3.1 词向量

林洲汉
上海交大电院
2024年秋季学期

目录

目录

词的表示

狗

狗：哺乳动物，种类很多，听觉嗅觉都很敏锐，善于看守门户。

这些表示不适合机器学习模型/神经网络直接使用

基础版：词的单热点表示

例句：我爱自然语言处理

我	爱	自	然	语	言	处	理
1	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0
0	0	1	0	0	0	0	0
0	0	0	1	0	0	0	0
0	0	0	0	1	0	0	0
0	0	0	0	0	1	0	0
0	0	0	0	0	0	1	0
0	0	0	0	0	0	0	1

我	爱	自然语言处理
1	0	0
0	1	0
0	0	1
0	0	0
0	0	0

只是机械地记下了词，
没法表达词的语义
没法表达词与词之间的远近亲疏关系

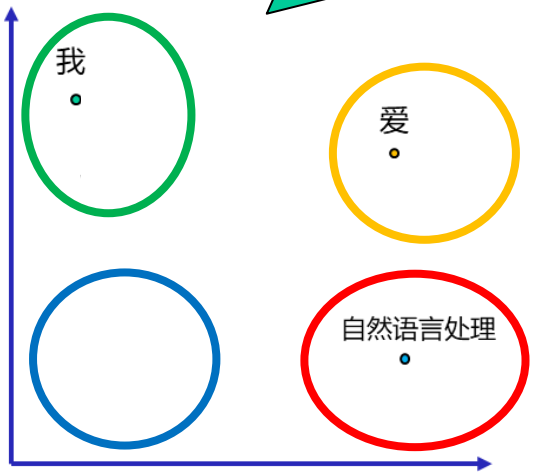
进阶版：词的分布式表示（词向量）

例句：

我们把像这样使用高维连续空间中的向量来表示单词的方法，称为**词的分布式表示**，这样的向量称为**词向量**。

我　爱　自然语言处理

0.94	0.25	0.54
0.45	0.35	0.20
0.25	0.54	0.93
0.02	0.19	0.13
0.30	0.93	0.37



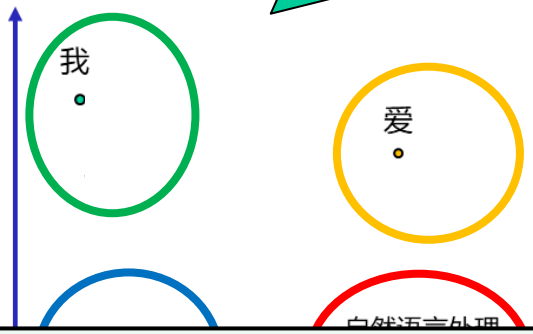
将单词用高维连续空间中的向量表示
用向量在高维空间中所处的位置代表词的语义
用向量间的远近关系表达词之间语义的远近亲疏

进阶版：词的分布式表示（词向量）

例句：

我们把像这样使用高维连续空间中的向量来表示单词的方法，称为**词的分布式表示**，这样的向量称为**词向量**。

我	爱	自然语言处理
0.94	0.25	0.54
0.45	0.35	0.20
0.25	0.54	0.93



词的连续向量表示为什么又称作“分布式表征（distributed representation）”？

单热点向量的表示方式中，每个词对应于特定的维度，这个词由且仅由这一维度表示，因此也被称为“局部语义表达”或“非分布式表达”。

与之相对的，词向量采用连续稠密向量的表示方式，词的语义分散在不同的维度中表达，因此被称为“分布式表征”。

目录

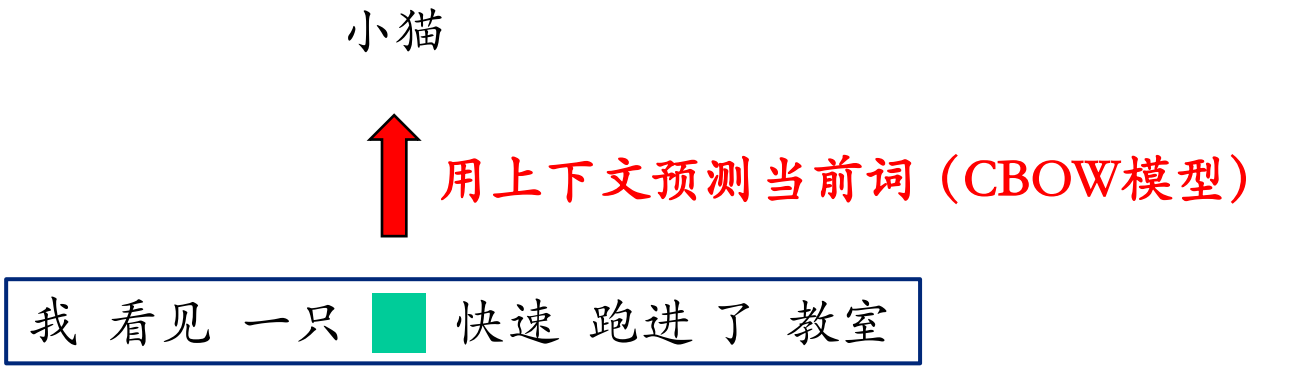
语义，从哪里来?

sinewy：

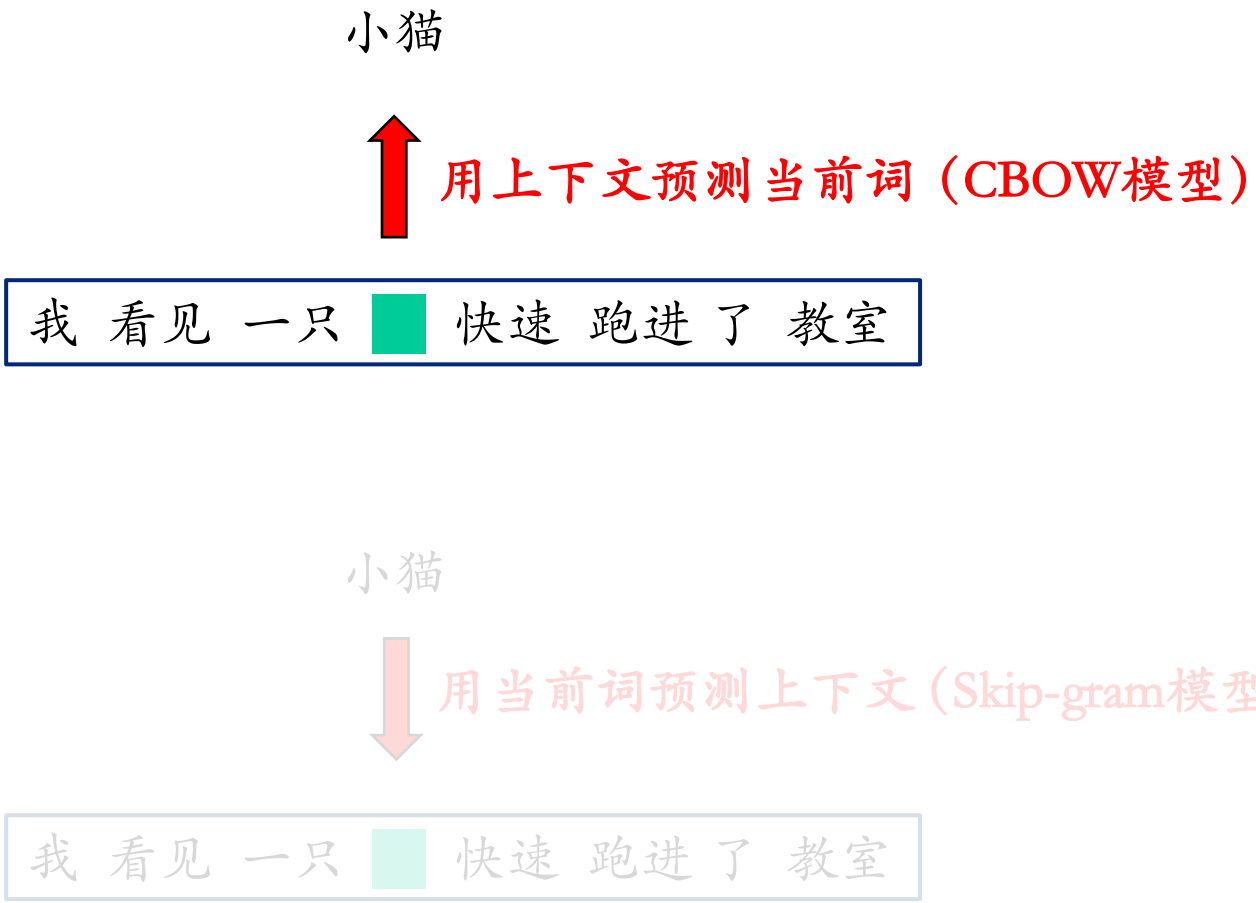
“You shall know a word by the company it keeps.”

——John Rupert Firth
英国语言学家
伦敦学派创始人

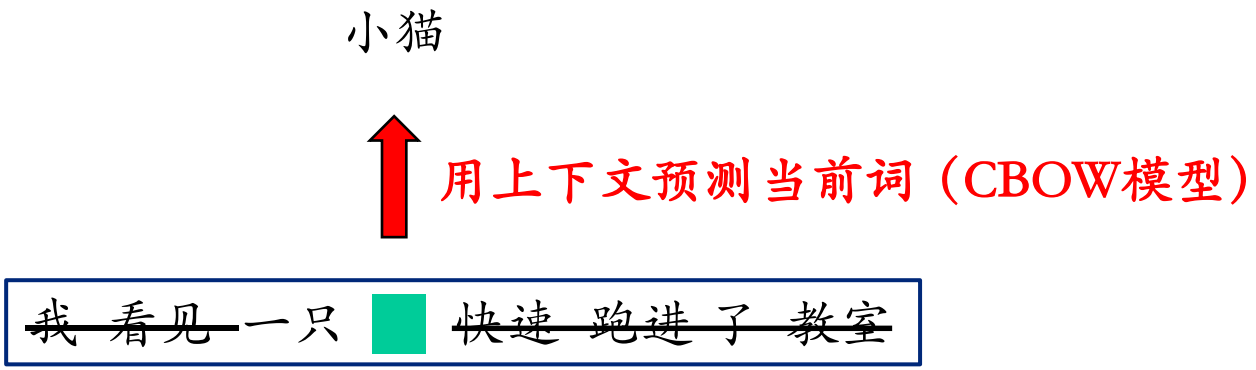
两种不同的词向量模型


小猫
用上下文预测当前词（CBOW模型）
我 看见 一只 快速 跑进 了 教室
小猫
用当前词预测上下文（Skip-gram模型）
我 看见 一只 快速 跑进 了 教室

CBOW（Continuous Bag-of-words）模型

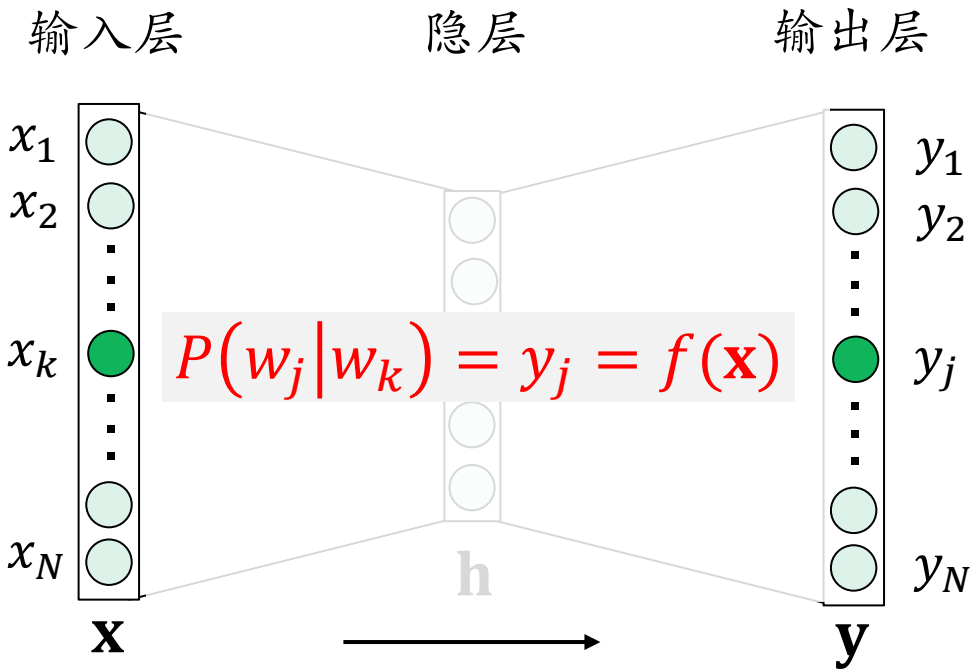


CBOW（Continuous Bag-of-words）模型



为简化表达，我们接下来首先讲解
上下文只包含了一个词的CBOW模型

CBOW（Continuous Bag-of-words）模型



- 输入$\mathbf{x}$为独热向量（列向量）

$$\mathbf{x} = [x_1, x_2, \dots, x_k, \dots, x_N]^T = [0,0, \dots, 1, \dots, 0]^T$$

