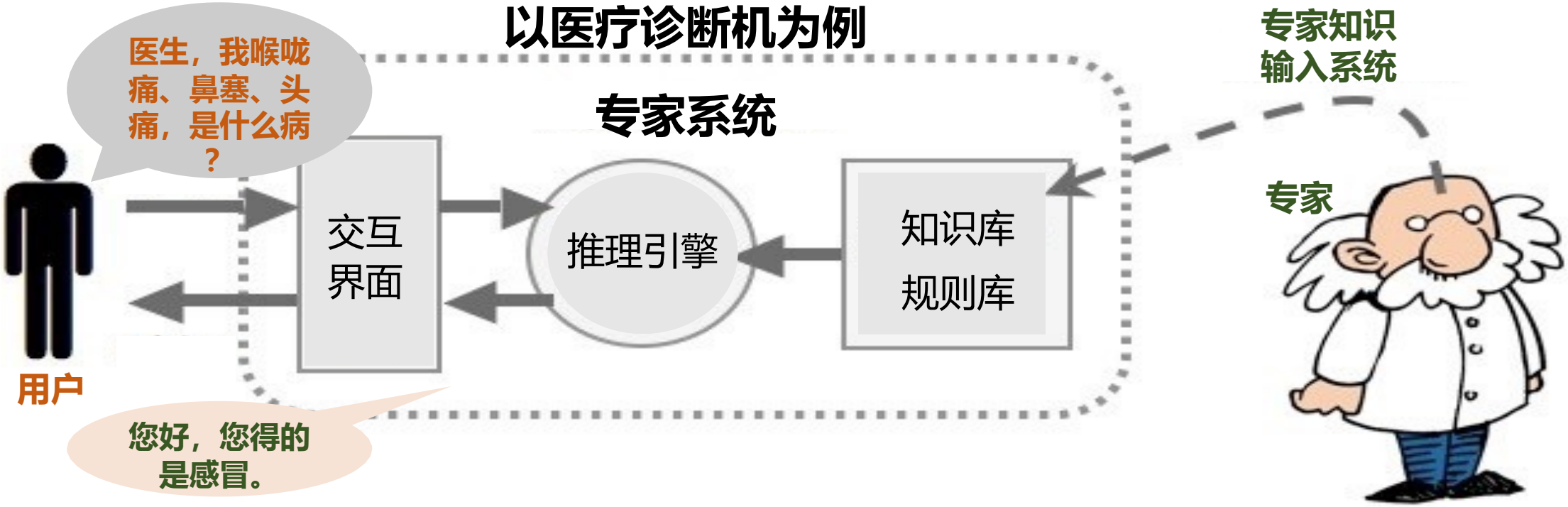
Richard Socher, John Bauer, Christopher D Manning, and Andrew Y Ng. 2013. Parsing with compositional vector grammars, ACL.

1970 — 80: 把全世界的知识都记录下来吧!

将知识基于规则表达

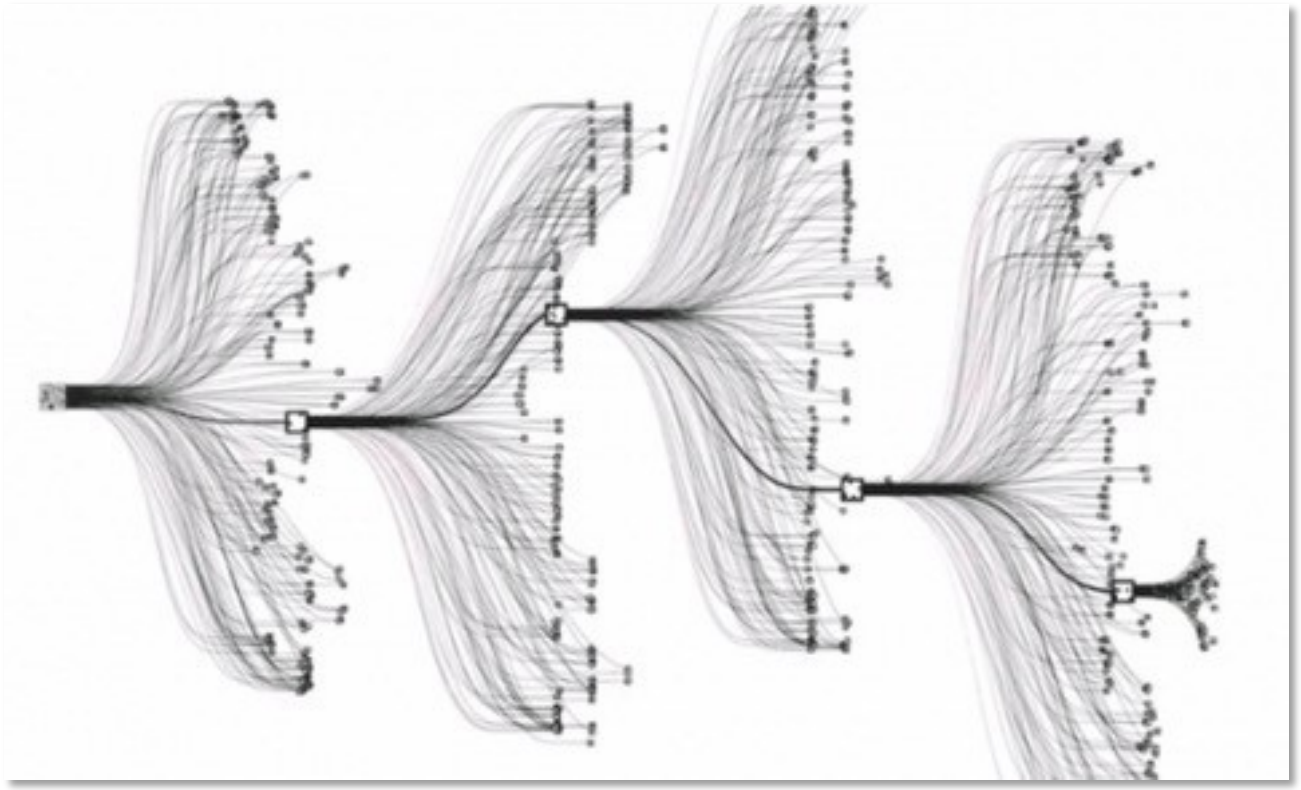
- 狗都有四条腿。
- 卡拉是条狗。

→ 所以卡拉有四条腿。



1990 –至今: 上帝也会掷骰子

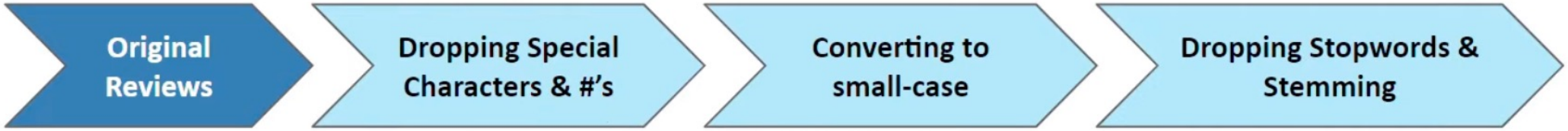
- **引入统计方法和机器学习技术，加速了语言处理的发展**
- **深度学习的兴起推动了自然语言处理的巨大飞跃，使得计算机更好地理解和生成人类语言**



目录

如何表示文本？ 第一步：数据清洗

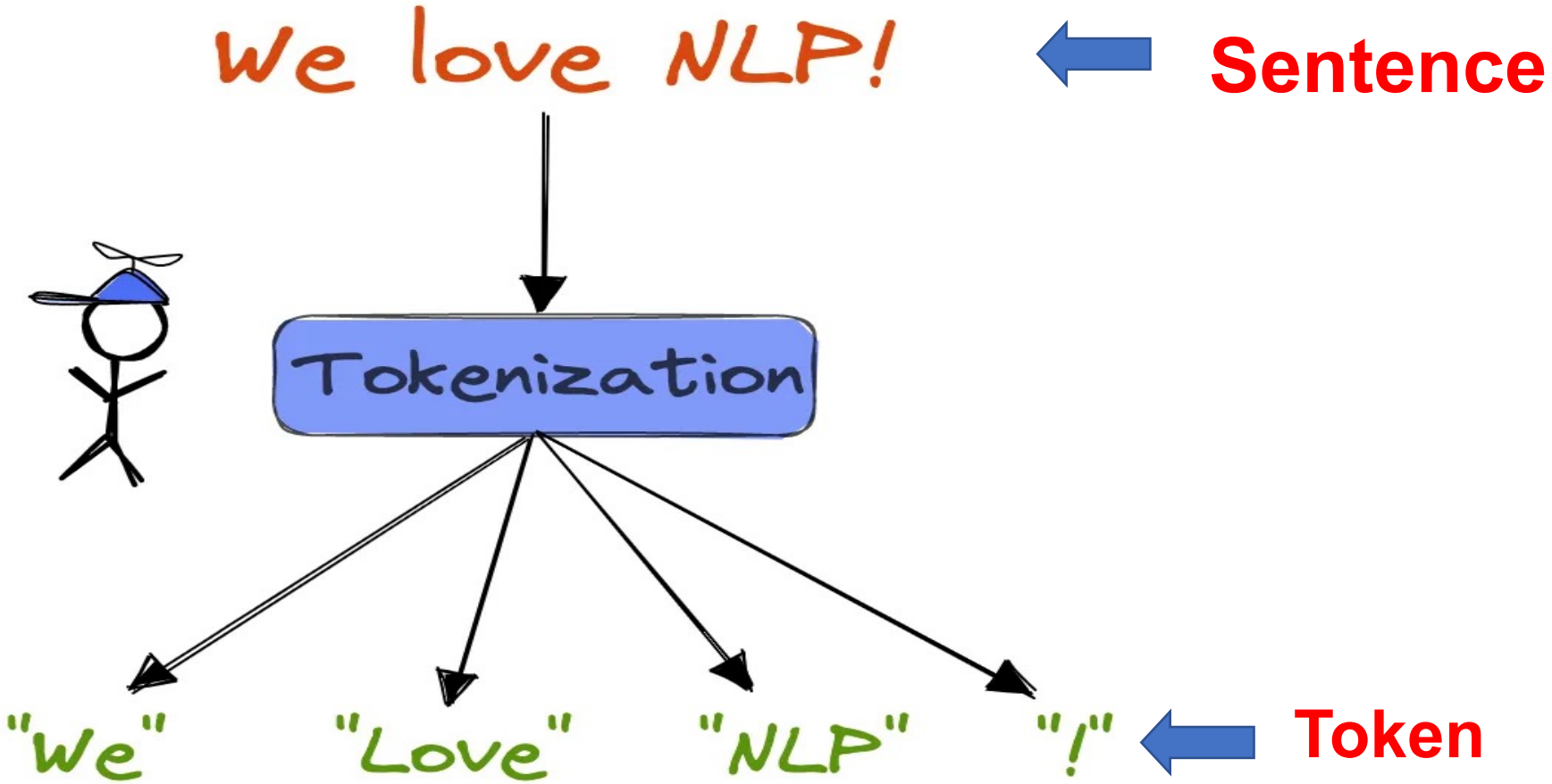
数据清洗



原始文本　　去除特殊字符　　转换成小写　　去除停止词，分词

如何表示文本？ 第二步：分词

Token、Tokenize和Tokenizer：



基于词的Tokenizer

Split on spaces

Let’s	do	tokenization!

Split on punctuation

Let	’s	do	tokenization	!
001	002	003	005	004

基于词的Tokenizer

Split on spaces

Let’s	do	tokenization!

Split on punctuation

Let	’s	do	tokenization	!

例如直接利用 Python 的 `split()` 函数按空格进行分词：

```
tokenized_text = "let's do tokenization".split()
print(tokenized_text)
```

缺点：这种策略的问题是会将所有出现过的单词都作为不同的 token，产生大量tokens。而实际上很多词是相关的，比如dog和dogs，如果给它们赋予不同的编号就无法表示出这种关联性。

基于字符的Tokenizer

基于字符的标记器将文本拆分为字符，而不是单词。这有两个主要好处：

- 词汇量要小得多。
- 词汇外（未知）标记要少得多，因为每个单词都可以由字符构建。

L	e	t	‘	s	d	o	t	o	k	e	n	i	z	a	t	i	o	n	!

爱 = “爪” + “友”

基于子词的Tokenizer

子词Tokenizer遵循原则：频繁使用的词不应该被拆分成较小的子词，但少见的词应该被分解成有意义的子词。

“annoyingly” = “annoying” + “ly”
“tokenization” = “token” + “ization”

Let’s </w>	do</w>	token	ization</w>	!</w>

大语言模型中的分词方法

- https://tiktokenizer.vercel.app/

- GPT中使用的最基础的方法是Byte Pair Encoding（https://github.com/karpathy/minbpe）

- 核心思想是用数据驱动的方式，在“字符”和“单词”之间找到一组最常用的“子词”作为词表，从而用有限的符号，高效地表示无限的文本。
  - **初始化**：将所有文本拆分为单个字符（或字节），并统计每个字符的出现频率。
  - **统计频次**：统计所有相邻符号对的出现次数。
  - **合并最频繁对**：将出现次数最多的符号对合并为一个新符号，加入词表。
  - **重复**：重复步骤 2-3，直到词表大小达到预设目标或没有可合并的对。

目录

词向量化

因为文本不能直接被模型计算，所以需要将文本转化为向量。
词是意义的基本单元，我们在上一小节讲述了如何分词。
顾名思义，词向量是用于表示单词意义的向量，
并且还可以被认为是单词的特征向量或表示。将单词映射到实向量的技术称为词嵌入（Word Embedding）。

文本转化为向量有两种表示方法：

- 转化为one-hot编码
- 转化为word embedding

独热编码（One-hot Encoding）

- 假设词表中不同词元的数目为N（即len(vocab)），词元索引的范围为0到N－1。如果词元的索引是整数i，那么我们将创建一个长度为N的全0向量，并将第i处的元素设置为1。
- 此向量是原始词元的一个独热向量。索引为0和2的独热向量如下所示：

```
F.one_hot(torch.tensor([0, 2]), len(vocab))
```

output：

```
tensor([[1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
         0, 0, 0, 0],
        [0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
         0, 0, 0, 0]])
```

