第8节 Word Vectors

Natural Language Processing with Deep Learning
CS224N/Ling284

1. NLP与深度学习

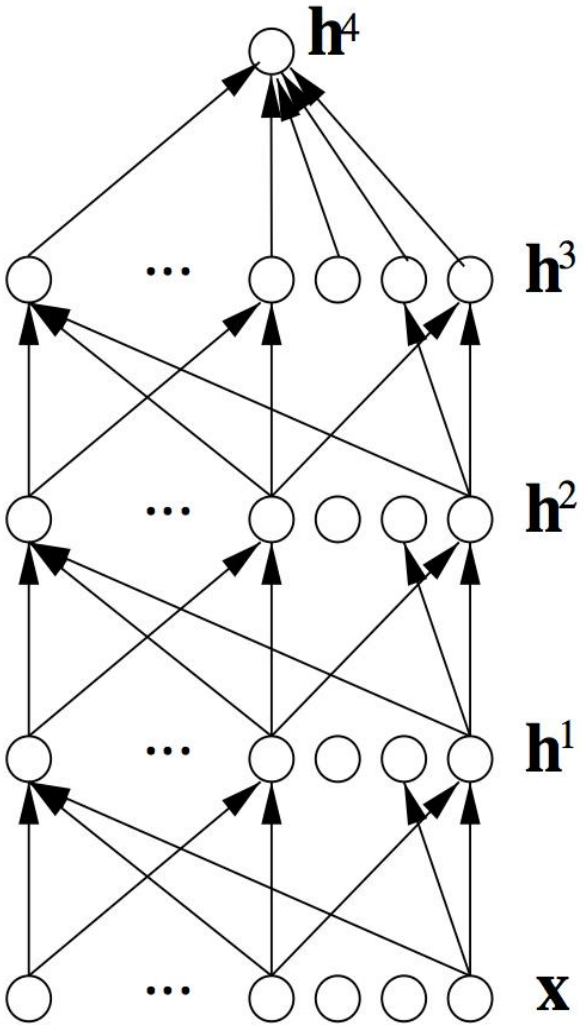
什么是深度学习（DL）？

深度学习是机器学习的一个子领域。机器学习只是优化权重，以最好地做出最终的预测。

表征学习（Representation learning）试图自动学习好的特征或表征。

深度学习算法试图学习（多层次）表示和输出。

从“原始”输入x（例如声音，字符或文字）。



探索深度学习的原因

手动设计的功能通常过多，不完整，且要花费很长时间进行设计和验证。
学习特征易于调整，且学习速度快。
深度学习为表示世界的视觉和语言信息提供了一个非常灵活，（几乎？）通用、可学习的框架。
深度学习可以无监督地（从原始文本）和有监督地（特定的标签，如正/负）学习。

从2010年开始，深度学习技术的表现优于其他机器学习技术。 为什么?

大量的训练数据的积累，对深度学习的研究有利。
更快的机器和多核CPU / GPU，支持了深度学习的研究。
新模型，算法，想法的出现：

- 更好，更灵活的中间表示（intermediate representations）学习；
- 有效的端到端联合系统学习；
- 使用上下文和在任务之间转移的有效学习方法。

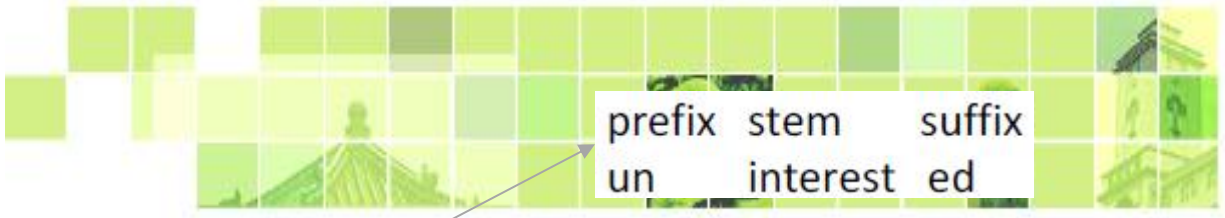
深度学习在NLP中的应用

深度NLP = 深度学习 + NLP

结合自然语言处理的思想和目标，采用表示学习和深度学习的方法来解决这些问题。

近年来NLP有了很大的改进与不同（后面说明）：

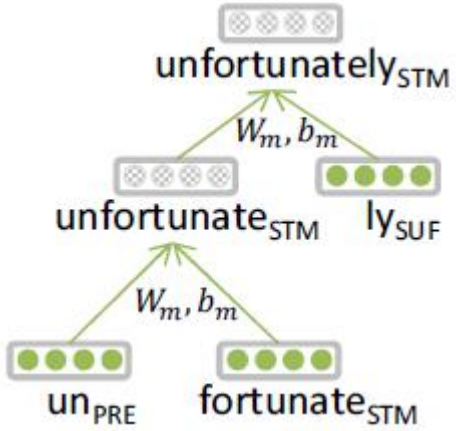
- 层次：语音，文字，语法，语义
- 工具：词性，实体，解析
- 应用：机器翻译，情感分析，对话代理，问答系统



NLP层级的表示：形态学

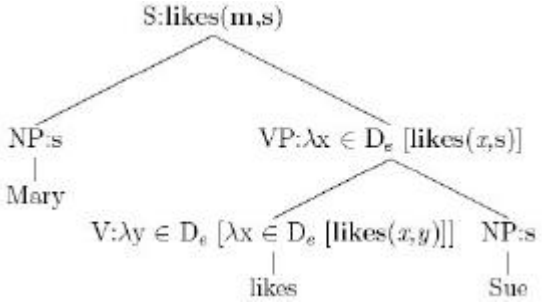
- 传统：词语是由语素组成的。
- 深度学习：每个语素都是一个向量，神经网络将两个向量组合成一个向量