- 输出$\mathbf{y}$为概率分布（列向量），满足

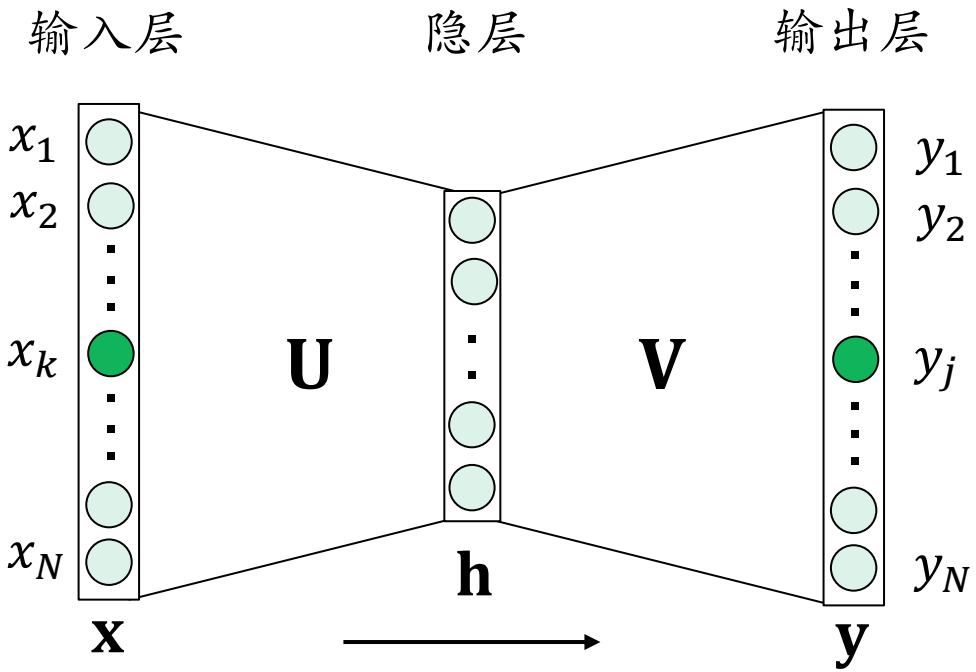
$$\forall t, y_t > 0, \text{且} \sum_{t=1}^{N} y_t = 1$$

- $\mathbf{h} \in \mathbb{R}^{H\times 1}$为隐层表征
- $\mathbf{U} \in \mathbb{R}^{H\times N}$和$\mathbf{V} \in \mathbb{R}^{N\times H}$为两个参数矩阵，且

$$\mathbf{V} = \begin{bmatrix} \mathbf{v}_1^{\mathrm{T}} \\ \mathbf{v}_2^{\mathrm{T}} \\ \vdots \\ \mathbf{v}_N^{\mathrm{T}} \end{bmatrix}$$

其中$\mathbf{v}_i \in \mathbb{R}^{H\times 1}$

CBOW（Continuous Bag-of-words）模型

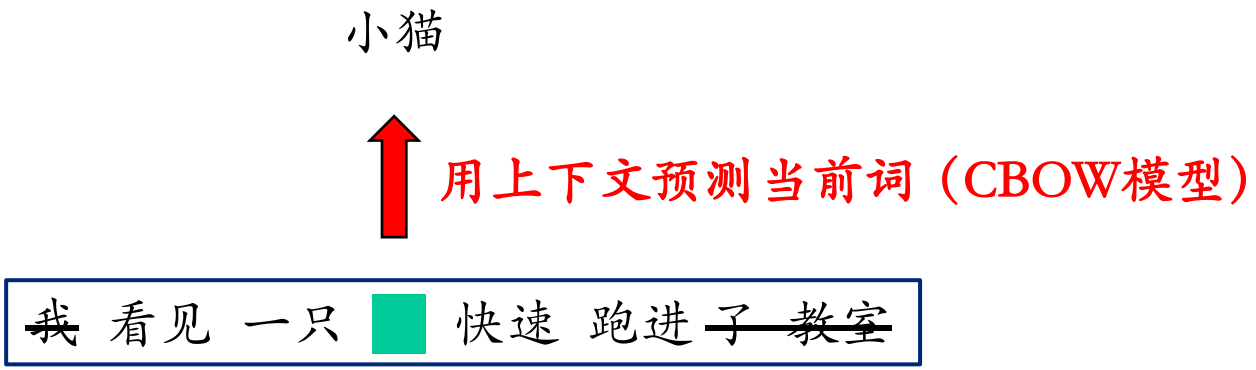


- 隐层向量 $\mathbf{h} = \mathbf{U}\mathbf{x}$
- 输出向量 $\mathbf{o} = \mathbf{V}\mathbf{h}$
- 输出向量经过概率归一化之后得到的概率分布向量

$$\mathbf{y} = \text{Softmax}(\boldsymbol{o}) \qquad y_j = \frac{\exp(o_j)}{\sum_{t=1}^{N} \exp(o_t)} \quad \text{Softmax}$$