单词向量表示 Word Embedding

expect =

$$\begin{pmatrix} 0.286 \\ 0.792 \\ -0.177 \\ -0.107 \\ 0.109 \\ -0.542 \\ 0.349 \\ 0.271 \\ 0.487 \end{pmatrix}$$

litoria

leptodactylidae

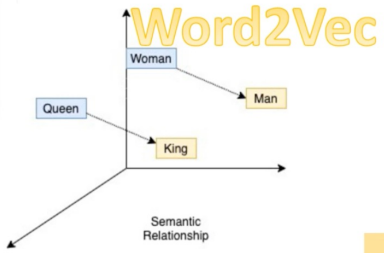
rana

eleutherodactylus

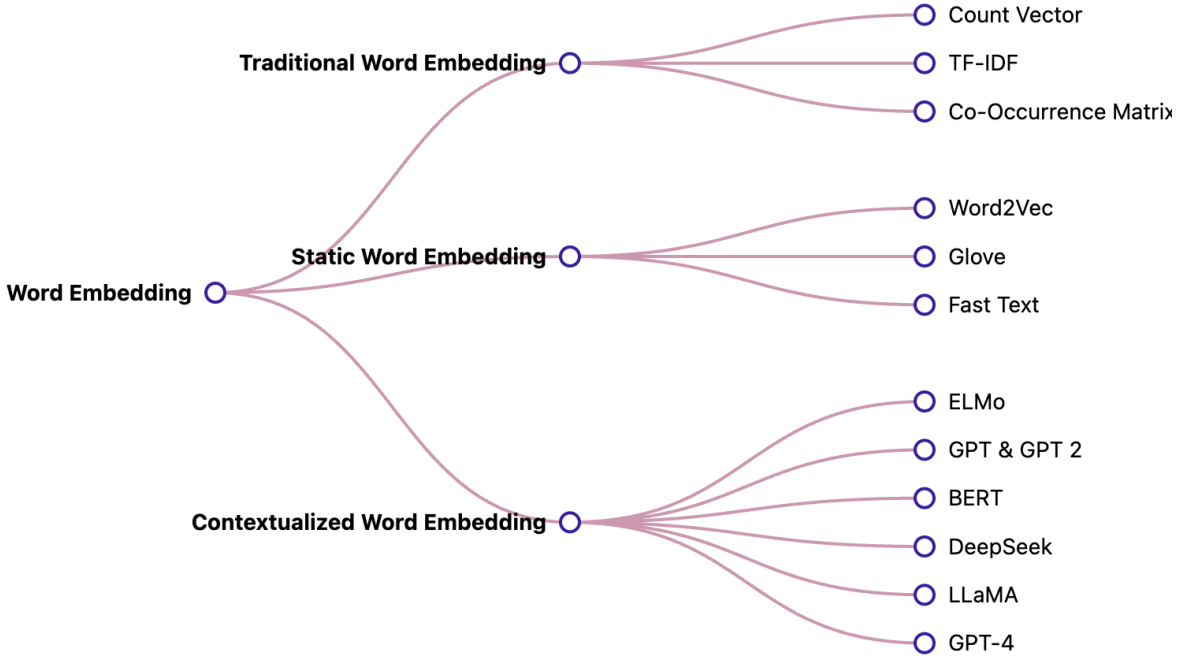
King - Man + Woman = Queen

词嵌入的几种方式

TF-IDF



GloVe



https://www.youtube.com/watch?v=H460qCxGTac&t=31s

词袋（Bag of words）

Learnerea best channel

it awesome

it great contents

I hate

Sentence	Learnerea	best	channel	it	awesome	great	contents	i	hate
Learnerea is the best	1	1	1	0	0	0	0	0	0
it is just awesome	0	0	0	1	1	0	0	0	0
it has great contents	0	0	0	1	0	1	1	0	0
I hate it	0	0	0	0	0	0	0	1	1

Token（词元）：unique words

频率-逆文档频率（TF-IDF）

- TF-IDF(term frequency–inverse document frequency)是一种用于信息检索与数据挖掘的常用加权技术，常用于挖掘文章中的关键词，而且算法简单高效，常被工业用于最开始的文本数据清洗。
- TF-IDF有两层意思，一层是"词频"（Term Frequency，缩写为TF），另一层是"逆文档频率"（Inverse Document Frequency，缩写为IDF）

第一步，计算词频：

$$\text{词频(TF)} = \frac{\text{某个词在文章中的出现次数}}{\text{文章的总词数}}$$

```python
def compute_tf(text):
    """计算词频（Term Frequency）"""
    # 对文本进行分词
    tokens = tokenize(text)
    # 计算每个词的出现次数
    token_counts = Counter(tokens)
    # 计算文档的总词数
    total_words = len(tokens)
    # 计算每个词的词频
    tf_dict = {word: count/total_words for word, count in token_counts.items()}
    return tf_dict
```

频率-逆文档频率（TF-IDF）

第二步，计算逆文档频率：

$$\text{逆文档频率(IDF)} = \log\left(\frac{\text{语料库的文档总数}}{\text{包含该词的文档数} + 1}\right)$$

```python
def compute_idf(documents):
    """计算逆文档频率（Inverse Document Frequency）"""
    # 统计每个词出现在多少个文档中
    word_doc_count = {}
    # 获取所有文档中的唯一词
    unique_words = set()

    for doc in documents:
        words = set(tokenize(doc))  # 使用集合去重
        unique_words.update(words)
        for word in words:
            word_doc_count[word] = word_doc_count.get(word, 0) + 1

    # 计算IDF
    total_docs = len(documents)
    idf_dict = {word: math.log(total_docs / word_doc_count[word])
                for word in word_doc_count}
    return idf_dict
```

频率-逆文档频率（TF-IDF）

第三步，计算TF-IDF：

$$\mathrm{TF-IDF} = \text{词频(TF)} \times \text{逆文档频率（IDF）}$$

- 当有TF(词频)和IDF(逆文档频率)后，将这两个词相乘，就能得到一个词的TF-IDF的值。

```python
def compute_tfidf(documents):
    """计算TF-IDF"""
    # 计算IDF
    idf_dict = compute_idf(documents)

    # 计算每个文档的TF-IDF
    tfidf_documents = []
    for doc in documents:
        tf_dict = compute_tf(doc)
        tfidf_dict = {word: tf * idf_dict[word]
                      for word, tf in tf_dict.items()}
        tfidf_documents.append(tfidf_dict)

    return tfidf_documents
```

频率-逆文档频率（TF-IDF）

示例：

```
documents = [
    "机器学习 是 人工智能 的 一个 分支",
    "深度学习 是 机器学习 的 一个 分支",
    "人工智能 是 计算机科学 的 一个 领域"
]
```

output：

```
文档 1 的TF-IDF值：
  机器学习: 0.0676
  人工智能: 0.0676
  分支: 0.0676
  是: 0.0000
  的: 0.0000
  一个: 0.0000
```

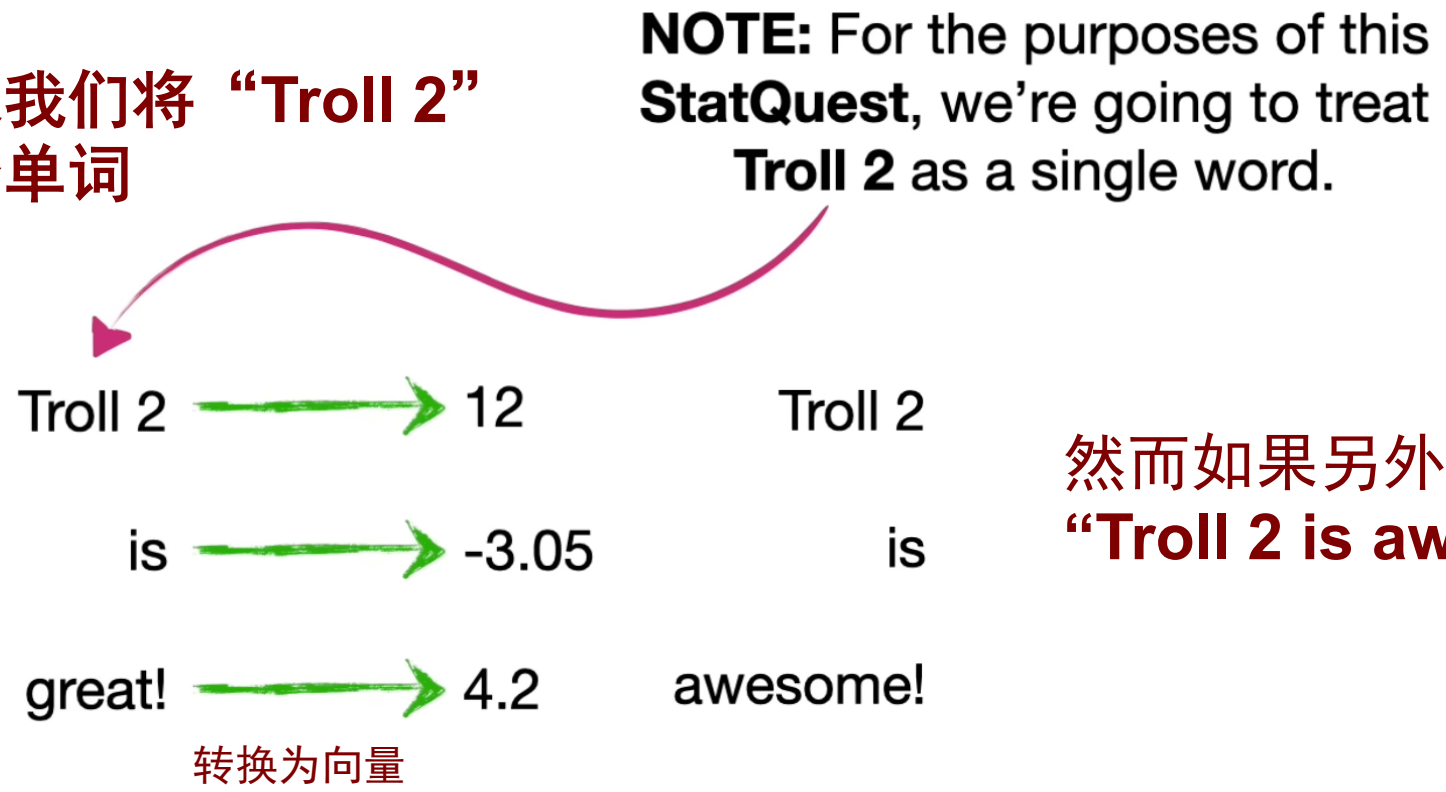
```
文档 2 的TF-IDF值：
  深度学习: 0.1831
  机器学习: 0.0676
  分支: 0.0676
  是: 0.0000
  的: 0.0000
  一个: 0.0000
```

```
文档 3 的TF-IDF值：
  计算机科学: 0.1831
  领域: 0.1831
  人工智能: 0.0676
  是: 0.0000
  的: 0.0000
  一个: 0.0000
```

- 某个词在文章中的TF-IDF越大，那么一般而言这个词在这篇文章的重要性会越高，所以通过计算文章中各个词的TF-IDF，由大到小排序，排在最前面的几个词，就是该文章的关键词。

Word2Vec

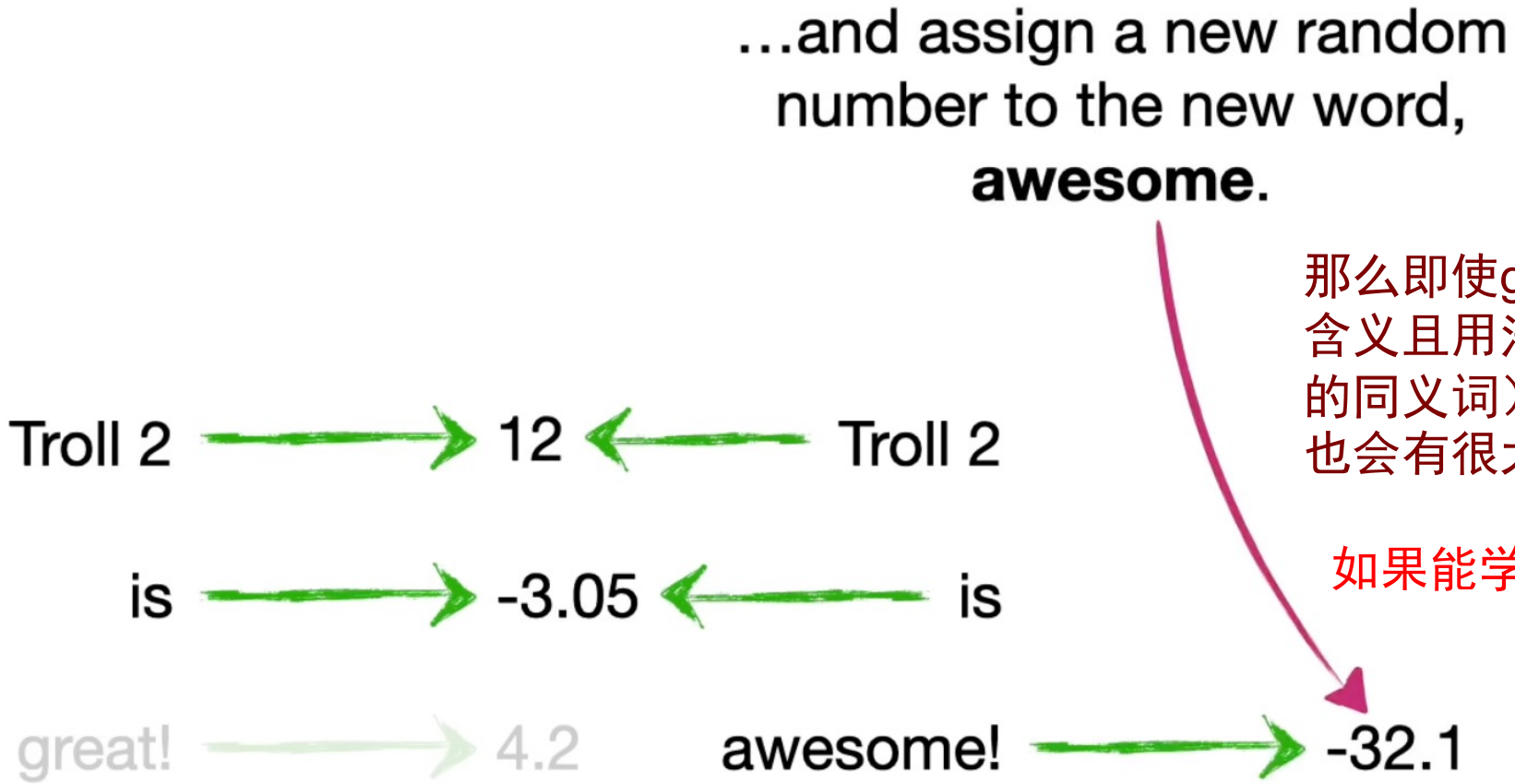
在本节课我们将“Troll 2”视为一个单词



然而如果另外一个人说“Troll 2 is awesome！”

Mikolov, Tomas, et al. "Efficient estimation of word representations in vector space." *arXiv preprint arXiv:1301.3781* (2013).

