

文本表征学习

3. 神经网络语言模型

EE1513

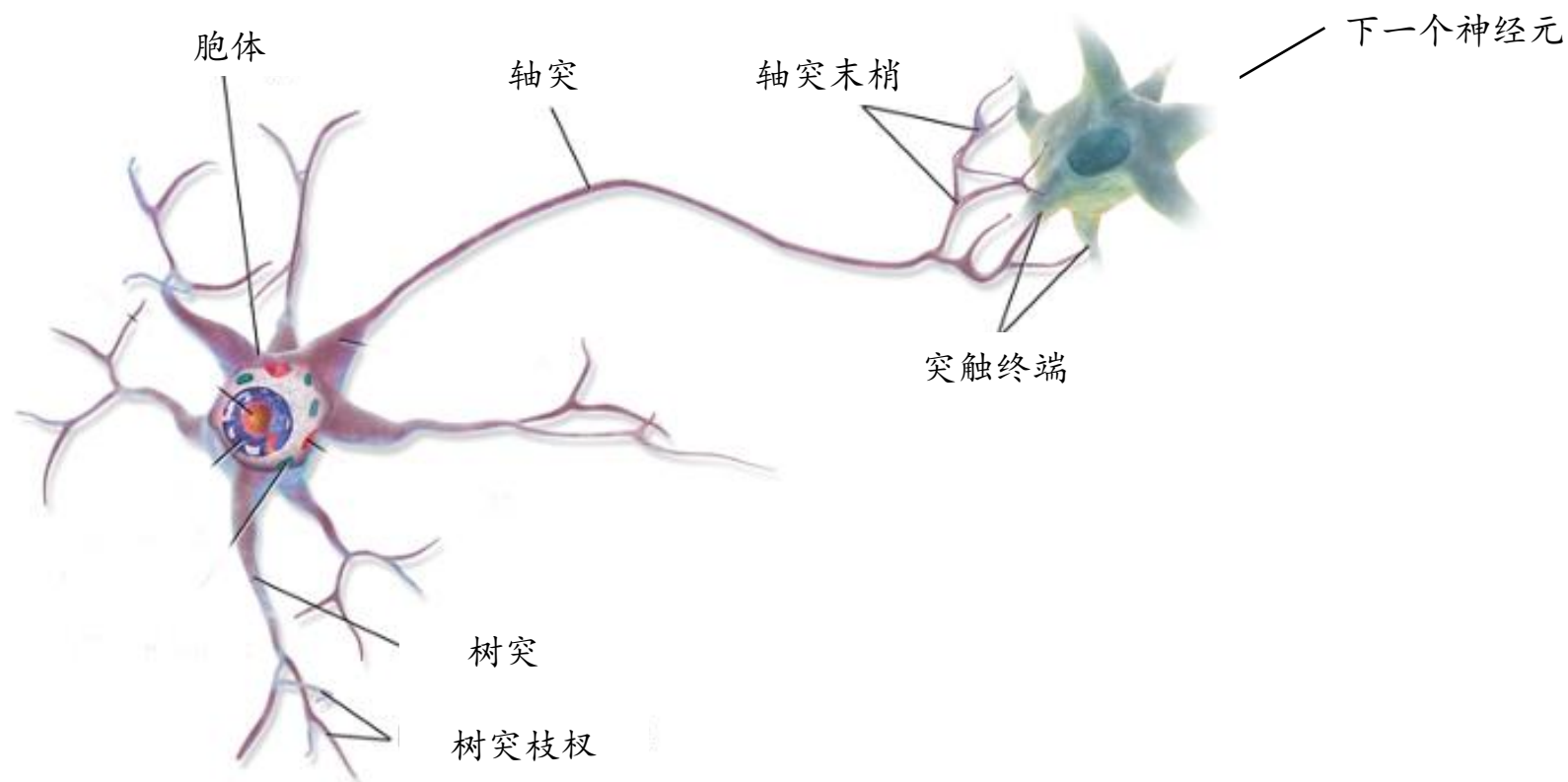
宋彦

大纲

- 深度神经网络
 - 前向算法
 - 反向传播
- 神经网络语言模型 (NLM)
 - 预测
 - 训练
 - 特点
- 神经网络语言模型的增强方法
- 应用

深度神经网络

人类大脑中的神经元



神经网络单元

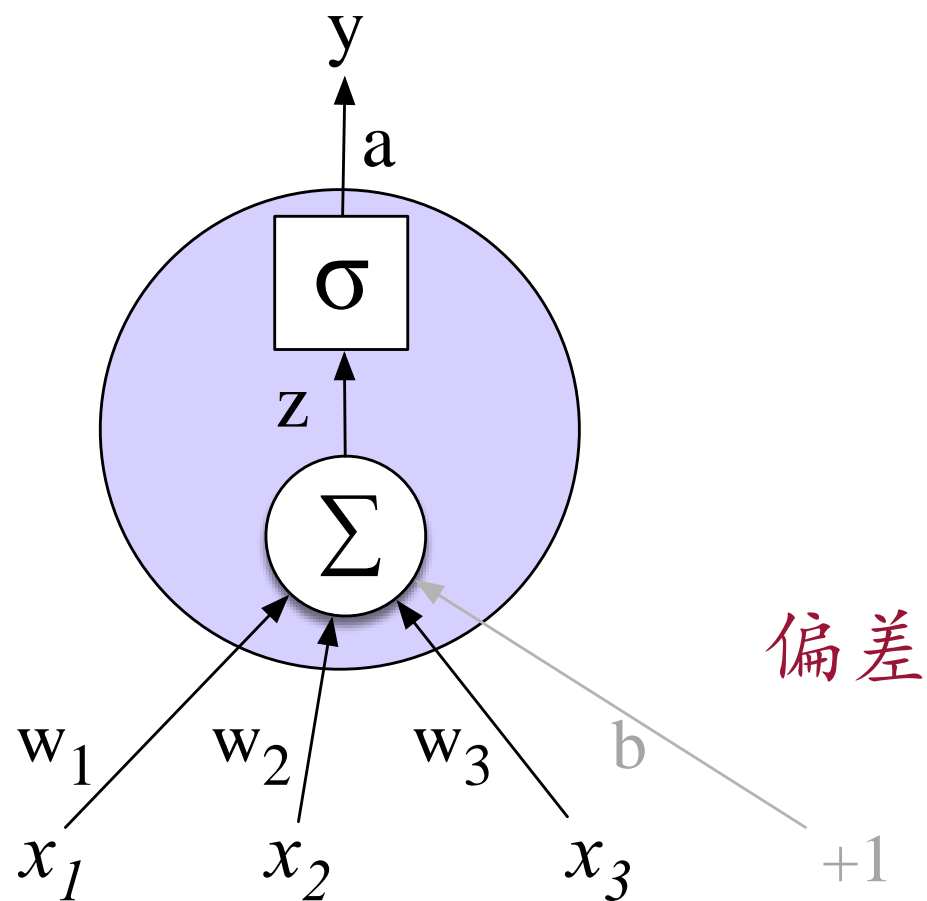
输出值

非线性变换

加权和

权重

输入层



感知机

- 符号：输入向量 \mathbf{x} 和输出向量 \mathbf{z} （他们用列向量表示）

$$\mathbf{x} = \begin{bmatrix} x_1 \\ \dots \\ x_j \\ \dots \\ x_n \end{bmatrix}, \mathbf{z} = \begin{bmatrix} z_1 \\ \dots \\ z_i \\ \dots \\ z_m \end{bmatrix}$$

n 和 m 分别代表输入和输出的向量维度

感知机

- 计算输出向量 \mathbf{z} 的过程为

$$\mathbf{a} = \mathbf{W}\mathbf{x} + \mathbf{b}$$

$$\mathbf{z} = g(\mathbf{a})$$

- $\mathbf{W} = (w_{i,j}) \in \mathbb{R}^{m \times n}$ 是权重矩阵
- $\mathbf{b} = [b_1, \dots, b_i, \dots, b_m]^T$ 是偏差向量(上角标 T 表示转置)
- \mathbf{W} 和 \mathbf{b} 是感知机的参数
- g 是激活函数
 - 在感知机中, $g(\mathbf{a}) = \mathbf{a}$

感知机

$$\mathbf{z} = g(\mathbf{W}\mathbf{x} + \mathbf{b}) = \mathbf{W}\mathbf{x} + \mathbf{b}$$

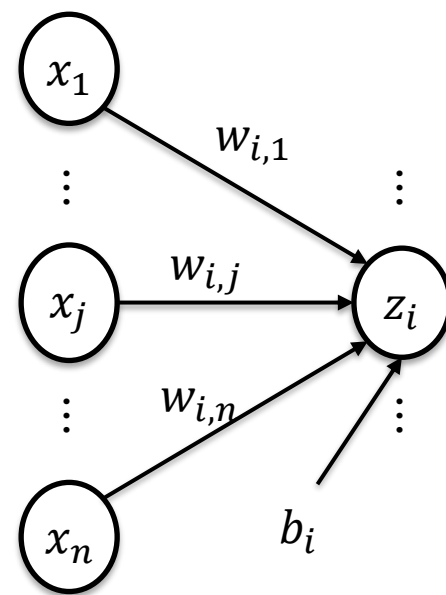
$$\begin{bmatrix} z_1 \\ \dots \\ z_i \\ \dots \\ z_m \end{bmatrix} = \begin{bmatrix} w_{1,1} & \dots & w_{1,j} & \dots & w_{1,n} \\ \dots & & \dots & & \dots \\ w_{i,1} & \dots & w_{i,j} & \dots & w_{i,n} \\ \dots & & \dots & & \dots \\ w_{m,1} & \dots & w_{m,j} & \dots & w_{m,n} \end{bmatrix} \begin{bmatrix} x_1 \\ \dots \\ x_j \\ \dots \\ x_n \end{bmatrix} + \begin{bmatrix} b_1 \\ \dots \\ b_i \\ \dots \\ b_m \end{bmatrix}$$

$$z_i = \sum_{j=1}^n w_{i,j} x_j + b_i$$

感知机

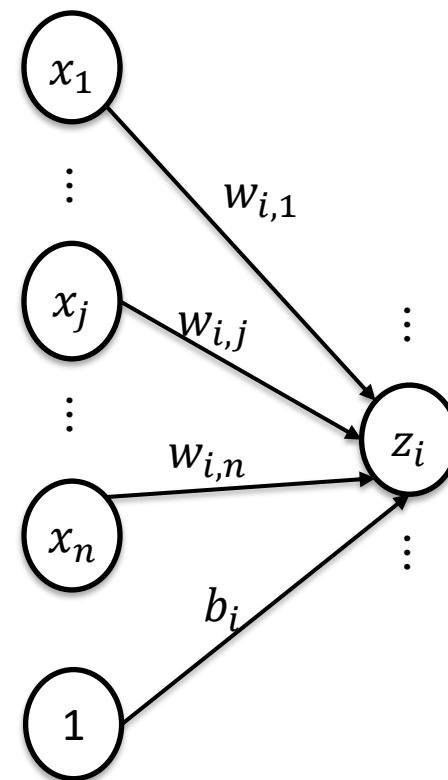
每一个节点表示一个神经元

$$z_i = \sum_{j=1}^n w_{i,j} x_j + b_i$$



输入

输出



输入

输出

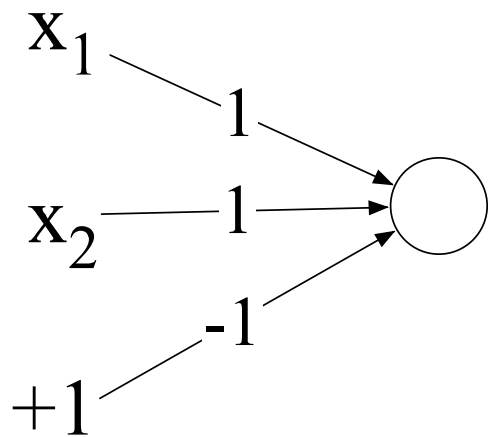
异或 (XOR) 问题

- 神经网络单元可以实现简单函数吗？
- 输入： x_1 和 x_2 ，输出： y

AND			OR			XOR		
x1	x2	y	x1	x2	y	x1	x2	y
0	0	0	0	0	0	0	0	0
0	1	0	0	1	1	0	1	1
1	0	0	1	0	1	1	0	1
1	1	1	1	1	1	1	1	0