- 我们使得模型预测的概率y_j在真实的中间词上尽可能大，别的词上尽可能小

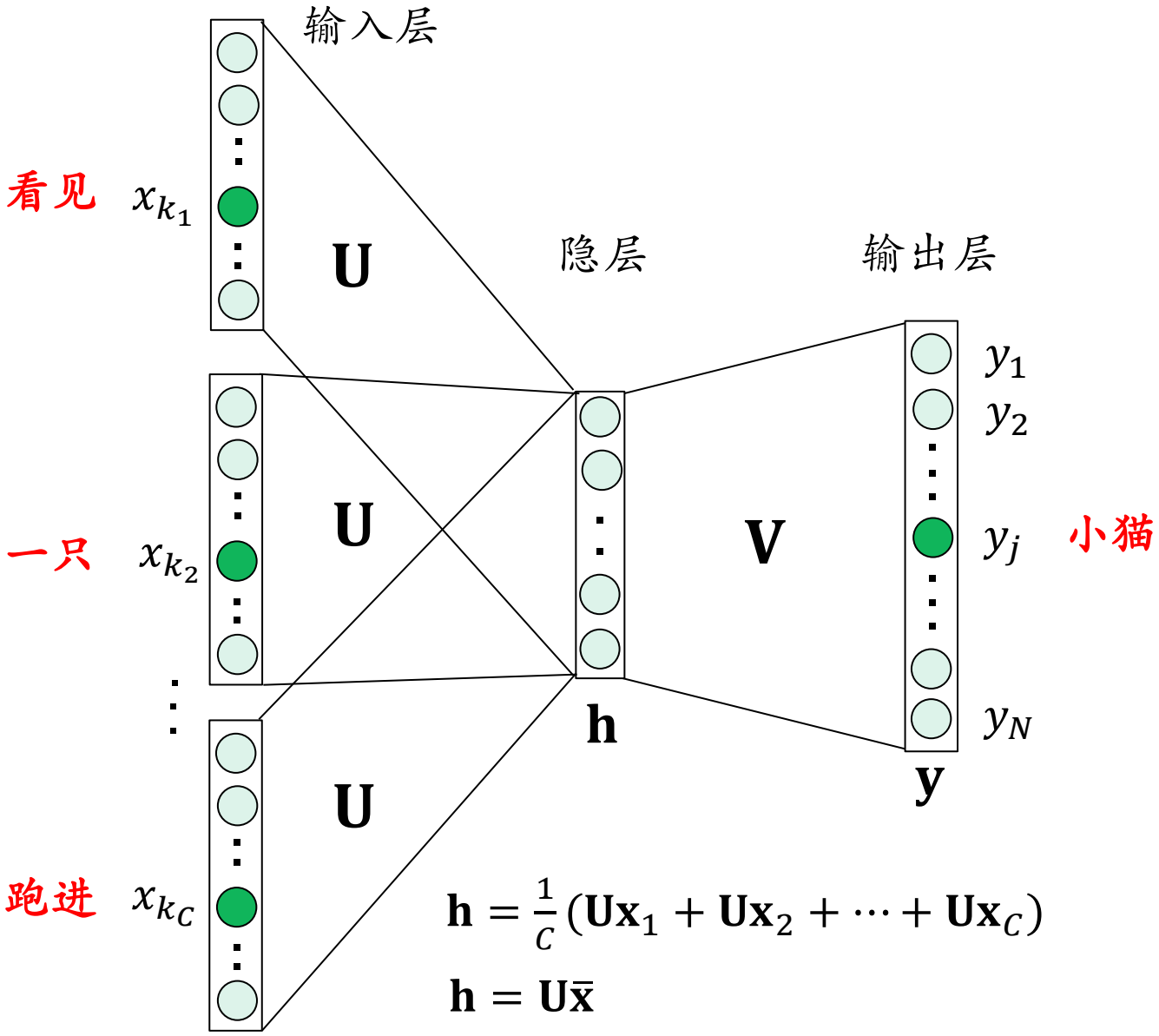
CBOW模型：考虑更多的上下文



对于更多的上下文，

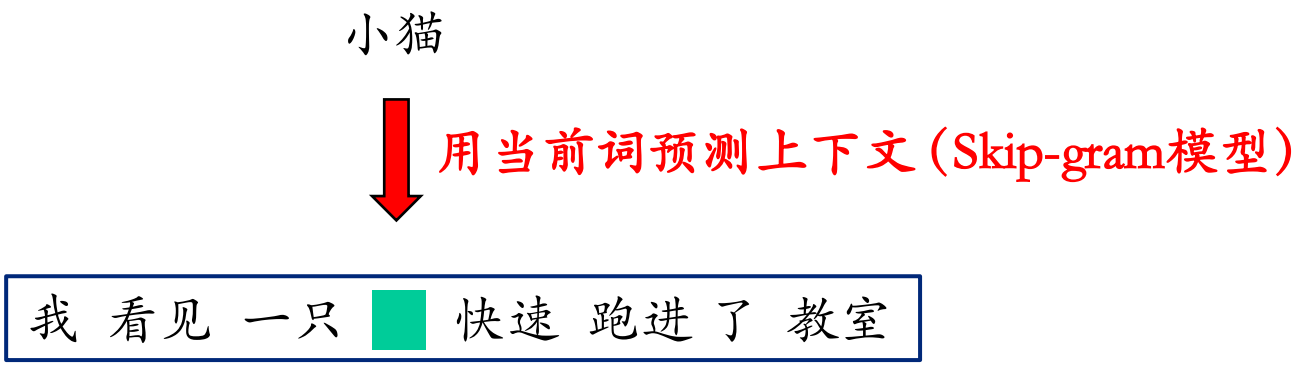
CBOW模型中的 **h** 取每个上下文单词所对应的 **h** 的平均

CBOW模型：考虑更多的上下文

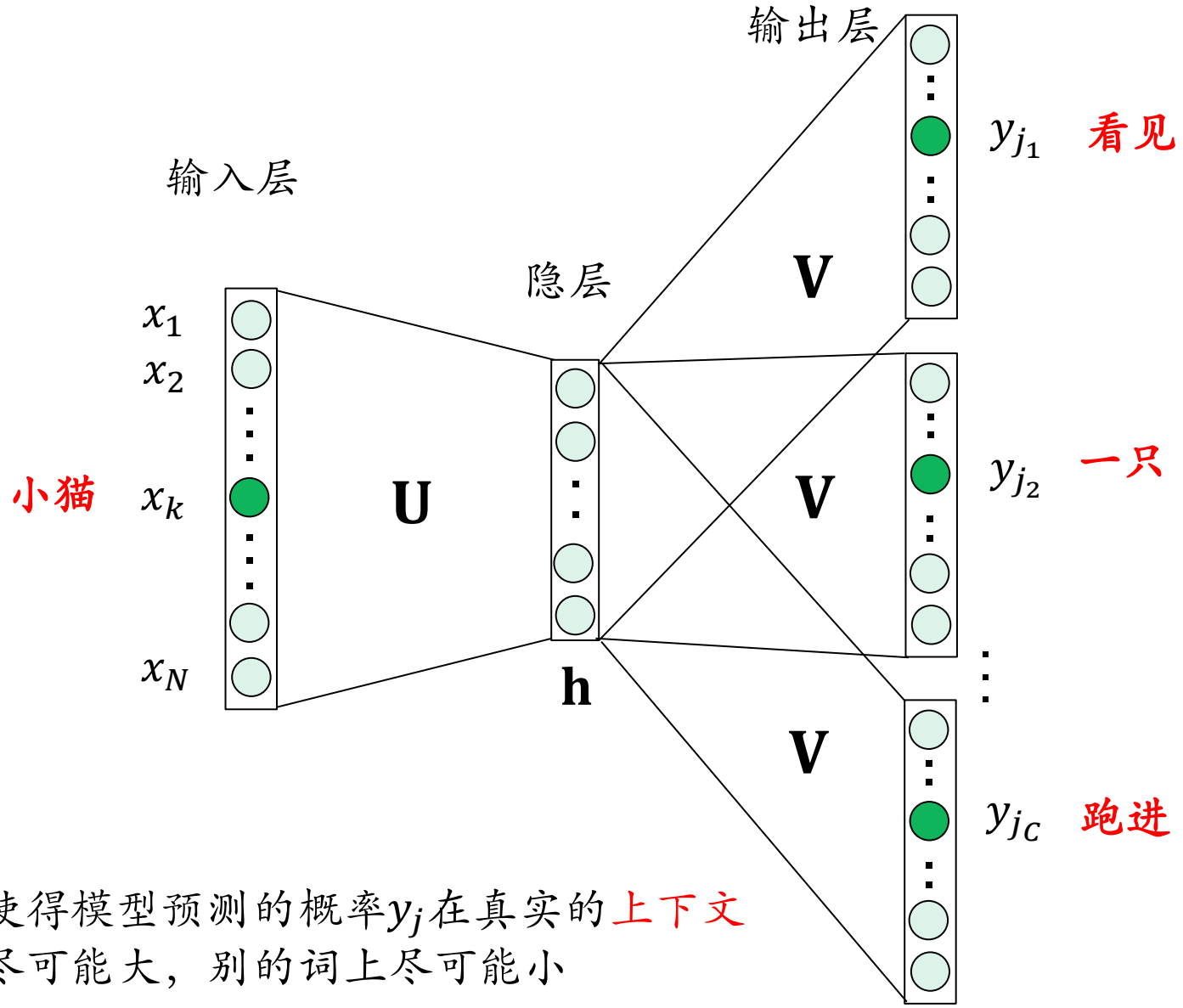


Skip-gram模型

小猫
用上下文预测当前词（CBOW模型）
我 看见 一只 快速 跑进 了 教室


小猫
用当前词预测上下文（Skip-gram模型）
我 看见 一只 快速 跑进 了 教室

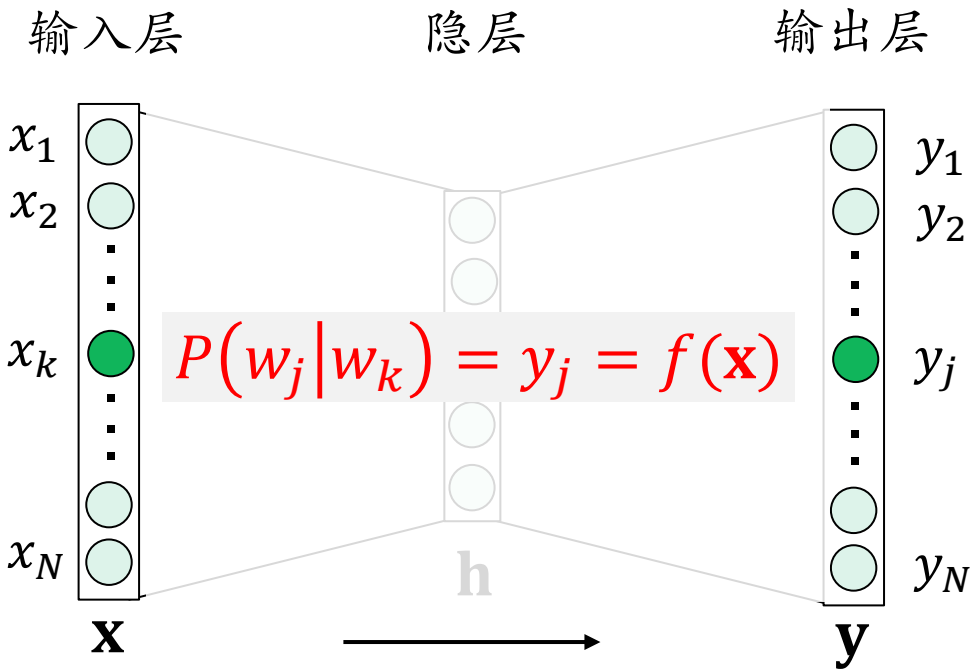
Skip-gram模型



我们使得模型预测的概率y_j在真实的上下文词上尽可能大，别的词上尽可能小

目录

反向传播算法（Back-propagation）



- 输入$\mathbf{x}$为独热向量（列向量）

$$\mathbf{x} = [x_1, x_2, \dots, x_k, \dots, x_N]^T = [0,0, \dots, 1, \dots, 0]^T$$

- 输出$\mathbf{y}$为概率分布（列向量），满足

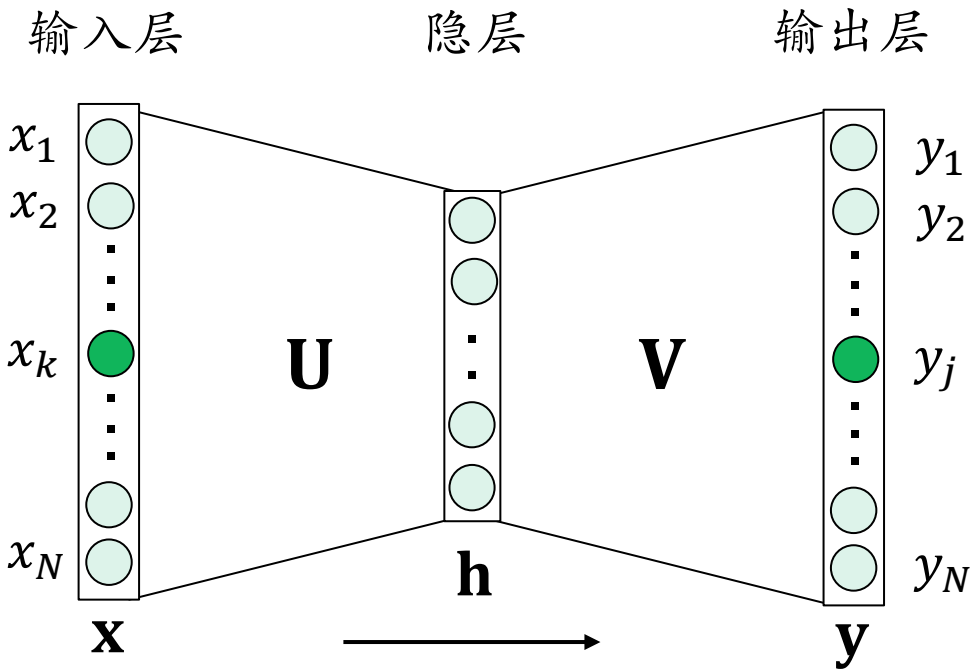
$$\forall t, y_t > 0, \text{且} \sum_{t=1}^{N} y_t = 1$$

- $\mathbf{h} \in \mathbb{R}^{H \times 1}$为隐层表征
- $\mathbf{U} \in \mathbb{R}^{H \times N}$和$\mathbf{V} \in \mathbb{R}^{N \times H}$为两个参数矩阵，且

$$\mathbf{V} = \begin{bmatrix} \mathbf{v}_1^T \\ \mathbf{v}_2^T \\ \vdots \\ \mathbf{v}_N^T \end{bmatrix}$$

其中$\mathbf{v}_i \in \mathbb{R}^{H \times 1}$

反向传播算法（Back-propagation）



$$\mathbf{y} = \text{Softmax}(\mathbf{o}) = \text{Softmax}(\mathbf{Vh}) = \text{Softmax}(\mathbf{VUx})$$

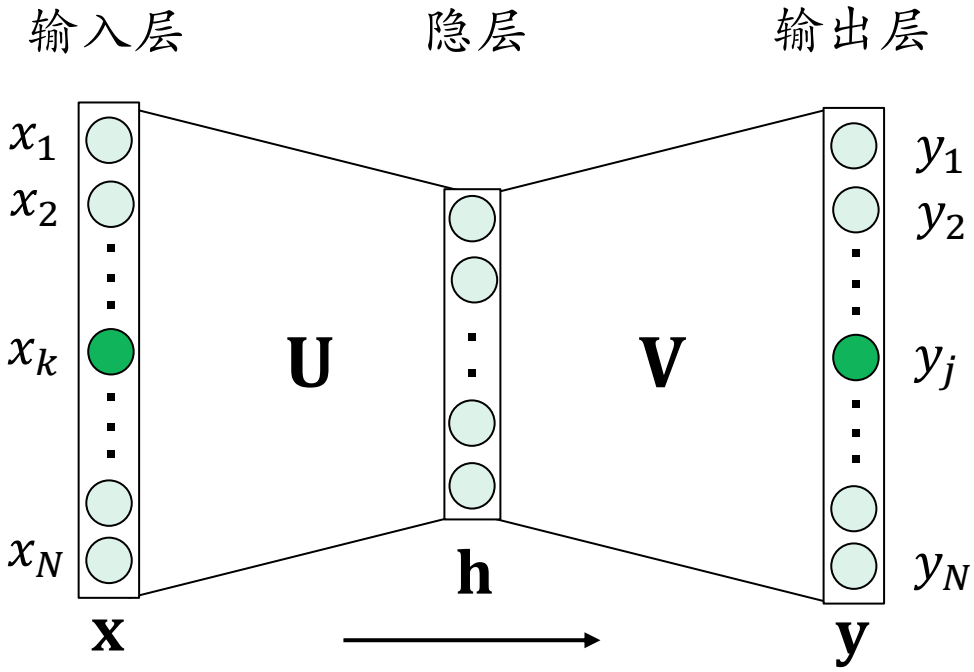
$$\frac{\partial \mathrm{E}}{\partial \mathbf{V}} = ?$$

$$\mathrm{E} = -\log y_j$$

$$\frac{\partial \mathrm{E}}{\partial \mathbf{U}} = ?$$

反向传播算法（Back-propagation）

$$\mathbf{y} = \text{Softmax}(\mathbf{o})$$
$$= \text{Softmax}(\mathbf{Vh})$$
$$= \text{Softmax}(\mathbf{VUx})$$