Word2Vec



那么即使great和awesome是相同的含义且用法也很相似（也就是great的同义词），他们获得的随机数编码也会有很大差异。

如果能学习到单词间的**相似性**呢？

将原有的词直接分配数值，而新词“**awesome**”分配一个新的随机数。

Word2Vec

great在此是正面的单词

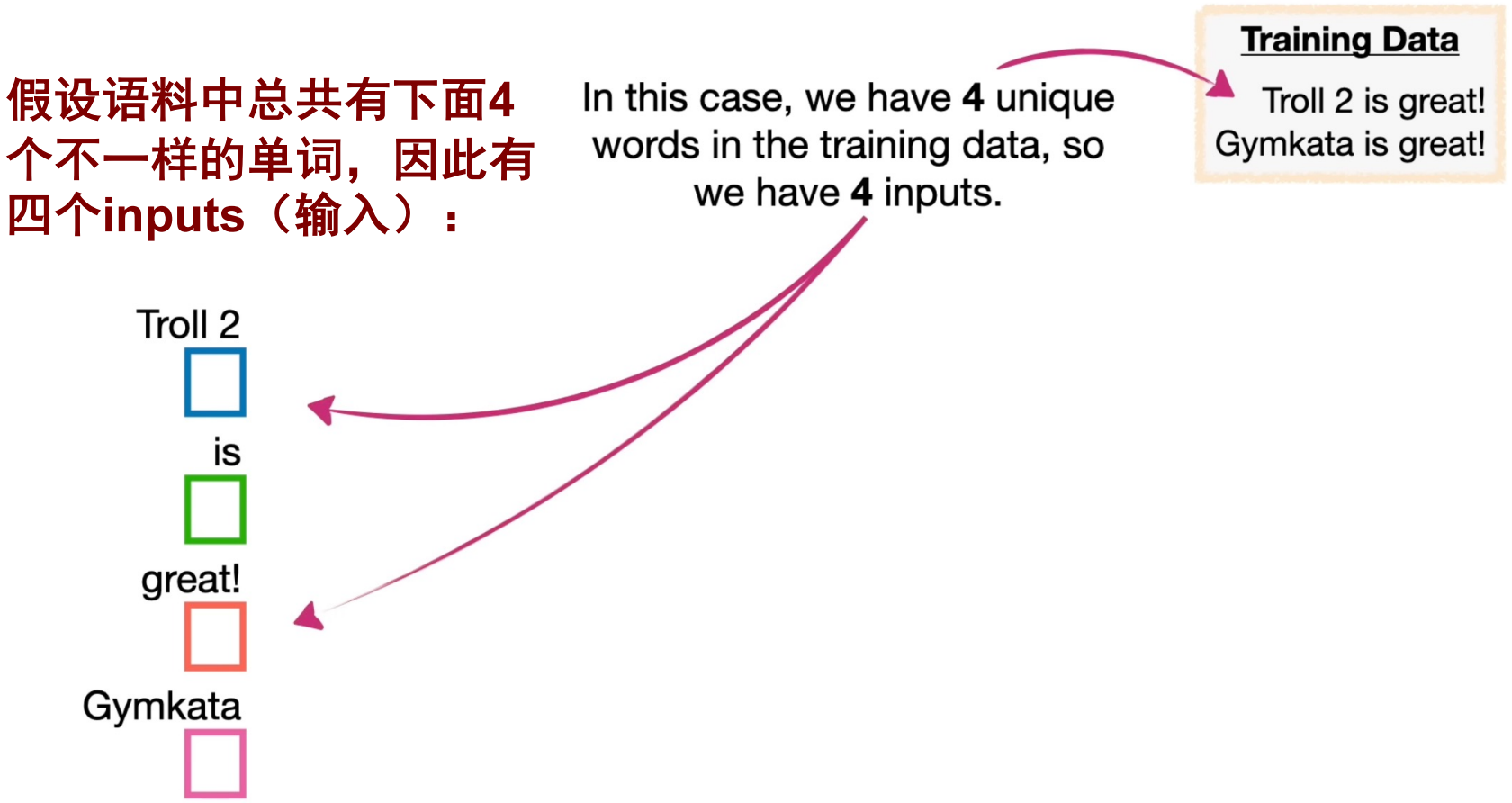
同时它也可以是负面的

FDU is great!

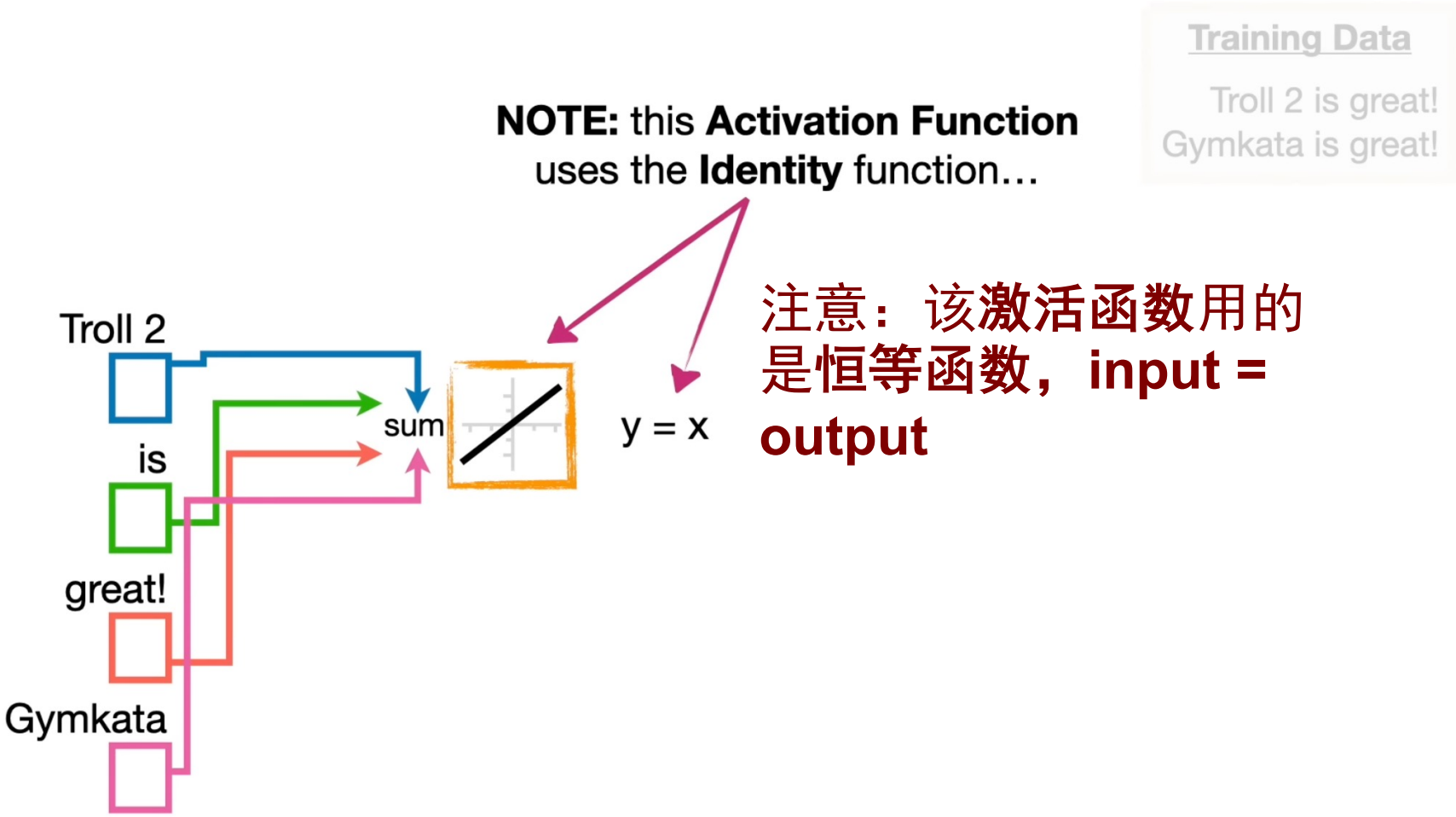
My cellphone's broken, **great**.

并且由于相同的单词可以在不同的语境中使用，或者变成复数形式，或以其他方式使用。因此为每个单词分配多个数字可能会更好，这样**神经网络**可以更容易地适应不同的语境。

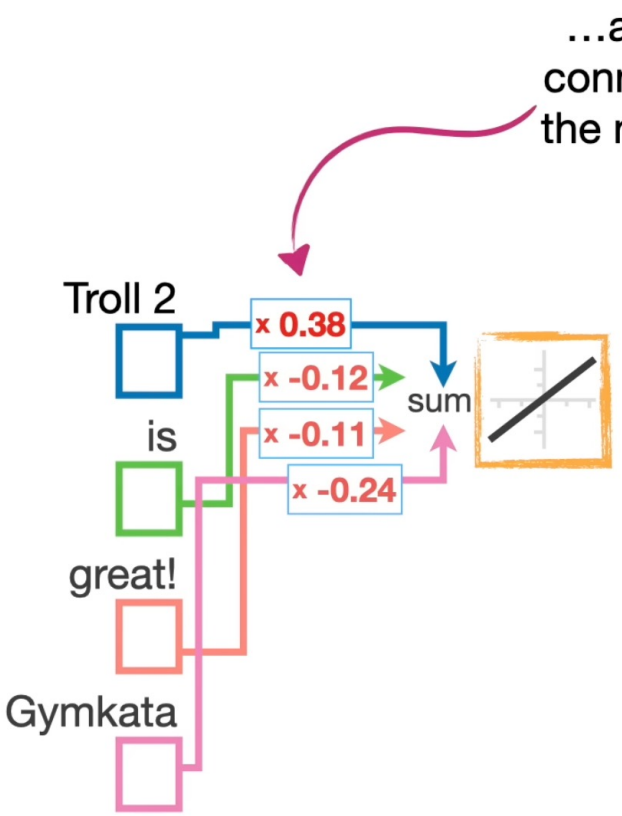
Word2Vec


假设语料中总共有下面4个不一样的单词，因此有四个inputs（输入）：
In this case, we have 4 unique words in the training data, so we have 4 inputs.
Training Data
Troll 2 is great!
Gymkata is great!
Troll 2
is
great!
Gymkata

Word2Vec


Training Data
Troll 2 is great!
Gymkata is great!
NOTE: this Activation Function
uses the Identity function…
Troll 2
is
great!
Gymkata
sum
y = x
注意：该激活函数用的是恒等函数，input = output

Word2Vec



Training Data

Troll 2 is great!
Gymkata is great!

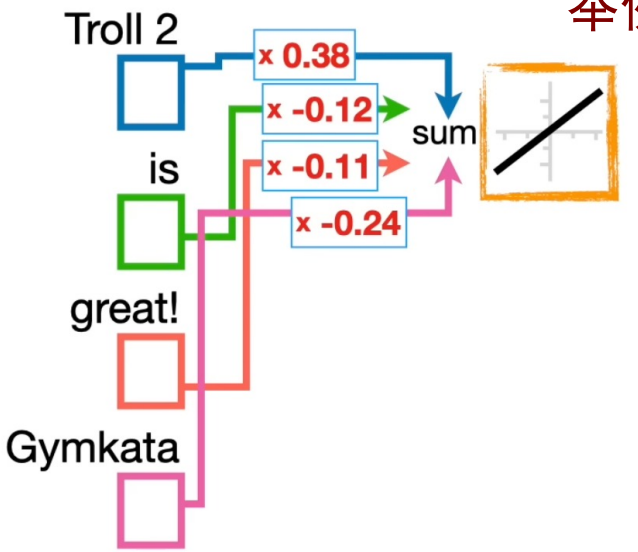
而这些连接上标注的**权重**最终会成为我们与每个单词关联的数字

Word2Vec

Training Data

Troll 2 is great!
Gymkata is great!

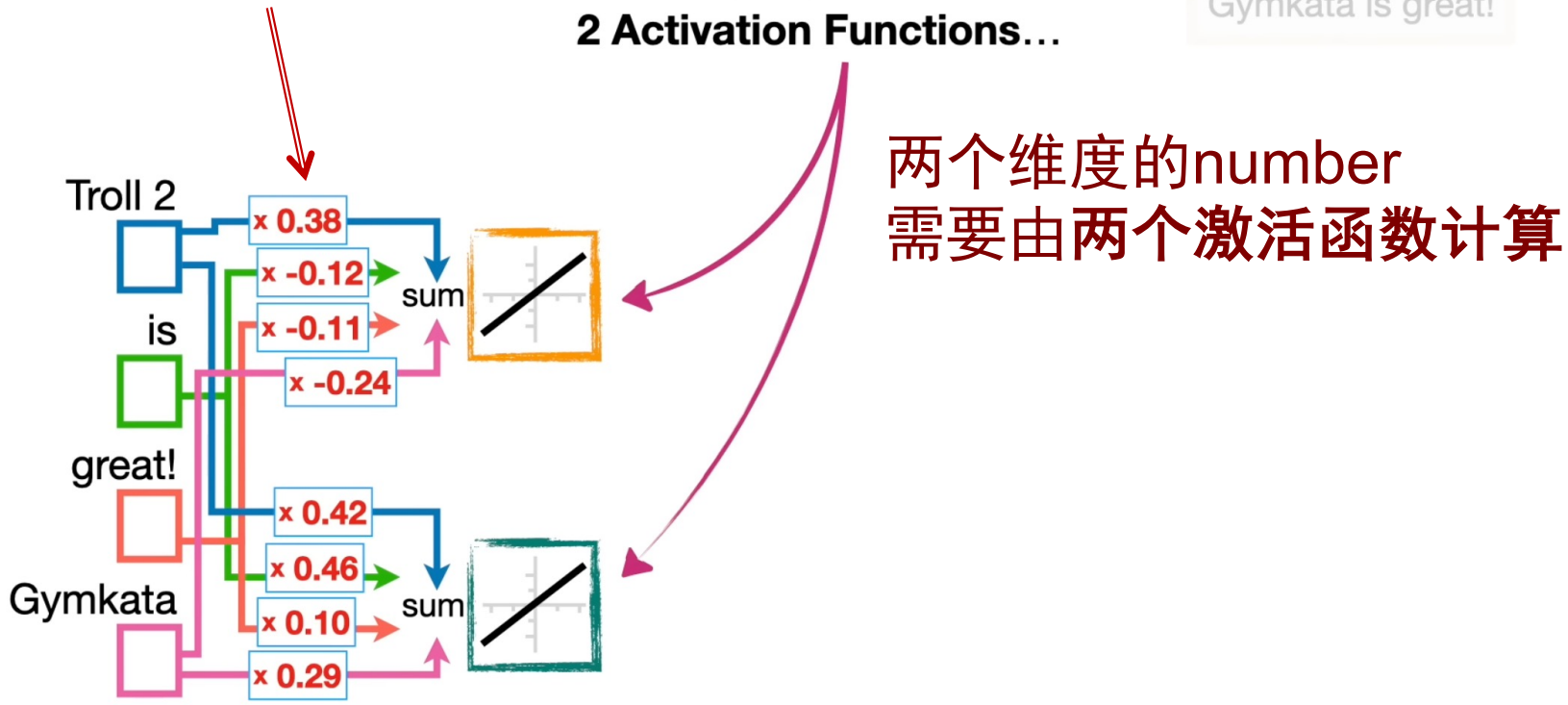
Now, in this example, we want to associate **2** numbers with each word…