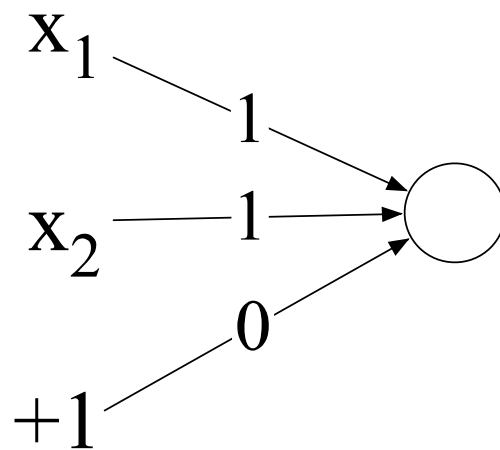
感知机用于和（AND）与或（OR）

$$y = \begin{cases} 0, & \text{if } w \cdot x + b \leq 0 \\ 1, & \text{if } w \cdot x + b > 0 \end{cases}$$



和

AND		
x1	x2	y
0	0	0
0	1	0
1	0	0
1	1	1



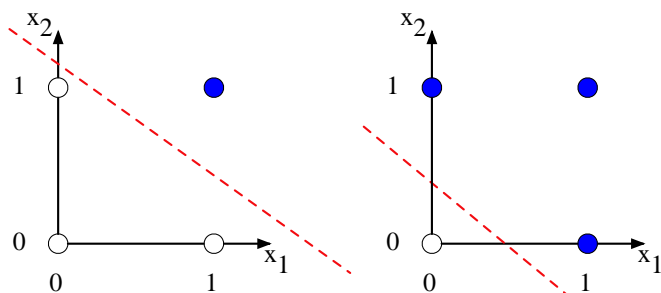
或

OR		
x1	x2	y
0	0	0
0	1	1
1	0	1
1	1	1

感知机是否能用于异或？

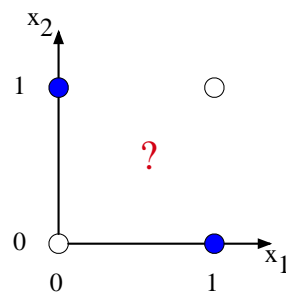
- 感知机是否能用于异或？
- 不行！为什么？
- 感知机的决策边界是线性的
- 异或不是线性可分的问题

XOR		
x1	x2	y
0	0	0
0	1	1
1	0	1
1	1	0



a) $x_1 \text{ AND } x_2$

b) $x_1 \text{ OR } x_2$

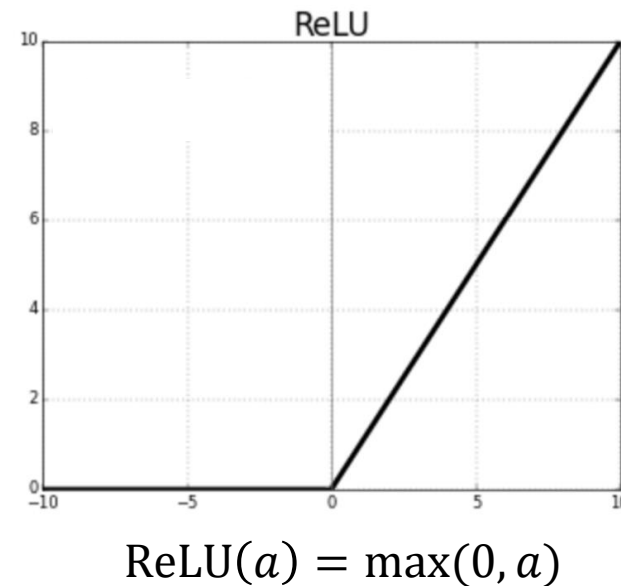
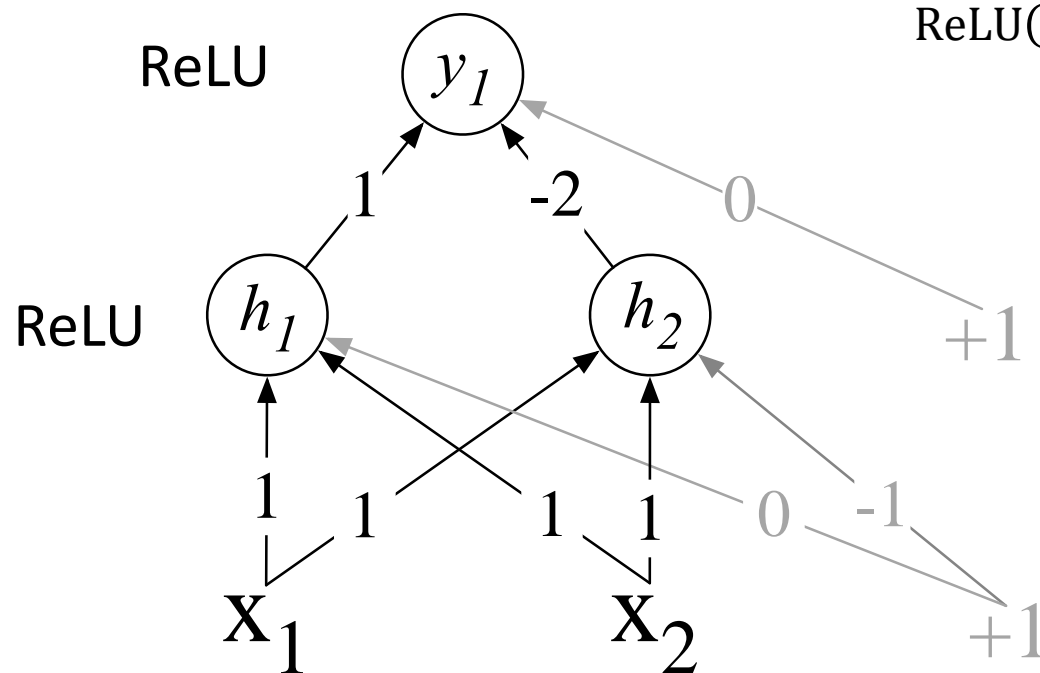


c) $x_1 \text{ XOR } x_2$

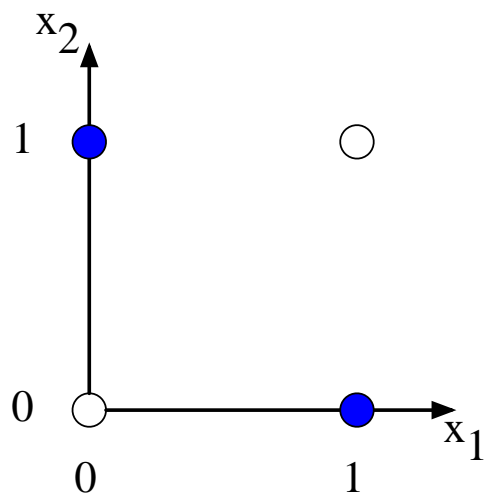
异或问题的解决方案

- 单个感知机无法解决异或问题
- 异或问题可以被多层神经网络解决

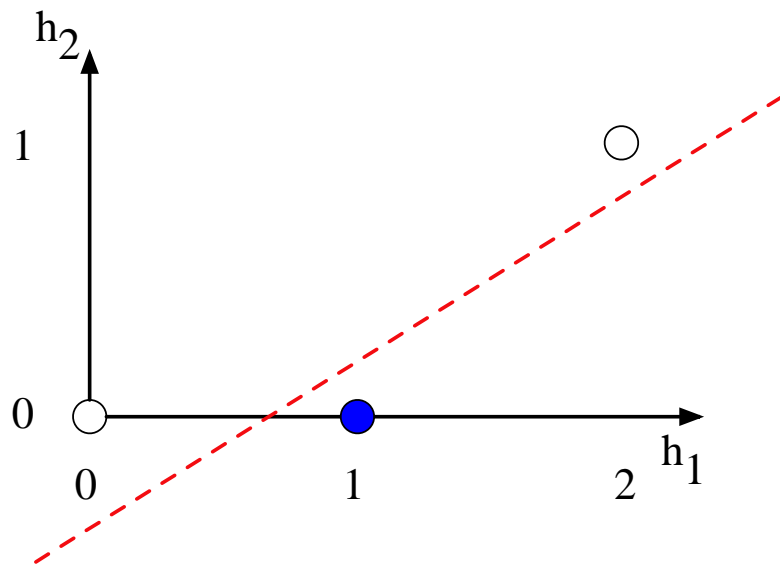
XOR		
x1	x2	y
0	0	0
0	1	1
1	0	1
1	1	0



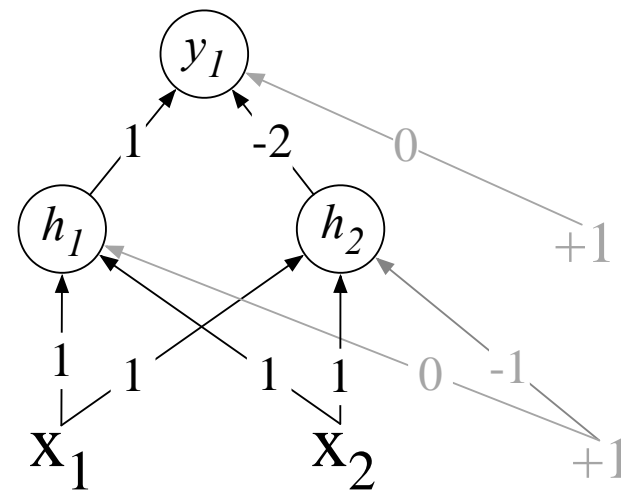
隐向量 h



a) The original x space



b) The new (linearly separable) h space

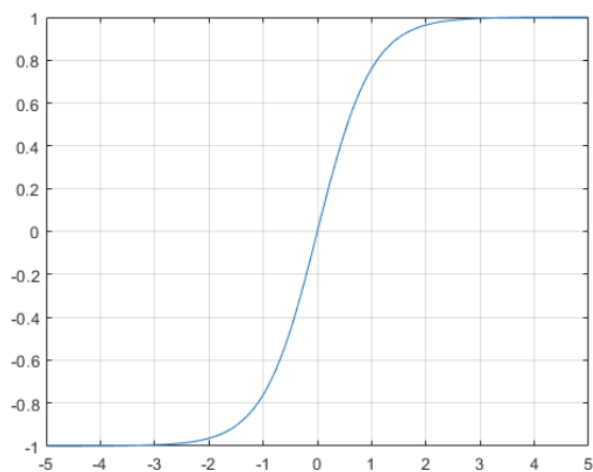


隐藏层将学习形成有用的表示

激活函数

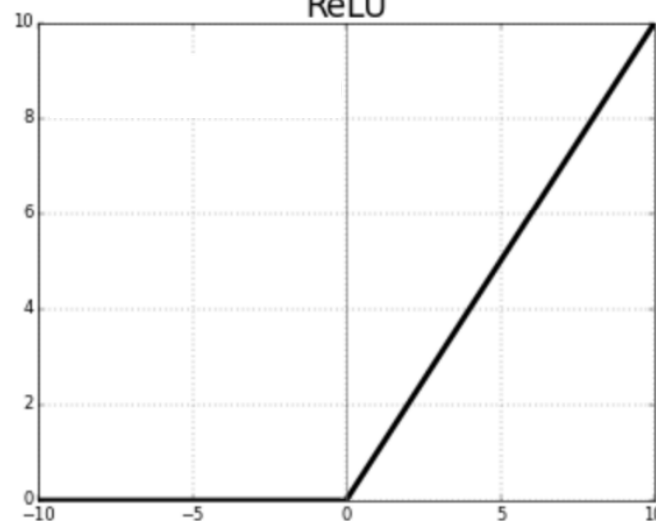
- 对于隐藏层的激活函数，人们经常使用 \tanh 和 ReLU 或者他们的变体