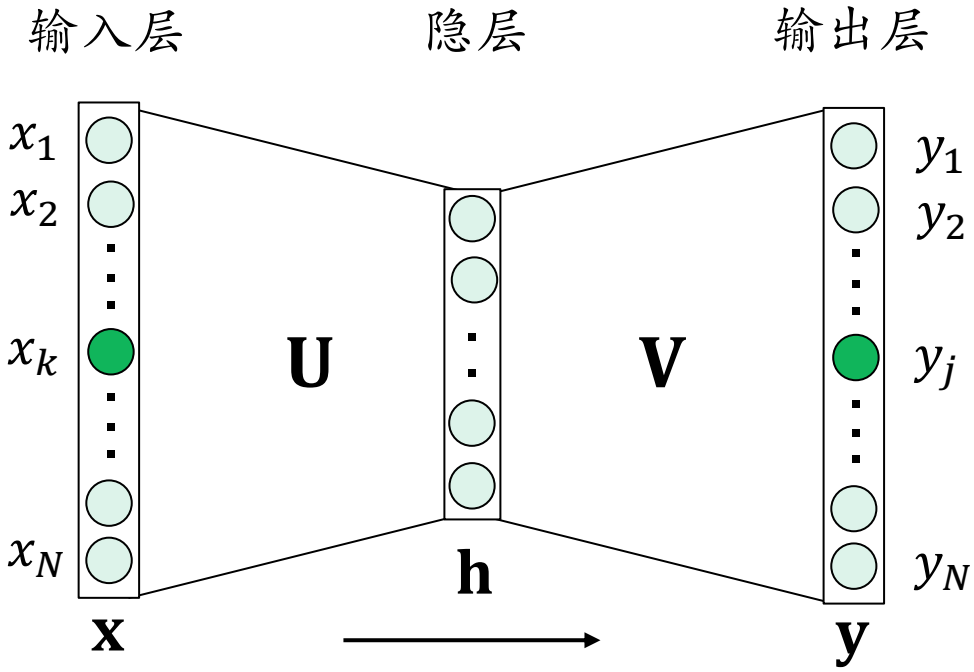
$$\mathrm{E} = -\log y_j$$
$$= -\log \frac{\exp(o_j)}{\sum_{t=1}^{\mathrm{N}} \exp(o_t)}$$
$$= -o_j + \log \sum_{t=1}^{N} \exp(o_t)$$
$$= -\mathbf{v_j}^{\mathrm{T}}\mathbf{Ux} + \log \sum_{t=1}^{N} \exp(\mathbf{v_t}^{\mathrm{T}}\mathbf{Ux})$$

$$\frac{\partial \mathrm{E}}{\partial \mathbf{V}} = \begin{cases} \frac{\partial \mathrm{E}}{\partial \mathbf{v_i}} = -\mathbf{Ux} + \frac{1}{\sum_{t=1}^{N} \exp(\mathbf{v_t}\mathbf{U}^{\mathrm{T}}\mathbf{x})} \exp(\mathbf{v_i}^{\mathrm{T}}\mathbf{Ux})\mathbf{Ux} & (\mathrm{i} = \mathrm{j}) \\ \frac{\partial \mathrm{E}}{\partial \mathbf{v_i}} = \frac{1}{\sum_{t=1}^{N} \exp(\mathbf{v_t}\mathbf{Ux})} \exp(\mathbf{v_i}^{\mathrm{T}}\mathbf{Ux})\mathbf{Ux} & (\mathrm{i} \neq \mathrm{j}) \end{cases}$$

y_i 和 $\mathbf{h}$

反向传播算法（Back-propagation）

$$\mathbf{y} = \text{Softmax}(\mathbf{o})$$
$$= \text{Softmax}(\mathbf{Vh})$$
$$= \text{Softmax}(\mathbf{VUx})$$



$$\mathrm{E} = -\log y_j$$
$$= -\log \frac{\exp(o_j)}{\sum_{t=1}^{\mathrm{N}} \exp(o_t)}$$
$$= -o_j + \log \sum_{t=1}^{N} \exp(o_t)$$
$$= -\mathbf{v_j}^{\mathrm{T}}\mathbf{Ux} + \log \sum_{t=1}^{N} \exp(\mathbf{v_t}^{\mathrm{T}}\mathbf{Ux})$$

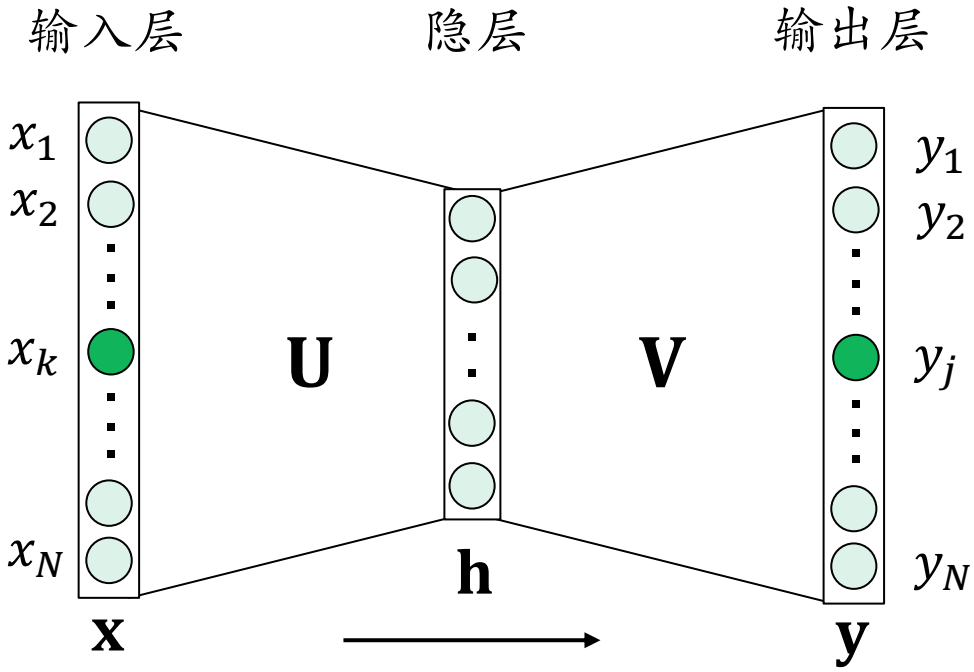
$$\frac{\partial \mathrm{E}}{\partial \mathbf{V}} = \begin{cases} \frac{\partial \mathrm{E}}{\partial \mathbf{v}_\mathrm{i}} = -\mathbf{Ux} + \frac{1}{\sum_{t=1}^{N} \exp(\mathbf{v_t}\mathbf{U}^{\mathrm{T}}\mathbf{x})} \exp(\mathbf{v}_\mathrm{i}^{\mathrm{T}}\mathbf{Ux})\mathbf{Ux} & = -\mathbf{h} + y_i\mathbf{h}\ (\mathrm{i} = \mathrm{j}) \\ \frac{\partial \mathrm{E}}{\partial \mathbf{v}_\mathrm{i}} = \frac{1}{\sum_{t=1}^{N} \exp(\mathbf{v_t}\mathbf{U}^{\mathrm{T}}\mathbf{x})} \exp(\mathbf{v}_\mathrm{i}^{\mathrm{T}}\mathbf{Ux})\mathbf{Ux} & = y_i\mathbf{h}\ (\mathrm{i} \neq \mathrm{j}) \end{cases}$$

y_i 和 $\mathbf{h}$

反向传播算法（Back-propagation）

$$\mathbf{y} = \text{Softmax}(\mathbf{o})$$
$$= \text{Softmax}(\mathbf{Vh})$$
$$= \text{Softmax}(\mathbf{VUx})$$

$$\mathrm{E} = -\log y_j$$
$$= -\log \frac{\exp(o_j)}{\sum_{t=1}^{\mathrm{N}} \exp(o_t)}$$
$$= -o_j + \log \sum_{t=1}^{N} \exp(o_t)$$
$$= -\mathbf{v_j}^{\mathrm{T}}\mathbf{Ux} + \log \sum_{t=1}^{N} \exp(\mathbf{v_t}^{\mathrm{T}}\mathbf{Ux})$$



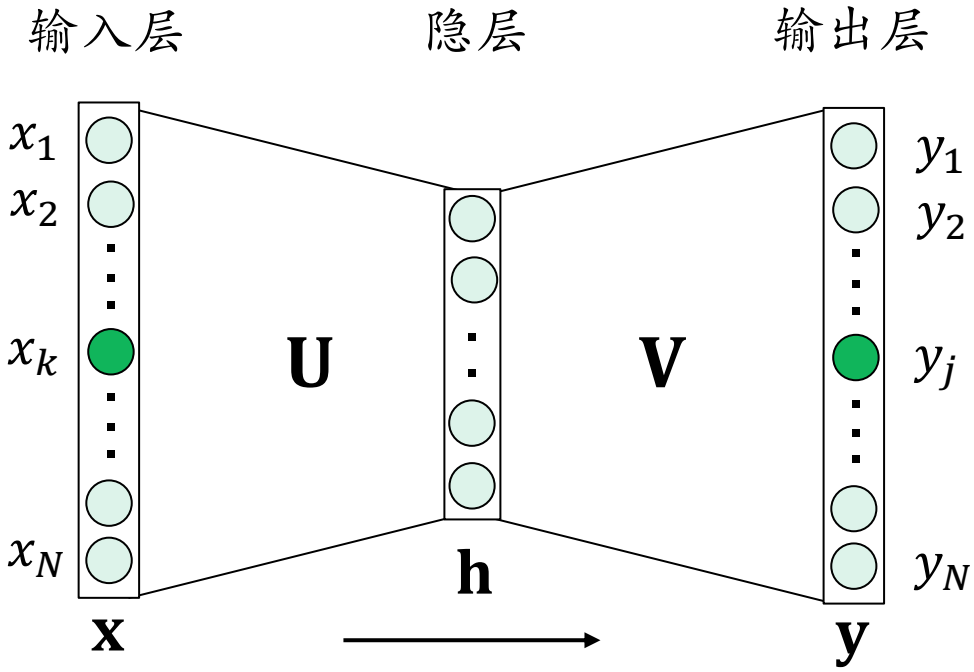
$$\frac{\partial \mathrm{E}}{\partial \mathbf{V}} = \begin{cases} \dfrac{\partial \mathrm{E}}{\partial \mathbf{v}_\mathrm{i}} = -\mathbf{h} + y_i\mathbf{h} \\ \dfrac{\partial \mathrm{E}}{\partial \mathbf{v}_\mathrm{i}} = y_i\mathbf{h} \end{cases}$$

$$\mathbf{e} \in \mathbb{R}^{N\times 1} = \begin{cases} y_i - 1 & (\mathrm{i} = \mathrm{j}) \\ y_i & (\mathrm{i} \neq \mathrm{j}) \end{cases}$$

反向传播算法（Back-propagation）

$$\mathbf{y} = \text{Softmax}(\mathbf{o})$$
$$= \text{Softmax}(\mathbf{Vh})$$
$$= \text{Softmax}(\mathbf{VUx})$$

$$\text{E} = -\log y_j$$
$$= -\log \frac{\exp(o_j)}{\sum_{t=1}^{\text{N}} \exp(o_t)}$$
$$= -o_j + \log \sum_{t=1}^{N} \exp(o_t)$$
$$= -\mathbf{v_j}^{\text{T}}\mathbf{Ux} + \log \sum_{t=1}^{N} \exp(\mathbf{v_t}^{\text{T}}\mathbf{Ux})$$