举例，现在我们要使每个word关联到**2**个number

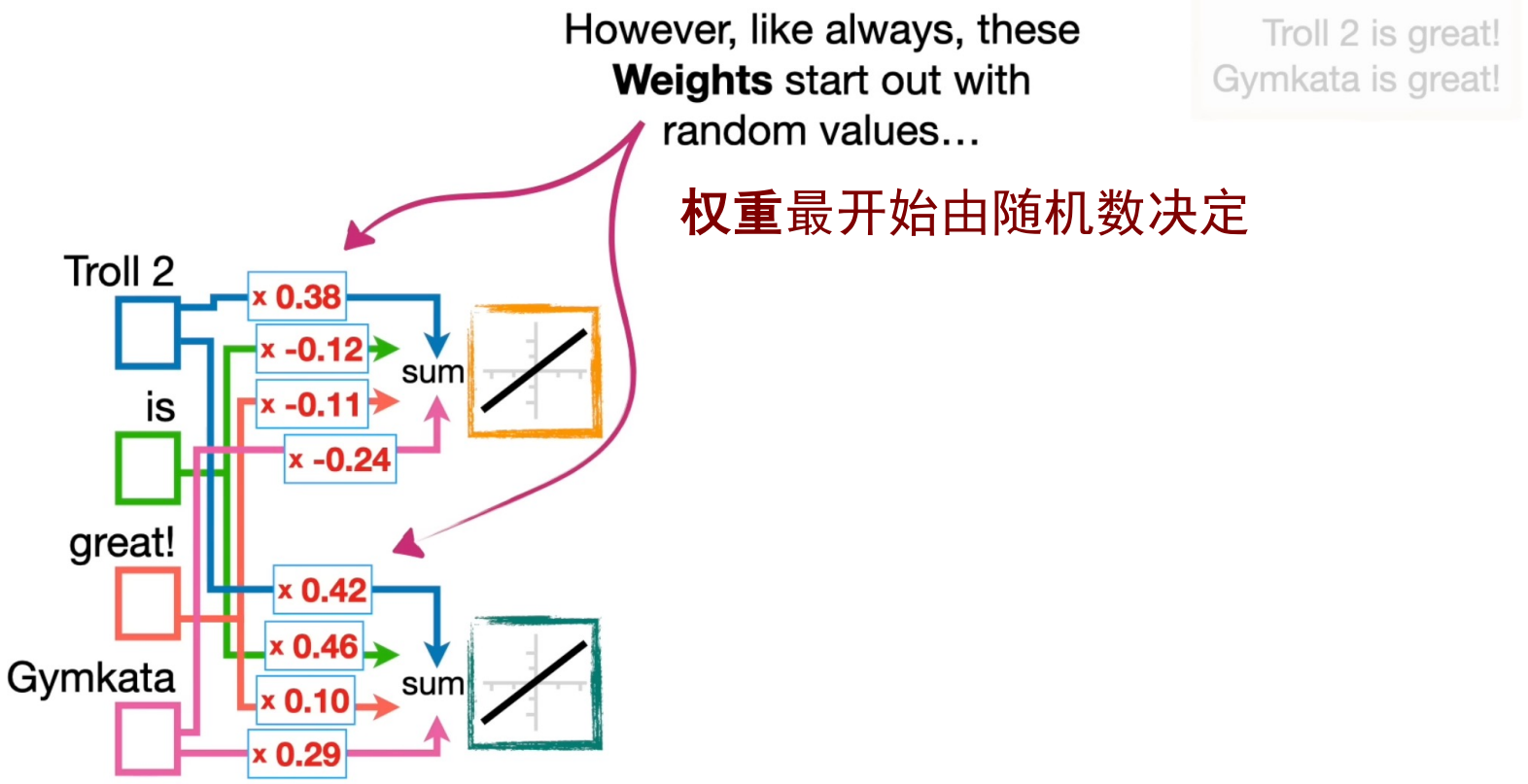
Word2Vec

权重最开始由随机数决定

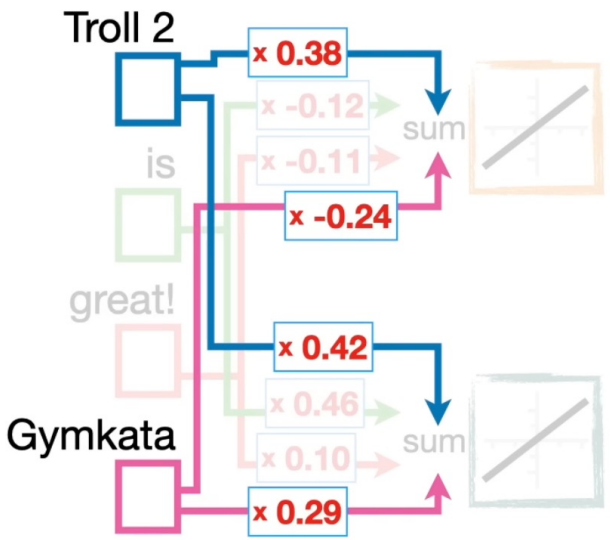


两个维度的number
需要由**两个激活函数计算**

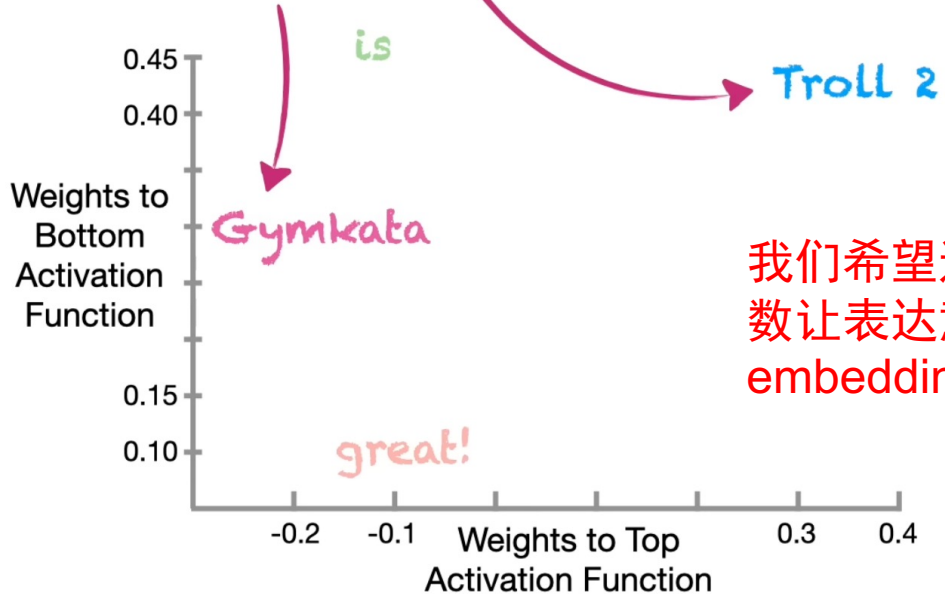
Word2Vec


Training Data
Troll 2 is great!
Gymkata is great!
However, like always, these **Weights** start out with random values…
权重最开始由随机数决定
Troll 2
is
great!
Gymkata
x 0.38
x -0.12
x -0.11
x -0.24
sum
x 0.42
x 0.46
x 0.10
x 0.29
sum

Word2Vec



Now, with this graph, we see that the words **Troll 2** and **Gymkata** are currently no more similar to each other as they are to any of the other words.



现在，通过这个图表，我们可以看到“**Troll 2**”和“**Gymkata**”这两个词彼此之间的相似度不高

我们希望通过更新weight参数让表达意思相似的单词的embedding vector更接近

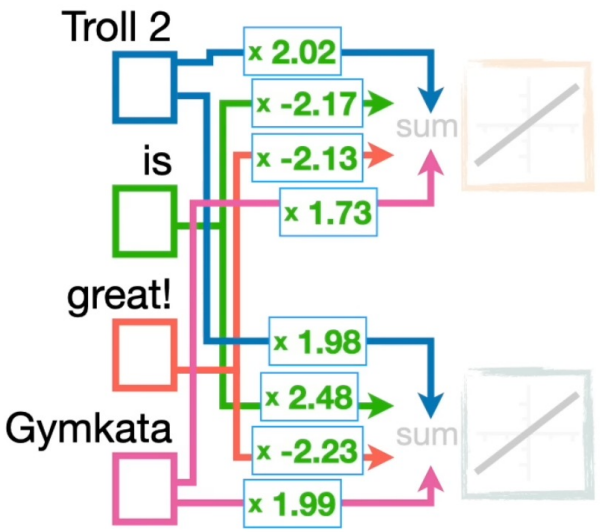
Word2Vec

相似的词拥有相似embedding（向量表示），**神经网络**在处理它们会变得更加容易，可能更容易学习和泛化。

Lastly, having similar words with similar embeddings means training a **Neural Network** to process language is easier…

Training Data

Troll 2 is great!
Gymkata is great!



那么如何使word embedding包含更多上下文信息？

Word2Vec

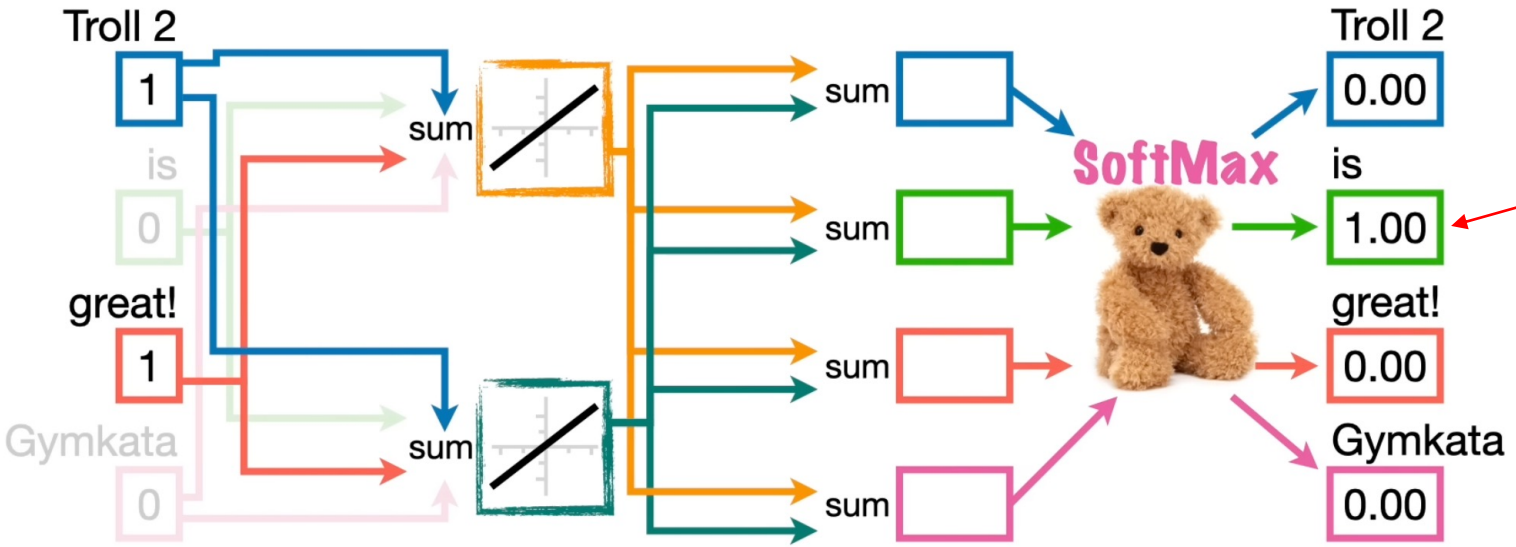
连续词袋模型（**Continuous Bag of Words**），通过使用周围的词来预测中间出现的词，从而增加上下文信息。

第一个经典方法：CBOW

The first method, called **Continuous Bag of Words**, increases the context by using the surrounding words to predict what occurs in the middle.

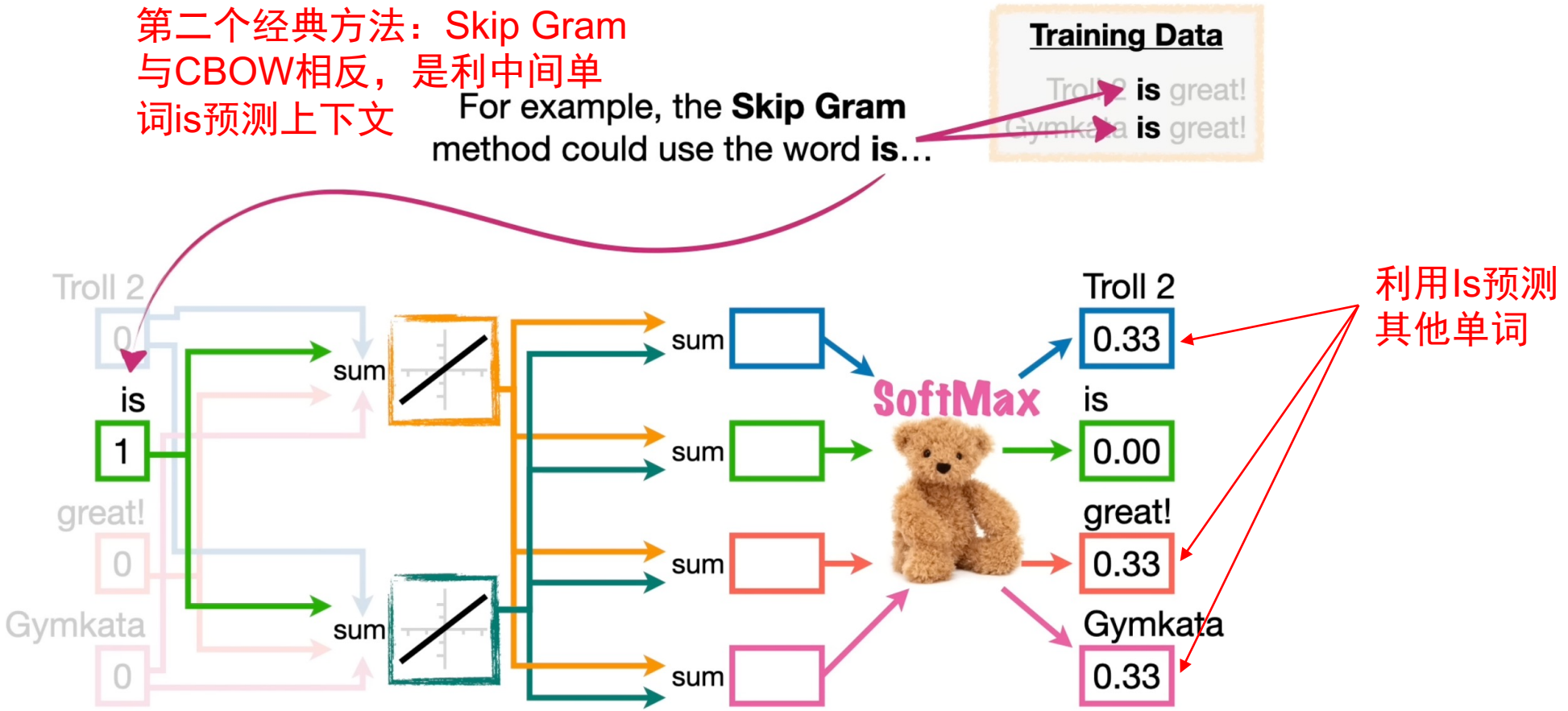
Training Data

Troll 2 is great!
Gymkata is great!