\tanh



$$\tanh(a) = \frac{e^a - e^{-a}}{e^a + e^{-a}}$$

ReLU



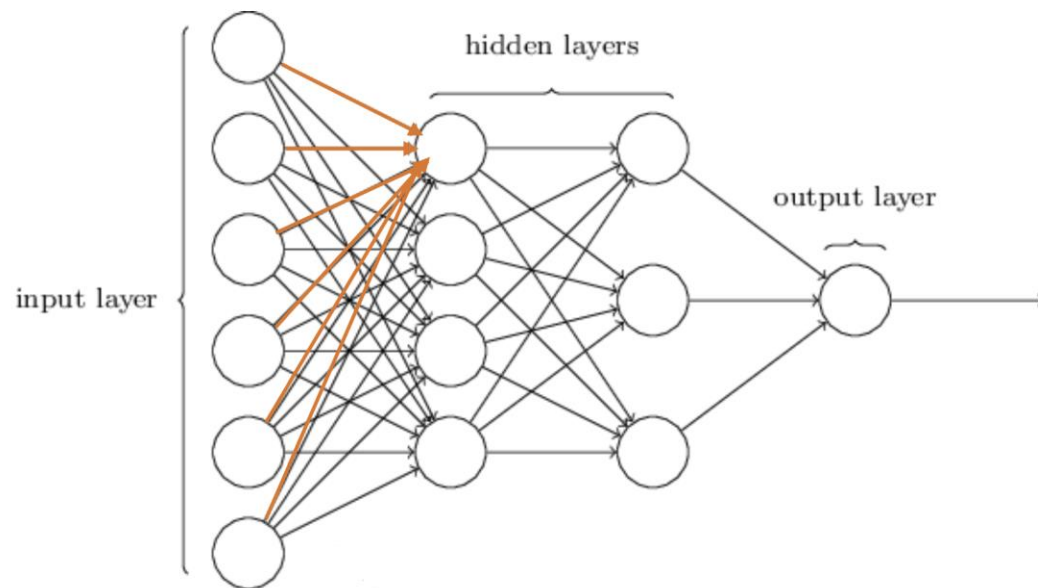
$$\text{ReLU}(a) = \max(0, a)$$

多层感知机 (MLP)

$$\mathbf{z} = g_L(\mathbf{W}_L g_{L-1}(\cdots g_2(\mathbf{W}_2 g_1(\mathbf{W}_1 \mathbf{x} + \mathbf{b}_1) + \mathbf{b}_2) \cdots) + \mathbf{b}_L)$$

- \mathbf{W}_l , \mathbf{b}_l , 和 g_l 分别代表第 l 层的权重矩阵、偏移向量、激活函数
- 所有的 \mathbf{W}_l 和 \mathbf{b}_l 都是模型参数
- \mathbf{z} 是最后一层的输出
- *softmax* 函数被经常应用于 \mathbf{z} 以得到最终输出 \mathbf{y}

$$\mathbf{y} = \text{softmax}(\mathbf{z})$$



Softmax 函数

- 给定向量 $\mathbf{z} = [z_1, \dots, z_i, \dots, z_n]^T$.
 $\mathbf{y} = \text{softmax}(\mathbf{z}) = [y_1, \dots, y_i, \dots, y_n]^T$

其中 y_i 是

$$y_i = \frac{e^{z_i}}{\sum_{j=1}^n e^{z_j}}$$

- 性质

- y_i 非负 ($y_i \geq 0$)
- y_i 的和是 1 ($\sum_{i=1}^n y_i = 1$)

y_i 看起来像是概率

MLP的训练

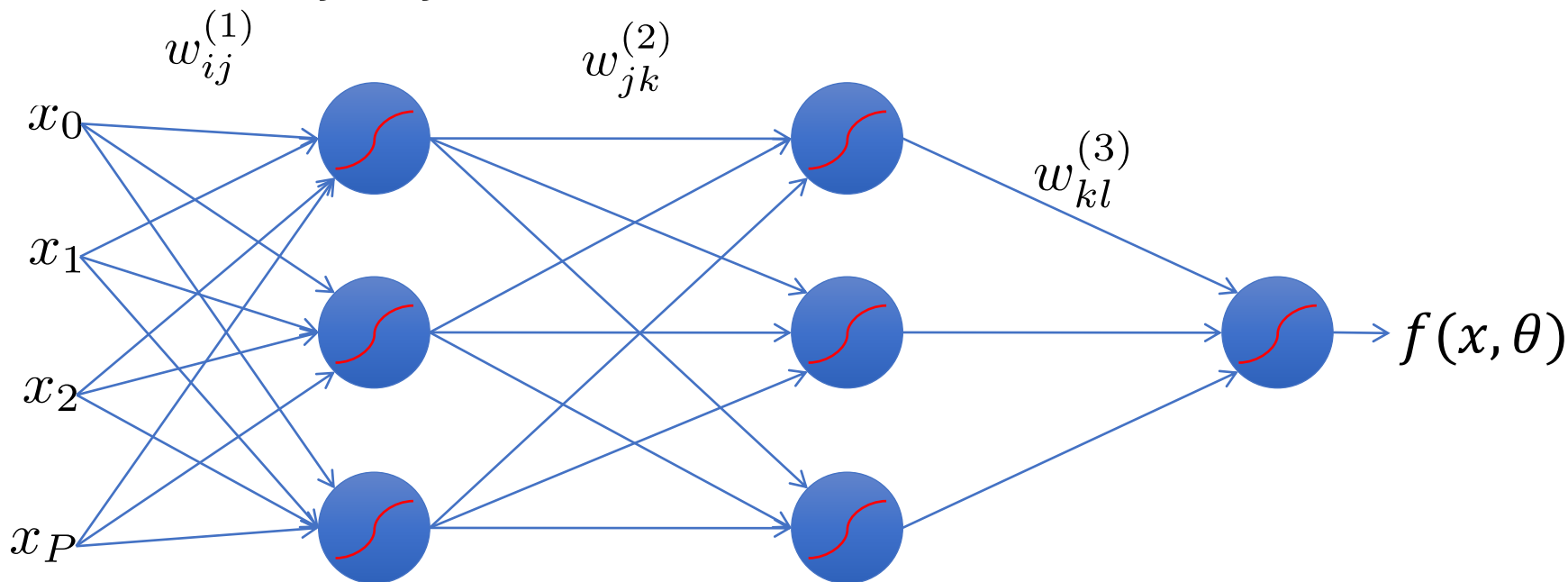
- 神经网络中所有需要被估计的参数记为 θ (包括 \mathbf{W} 和 \mathbf{b})
- 目的是使得预测的结果 y 尽可能靠近人工标注结果 y^*
- 使用损失函数计算 y 和 y^* 之间的距离
$$L(y^*, y)$$

例如, 交叉熵损失函数

- 使得 θ 最小化 L
- 为此, 我们需要通过反向传播计算 L 的梯度

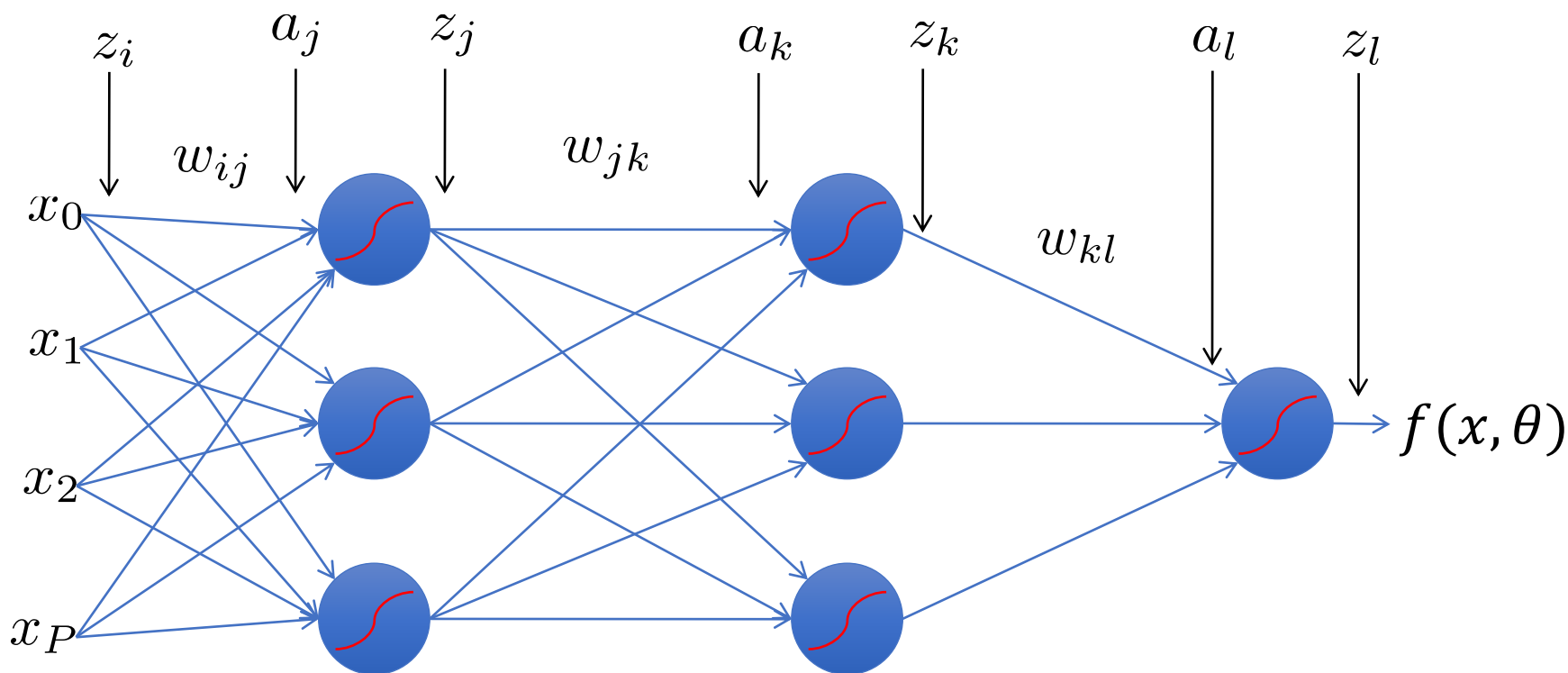
反向传播

- 假设我们有三层神经网络
- 模型参数: $\theta = \{w_{ij}^{(1)}, w_{jk}^{(2)}, w_{kl}^{(3)}\}$
- 简记为: $\theta = \{w_{ij}, w_{jk}, w_{kl}\}$