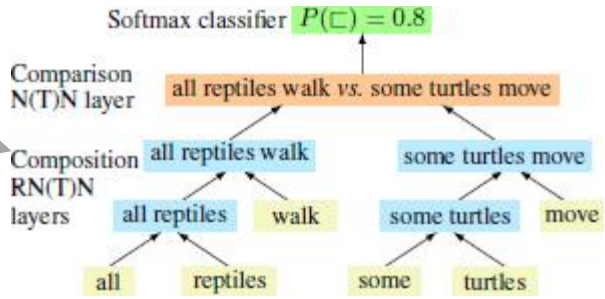
注：最大的语法单位是句子，比句子小的语法单位，依次是短语、词、语素。



NLP层级的表示：语义

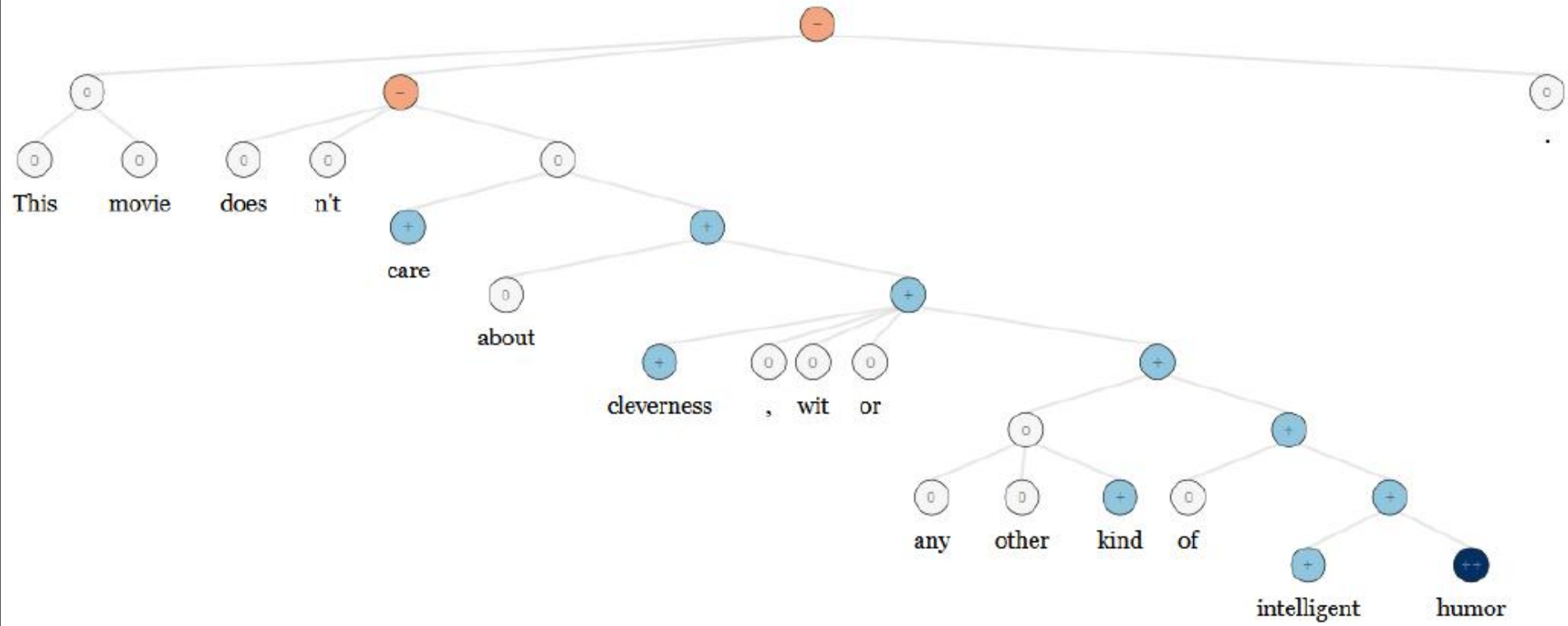
- 传统：Lambda演算：精心设计的功能、作为输入具体的其他功能、没有语言的相似性或模糊性的概念。
- 深度学习：每个单词，每个词组和每个逻辑表达式都是一个向量，神经网络将两个向量组合成一个向量。





NLP应用：情感分析

- 传统：精心策划的情感词典结合袋表示（忽略词序）或人工设计否定特征（不会捕捉所有内容）
- 深度学习：可以使用用于形态学、语法和逻辑语义学的相同的深度学习模型（递归NN ）



NLP应用：问答系统

- 传统：很多用于捕捉世界和其他知识的特征，例如正则表达式。

Is main verb trigger?

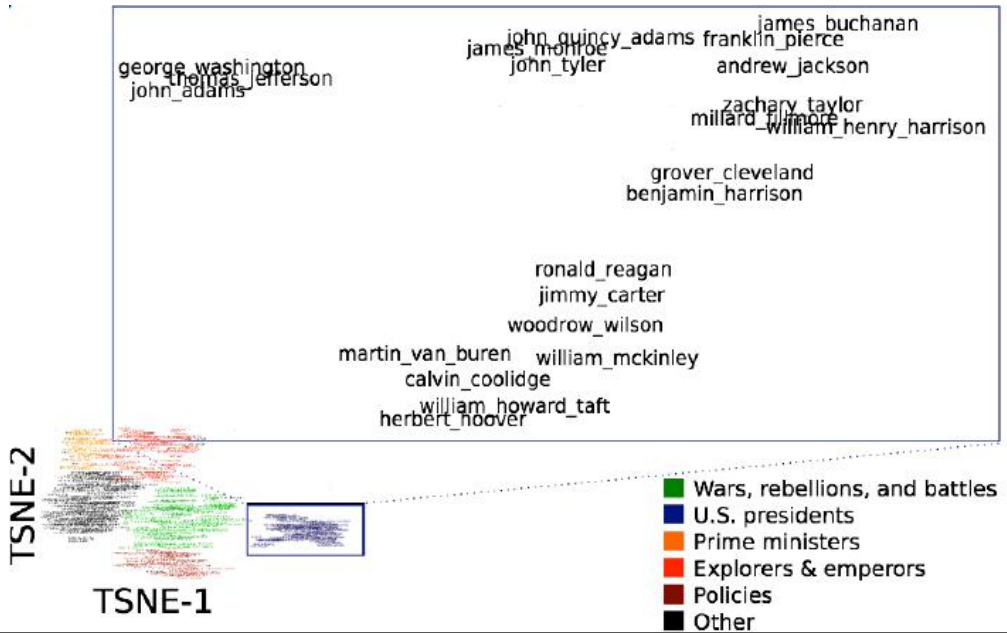
Yes

Condition	Regular Exp.
Wh- word subjective?	AGENT
Wh- word object?	THEME

No

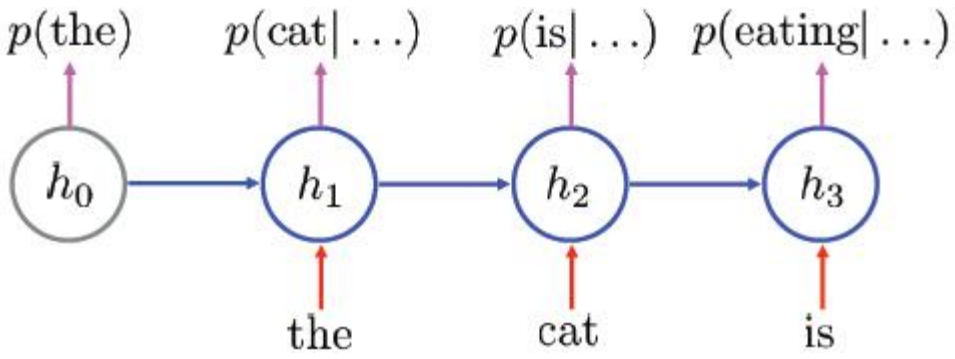
Condition	Regular Exp.
default	(ENABLE|SUPER)+
DIRECT	(ENABLE|SUPER)
PREVENT	(ENABLE|SUPER)* PREVENT(ENABLE|SUPER)*

- 深度学习：可以使用深度学习架构。事实存储在向量中。



NLP应用：对话代理/回复生成

- 一个简单而成功的例子是Google Inbox APP中提供的自动回复功能。
- 这是一个递归神经网络的实例，是强大且通用的神经语言模型的应用。



NLP应用：机器翻译

- 过去曾经尝试过很多层次的翻译：传统的MT系统是非常庞大的复杂系统。
- 神经机器翻译：源语句被映射到向量，然后输出生成的语句。现在被应用在谷歌翻译（等）的一些语言翻译上，大大降低了错误率！