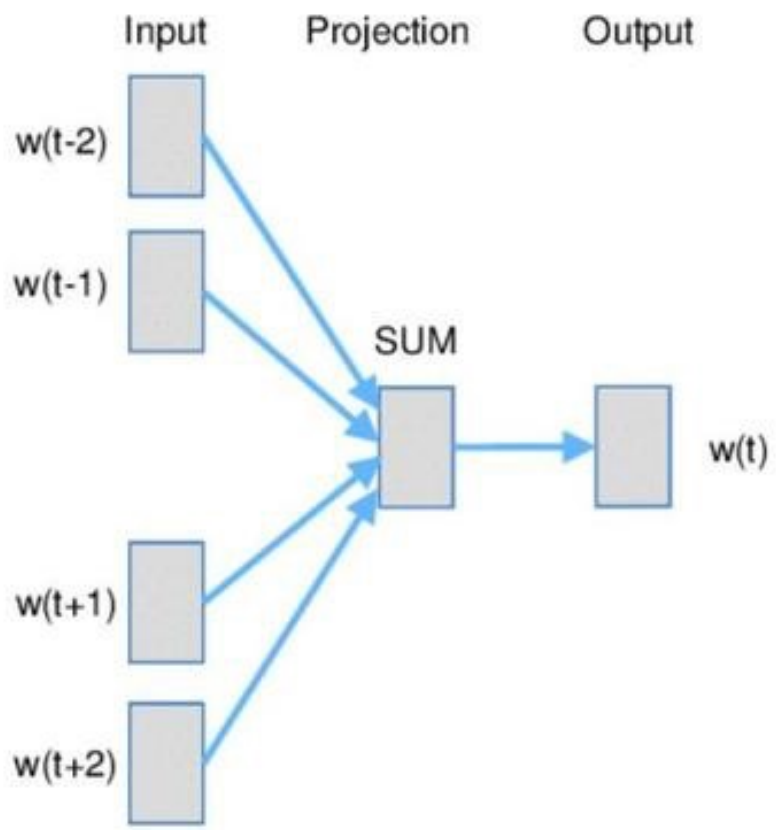


利用周围的单词（即Troll 2和great）预测中间的某个单词（即is）

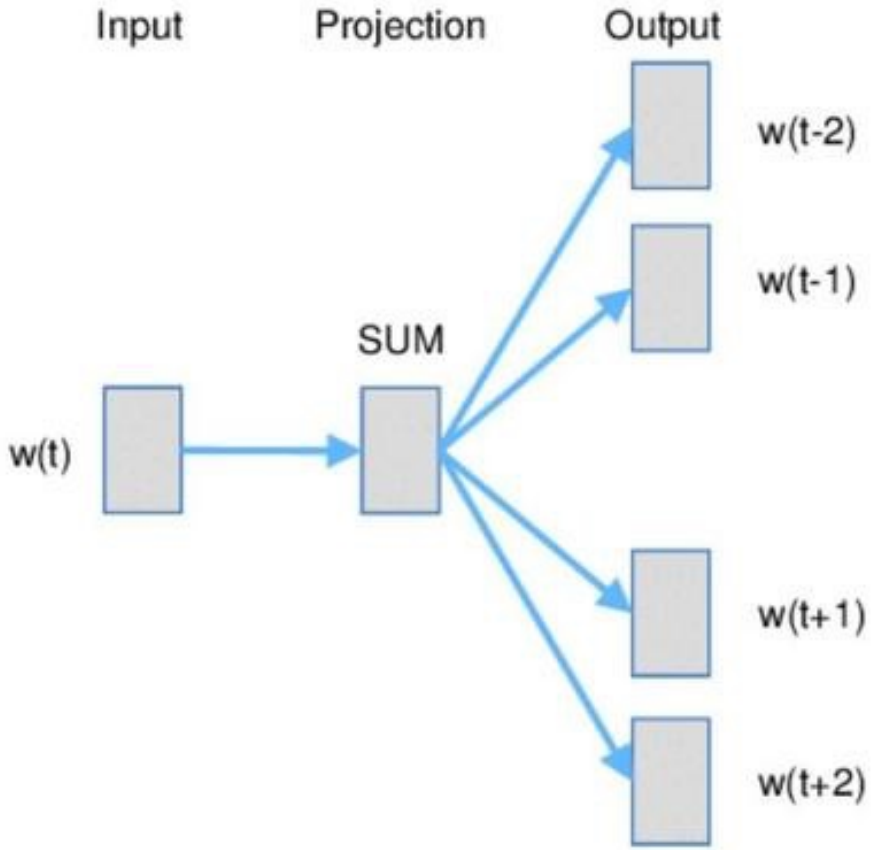
Word2Vec


第二个经典方法：Skip Gram
与CBOW相反，是利中间单
词is预测上下文
For example, the Skip Gram method could use the word is…
Training Data
is great!
is great!
Troll 2
0
is
1
great!
0
Gymkata
0
sum
sum
sum
sum
sum
SoftMax
Troll 2
0.33
is
0.00
great!
0.33
Gymkata
0.33
利用Is预测
其他单词

Word2Vec



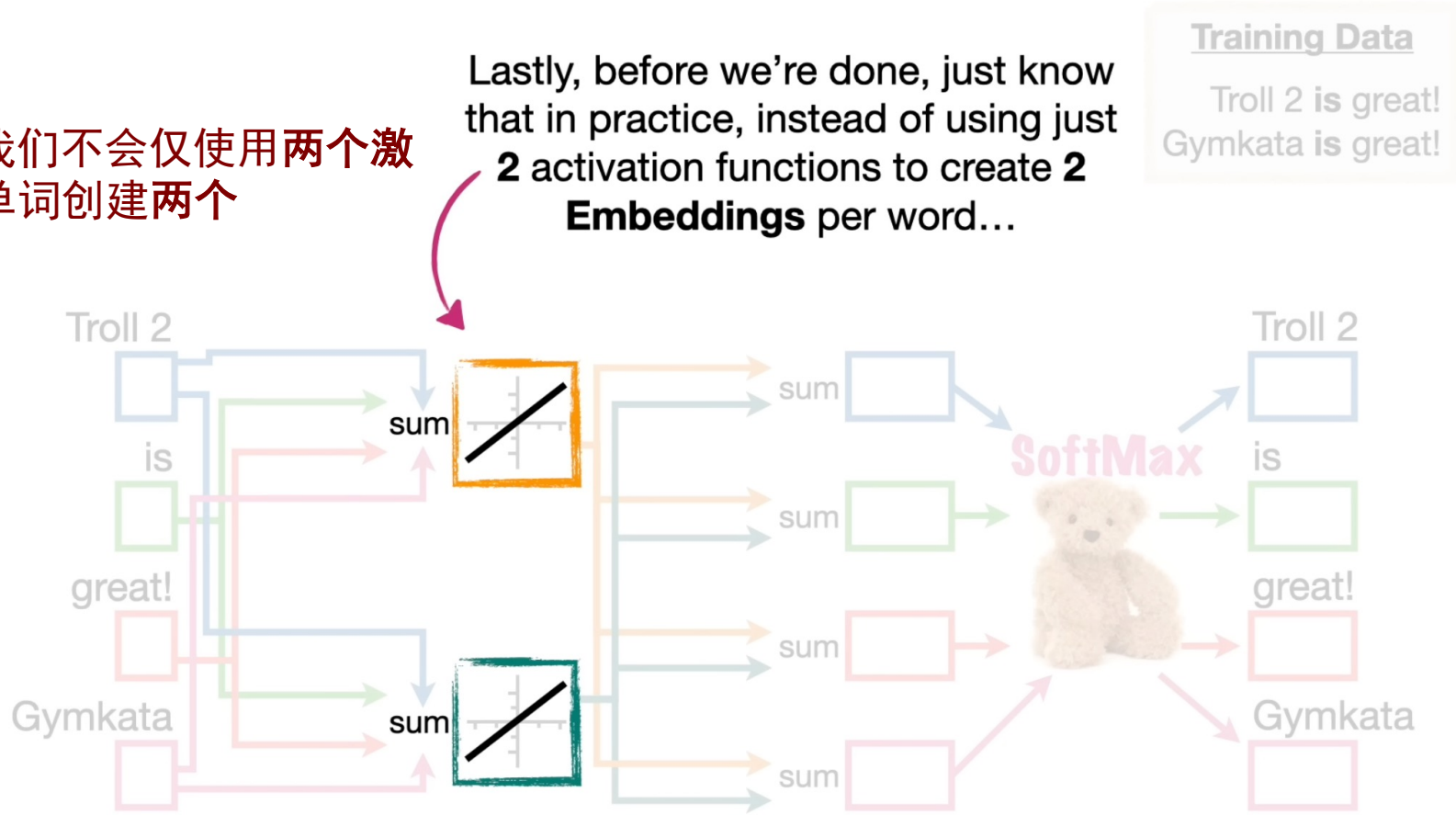
CBOW Model



Skip-gram Model

Word2Vec

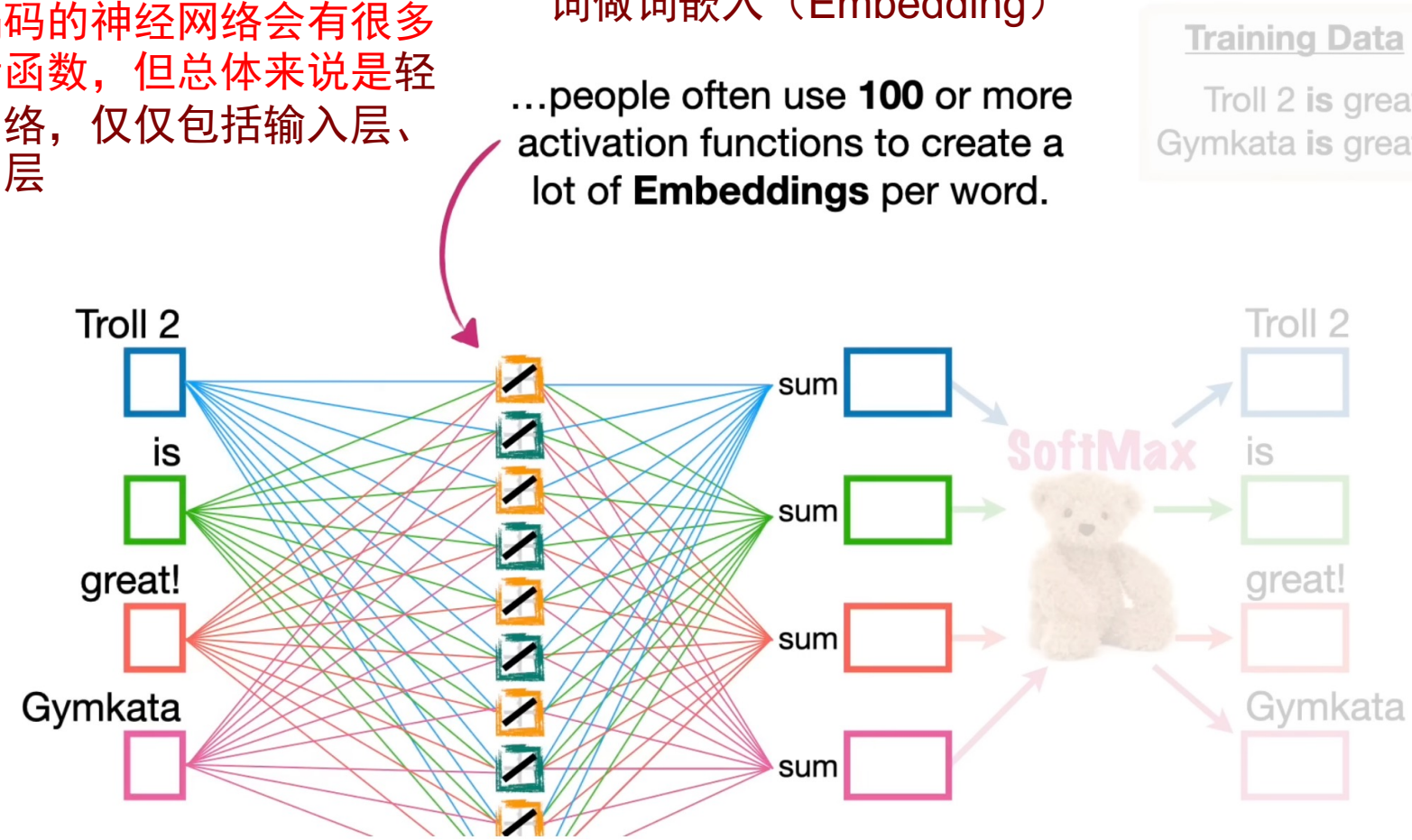
在实际操作中，我们不会仅使用**两个激活函数**来为每个单词创建**两个Embeddings**