反向传播

- 模型参数: $\theta = \{w_{ij}, w_{jk}, w_{kl}\}$
- 记 a 和 z 分别为每个节点的输入和输出



反向传播

$$a_j = \sum_i w_{ij} z_i$$

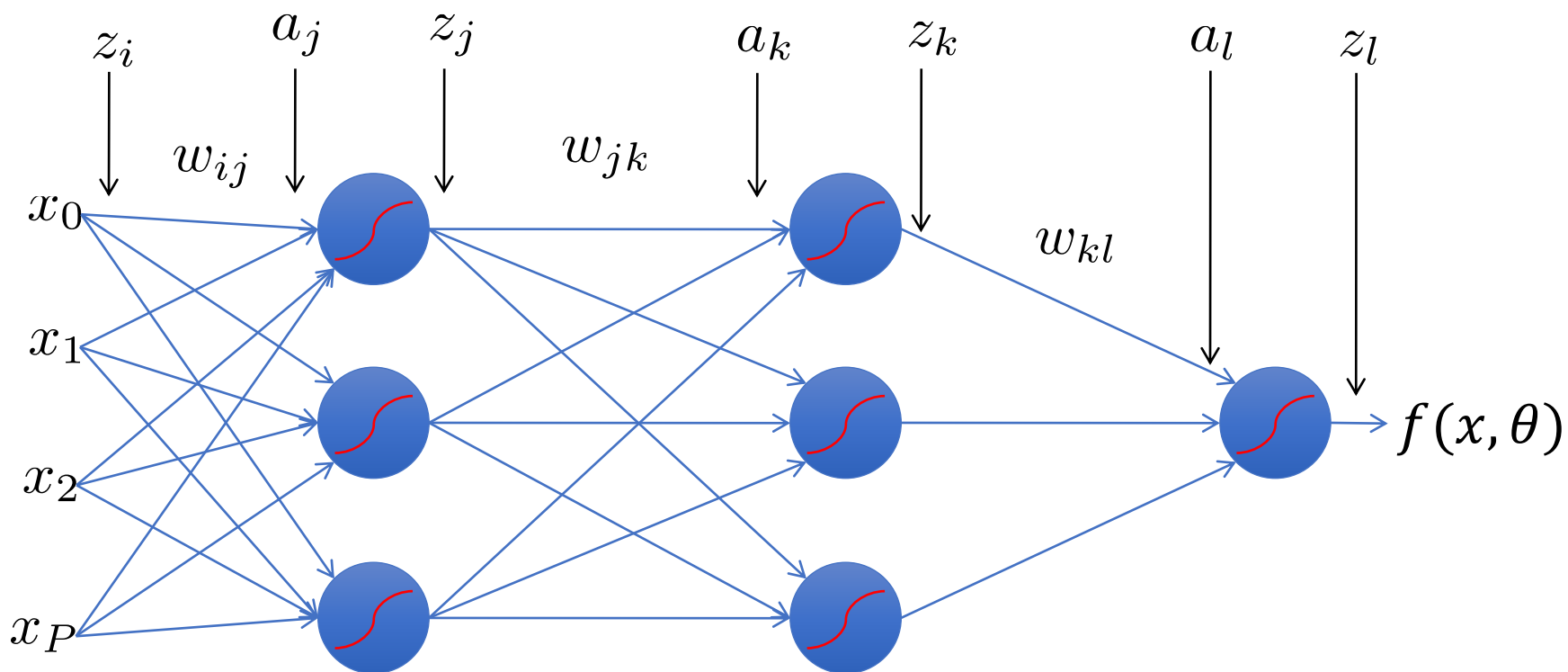
$$a_k = \sum_j w_{jk} z_j$$

$$a_l = \sum_k w_{kl} z_k$$

$$z_j = g(a_j)$$

$$z_k = g(a_k)$$

$$z_l = g(a_l)$$



训练：最小化损失函数

$$R(\theta) = \frac{1}{N} \sum_{n=0}^N L(y_n - f(x_n))$$

经验风险函数

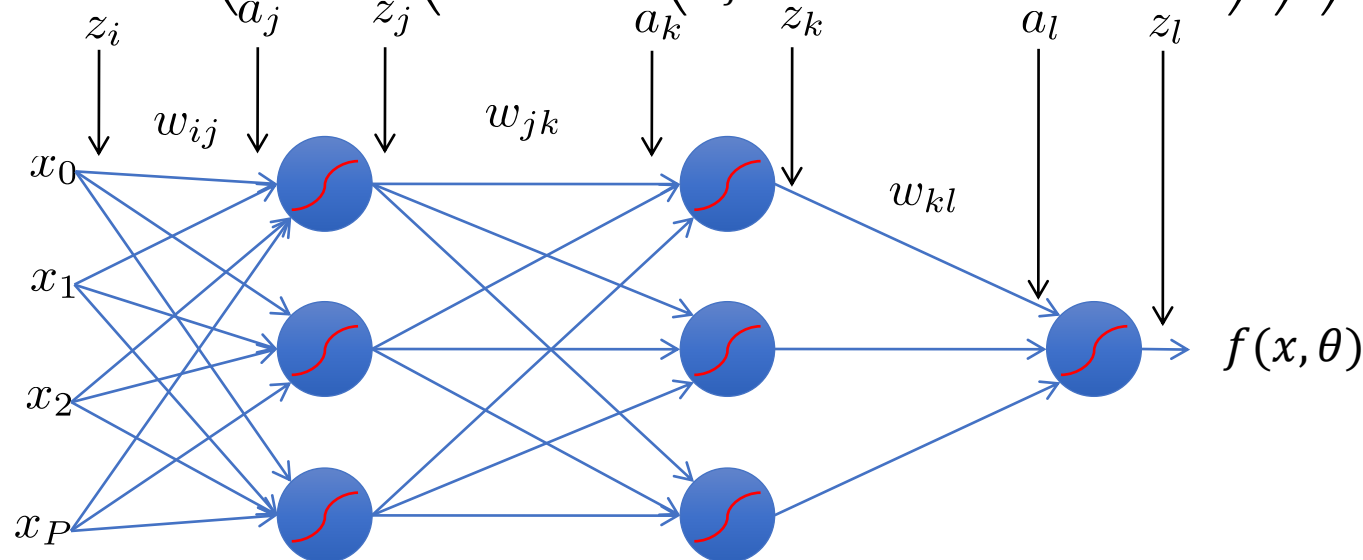
R : 所有的损失

θ : 所有的参数

N : 训练实例的数量

$$= \frac{1}{N} \sum_{n=0}^N \frac{1}{2} (y_n - f(x_n))^2$$

$$= \frac{1}{N} \sum_{n=0}^N \frac{1}{2} \left(y_n - g \left(\sum_k w_{kl} g \left(\sum_j w_{jk} g \left(\sum_i w_{ij} x_{n,i} \right) \right) \right) \right)^2$$



反向传播

优化最后一层参数 w_{kl}

$$L_n = \frac{1}{2} (y_n - f(x_n))^2$$

第 n 个实例的损失

R : 所有的损失

$$\frac{\partial R}{\partial w_{kl}} = \frac{1}{N} \sum_n \left[\frac{\partial L_n}{\partial a_{l,n}} \right] \left[\frac{\partial a_{l,n}}{\partial w_{kl}} \right]$$

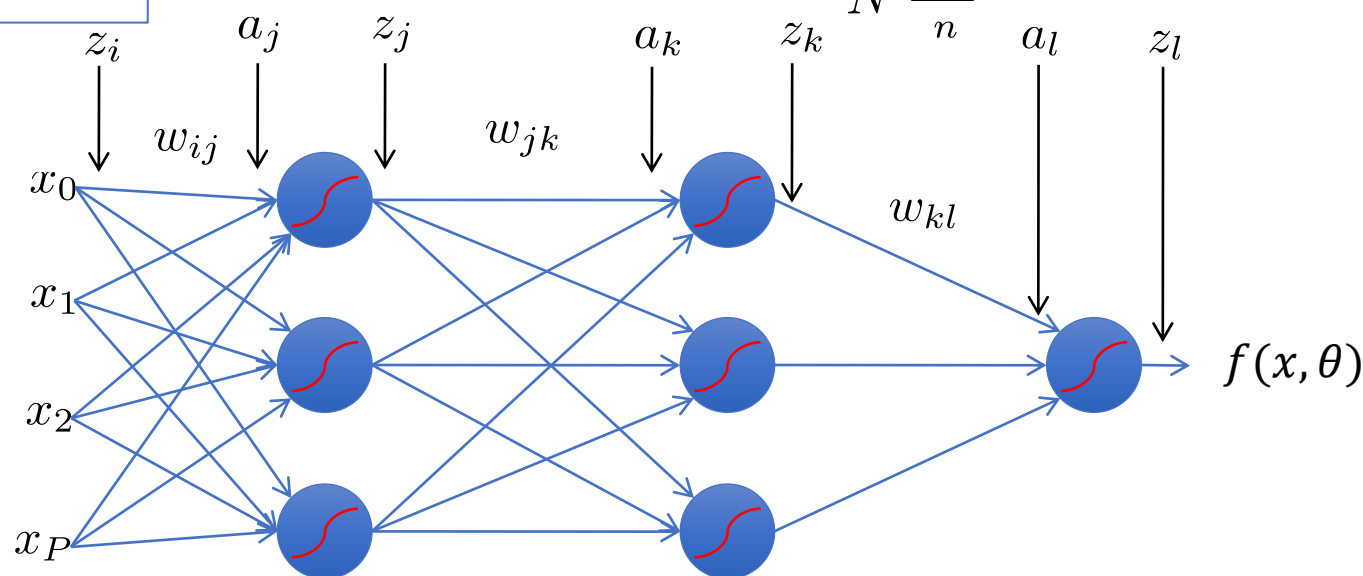
链式法则

$$\frac{\partial R}{\partial w_{kl}} = \frac{1}{N} \sum_n \left[\frac{\partial \frac{1}{2} (y_n - g(a_{l,n}))^2}{\partial a_{l,n}} \right] \left[\frac{\partial z_{k,n} w_{kl}}{\partial w_{kl}} \right] = \frac{1}{N} \sum_n [-(y_n - z_{l,n}) g'(a_{l,n})] z_{k,n}$$

$$z_{l,n} = g(a_{l,n})$$

g 是激活函数

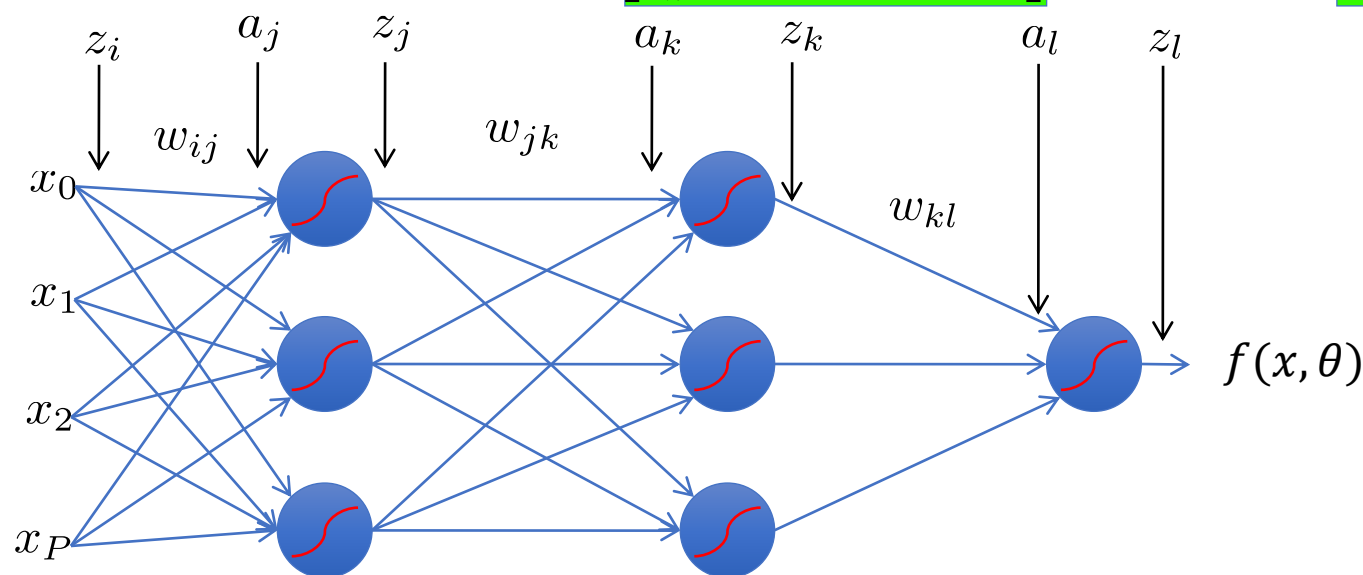
$$= \frac{1}{N} \sum_n \delta_{l,n} z_{k,n}$$