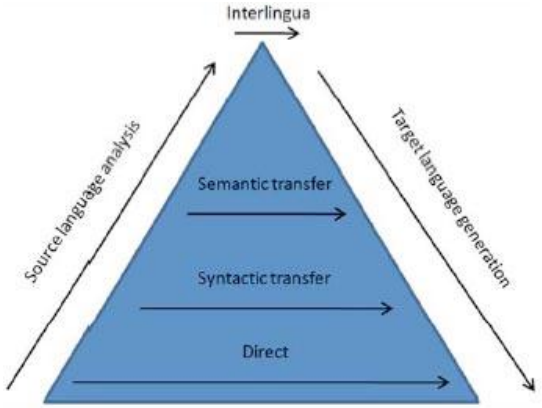
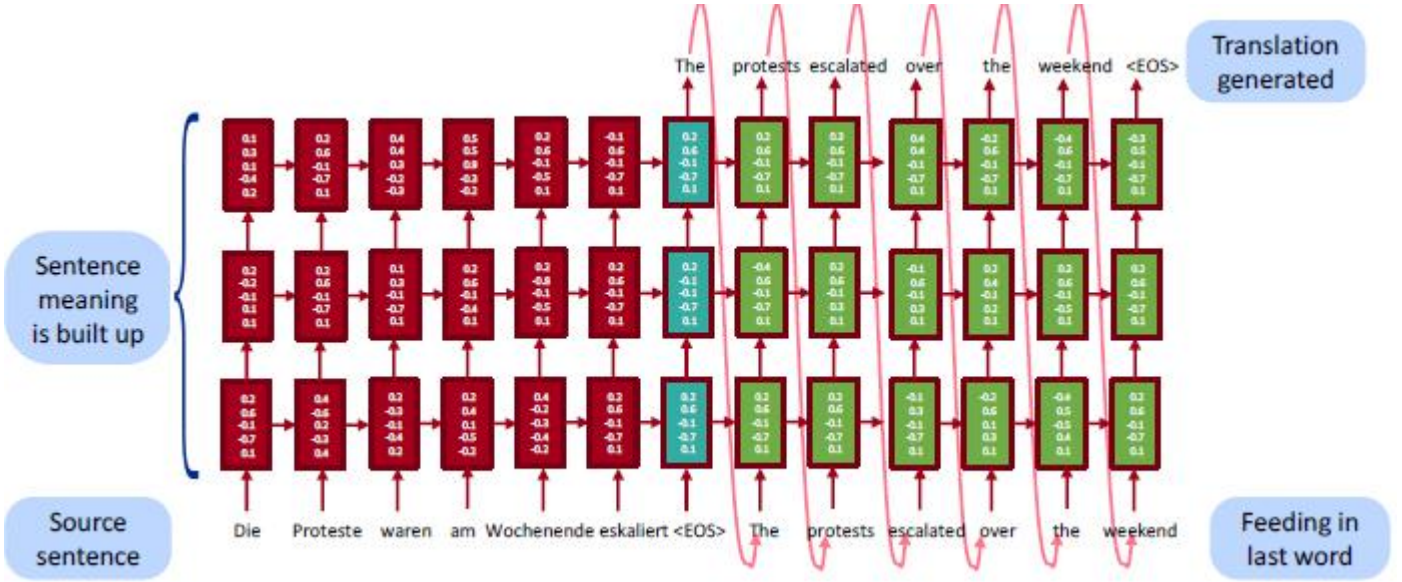
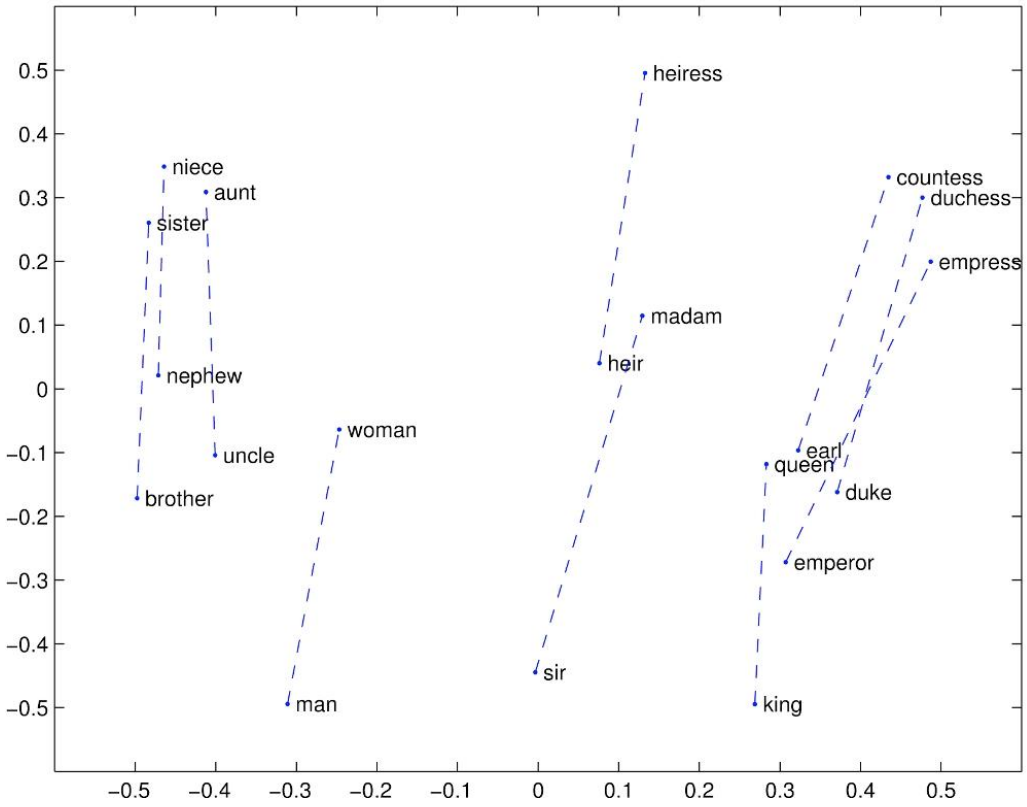


Figure 1: The Vauquois triangle



结论：在各个层级上的表示方式——向量

学习单词的矢量表示以及它们实际代表什么；神经网络是如何工作的，以及如何将这些向量用于所有NLP层级和其他更多不同的应用上。



2. Word Vectors

计算机中如何表达一个词的含义?

WordNet是nltk中经常使用的一个语料库，它包含一个同义词集（synonym sets）和上下位关系（hypernyms，“is-a”）

e.g. synonym sets containing “good”:

```
from nltk.corpus import wordnet as wn
poses = { 'n':'noun', 'v':'verb', 's':'adj (s)', 'a':'adj', 'r':'adv'}
for synset in wn.synsets("good"):
    print("{}: {}".format(poses[synset.pos()],
            ", ".join([l.name() for l in synset.lemmas()])))
```

```
noun: good
noun: good, goodness
noun: good, goodness
noun: commodity, trade_good, good
adj: good
adj (sat): full, good
adj: good
adj (sat): estimable, good, honorable, respectable
adj (sat): beneficial, good
adj (sat): good
adj (sat): good, just, upright
…
adverb: well, good
adverb: thoroughly, soundly, good
```

e.g. hypernyms of “panda”:

```
from nltk.corpus import wordnet as wn
panda = wn.synset("panda.n.01")
hyper = lambda s: s.hypernyms()
list(panda.closure(hyper))
```

```
[Synset('procyonid.n.01'),
Synset('carnivore.n.01'),
Synset('placental.n.01'),
Synset('mammal.n.01'),
Synset('vertebrate.n.01'),
Synset('chordate.n.01'),
Synset('animal.n.01'),
Synset('organism.n.01'),
Synset('living_thing.n.01'),
Synset('whole.n.02'),
Synset('object.n.01'),
Synset('physical_entity.n.01'),
Synset('entity.n.01')]
```

但这种离散的词义表示方法存在一些问题：

- 同义词间细微差别的表示无法满足，如：expert, good, practiced, proficient, skillful…
- 缺少新词（不可能永远保持最新）：wicked, badass, nifty, crack, wizard, ninja…
- 可能过于人为主观
- 需要人力来创造和适应
- 很难计算准确的单词相似性

不论是基于大量规则的方式，还是统计性NLP方式，大都是在基本词粒度上处理成**one-hot**形式，即0和1组成的矢量（矢量维度 = 字典中的单词数）

例如，

motel = [0 0 0 0 0 0 0 0 0 0 1 0 0 0 0]

hotel = [0 0 0 0 0 0 0 1 0 0 0 0 0 0 0]