Word2Vec

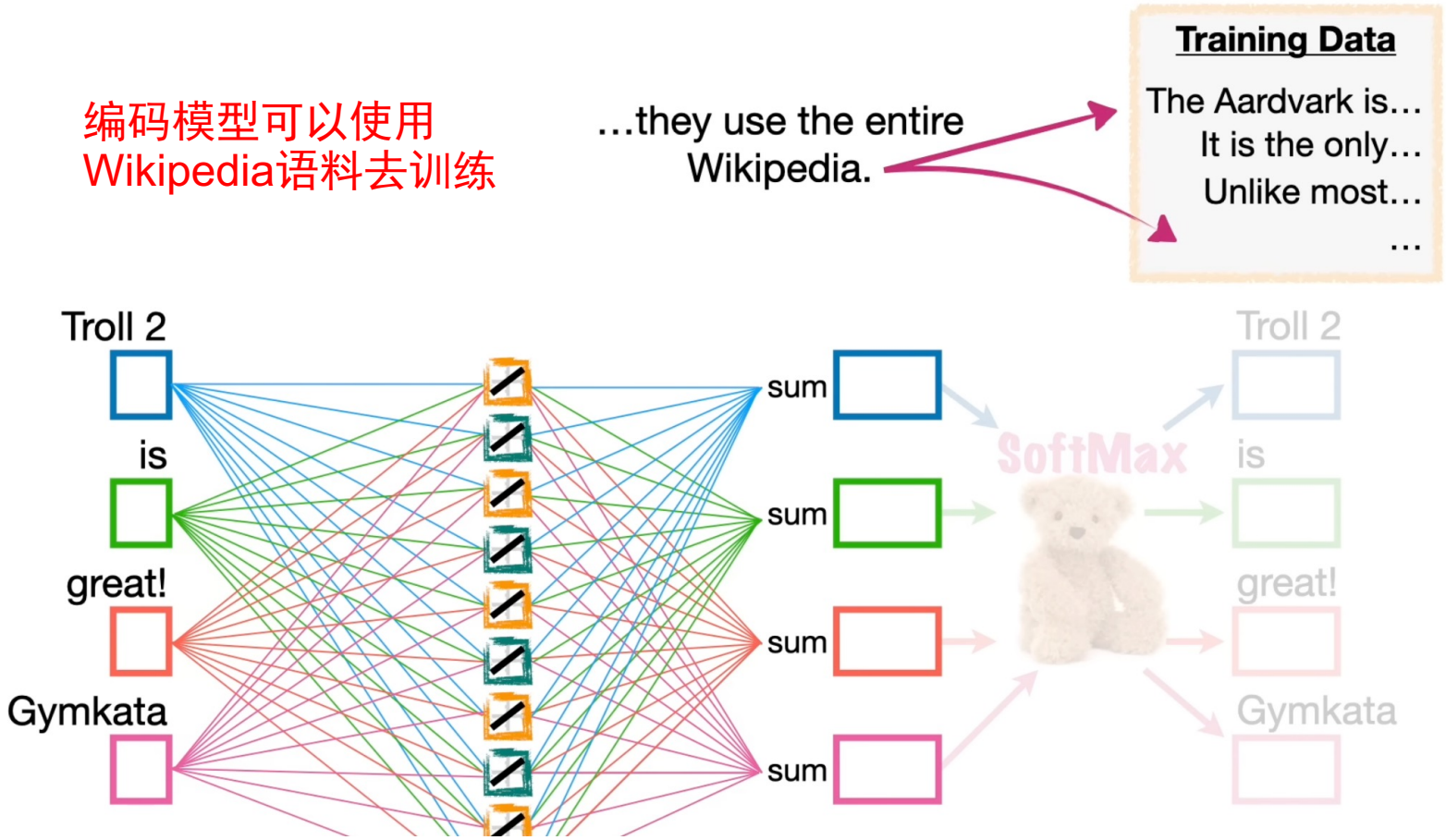
实际中用于编码的神经网络会有很多神经元和激活函数，但总体来说是轻量级的神经网络，仅仅包括输入层、隐藏层和输出层