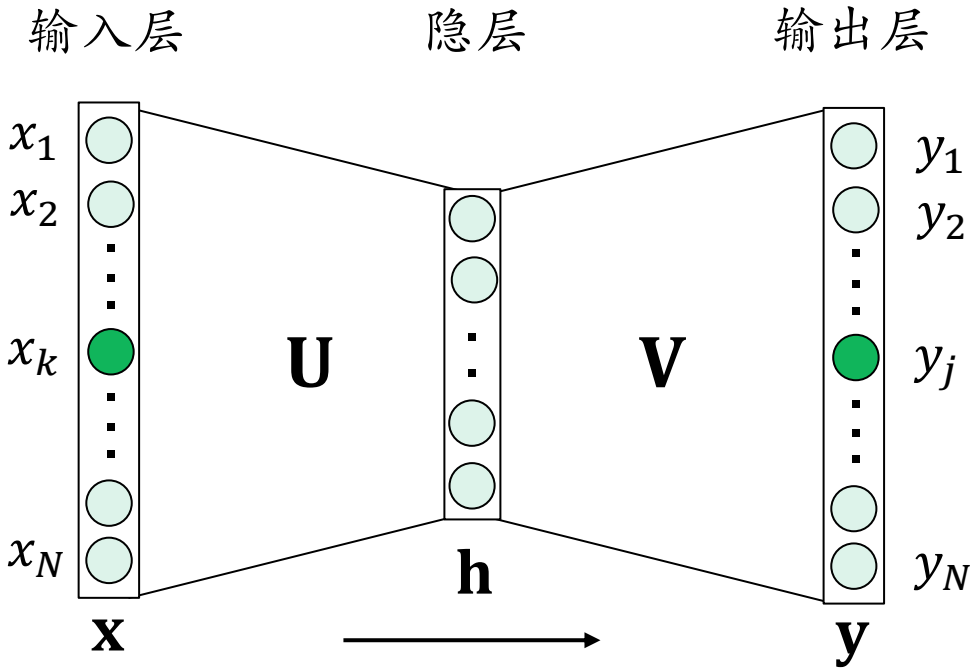


$$\frac{\partial \text{E}}{\partial \mathbf{V}} = \mathbf{e}\mathbf{h}^{\text{T}}$$

$$\mathbf{e} \in \mathbb{R}^{N\times 1} = \begin{cases} y_i - 1 & (\text{i} = \text{j}) \\ y_i & (\text{i} \neq \text{j}) \end{cases}$$

反向传播算法（Back-propagation）

$$\mathbf{y} = \text{Softmax}(\mathbf{o})$$
$$= \text{Softmax}(\mathbf{Vh})$$
$$= \text{Softmax}(\mathbf{VUx})$$



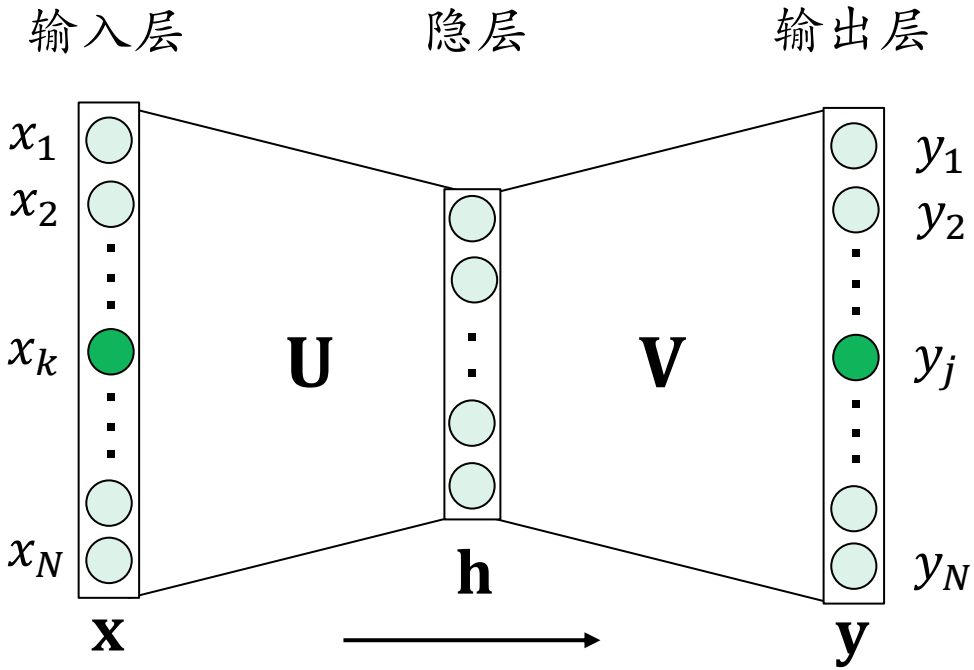
$$\mathrm{E} = -\log y_j$$
$$= -\log \frac{\exp(o_j)}{\sum_{t=1}^{\mathrm{N}} \exp(o_t)}$$
$$= -o_j + \log \sum_{t=1}^{N} \exp(o_t)$$
$$= -\mathbf{v_j}^{\mathrm{T}}\mathbf{Ux} + \log \sum_{t=1}^{N} \exp(\mathbf{v_t}^{\mathrm{T}}\mathbf{Ux})$$

$$\frac{\partial \mathrm{E}}{\partial \mathbf{U}} = -\mathbf{x}\mathbf{v_j}^{\mathrm{T}} + \frac{1}{\sum_{t=1}^{N} \exp(\mathbf{v_t}\mathbf{Ux})} \sum_{t=1}^{N} \exp(\mathbf{v_t}\mathbf{Ux})\mathbf{x}\mathbf{v_t}^{\mathrm{T}}$$
$$= \mathbf{x}\left(\frac{1}{\sum_{t=1}^{N} \exp(\mathbf{v_t}\mathbf{Ux})} \sum_{t=1}^{N} \exp(\mathbf{v_t}\mathbf{Ux})\, \mathbf{v_t}^{\mathrm{T}} - \mathbf{v_j}^{\mathrm{T}}\right) = \mathbf{V}^{\mathrm{T}}\mathbf{e}\mathbf{x}^{\mathrm{T}}$$

反向传播算法（Back-propagation）

$$\mathbf{y} = \text{Softmax}(\mathbf{o})$$
$$= \text{Softmax}(\mathbf{Vh})$$
$$= \text{Softmax}(\mathbf{VUx})$$

$$\mathrm{E} = -\log y_j$$
$$= -\log \frac{\exp(o_j)}{\sum_{t=1}^{\mathrm{N}} \exp(o_t)}$$
$$= -o_j + \log \sum_{t=1}^{N} \exp(o_t)$$
$$= -\mathbf{v_j}^{\mathrm{T}}\mathbf{Ux} + \log \sum_{t=1}^{N} \exp(\mathbf{v_t}^{\mathrm{T}}\mathbf{Ux})$$



$$\begin{cases} \dfrac{\partial \mathrm{E}}{\partial \mathbf{V}} = \mathbf{e}\mathbf{h}^{\mathrm{T}} \\ \dfrac{\partial \mathrm{E}}{\partial \mathbf{U}} = \mathbf{V}^{\mathrm{T}}\mathbf{e}\mathbf{x}^{\mathrm{T}} \end{cases}$$

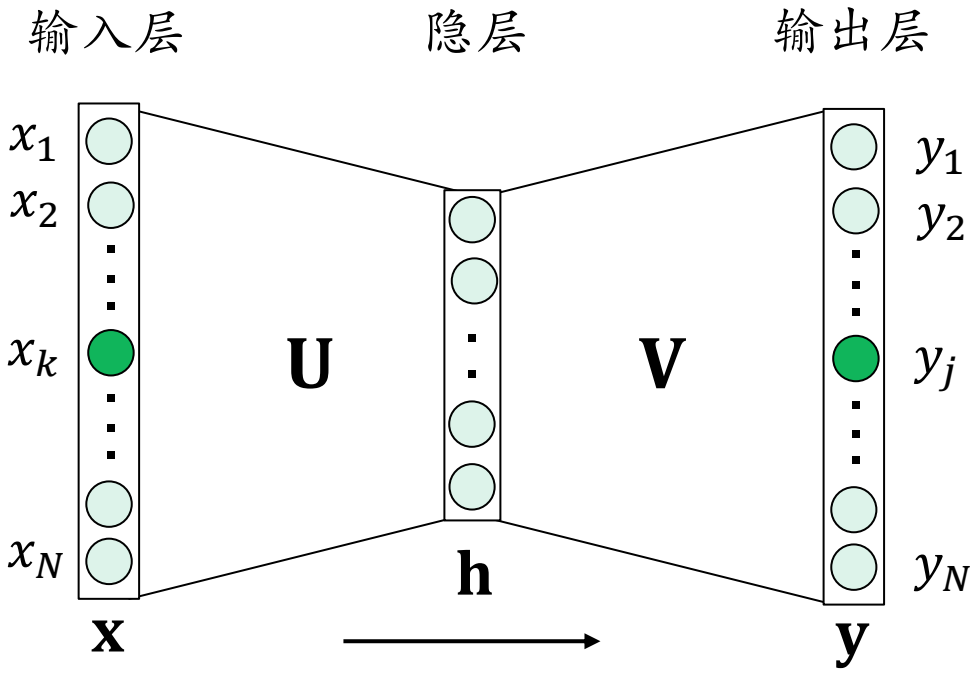
设置学习率η

$$\begin{cases} \mathbf{V} = \mathbf{V} - \eta \dfrac{\partial \mathrm{E}}{\partial \mathbf{V}} = \mathbf{V} - \eta\mathbf{e}\mathbf{h}^{\mathrm{T}} \\ \mathbf{U} = \mathbf{U} - \eta \dfrac{\partial \mathrm{E}}{\partial \mathbf{U}} = \mathbf{U} - \eta\mathbf{V}^{\mathrm{T}}\mathbf{e}\mathbf{x}^{\mathrm{T}} \end{cases}$$

反向传播算法（Back-propagation）

现在问题来了：
明明是求导，为什么叫他“反向传播”呢？

反向传播，
指的是预测误差（**e**）的反向传播。



$$\begin{cases} \dfrac{\partial \mathrm{E}}{\partial \mathbf{V}} = \mathbf{e}\mathbf{h}^{\mathrm{T}} \\ \dfrac{\partial \mathrm{E}}{\partial \mathbf{U}} = \mathbf{V}^{\mathrm{T}}\mathbf{e}\mathbf{x}^{\mathrm{T}} \end{cases}$$

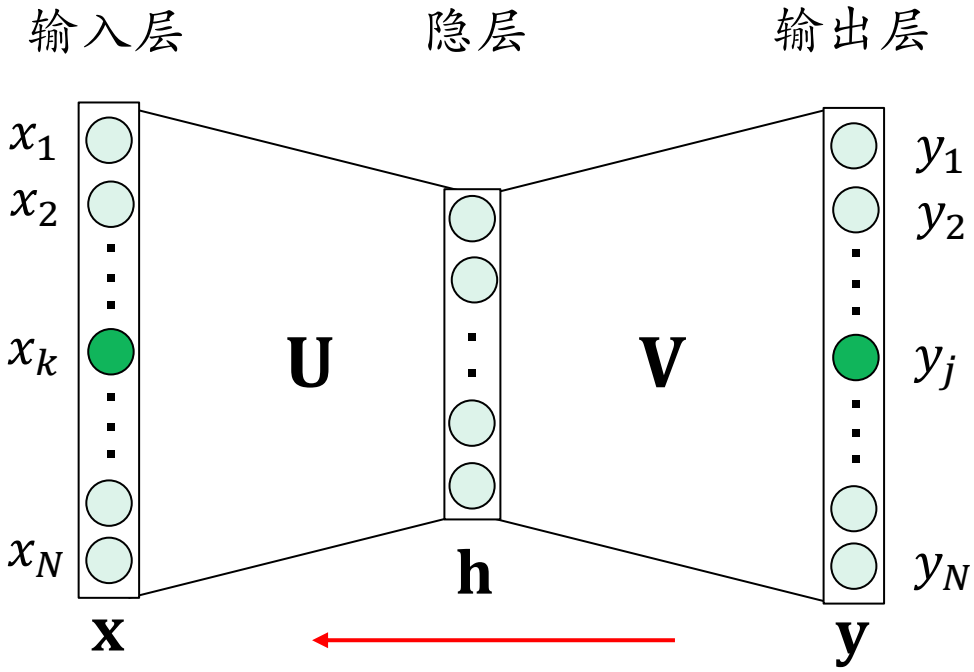
设置学习率η

$$\begin{cases} \mathbf{V} = \mathbf{V} - \eta \dfrac{\partial \mathrm{E}}{\partial \mathbf{V}} = \mathbf{V} - \eta\mathbf{e}\mathbf{h}^{\mathrm{T}} \\ \mathbf{U} = \mathbf{U} - \eta \dfrac{\partial \mathrm{E}}{\partial \mathbf{U}} = \mathbf{U} - \eta\mathbf{V}^{\mathrm{T}}\mathbf{e}\mathbf{x}^{\mathrm{T}} \end{cases}$$