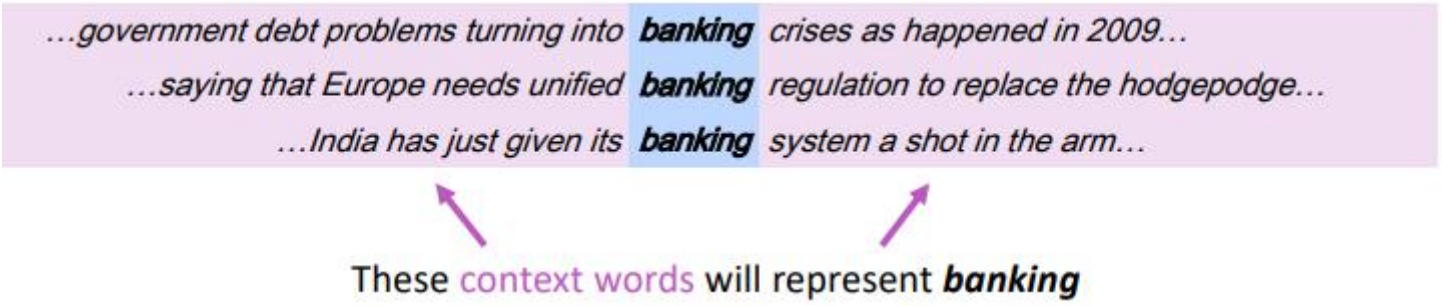
如果用户搜索[Seattle motel]，我们期望匹配的内容包含“Seattle hotel”。但是，motel和hotel是意思相近的两个词，可是两个词的one-hot向量却是正交的。显然**one-hot不能表示词之间的相似性**

我们可以考虑通过语料库找到同义词集，但这不是一个好的做法。

在one-hot的基础上进行进一步的处理：考虑使用一种维度较低并且有递推关系的向量来表示词，相似的词具有相似的向量。

基于分布相似性的表示（Distributed representation）

从一个词周围词的集合当中可以提取出有价值的信息（当一个单词出现在文本中，它的上下文是显示在附近（固定大小的窗口中）的一组单词）：



Dristributed representation可以解决one hot representation的问题，它的思路是通过训练，将每个词都映射到一个较短的词向量上来。所有的这些词向量就构成了向量空间，进而可以使用统计学的方法来研究词之间的关系。这词向量维度需要我们在训练时指定。

将稀疏的的向量，压缩到一个稠密的向量中，以此来表示一个词，至于这里每维特征的意义，对于人来说是感知不出来的，它是从计算机的角度来看的。

word vectors（词向量）有时也被称做

word embeddings（词嵌入，即将高维词向量嵌入到一个低维空间）

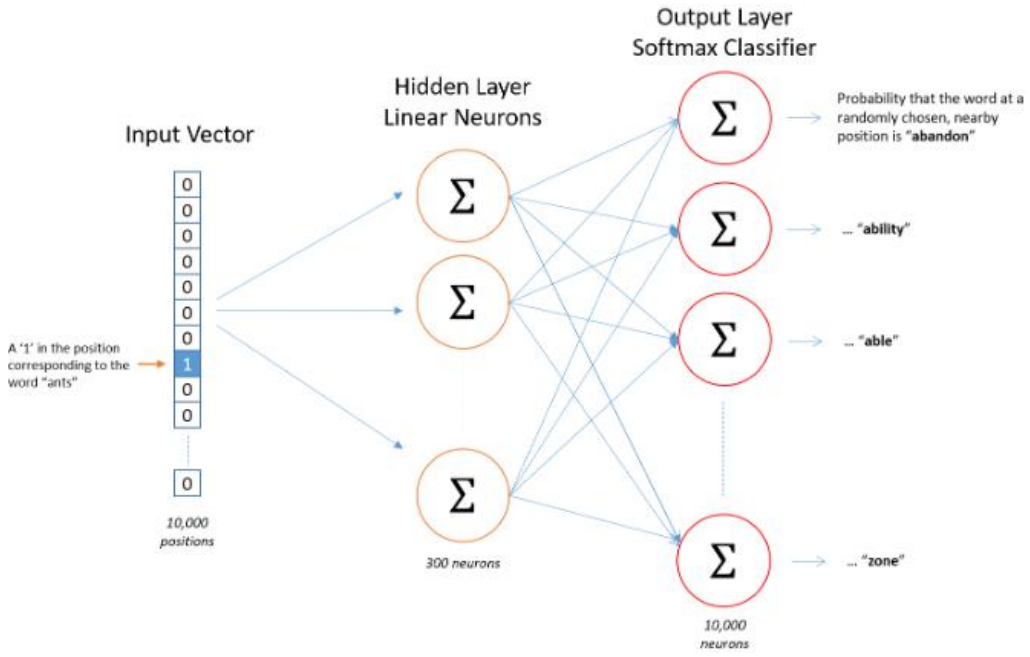
$$banking = \begin{pmatrix} 0.286 \\ 0.792 \\ -0.177 \\ -0.107 \\ 0.109 \\ -0.542 \\ 0.349 \\ 0.271 \end{pmatrix}$$

字符串形式的词语其实是更高维、更稀疏的向量。若词汇表大小为N，每个字符串形式的词语字典序为i，则其被表示为一个N维向量，该向量的第i维为1，其他维都为0。汉语的词汇量大约在十万这个量级，十万维的向量对计算来讲绝对是个维度灾难。而word2vec得到的词向量，则可以自由控制维度，一般是100左右

Word2vec实际是一个简单化的神经网络

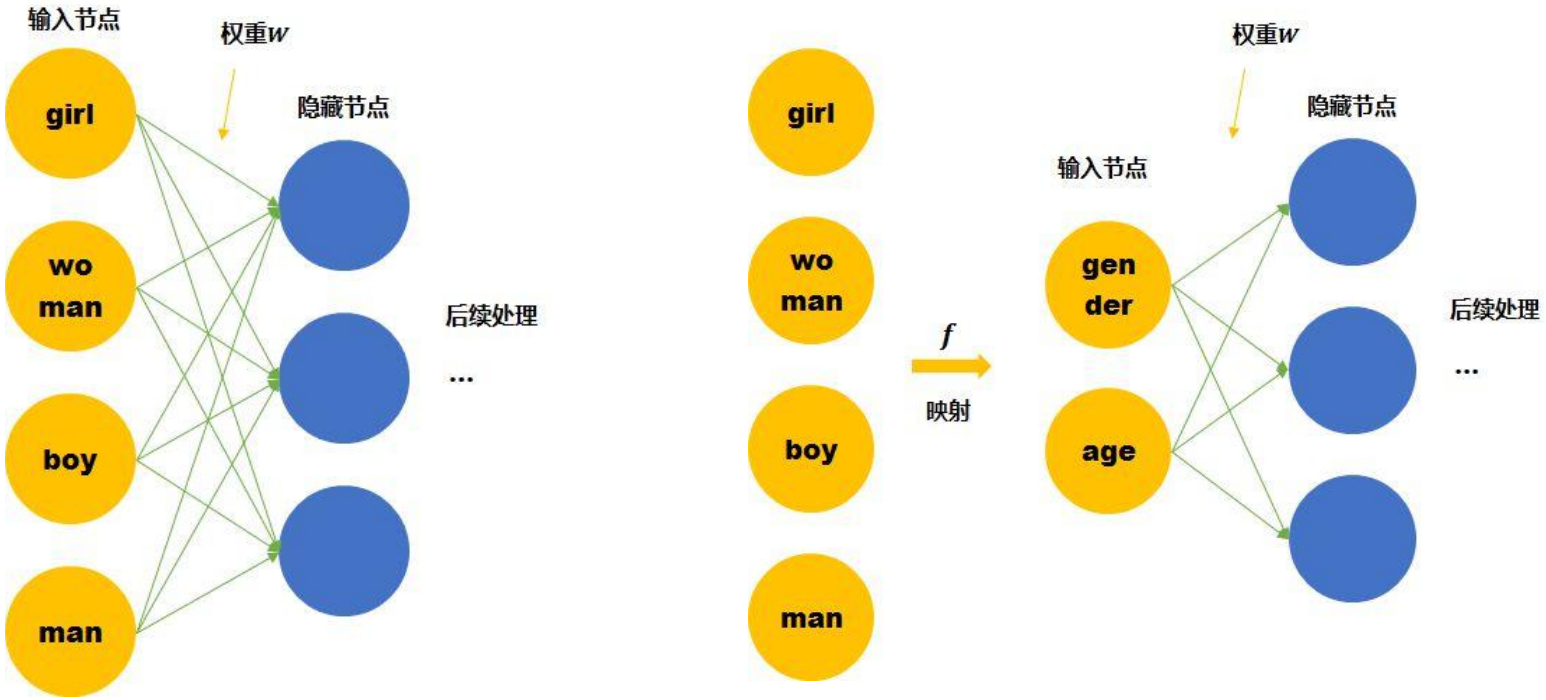
假设有一系列样本(x,y)，这里 x 是词语，y 是词性，要构建 f(x)->y 的映射。把 x 看做一个句子里的一个词语，y 是这个词语的上下文词语，那么这里的 f，便是语言模型（language model），这个模型的目的，就是判断 (x,y) 这个样本，是否符合自然语言的法则。