现实生活中人们通常用100个或更多激活函数去为每一个单词做词嵌入（Embedding）



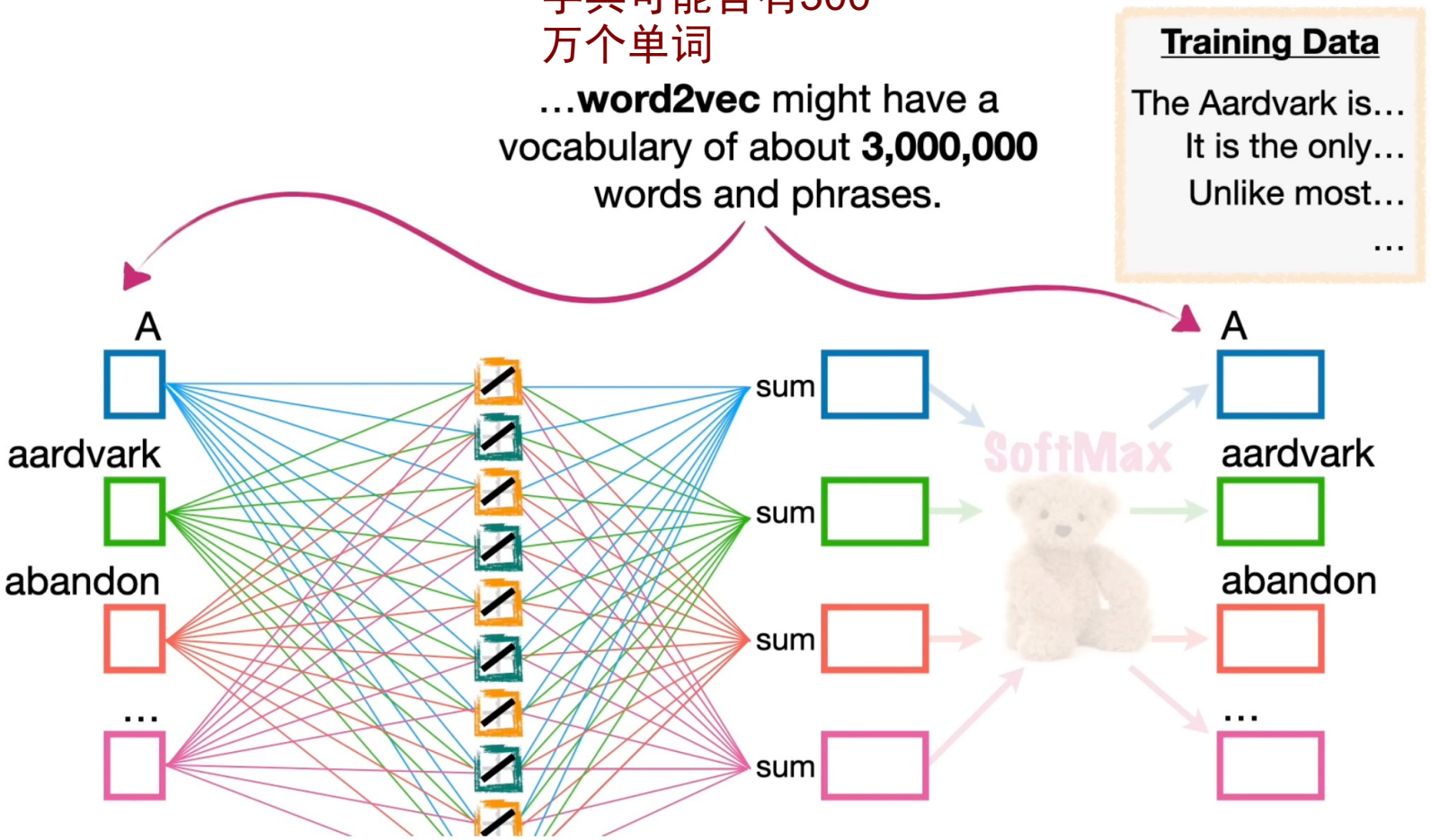
Word2Vec

编码模型可以使用
Wikipedia语料去训练



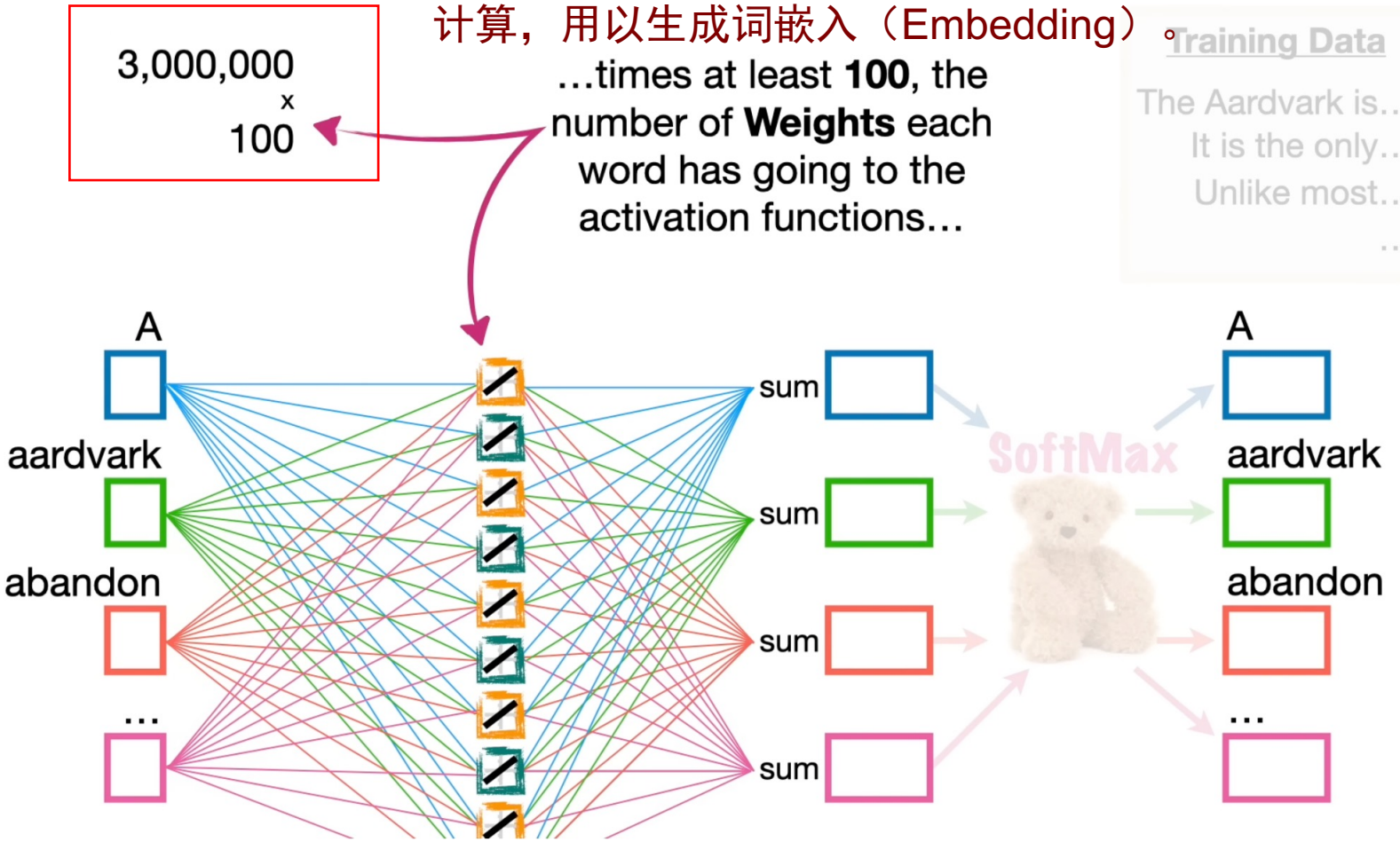
Word2Vec

字典可能含有300万个单词



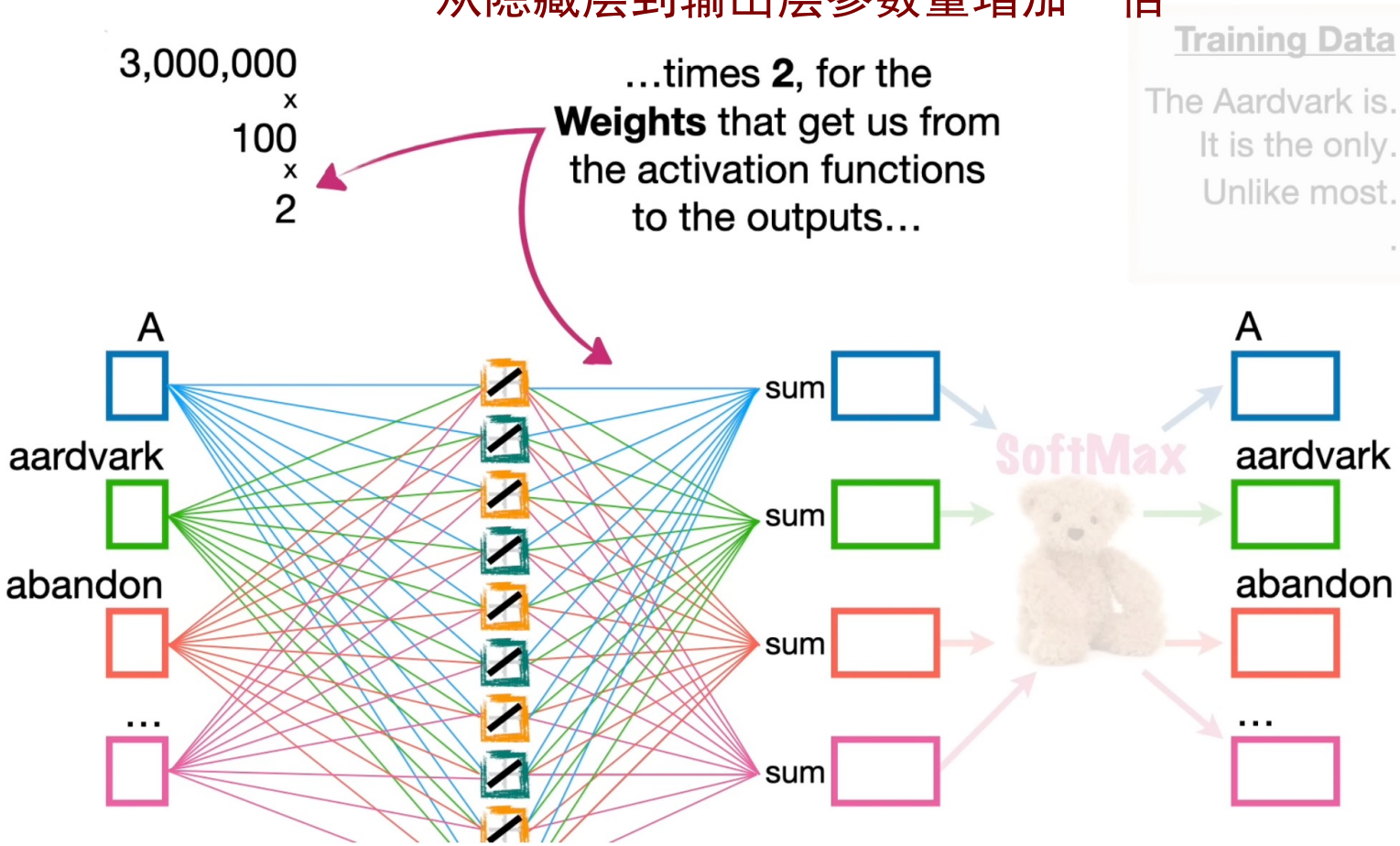
Word2Vec

每个单词至少会有100个**权重**参与激活函数的计算，用以生成词嵌入（Embedding）。

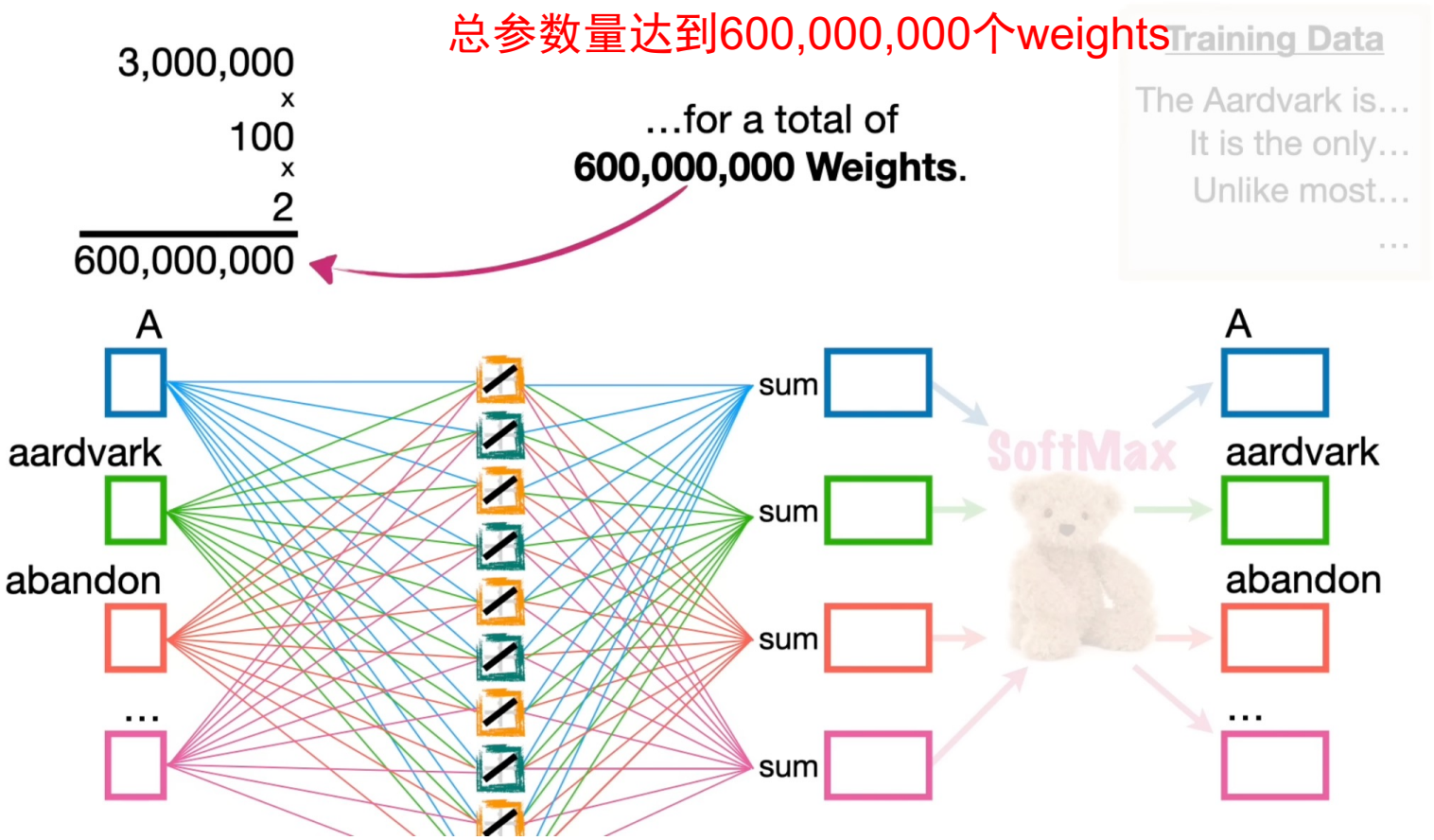


Word2Vec

从隐藏层到输出层参数量增加一倍



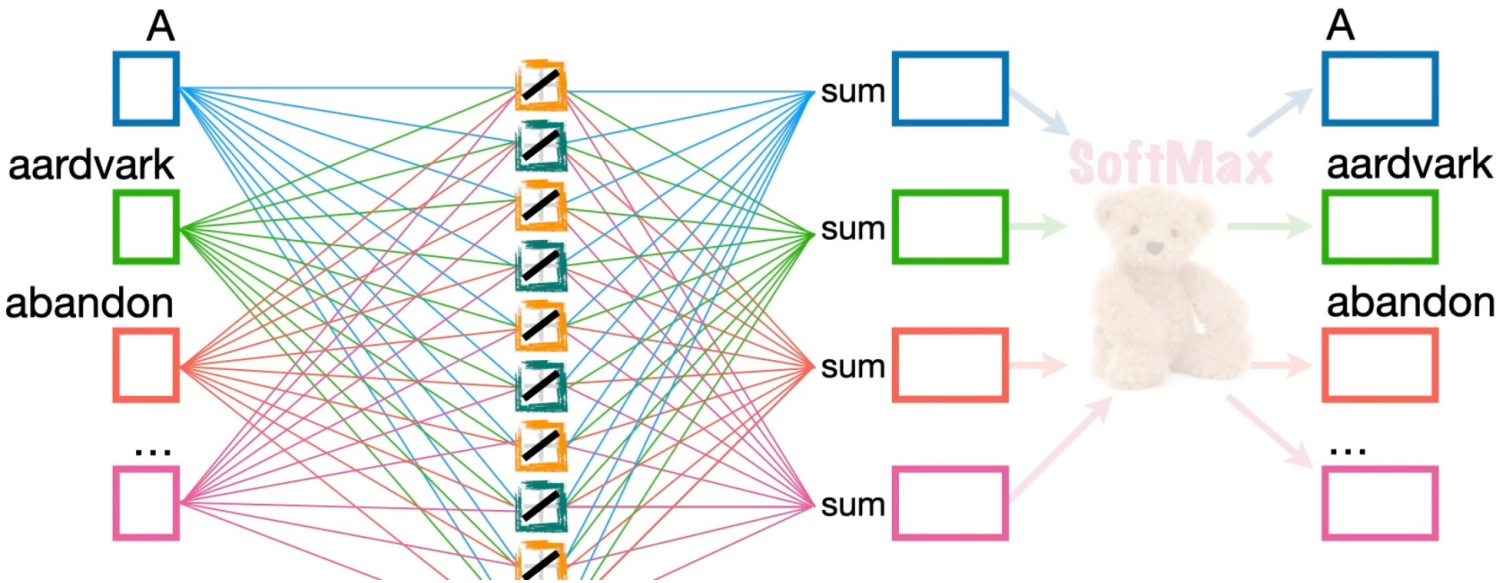
Word2Vec


总参数量达到600,000,000个weights
3,000,000
x
100
x
2
600,000,000
…for a total of
600,000,000 Weights.
Training Data
The Aardvark is…
It is the only…
Unlike most…
…
A
aardvark
abandon
…
sum
sum
sum
sum
SoftMax
A
aardvark
abandon
…

Word2Vec

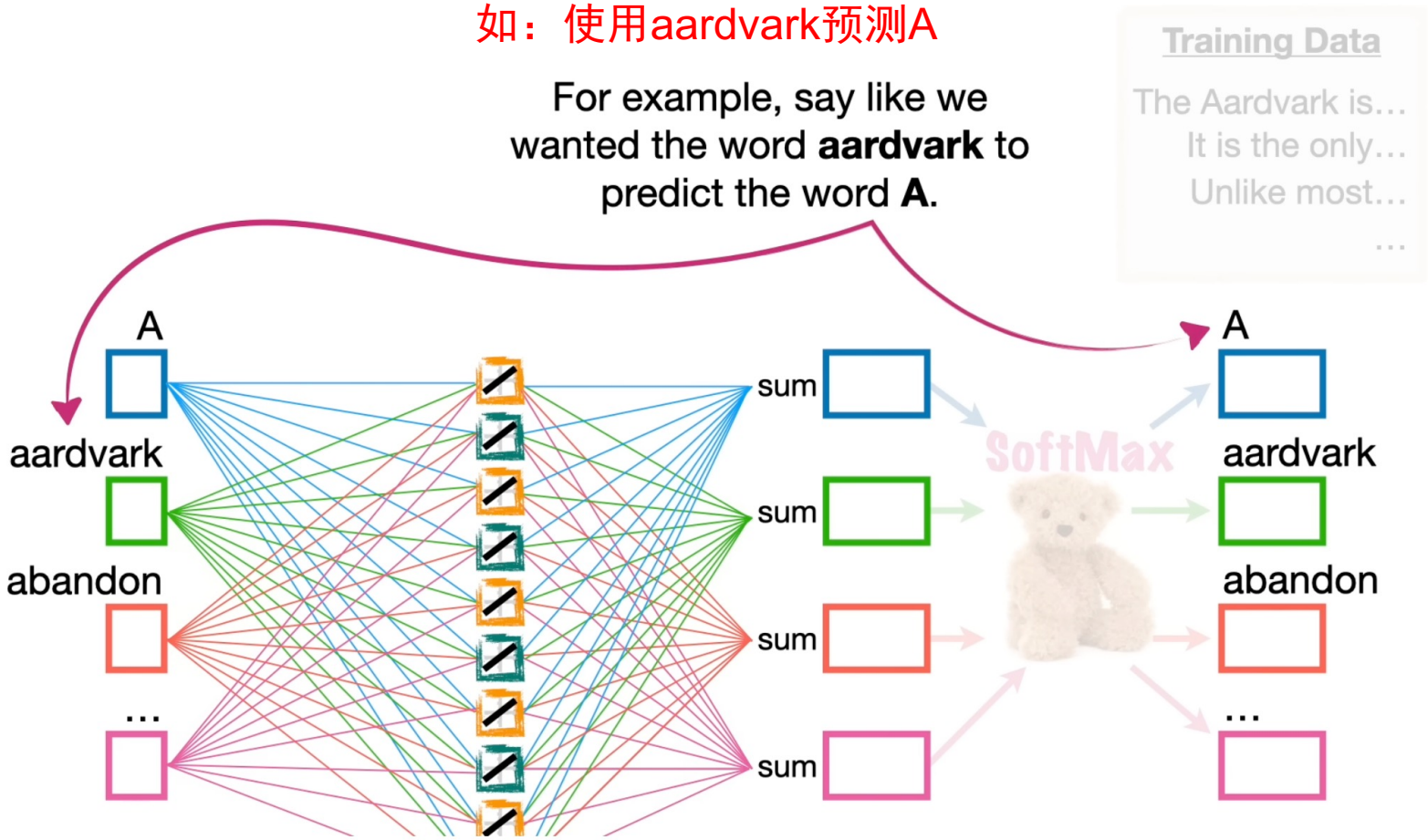
加速方法：Negative Sampling

However, one way that **word2vec** speeds things up is to use something called **Negative Sampling**.