反向传播

对所有前面的层重复相同的过程

$$\begin{aligned}\frac{\partial R}{\partial w_{kl}} &= \frac{1}{N} \sum_n \left[\frac{\partial L_n}{\partial a_{l,n}} \right] \left[\frac{\partial a_{l,n}}{\partial w_{kl}} \right] = \frac{1}{N} \sum_n \left[-(y_n - z_{l,n}) g'(a_{l,n}) \right] z_{k,n} = \frac{1}{N} \sum_n \delta_{l,n} z_{k,n} \\ \frac{\partial R}{\partial w_{jk}} &= \frac{1}{N} \sum_n \left[\frac{\partial L_n}{\partial a_{k,n}} \right] \left[\frac{\partial a_{k,n}}{\partial w_{jk}} \right] = \frac{1}{N} \sum_n \left[\sum_l \delta_{l,n} w_{kl} g'(a_{k,n}) \right] z_{j,n} = \frac{1}{N} \sum_n \delta_{k,n} z_{j,n} \\ \frac{\partial R}{\partial w_{ij}} &= \frac{1}{N} \sum_n \left[\frac{\partial L_n}{\partial a_{j,n}} \right] \left[\frac{\partial a_{j,n}}{\partial w_{ij}} \right] = \frac{1}{N} \sum_n \left[\sum_k \delta_{k,n} w_{jk} g'(a_{j,n}) \right] z_{i,n} = \frac{1}{N} \sum_n \delta_{j,n} z_{i,n}\end{aligned}$$



学习方法：梯度下降

- 函数 R 有 n 个参数 $\theta = [w_1, w_2 \cdots w_n]$

- R 的梯度计算方式为

$$\nabla R(\theta) = \left[\frac{\partial R}{\partial w_1}, \frac{\partial R}{\partial w_2}, \cdots, \frac{\partial R}{\partial w_n} \right]$$

- 把每个参数 w 向着梯度相反的方向更新

$$\theta = \theta - \eta \cdot \nabla R(\theta)$$

其中 η 是学习率，用于控制每次更新的大小

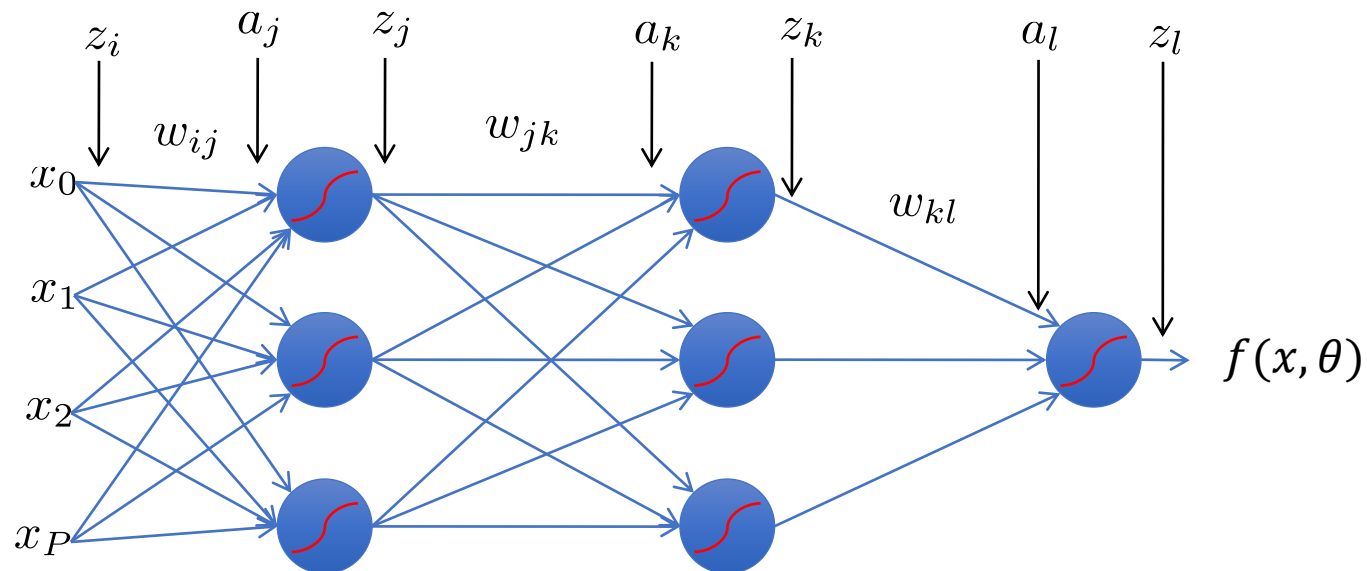
$$w_i = w_i - \eta \cdot \frac{\partial R}{\partial w_i}$$

梯度下降

$$w_{ij}^{t+1} = w_{ij}^t - \eta \frac{\partial R}{\partial w_{ij}}$$

$$w_{jk}^{t+1} = w_{jk}^t - \eta \frac{\partial R}{\partial w_{jk}}$$

$$w_{kl}^{t+1} = w_{kl}^t - \eta \frac{\partial R}{\partial w_{kl}}$$



神经网络语言模型

回忆：什么是语言模型

- 一个一般的定义：对语言进行建模的方式
- N-gram 语言模型：句子概率通过下式计算

$$P(w_1 \cdots w_n) \approx \prod_{i=1}^n P(w_i | w_{i-k} \cdots w_{i-1})$$

其中的假设为：当前词仅受到前 $k - 1$ 个词的影响

- 我们是否可以使用基于神经网络的语言模型？

神经网络语言模型

- 词 w_i 被 f_e 映射到低维空间的向量 z_i

$$f_e: w_i \rightarrow z_i$$

- 使用得到的向量和神经网络 (f) 计算条件概率

$$P(w_n | w_1 \cdots w_{n-1}) = f_{w_n}(z_1, \cdots, z_{n-1})$$

f_{w_n} 表示神经网络 f 计算得到的 w_n 的概率

- 使用神经网络语言模型的文本生成

$$\hat{w}_n = \arg \max_{w \in V} P(w | w_1 \cdots w_{n-1}) = \arg \max_{w \in V} f_w(z_1, \cdots, z_{n-1})$$

连续空间语言模型的学习

- 输入：
 - 历史词汇
- 输出：
 - 目标词
- 计算词概率的函数：
 - 线性转换
 - 前馈神经网络
 - 递归神经网络
 - Continuous bag-of-words
 - Skip-gram
 - ...

$$P(w_n | w_1 \cdots w_{n-1}) = f(e_1, \cdots, e_{n-1})$$

连续空间语言模型的学习

- 我们如何学习词汇表中每个单词的单词表示 z ?
- 我们如何在给定单词历史情况下，预测下一个单词或其表示 \hat{z}_t 的模型
- 同时学习模型和表示

词的向量空间表示

- 使用向量表示比较两个词：

- 点积
- 余弦相似度
- 欧氏距离

- 位置 t 处的双线性评分函数：

- 模型（参数记为 θ ）预测下一个词
- 用于计算单词的 v 的概率的偏差 b_v
- 给定一个预测向量 $\hat{\mathbf{z}}_t$ ，实际预测的词是最近邻的词 \hat{z}_t
- 在大词汇表（包含百万词）中搜索在计算上可能很昂贵……

$$s(\mathbf{w}_1^{t-1}, v; \theta) = s(\hat{\mathbf{z}}_t, v) = s_\theta(v) = \hat{\mathbf{z}}_t^T \mathbf{z}_v + b_v$$

词的向量空间表示

- 归一化的概率:

- 使用 softmax 函数

$$P(w_t = v \mid \mathbf{w}_1^{t-1}) = \frac{e^{s(\hat{\mathbf{z}}_t, v)}}{\sum_{v'=1}^V e^{s(\hat{\mathbf{z}}_t, v')}}}$$

- 位置 t 处的双线性评分函数:

$$s(\mathbf{w}_1^{t-1}, v; \boldsymbol{\theta}) = s(\hat{\mathbf{z}}_t, v) = s_{\boldsymbol{\theta}}(v) = \hat{\mathbf{z}}_t^T \mathbf{z}_v + b_v$$

- 模型（参数记为 $\boldsymbol{\theta}$ ）预测下一个词
 - 用于计算单词的 v 的概率的偏差 b_v
 - 给定一个预测向量 $\hat{\mathbf{z}}_t$ ，实际预测的词是最近邻的词 $\hat{\mathbf{z}}_t$
 - 在大词汇表（包含百万词）中搜索在计算上可能很昂贵

损失函数

- 对数似然模型：
 - 数值更稳定

$$\log P(w_1, w_2, \dots, w_{t-1}, w_T) = \log \left(\prod_{t=1}^T P(w_t | \mathbf{w}_1^{t-1}) \right) = \sum_{t=1}^T \log P(w_t | \mathbf{w}_1^{t-1})$$