Word2vec的最终目的，不是要把 f 训练得多么完美，而是只关心模型训练完后的副产物——模型参数（这里特指神经网络的权重），并将这些参数，作为输入 x 的某种向量化的表示，这个向量便叫做——词向量。



在word2vec出现之前，已经有用神经网络DNN来用训练词向量进而处理词与词之间的关系了。采用的方法一般是一个三层的神经网络结构（当然也可以多层），分为输入层，隐藏层和输出层(softmax层)

我们以一个实际例子来看，神经网络需要学习的连接线的权重，word2vec的任务是降低训练的数据量，比如下面的例子将一个4✕3的学习任务缩小到2✕3



参考：Word Embedding--介绍 https://zhuanlan.zhihu.com/p/27830489

Word2vec

基本思想

- 假设我们有一个规模足够大的文本语料
- 固定词汇表中的每个词都用矢量表示
- 扫描文本的每个位置t，其中有一个中心词c和一个上下文词o
- 使用c和o的向量相似度来计算c给定时o出现的概率
- 不断调整词向量，得到最大化的可能

对于每个位置 $t=1,...,T$，给定一个固定窗口大小m和一个中心词 w_j，预测上下文的单词，可以表示为：

$$Likelihood = L(\theta) = \prod_{t=1}^{T} \prod_{\substack{-m \le j \le m \\ j \neq 0}} P(w_{t+j} \mid w_t, \theta)$$

- θ是需要优化的变量
- 目标函数（objective function）或者叫成本函数是likelihood的负对数

$$J(\theta) = -\frac{1}{T} \log L(\theta) = -\frac{1}{T} \sum_{t=1}^{T} \sum_{\substack{-m \le j \le m \\ j \neq 0}} \log P(w_{t+j} \mid w_t, \theta)$$

最小化目标函数 $\Leftrightarrow$ 最大化预测准确性

为了最小化 $J(\theta) = -\frac{1}{T}\log L(\theta) = -\frac{1}{T}\sum_{t=1}^{T}\sum_{\substack{-m\le j\le m \\ j\neq 0}}\log P(w_{t+j} \mid w_t, \theta)$

第一步是计算 $P(w_{t+j} \mid w_t, \theta)$

每个单词w中我们将使用两个向量：

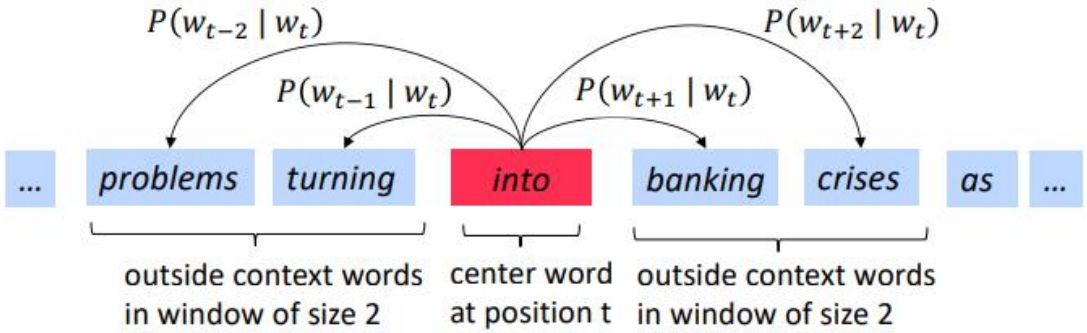
- v_w when w is a center word
- u_w when w is a context word

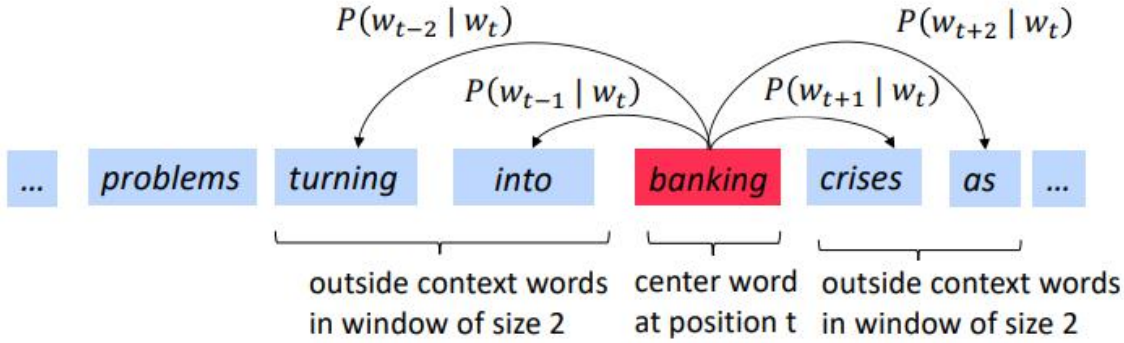
那么，对于每个中心词c和上下文词o，有：

点乘比较o和c之间的相似度：
$u^T v = u \cdot v$ 点乘的意义是两个向量越是近似，则点乘结果越大。

$$P(o \mid c) = \frac{\exp(u_o^T \mathrm{v}_c)}{\sum_{w \in V} \exp(u_w^T \mathrm{v}_c)}$$