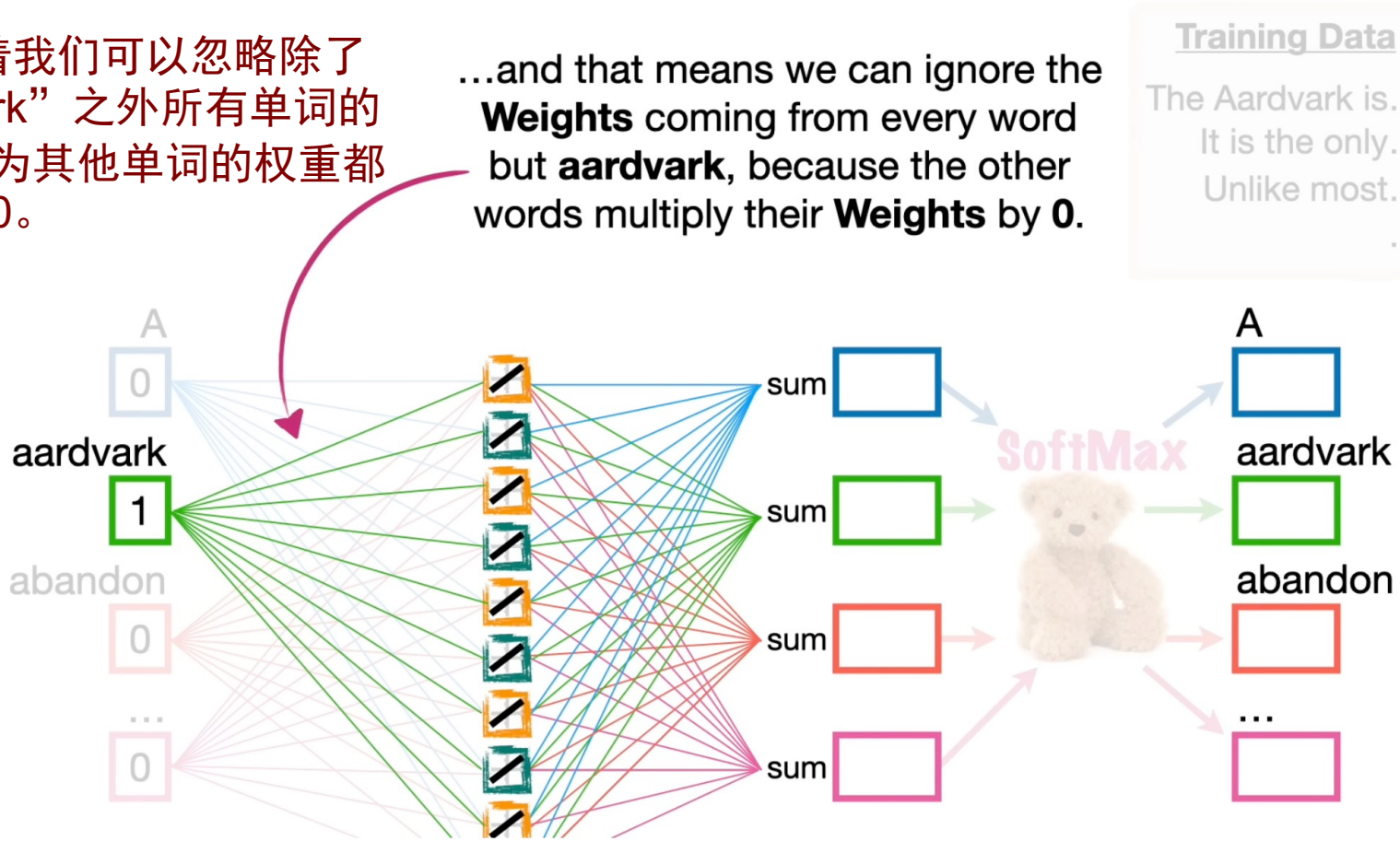
Word2Vec

如：使用aardvark预测A



Word2Vec

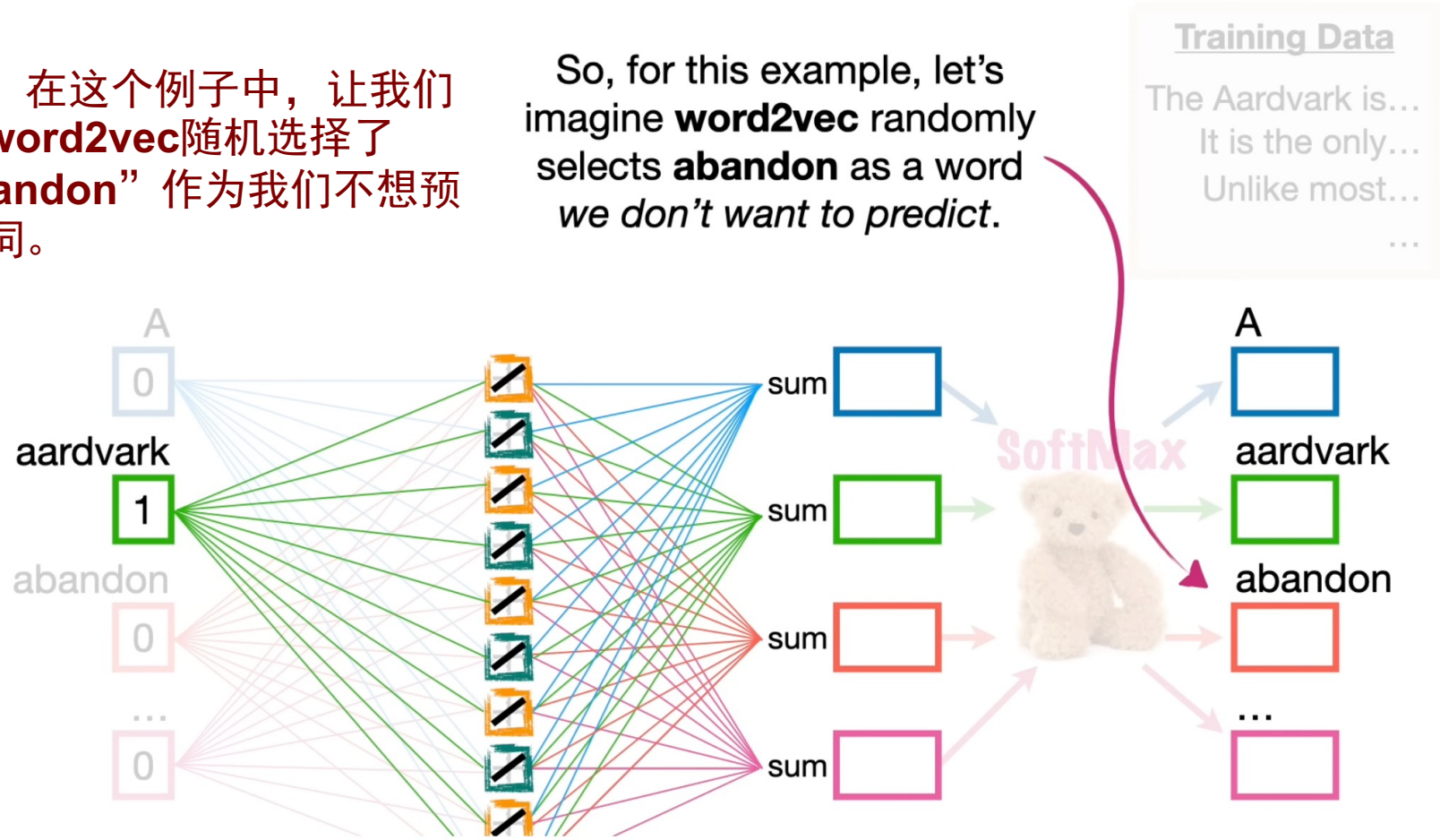
..这意味着我们可以忽略除了“aardvark”之外所有单词的权重，因为其他单词的权重都被乘以了0。



前面可以忽略与aardvark不相关的参数

Word2Vec

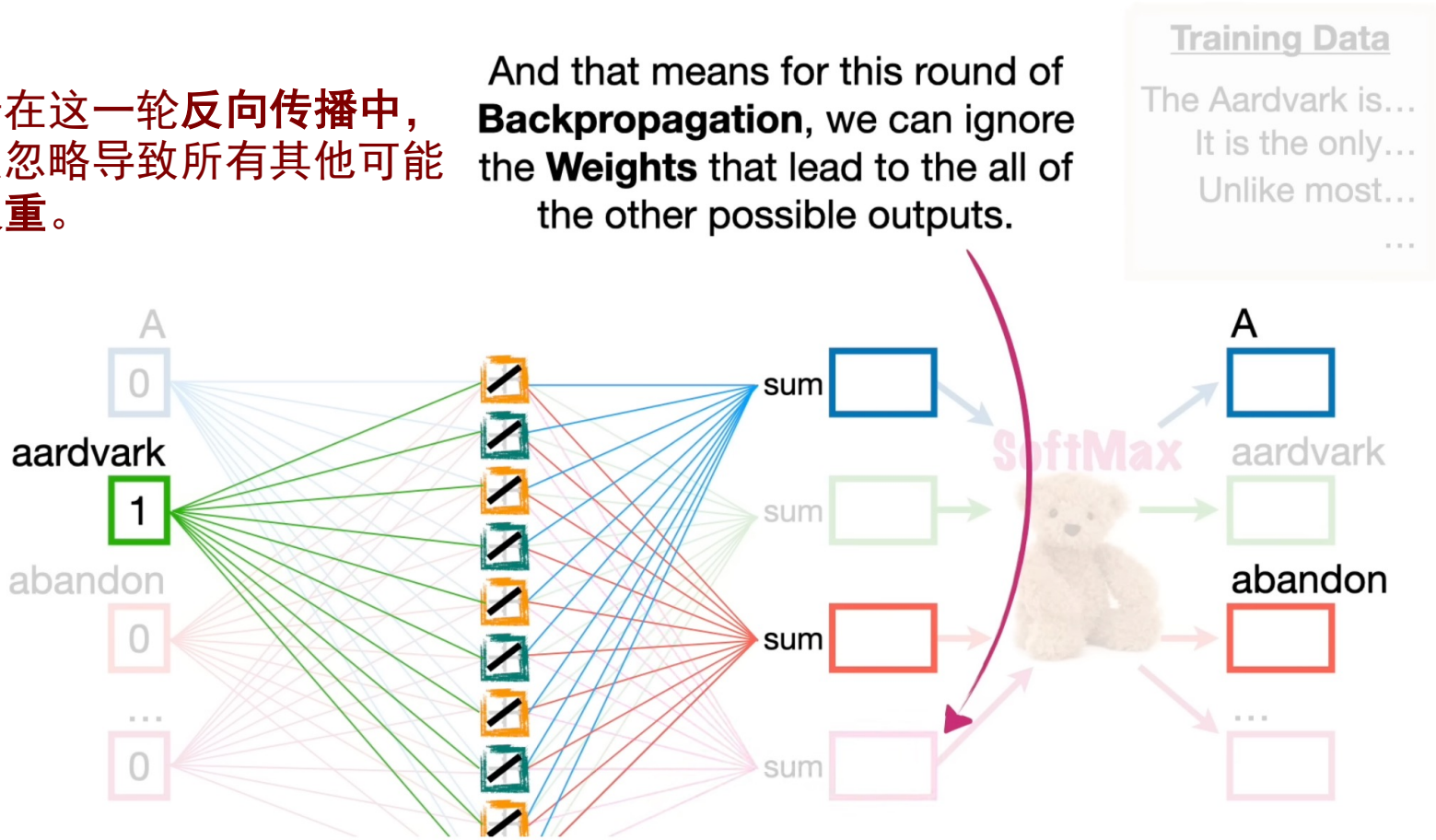
那么，在这个例子中，让我们假设**word2vec**随机选择了“**abandon**”作为我们不想预测的词。



后面只保留一个想要预测的和一个不想要预测的单词

Word2Vec

这意味着在这一轮**反向传播中**，我们可以忽略导致所有其他可能输出的**权重**。



这样可以忽略掉大量的其他样本（299998个）

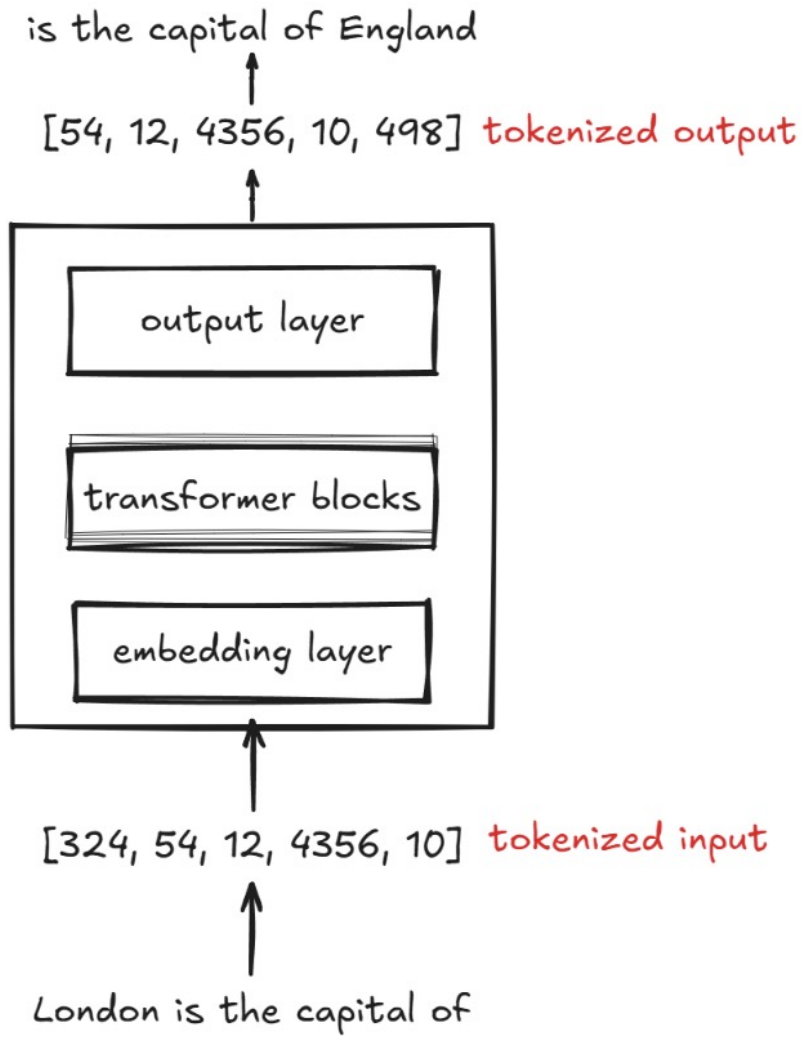
对比学习理念

https://www.youtube.com/watch?v=viZrOnJclY0

大语言模型中的词向量化方法

- 动态在训练中更新查找表（GPT4：词汇表12万，向量维度7168）

- “While we can use pretrained models such as Word2Vec to generate embeddings for machine learning models, LLMs commonly produce their own embeddings that are part of the input layer and are updated during training. The advantage of optimizing the embeddings as part of the LLM training instead of using Word2Vec is that the embeddings are optimized to the specific task and data at hand.” - 《**Build a large language model (From scratch)**》



练习时间

5. 在自然语言处理（NLP）任务中，文本数据需要转换为可供计算机处理的数值表示。词向量（Word Embeddings）作为一种分布式表示方法，被广泛应用于各种 NLP 任务。

（1）（4 分）请说明为什么在 NLP 中需要将文本转换为词向量表示，而不仅仅使用原始文本或简单的特征表示。

（2）（5 分）与基于独热编码（One-Hot Encoding）的词表示方法相比，词向量具备哪些显著优势？

（3）（5 分）与传统的词袋模型（BoW）方法相比，Word2Vec 具备哪些显著优势？

练习时间

5. 在自然语言处理（NLP）任务中，文本数据需要转换为可供计算机处理的数值表示。词向量（Word Embeddings）作为一种分布式表示方法，被广泛应用于各种 NLP 任务。

（1）（4 分）请说明为什么在 NLP 中需要将文本转换为词向量表示，而不仅仅使用原始文本或简单的特征表示。

（2）（5 分）与基于独热编码（One-Hot Encoding）的词表示方法相比，词向量具备哪些显著优势？

（3）（5 分）与传统的词袋模型（BoW）方法相比，Word2Vec 具备哪些显著优势？

答案：

(1) 为什么需要将文本转换为词向量表示（4 分）

计算机无法直接处理原始文本，文本是非结构化数据，计算机无法直接理解和处理，需要将其转换为数值形式以用于机器学习和深度学习算法。

可以捕捉词语语义信息，相比简单的特征表示（如独热编码），词向量通过分布式表示将词语嵌入到连续的低维向量空间中，使得语义相近的词在向量空间中彼此接近，能更好地捕捉上下文和语义关系。

(2) 词向量相对于独热编码的显著优势（5 分）

维度更低，计算更高效，独热编码会生成一个高维稀疏向量（维度等于词汇表大小），而词向量的维度较低（例如 100~300），大幅减少内存和计算资源消耗。

捕捉语义相似性，独热编码无法表示词之间的语义关系，而词向量通过训练捕捉到了词的上下文语义信息，使得相似词具有相近的向量表示。

避免稀疏性问题，独热编码产生稀疏矩阵，难以捕捉全局语义信息，而词向量是密集表示，能更高效地表示语义特征。

(3) Word2Vec 相对于词袋模型的显著优势（5 分）

捕捉词语上下文关系，词袋模型只考虑单词的频率或权重，忽略词序和上下文关系；而 Word2Vec 基于上下文窗口训练，能够捕捉词语的上下文语义信息。

支持迁移学习，Word2Vec 训练的词向量可以在不同任务间共享，降低对标注数据的需求；而词袋模型方法通常需要针对具体任务重新计算。

低维且密集，Word2Vec 生成低维密集向量，具有更好的计算效率和存储优势；词袋模型生成高维稀疏向量，计算成本较高且容易过拟合。

项目推荐

- 生物医学工程创新设计竞赛，报名地址www.bmedesign.cn
- 若干本科生命题项目（医疗AI，脑机接口），**参与可替代大作业二**
- 报名截止：2026年4月24日23点59分

谢谢！