反向传播算法（Back-propagation）

现在问题来了：
明明是求导，为什么叫他“反向传播”呢？

反向传播，
指的是预测误差（**e**）的
反向传播。



$$\begin{cases} \dfrac{\partial \mathrm{E}}{\partial \mathbf{V}} = \mathbf{e}\mathbf{h}^{\mathrm{T}} \\ \dfrac{\partial \mathrm{E}}{\partial \mathbf{U}} = \mathbf{V}^{\mathrm{T}}\mathbf{e}\mathbf{x}^{\mathrm{T}} \end{cases}$$

在这个例子里，我们可以把每一个权值矩阵的梯度分解成两部分，即：

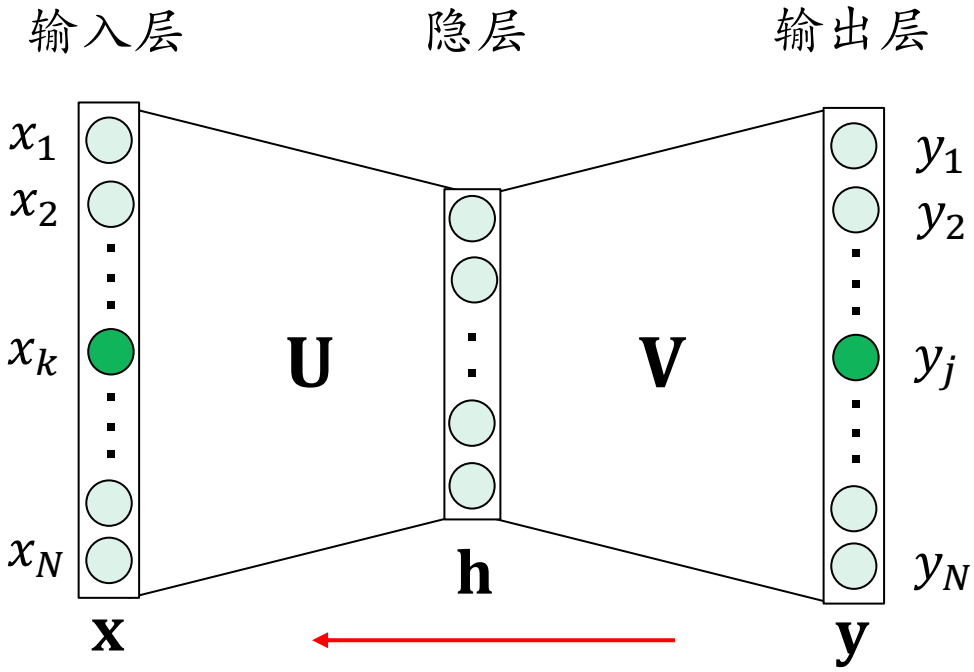
1. 该层的输入，和
2. 该层接收到的误差信号

如果层数继续加深，你会发现每一层的梯度都满足这个形式。且**e**每一次都是乘上前一层权值的转置。

反向传播算法（Back-propagation）

现在问题来了：
明明是求导，为什么叫他“反向传播”呢？

反向传播，
指的是预测误差（**e**）的
反向传播。



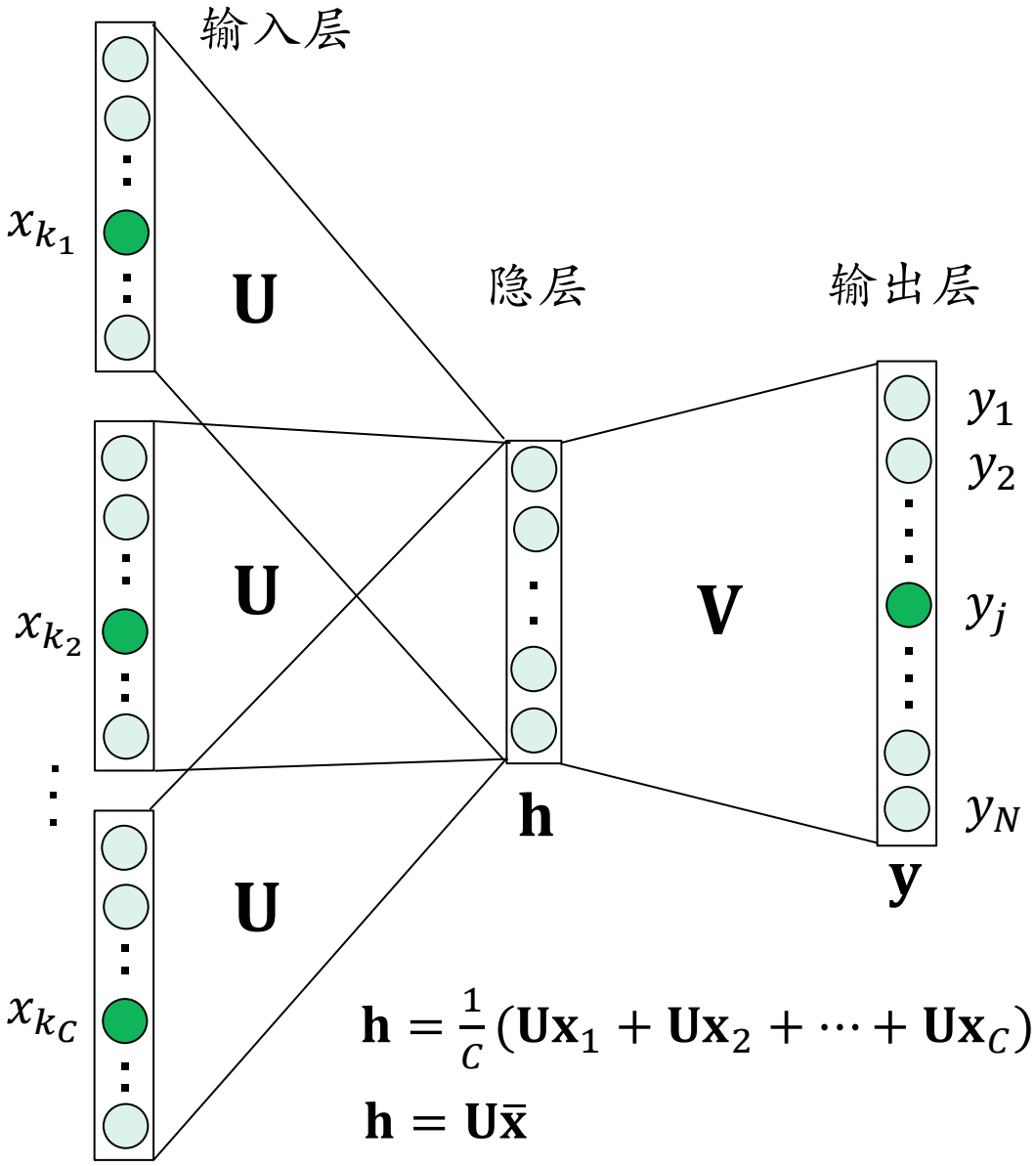
$$\begin{cases} \dfrac{\partial \mathrm{E}}{\partial \mathbf{V}} = \mathbf{e}\mathbf{h}^{\mathrm{T}} \\ \dfrac{\partial \mathrm{E}}{\partial \mathbf{U}} = \mathbf{V}^{\mathrm{T}}\mathbf{e}\mathbf{x}^{\mathrm{T}} \end{cases}$$

在这个例子里，我们可以把每一个权值矩阵的梯度分解成两部分，即：

1. 该层的输入，和
2. 该层接收到的误差信号

注意，如果中间层存在激活函数，则**e**向前传播时还要再乘上激活函数的导数

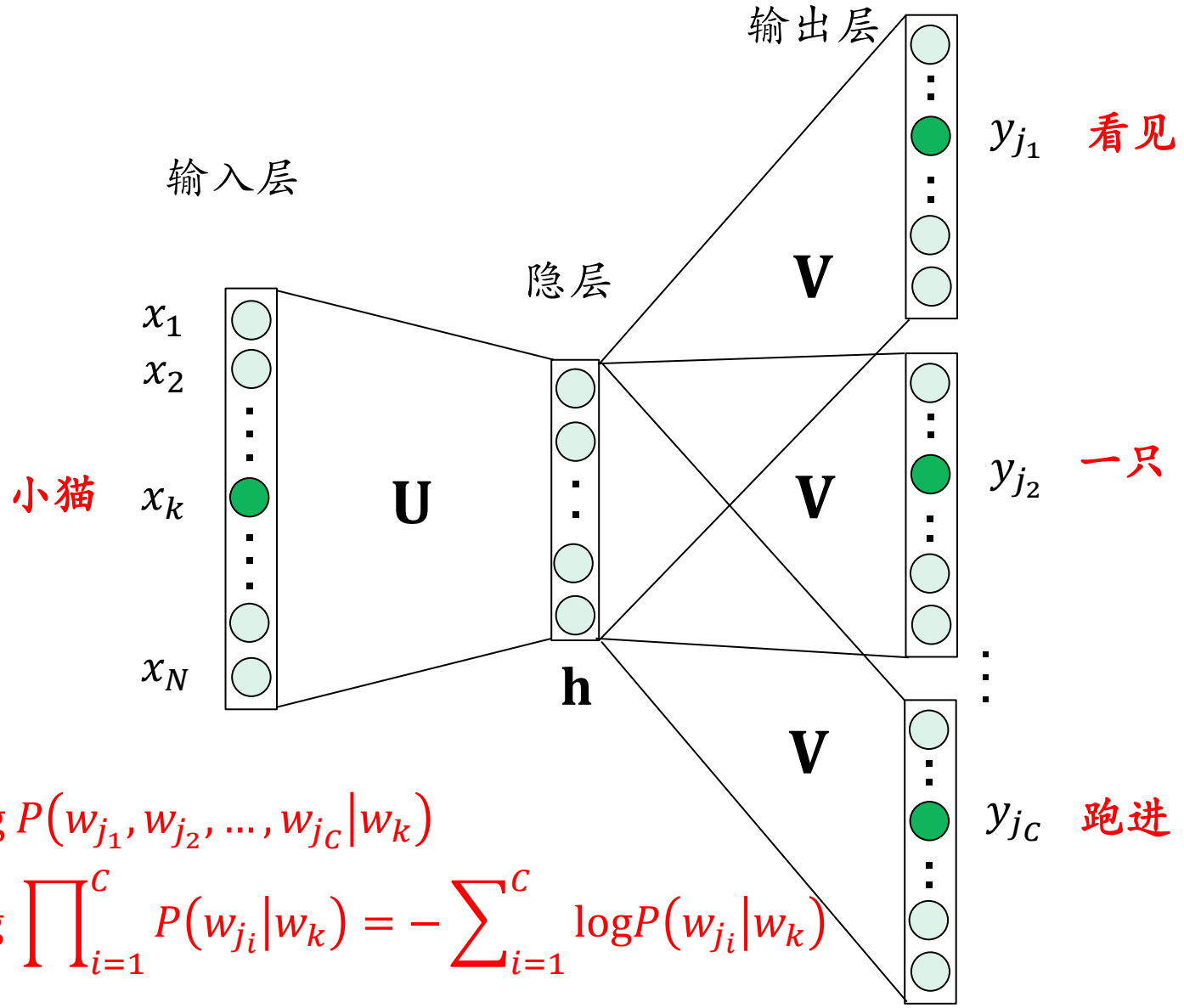
CBOW模型：考虑更多的上下文



$$\mathbf{h} = \frac{1}{C}(\mathbf{U}\mathbf{x}_1 + \mathbf{U}\mathbf{x}_2 + \cdots + \mathbf{U}\mathbf{x}_C)$$

$$\mathbf{h} = \mathbf{U}\bar{\mathbf{x}}$$

Skip-gram模型



$$E = -\log P(w_{j_1}, w_{j_2}, \dots, w_{j_C} | w_k)$$

$$= -\log \prod_{i=1}^{C} P(w_{j_i} | w_k) = -\sum_{i=1}^{C} \log P(w_{j_i} | w_k)$$