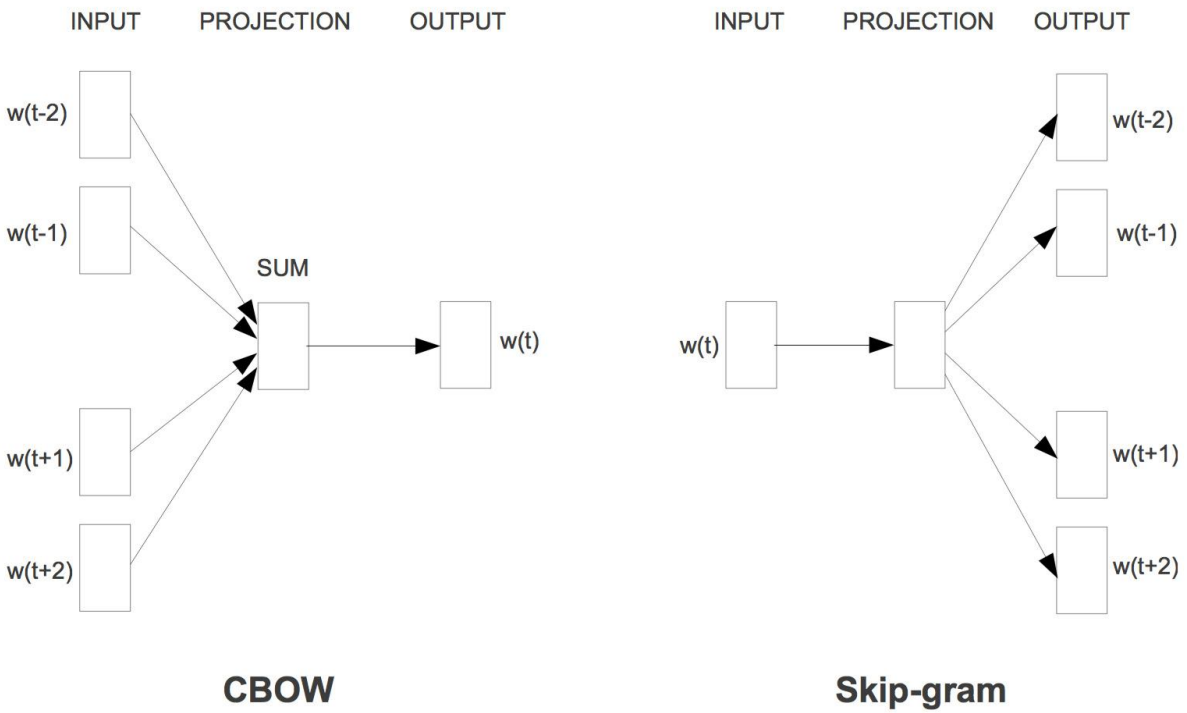
例如计算 $P(w_{t+i} \mid w_t)$





Word2vec两个主要模型

- 用输入单词作为中心单词去预测周边单词的方式叫做：The Skip-Gram Model（SG）
- 用输入单词作为周边单词去预测中心单词的方式叫做：Continuous Bag of Words (CBOW)

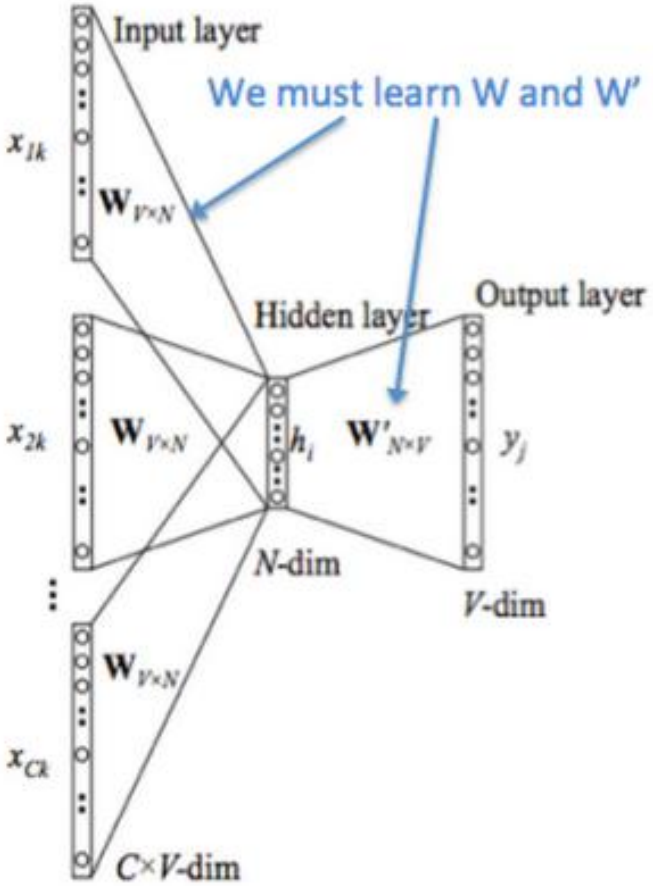


Continuous Bag of Words (CBOW)

CBOW又称连续词袋模型，是一个三层神经网络。如下图所示，该模型的特点是输入已知上下文，输出对当前单词的预测。

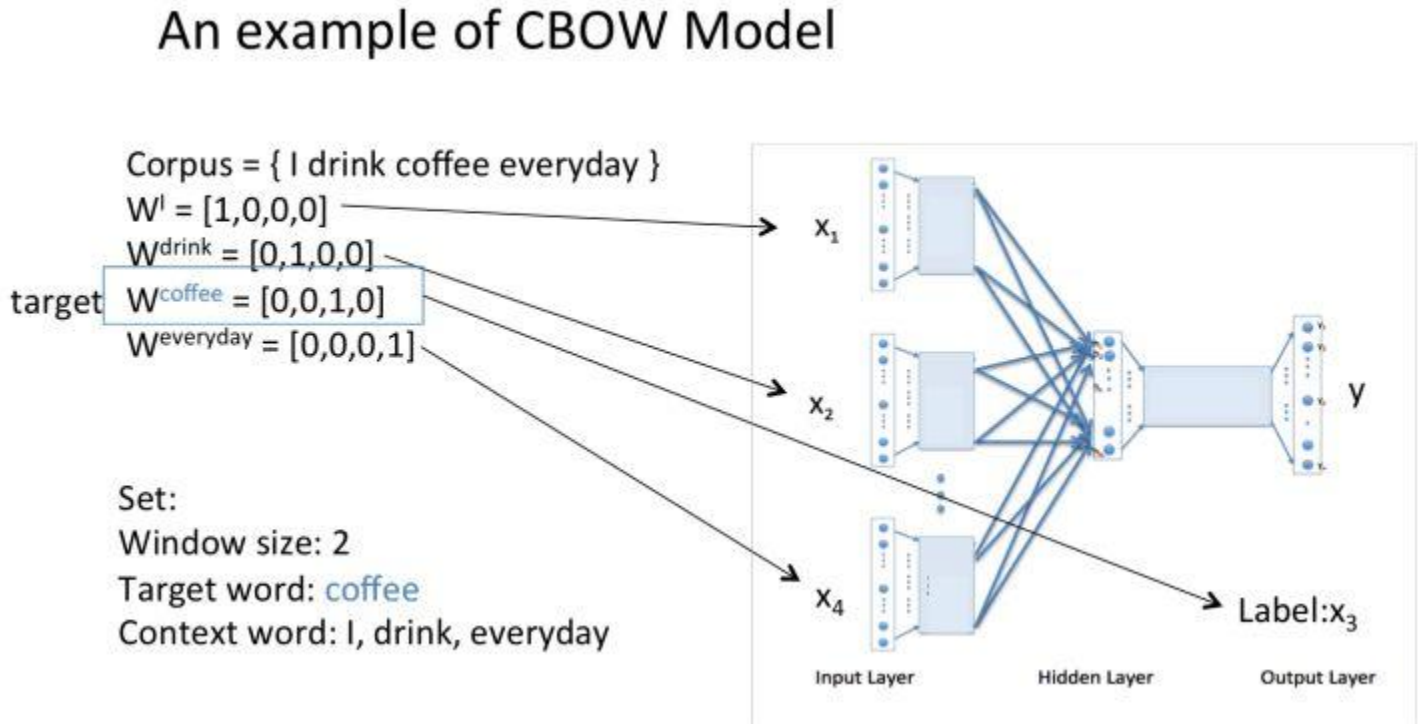
Notation for CBOW Model:

- w_i: Word i from vocabulary V
- $\mathcal{V} \in \mathbb{R}^{n \times |V|}$: Input word matrix
- v_i: i-th column of $\mathcal{V}$, the input vector representation of word w_i
- $\mathcal{U} \in \mathbb{R}^{n \times |V|}$: Output word matrix
- u_i: i-th row of $\mathcal{U}$, the output vector representation of word w_i



https://cs224d.stanford.edu/lecture_notes/notes1.pdf

具体例子



词袋模型先将句子分词，然后对每个词进行编码，常见的有one-hot、TF-IDF、Huffman编码，假设词与词之间没有先后关系

具体例子

An example of CBOW Model

Corpus = { I drink coffee everyday }

Initialize:

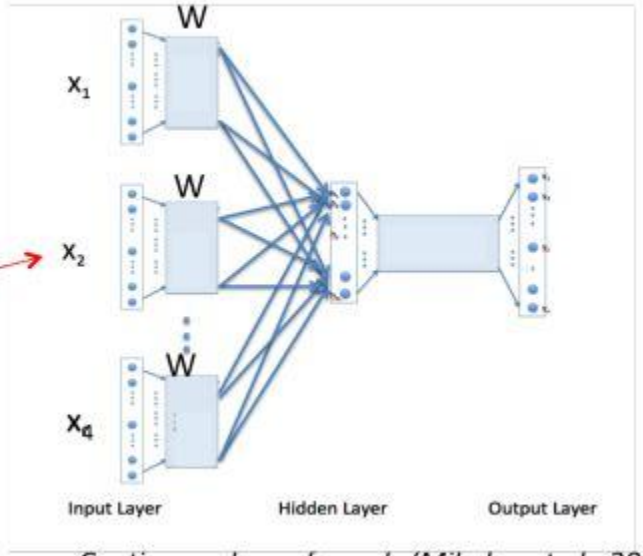
$$W = \begin{bmatrix} 1 & 2 & 3 & 0 \\ 1 & 2 & 1 & 2 \\ -1 & 1 & 1 & 1 \end{bmatrix}$$

Ex:

$W^{drink} = [0,1,0,0]$

$$Wx_2 = v_2$$

$$\begin{bmatrix} 1 & 2 & 3 & 0 \\ 1 & 2 & 1 & 2 \\ -1 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 2 \\ 2 \\ 1 \end{bmatrix}$$



Continuous bag-of-words (Mikolov et al., 2013)

具体例子

An example of CBOW Model

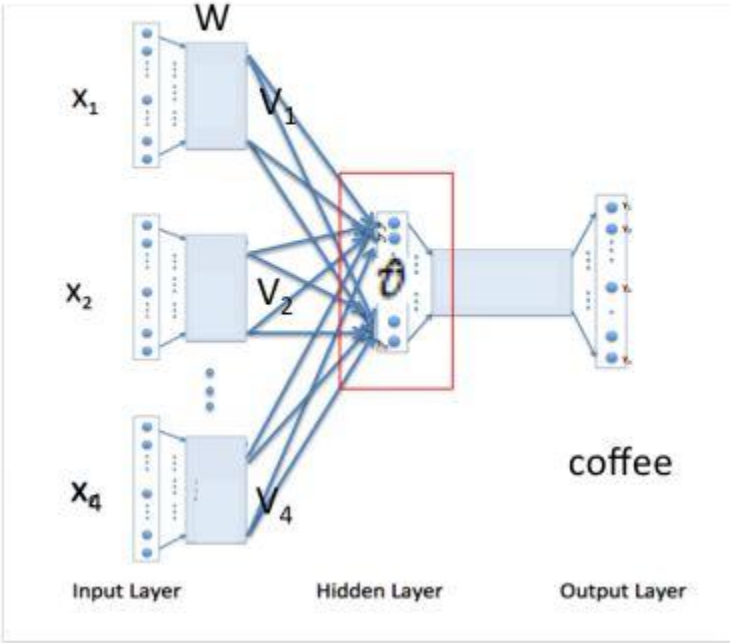
Corpus = { I drink coffee everyday }

Initialize:

$$W = \begin{bmatrix} 1 & 2 & 3 & 0 \\ 1 & 2 & 1 & 2 \\ -1 & 1 & 1 & 1 \end{bmatrix}$$

$$\frac{V_1 + V_2 + V_4}{3} = \hat{v}$$

$$\frac{1}{3}\left(\begin{bmatrix} 1 \\ 1 \\ -1 \end{bmatrix} + \begin{bmatrix} 2 \\ 2 \\ 1 \end{bmatrix} + \begin{bmatrix} 0 \\ 2 \\ 1 \end{bmatrix}\right) = \begin{bmatrix} 1 \\ 1.67 \\ 0.33 \end{bmatrix}$$



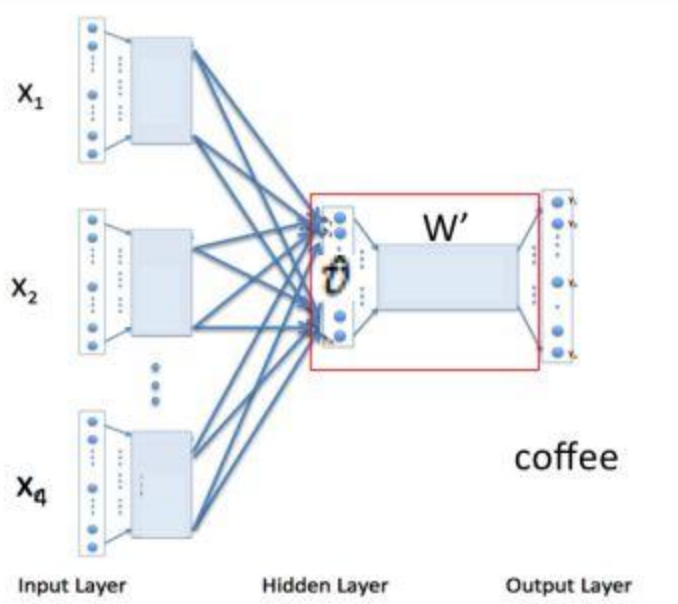
具体例子

An example of CBOW Model

Corpus = { I drink coffee everyday }

Initialize:

$$W' = \begin{bmatrix} 1 & 2 & -1 \\ -1 & 2 & -1 \\ 1 & 2 & 2 \\ 0 & 2 & 0 \end{bmatrix}$$



$$W'\hat{v} = U_0$$

$$\begin{bmatrix} 1 & 2 & -1 \\ -1 & 2 & -1 \\ 1 & 2 & 2 \\ 0 & 2 & 0 \end{bmatrix} \begin{bmatrix} 1.00 \\ 1.67 \\ 0.33 \end{bmatrix} = \begin{bmatrix} 4.01 \\ 2.01 \\ 5.00 \\ 3.34 \end{bmatrix} \begin{matrix} u_1 \\ u_2 \\ u_c \\ u_4 \end{matrix}$$

具体例子

An example of CBOW Model

Output: Probability distribution

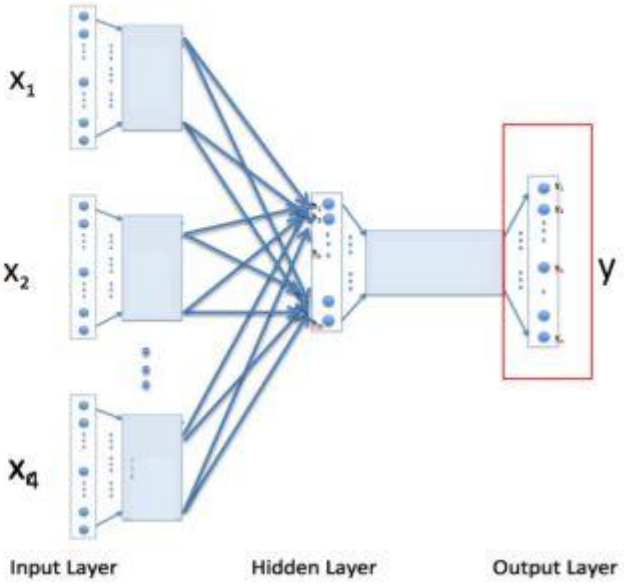
$$softmax(\mathbf{u}_o) = \mathbf{y}$$

$$softmax\left(\begin{bmatrix} 4.01 \\ 2.01 \\ 5.00 \\ 3.34 \end{bmatrix}\right) = \begin{bmatrix} 0.23 \\ 0.03 \\ 0.62 \\ 0.12 \end{bmatrix}$$

Probability of “coffee”