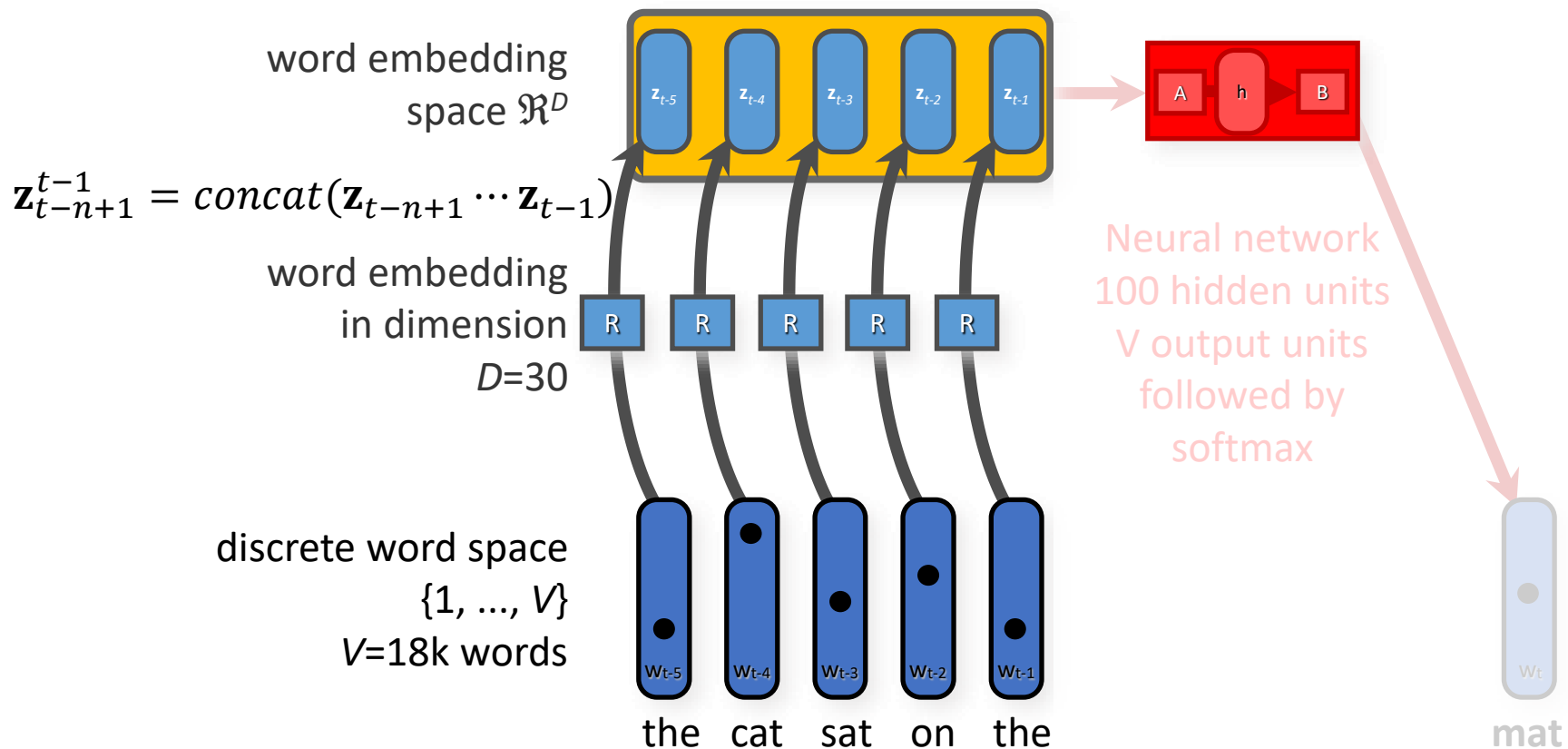
- 最大化损失函数：
 - 对数似然

$$P(w_t = w | \mathbf{w}_1^{t-1}) = \frac{e^{s_{\theta}(w)}}{\sum_{v=1}^V e^{s_{\theta}(v)}}$$

$$L_t = \log P(w_t = w | \mathbf{w}_1^{t-1}) = s_{\theta}(w) - \log \sum_{v=1}^V e^{s_{\theta}(v)}$$

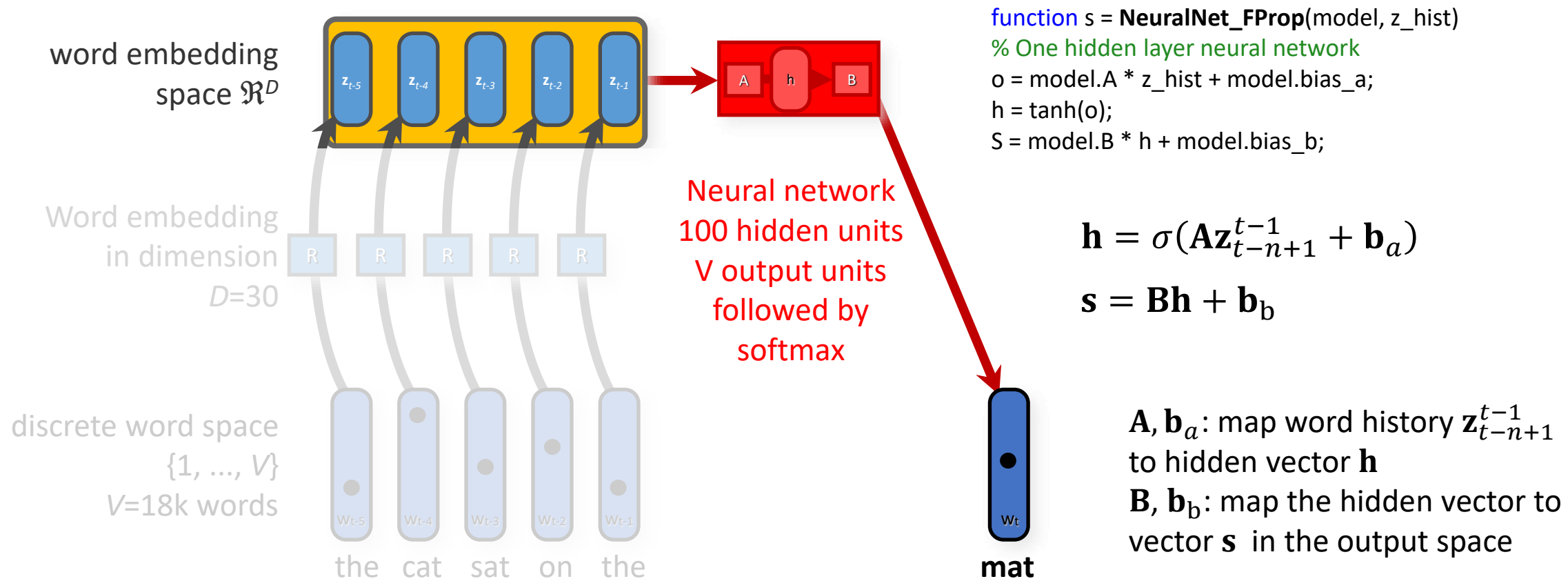
- 一般来说，损失定义为：正确答案的分数+归一化项
- 归一化项的计算成本很高

神经网络语言模型

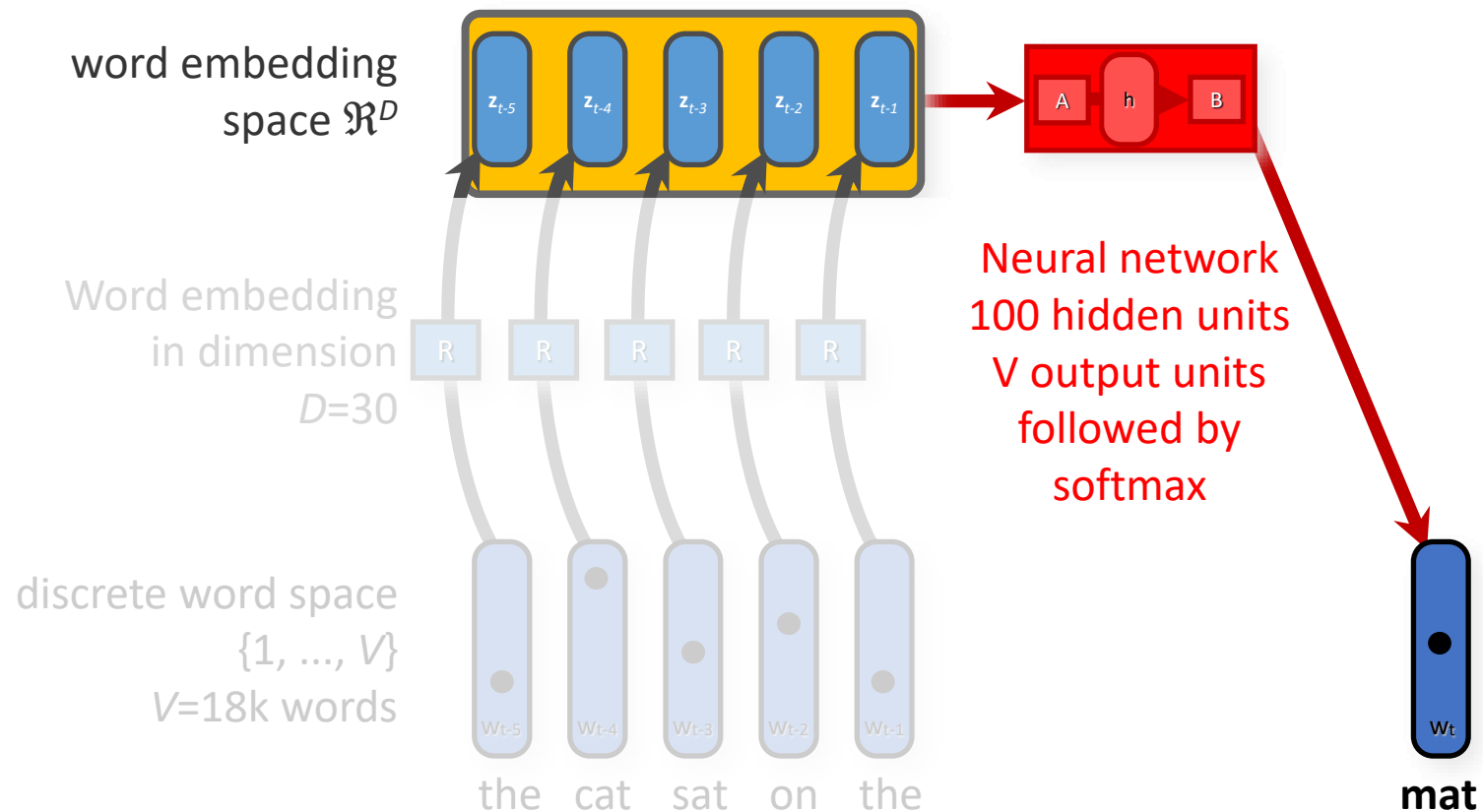


```
function z_hist = Embedding_FProp(model, w)
% Get the embeddings for all words in w
z_hist = model.R(:, w);
z_hist = reshape(z_hist, length(w)*model.dim_z, 1);
```

神经网络语言模型



神经网络语言模型



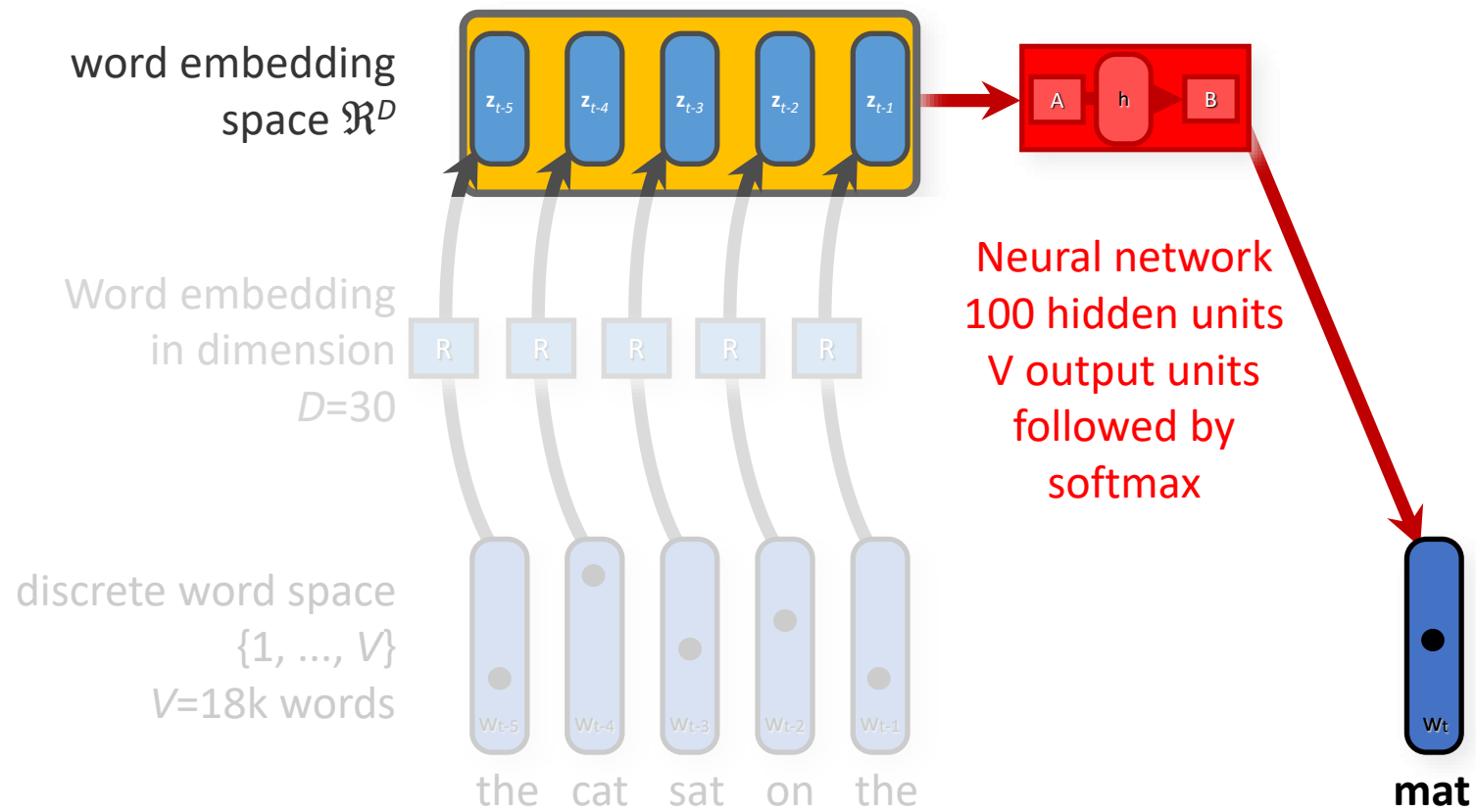
```
function p = Softmax_FProp(s)
% Probability estimation
p_num = exp(s);
p = p_num / sum(p_num);
```