https://arxiv.org/pdf/1411.2738.pdf

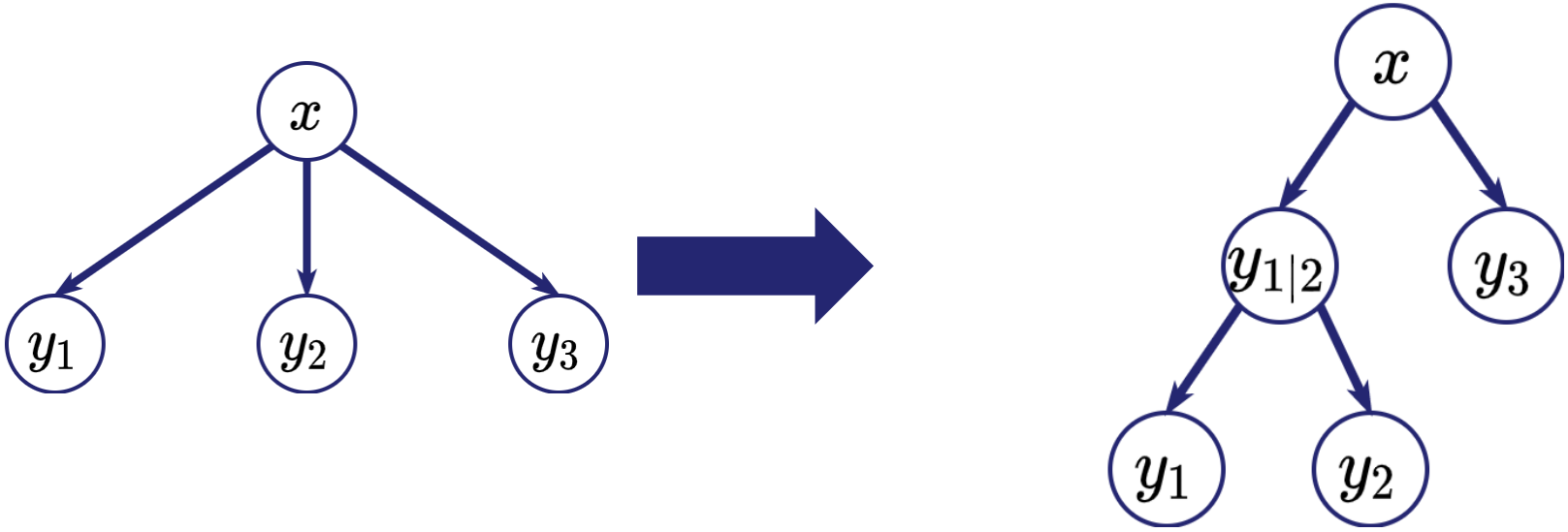
目录

词向量模型的优化

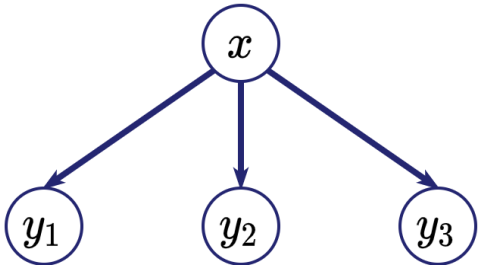
- 由于实际应用中涉及的词表很大，而softmax中又涉及大量无法约简的指数运算，因此输出端的计算开销非常大
- 为了实现高效的训练，有两种降低输出维度的方法
  - 分层柔性最大化（hierarchical softmax）
  - 负采样（negative sampling）
- 这两种方法的思想都是将一个多分类任务转化为多个二分类任务

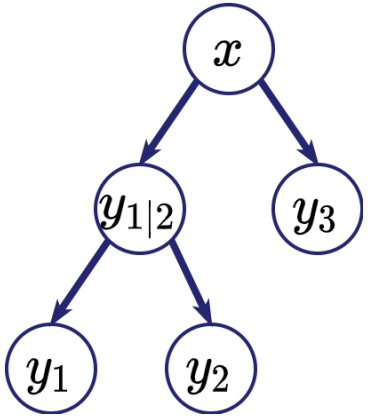
词向量模型的优化：Hierarchical Softmax

- Hierarchical Softmax
  - 将多分类问题转换为多个二分类问题，组成一棵二叉分类树
  - 以三分类问题为例



词向量模型的优化：Hierarchical Softmax





用二进制串表示二叉分类树上的分类路径，用0表示走左分支，1表示走右分支

$$\hat{P}(y_1) = \text{softmax}_1(y) = \frac{\exp({}^1y)}{\sum \exp({}^iy)}$$

$$\hat{P}(y_2) = \text{softmax}_2(y)$$

$$\hat{P}(y_3) = \text{softmax}_3(y)$$

$$\begin{aligned}\tilde{P}(y_1) &= \tilde{P}(00) \\ &= (1-\sigma(y_3))(1-\sigma(y_2)) \\ \tilde{P}(y_2) &= \tilde{P}(01) \\ &= (1-\sigma(y_3))\sigma(y_2) \\ \tilde{P}(y_3) &= \tilde{P}(1) \\ &= \sigma(y_3)\end{aligned}$$