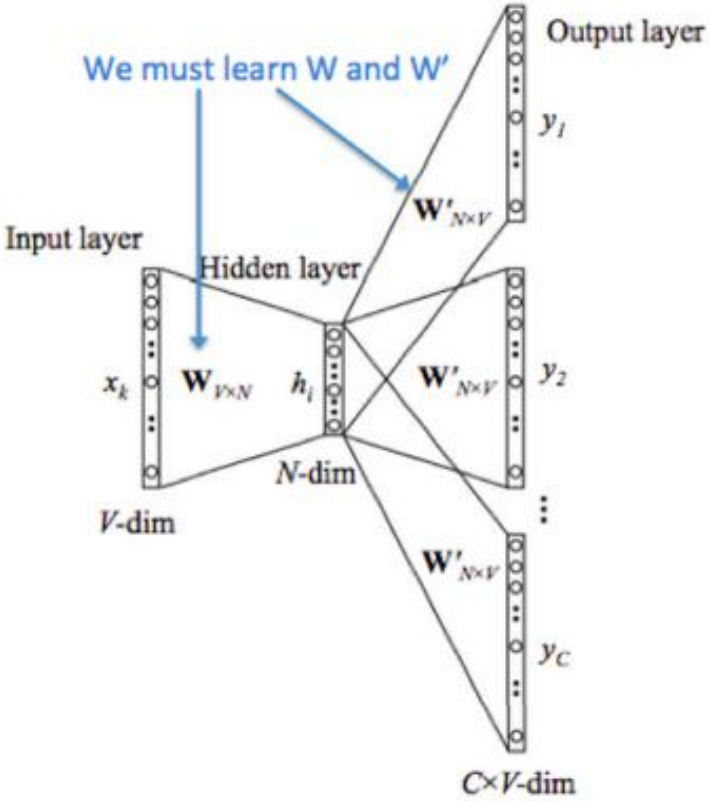
We desire probability generated to match the true probability(label) x_3 [0,0,1,0]

Use gradient descent to update W and W’



Skip-Gram

Skip-gram逆转了CBOW的因果关系，模型输入层不再是多个词向量，而是只有一个词向量；需要用当前词预测上下文窗口中的每一个词

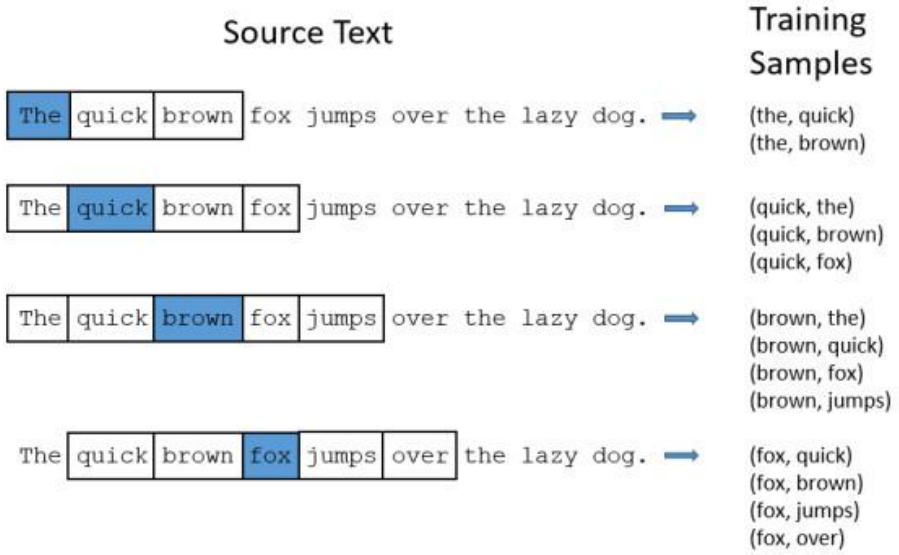


Notation for Skip-Gram Model:

- w_i: Word i from vocabulary V
- $\mathcal{V} \in \mathbb{R}^{n\times|V|}$: Input word matrix
- v_i: i-th column of $\mathcal{V}$, the input vector representation of word w_i
- $\mathcal{U} \in \mathbb{R}^{n\times|V|}$: Output word matrix
- u_i: i-th row of $\mathcal{U}$, the output vector representation of word w_i

对高频词抽样

对于句子The quick brown fox jumps over the laze dog，如果使用大小为2的窗口，那么可以得到以下训练样本。



经常出现的词语，如“我的”、“你的”和“他的”，无法给附近的单词提供太多的上下文信息。如果我们放弃其中的一些单词，我们就可以从我们的数据中移除一些噪声noise，以得到更快的训练和更好的表现

由于在文本中 the这样的常用词出现概率很大，因此我们将会有大量的（the，...）这样的训练样本，而这些样本数量远远超过了我们学习the这个词向量所需的训练样本数

成本/目标函数

优化（最小化）目标/成本函数：梯度下降法

例子：求函数的最小值

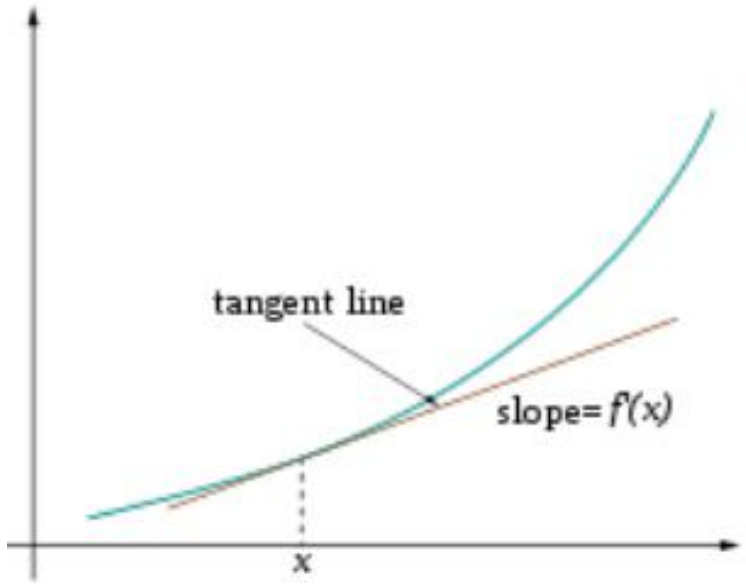
$f(x) = x^4-3x^3+2$，导数 $f'(x) = 4x^3-9x^2$

```
x_old = 0
x_new = 6 # The algorithm starts at x=6
eps = 0.01 # step size
precision = 0.00001

def f_derivative(x):
    return 4 * x**3 - 9 * x**2

while abs(x_new - x_old) > precision:
    x_old = x_new
    x_new = x_old - eps * f_derivative(x_old)

print("Local minimum occurs at", x_new)
```



不断减去一小部分的梯度趋向到最小值

梯度下降

- 要在所有训练数据下内最小化 J(θ)，需要我们计算所有窗口的梯度
- 更新将针对 θ 的每个元素： $\theta_j^{new} = \theta_j^{old} - \alpha \frac{\partial}{\partial \theta_j^{old}} J(\theta)$
- 步长α的选择
- 所有参数的矩阵表示法：

$$\theta^{new} = \theta^{old} - \alpha \frac{\partial}{\partial \theta^{old}} J(\theta)$$

$$\theta^{new} = \theta^{old} - \alpha \nabla_\theta J(\theta)$$

```
while True:
    theta_grad = evaluate_gradient(J,corpus,theta)
    theta = theta - alpha * theta_grad
```

随机梯度下降（SGD）

- 上述方法在语料库窗口很大时，更新耗时太长
- 因此选择SGD，在每个窗口 t 之后更新参数

$$\theta^{new} = \theta^{old} - \alpha \nabla_\theta J_t(\theta)$$

```
while True:
    window = sample_window(corpus)
    theta_grad = evaluate_gradient(J,window,theta)
    theta = theta - alpha * theta_grad
```

Thank you!