$$P(w_t | \mathbf{w}_{t-n+1}^{t-1}) = \frac{e^{s_\theta(w_t)}}{\sum_v e^{s_\theta(v)}}$$

$P(w_t | \mathbf{w}_{t-n+1}^{t-1})$: the probability of word w_t given the word history $w_{t-n+1} \cdots w_{t-1}$ in a window size

$s_\theta(v)$: the value of the position in \mathbf{s} corresponding to the word v in the vocabulary

神经网络语言模型



Complexity: $(n-1) \times D + (n-1) \times D \times H + H \times V$

$$\mathbf{h} = \sigma(\mathbf{A}\mathbf{z}_{t-n+1}^{t-1} + \mathbf{b}_a)$$

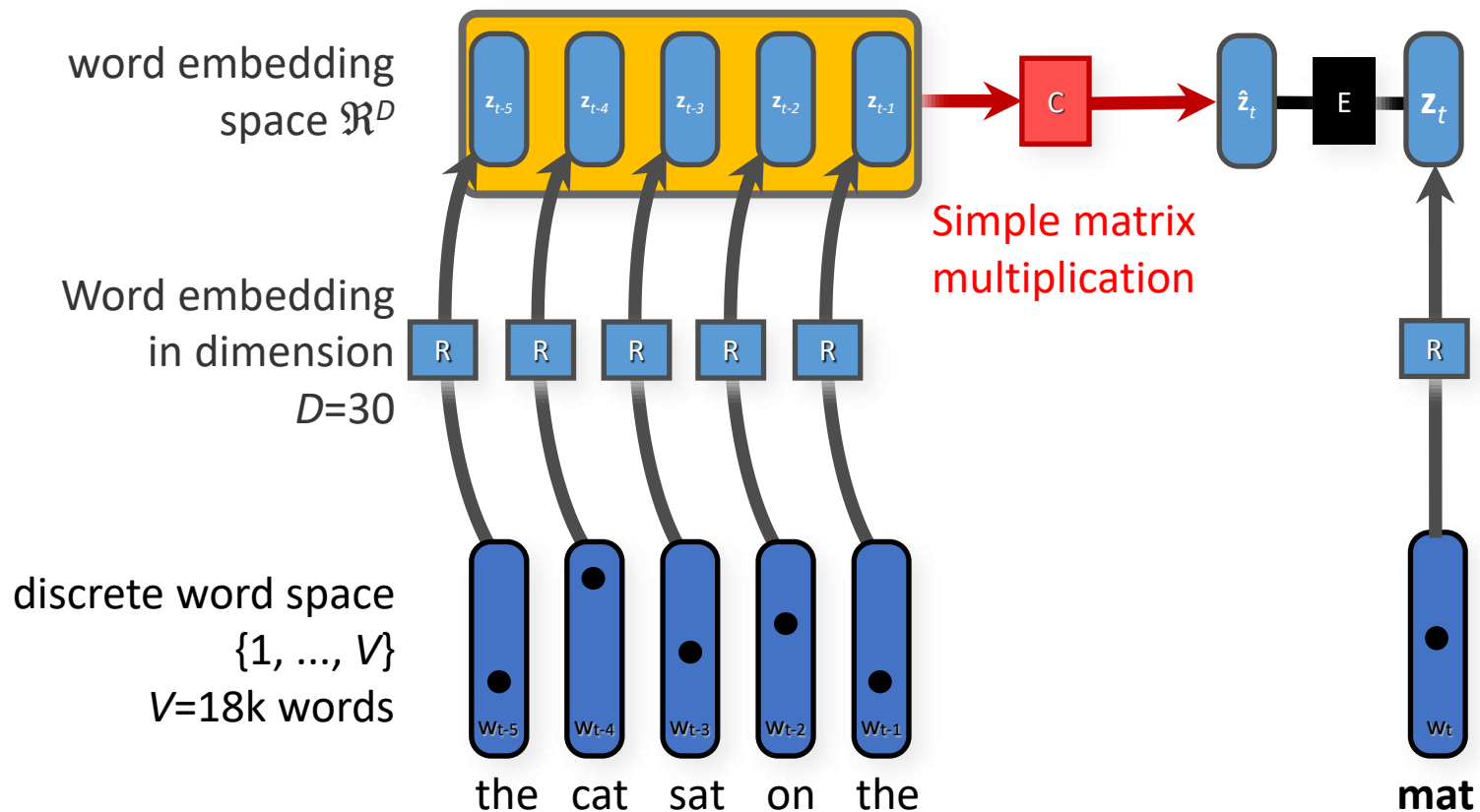
$$\mathbf{s} = \mathbf{B}\mathbf{h} + \mathbf{b}_b$$

$$P(w_t | \mathbf{w}_{t-n+1}^{t-1}) = \frac{e^{s_\theta(w_t)}}{\sum_v e^{s_\theta(v)}}$$

Outperforms best n-grams
(Class-based Kneser-Ney
back-off 5-grams) by 7%

Took months to train
(in 2001-2002) on AP News
corpus (14M words)

神经网络语言模型

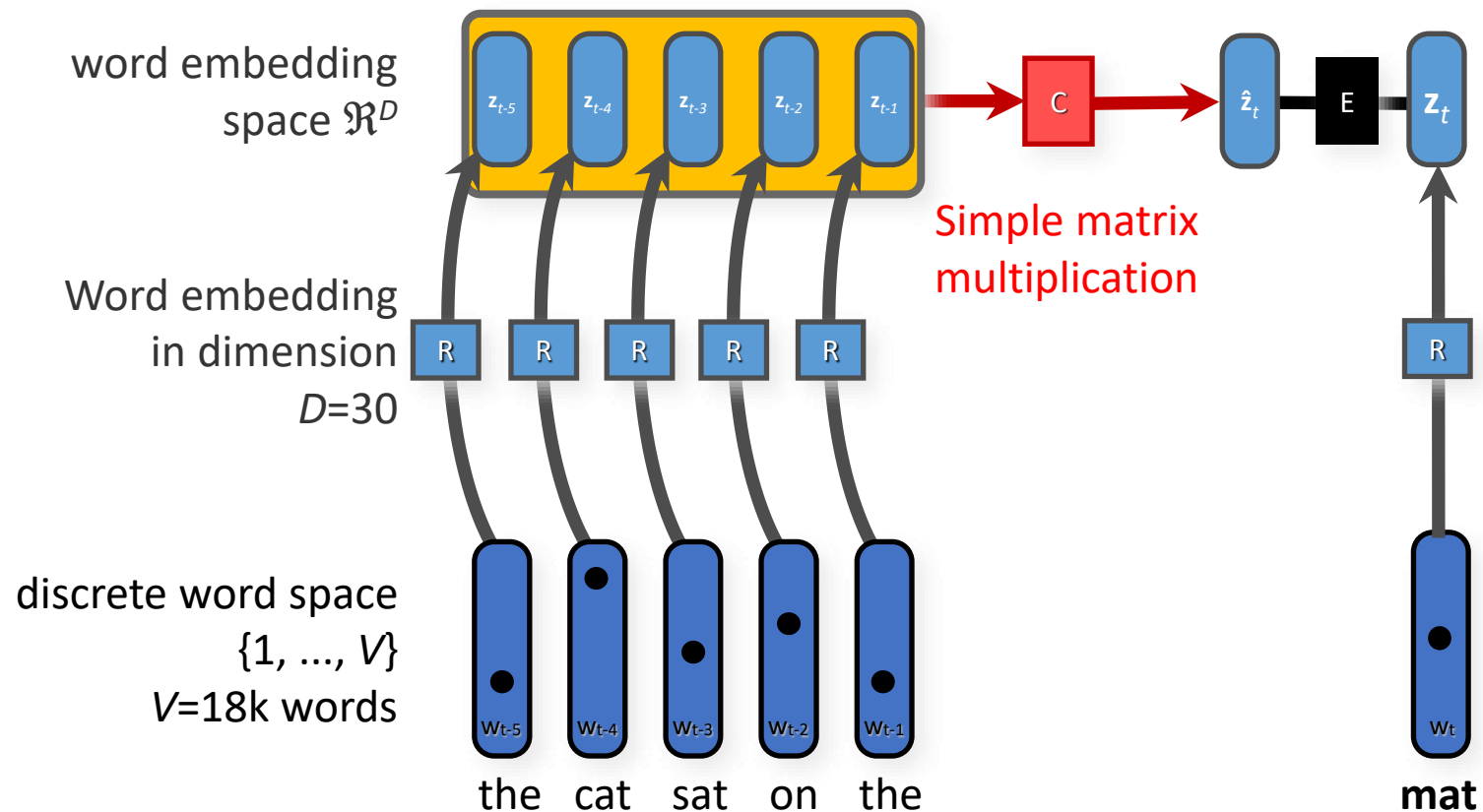


```
function z_hat = FProp(model, z_hist)
% Simple linear transform
Z_hat = model.C * z_hist + model.bias_c;
```

$$\hat{\mathbf{z}}_t = \mathbf{C}\mathbf{z}_{t-n+1}^{t-1} + \mathbf{b}_c$$

Perform a linear transformation of \mathbf{z}_{t-n+1}^{t-1} with the same dimension