词向量模型的优化：Hierarchical Softmax

- 任意构造的二叉树不一定能达到最优的效率
- 效率上最优的二叉树应满足平均最少的分类次数，即平均最短的二进制路径长度

- 哈夫曼树在通信中被用于设计信源已知情况下的最优信道编码，基于哈夫曼树的哈夫曼编码具有最短的平均码长
- 类似的，二叉哈夫曼树就是最优的二叉分类树

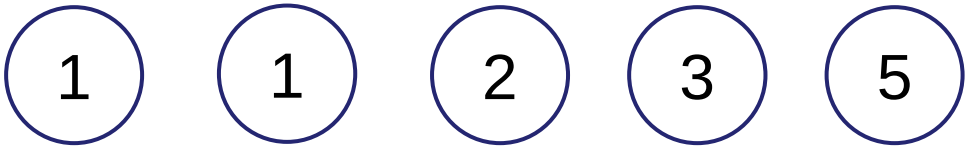
词向量模型的优化：Hierarchical Softmax

- 任意构造的二叉树不一定能达到最优的效率
- 效率上最优的二叉树应满足平均最少的分类次数，即平均最短的二进制路径长度

- 哈夫曼树在通信中被用于设计信源已知情况下的最优信道编码，基于哈夫曼树的哈夫曼编码具有最短的平均码长
- 类似的，二叉哈夫曼树就是最优的二叉分类树

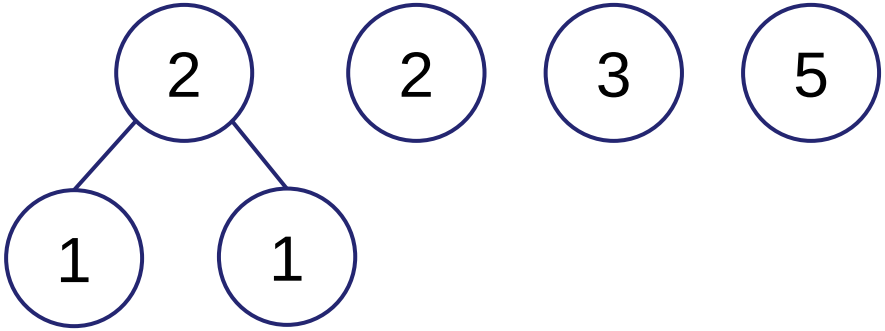
词向量模型的优化：Hierarchical Softmax

- 哈夫曼树可以利用贪心法，不断地合并权重最小的子树得到
- 以五分类情况为例
- 假设5个类别的样本数之比为1:1:2:5:3



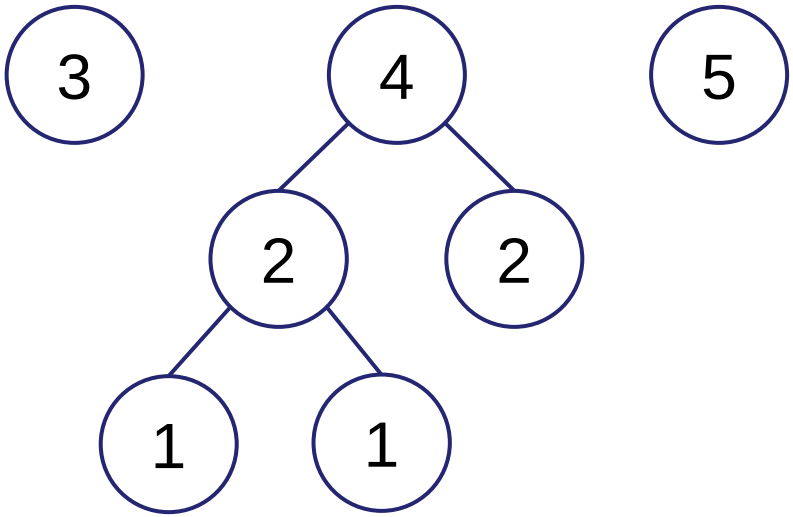
词向量模型的优化：Hierarchical Softmax

- 哈夫曼树可以利用贪心法，不断地合并权重最小的子树得到
- 以五分类情况为例
- 假设5个类别的样本数之比为1:1:2:5:3



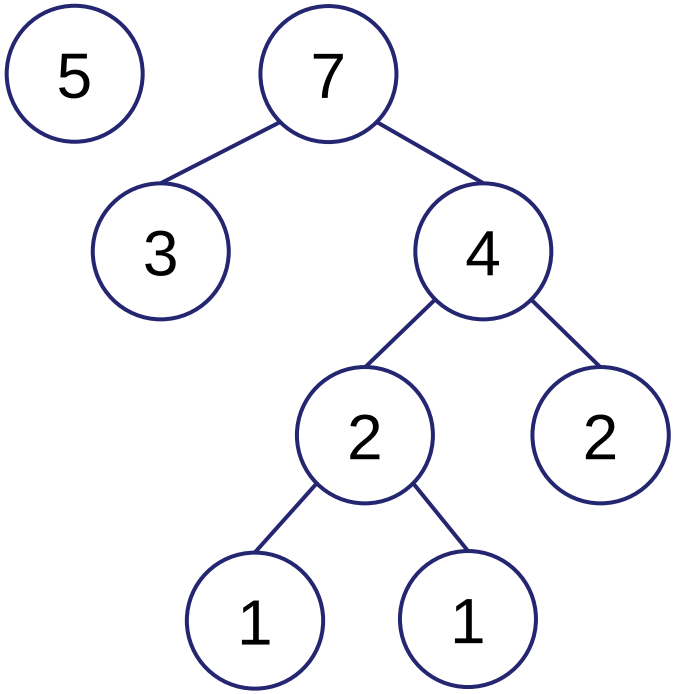
词向量模型的优化：Hierarchical Softmax

- 哈夫曼树可以利用贪心法，不断地合并权重最小的子树得到
- 以五分类情况为例
- 假设5个类别的样本数之比为1:1:2:5:3



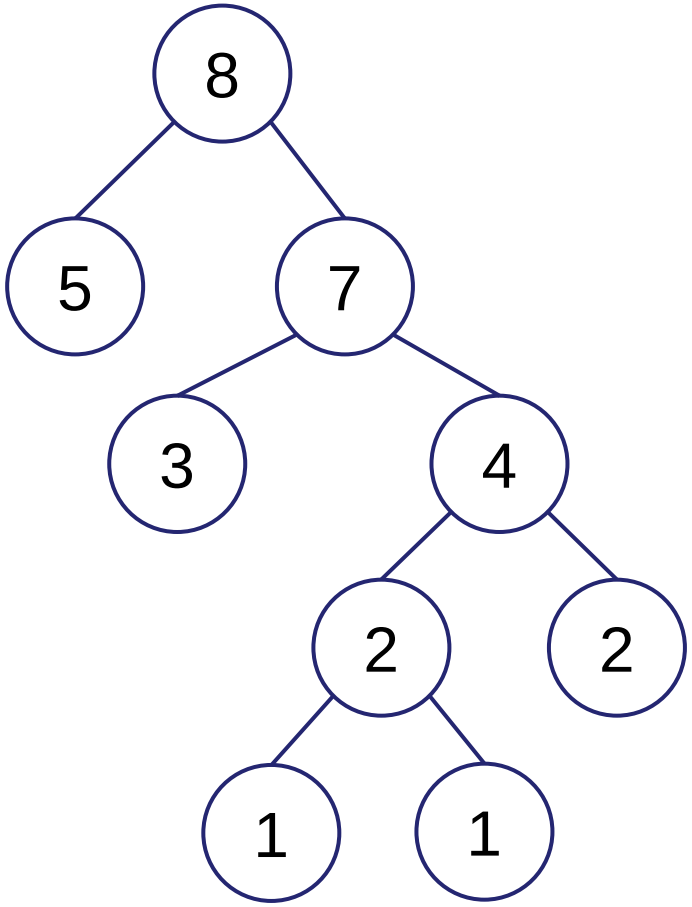
词向量模型的优化：Hierarchical Softmax

- 哈夫曼树可以利用贪心法，不断地合并权重最小的子树得到
- 以五分类情况为例
- 假设5个类别的样本数之比为1:1:2:5:3



词向量模型的优化：Hierarchical Softmax

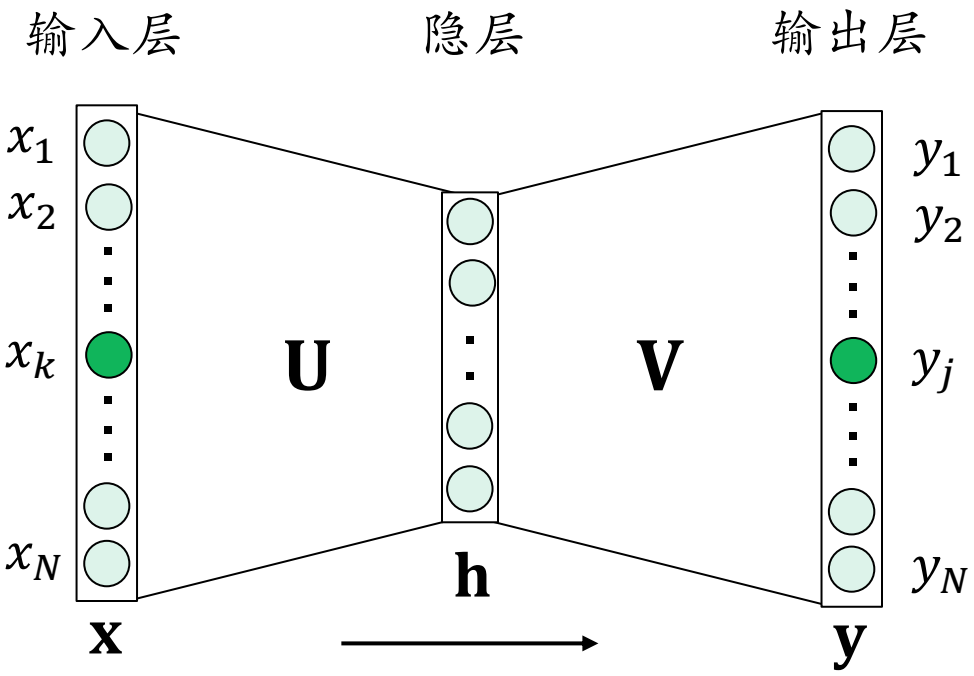
- 假设5个类别的样本数之比为1:1:2:5:3



此时的平均分类次数，也即该树叶子节点的加权路径长度为：

$$\frac{1}{12}(5\times 1+3\times 2+1\times 4+1\times 4+2\times 3)\approx 2.08$$

词向量模型的优化：Negative Sampling



大量的计算来自于输出层里的归一化：

$$y_j = \frac{\exp(o_j)}{\sum_{t=1}^{N} \exp(o_t)}$$

但是实际上我们并不需要真正去计算上面这个概率，我们只需要模型能够在正确单词上的输出尽可能大，错误单词上的输出尽可能小，就行了。

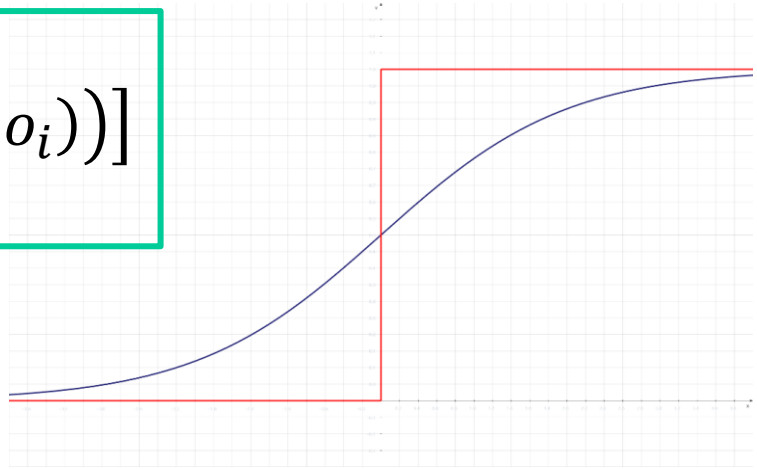
词向量模型的优化：Negative Sampling

我们把优化函数重新设计成这样：

1. 最大化正确单词上的输出
2. 定义一个噪声分布，每次最大化上面那个目标的同时，按照这个噪声分布去词汇表里随机取几个词，然后最小化这些单词上的输出。

不需要归一化，甚至不需要算每个单词的输出概率！

$$\mathrm{E} = \log \sigma(o_j) + \sum_{i=1}^{K} \mathbb{E}_{i \sim P_{noise}(i)} \left[\log(\sigma(-o_i))\right]$$



最大化正确单词上的输出

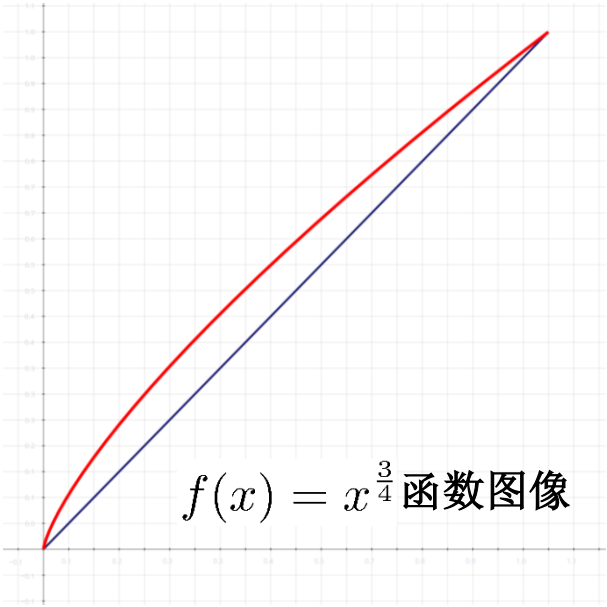
从噪声分布P_{noise}中采样K个样本，并最小化这些噪声单词上的输出

$$\sigma(y) = \frac{e^y}{e^y + 1} = \frac{1}{1 + e^{-y}}$$

词向量模型的优化：Negative Sampling

- 噪声分布P_{noise}采用如下分布：

$$P(w_i|\mathcal{C}) = \frac{\alpha_i^{\frac{3}{4}}}{\sum_j \alpha_j^{\frac{3}{4}}}$$



- 其中α_i表示词i在语料库中出现频率
- 采用小于1的幂形式可以适当增加罕见词被采样到的概率，有助于改善训练的词向量的性能

词向量的提出者

Efficient Estimation of Word Representations in Vector Space

Tomas Mikolov
Google Inc., Mountain View, CA
tmikolov@google.com

Kai Chen
Google Inc., Mountain View, CA
kaichen@google.com

Greg Corrado
Google Inc., Mountain View, CA
gcorrado@google.com

Jeffrey Dean
Google Inc., Mountain View, CA
jeff@google.com

Distributed Representations of Words and Phrases and their Compositionality

Tomas Mikolov
Google Inc.
Mountain View
mikolov@google.com

Ilya Sutskever
Google Inc.
Mountain View
ilyasu@google.com

Kai Chen
Google Inc.
Mountain View
kai@google.com

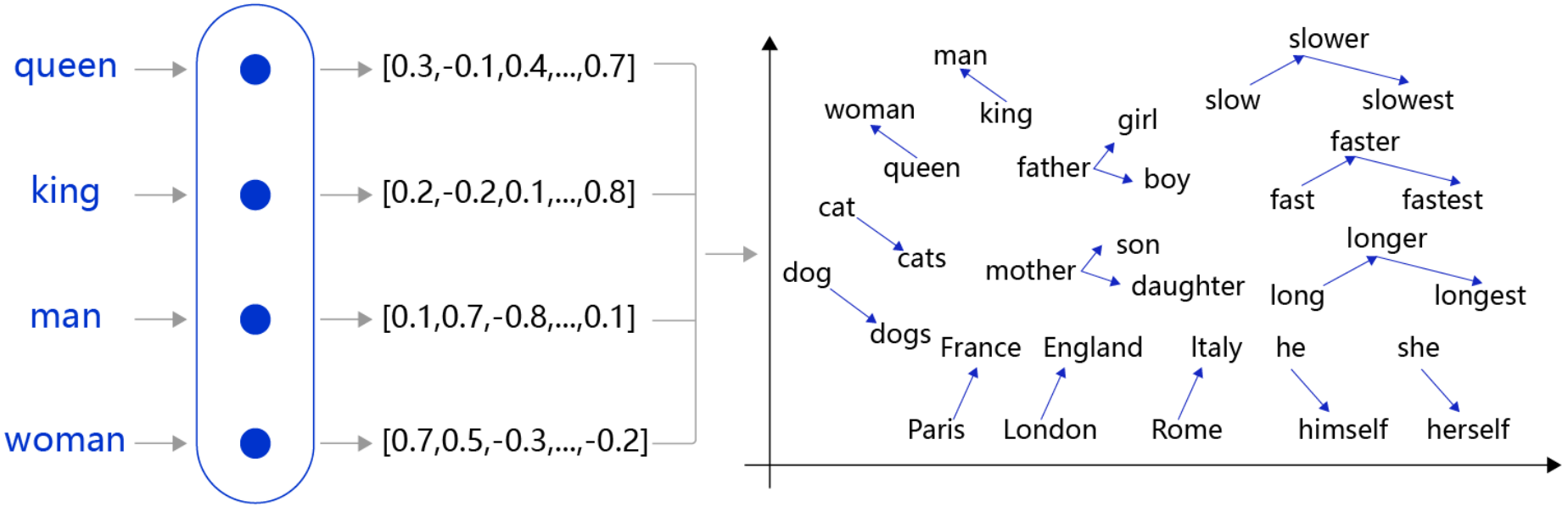
Greg Corrado
Google Inc.
Mountain View
gcorrado@google.com

Jeffrey Dean
Google Inc.
Mountain View
jeff@google.com

Tomas Mikolov

目录

词向量的性质



$$\mathbf{u}(\text{queen}) - \mathbf{u}(\text{woman}) \approx \mathbf{u}(\text{king}) - \mathbf{u}(\text{man})$$
$$\mathbf{u}(\text{France}) - \mathbf{u}(\text{Paris}) \approx \mathbf{u}(\text{England}) - \mathbf{u}(\text{London})$$
$$\ldots$$

目录

词向量模型的缺点

- 他手里拿了一个**苹果**正在吃
- 他手里拿了一个**苹果**然后插上了充电器

$\mathbf{u}(\text{苹果}_1)$ $\mathbf{u}(\text{苹果}_2)$ …

1. 没法提前知道每个词具体不同意思的数量
2. 没法提前知道当前句子里的词对应的意思是哪一个

要解决上述问题，我们需要词向量的高级版：与上下文相关的词向量（context-dependent word vectors）

4.1动态词向量/预训练模型

目录

常用工具

- **最早的word2vec原版代码（2013年）：**
  - https://code.google.com/archive/p/word2vec/
- **CBOW和Skip-gram的TensorFlow实现：**
  - https://www.tensorflow.org/tutorials/text/word2vec
- **CBOW的Pytorch实现：**
  - https://pytorch.org/tutorials/beginner/nlp/word_embeddings_tutorial.html#exercise-computing-word-embeddings-continuous-bag-of-words