神经网络语言模型



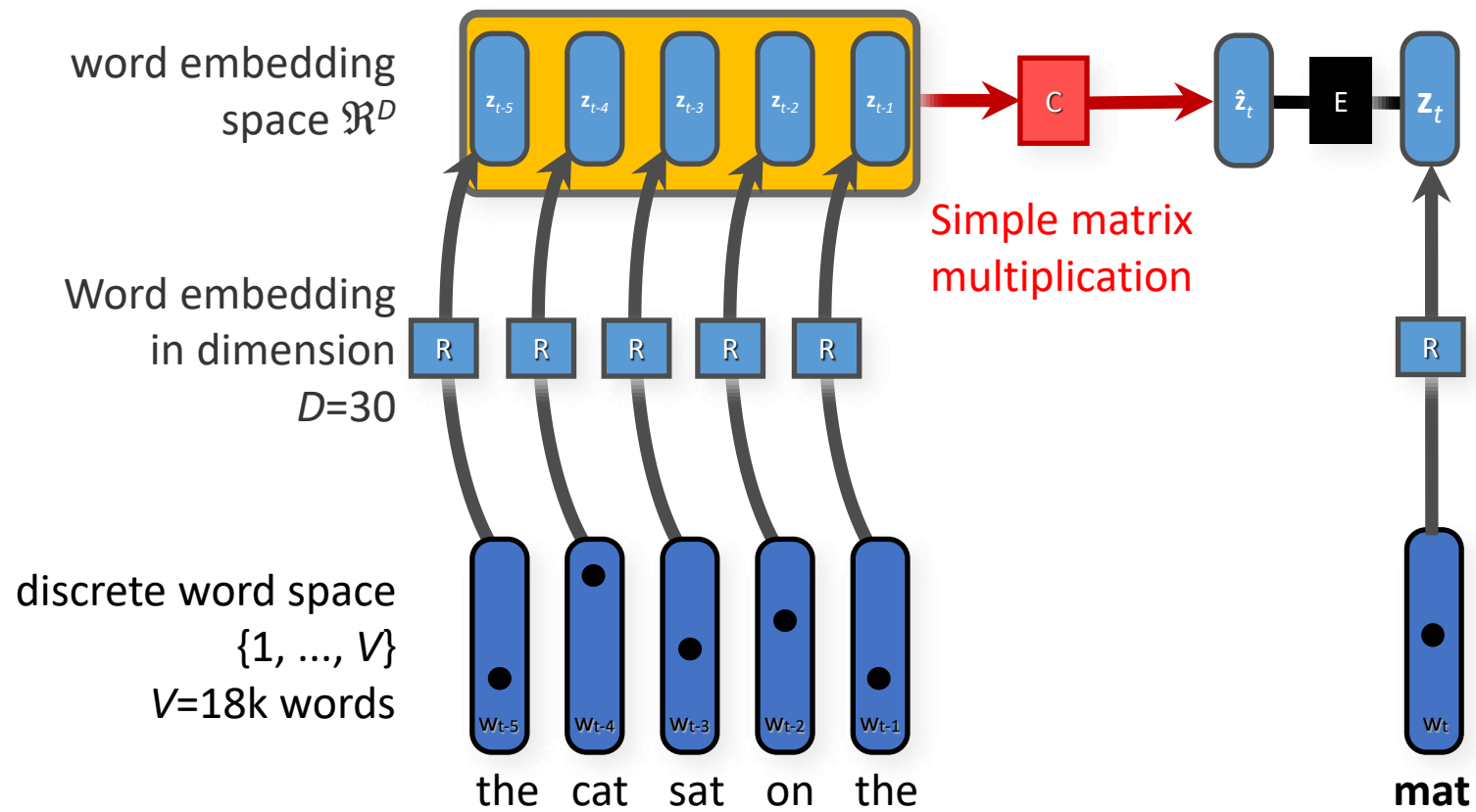
function $s = \dots$
Score_FProp(z_hat , model)
 $s = \text{model.R}' * z_hat + \text{model.bias}_v$

$$s_{\theta}(v) = \hat{\mathbf{z}}_t^T \mathbf{z}_v + \mathbf{b}_v$$

$$P(w_t | \mathbf{w}_{t-n+1}^{t-1}) = \frac{e^{s_{\theta}(w_t)}}{\sum_v e^{s_{\theta}(v)}}$$

$\hat{\mathbf{z}}_t^T$ is the transpose of $\hat{\mathbf{z}}_t$
 \mathbf{z}_v is the word vector of word v
 \mathbf{b}_v is the bias vector of word v

神经网络语言模型



$$\hat{\mathbf{z}}_t = \mathbf{C}\mathbf{z}_{t-n+1}^{t-1} + \mathbf{b}_c$$

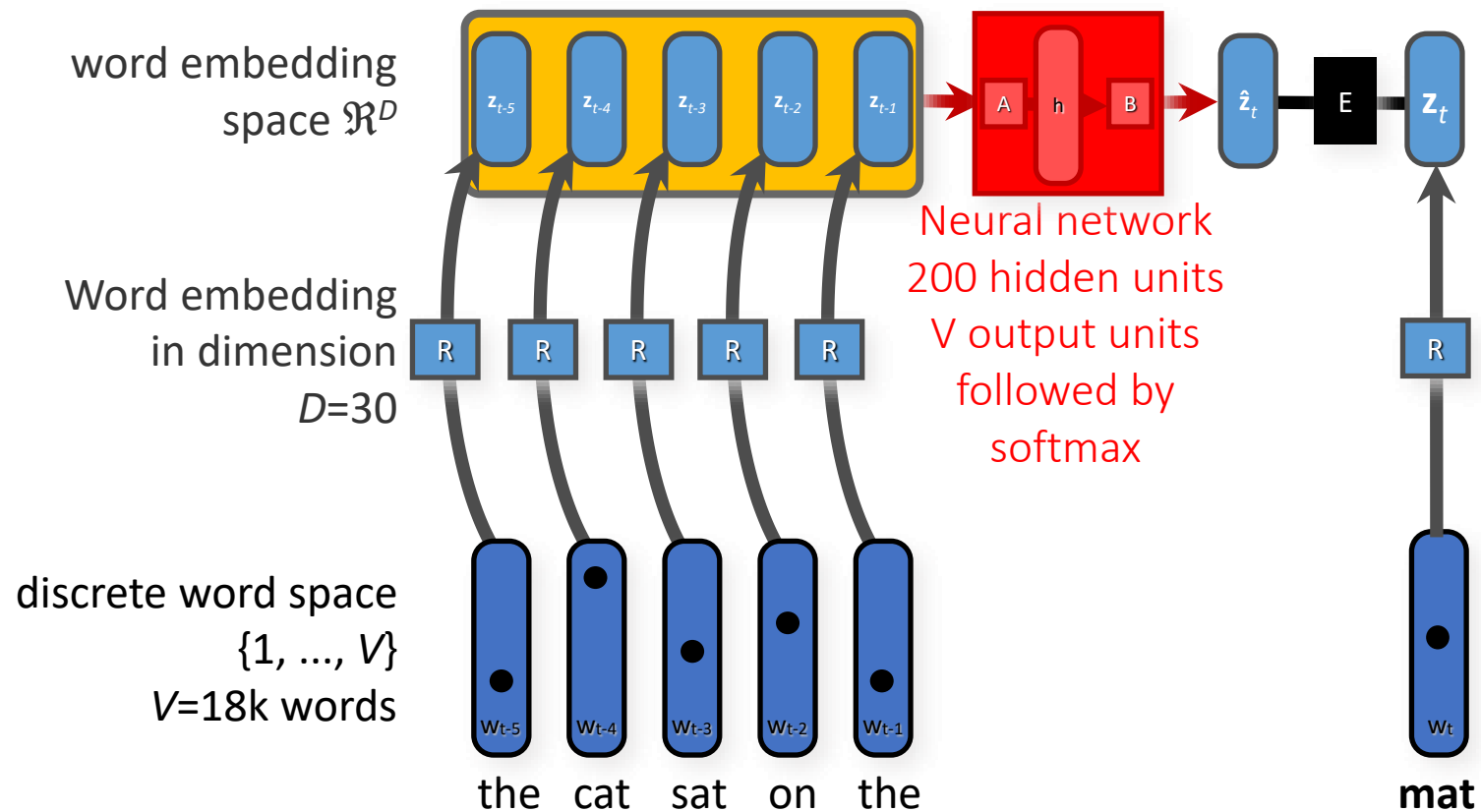
$$s_\theta(v) = \hat{\mathbf{z}}_t^T \mathbf{z}_v + \mathbf{b}_v$$

$$P(w_t | \mathbf{w}_{t-n+1}^{t-1}) = \frac{e^{s_\theta(w_t)}}{\sum_v e^{s_\theta(v)}}$$

Slightly better than
best n-grams
(Class-based Kneser-Ney
back-off 5-grams)

Takes days to train
(in 2007) on AP News
corpus (14 million words)

神经网络语言模型



Complexity: $(n-1) \times D + (n-1) \times D \times H + H \times D + D \times V$

$$\mathbf{h} = \sigma(\mathbf{A}\mathbf{z}_{t-n+1}^{t-1} + \mathbf{b}_a)$$

$$\hat{\mathbf{z}}_t = \mathbf{B}\mathbf{h} + \mathbf{b}_b$$

$$s_\theta(v) = \hat{\mathbf{z}}_t^T \mathbf{z}_v + \mathbf{b}_v$$

$$P(w_t | \mathbf{w}_{t-n+1}^{t-1}) = \frac{e^{s_\theta(w_t)}}{\sum_v e^{s_\theta(v)}}$$

Outperforms best n-grams
(Class-based Kneser-Ney
back-off 5-grams) by 24%

Took weeks to train
(in 2009-2010) on AP News
corpus (14M words)

神经网络语言模型

- \mathbf{z}_{t-n+1}^{t-1} : 在滑动窗口中的上下文词的表征
- A, b_a : 用于把 \mathbf{z}_{t-n+1}^{t-1} 映射为高维向量 h 的参数
- B, b_b : 把 h 映射回低维向量的参数
- b_v : 词 v 的偏差向量
- softmax: 归一化输出
- \mathbf{z}_{t-n+1}^{t-1} 和 A 对神经语言模型十分重要!

$$\mathbf{h} = \sigma(\mathbf{A}\mathbf{z}_{t-n+1}^{t-1} + \mathbf{b}_a)$$

$$\hat{\mathbf{z}}_t = \mathbf{B}\mathbf{h} + \mathbf{b}_b$$

$$s_\theta(v) = \hat{\mathbf{z}}_t^T \mathbf{z}_v + \mathbf{b}_v$$

$$P(w_t | \mathbf{w}_{t-n+1}^{t-1}) = \frac{e^{s_\theta(w_t)}}{\sum_v e^{s_\theta(v)}}$$

神经网络语言模型

- \mathbf{z}_{t-n+1}^{t-1}
 - 决定了一个词能承载多少信息
 - 一般来说，较低的维度意味着较少的信息；较高的维度使其更难衡量词语之间的相似性
 - 需要找到一个合适的语境窗口和词的表示维度
- A :
 - 决定了隐藏向量 \mathbf{h} 的质量，从而进一步影响神经语言模型的质量。
 - 人们可以通过改善 A 来获得一个更好的神经语言模型
 - 使用另一个架构，添加额外的知识，...

$$\mathbf{h} = \sigma(\mathbf{A}\mathbf{z}_{t-n+1}^{t-1} + \mathbf{b}_a)$$

$$\hat{\mathbf{z}}_t = \mathbf{B}\mathbf{h} + \mathbf{b}_b$$

$$s_\theta(v) = \hat{\mathbf{z}}_t^T \mathbf{z}_v + \mathbf{b}_v$$

$$P(w_t | \mathbf{w}_{t-n+1}^{t-1}) = \frac{e^{s_\theta(w_t)}}{\sum_v e^{s_\theta(v)}}$$

神经网络语言模型的学习过程

- 找到神经语言模型的参数 θ ，使其最大化观察到的数据的对数可能性。

$$L_t = \log P(w_t = w | \mathbf{w}_1^{t-1}) = s_{\theta}(w) - \log \sum_{v=1}^V e^{s_{\theta}(v)}$$

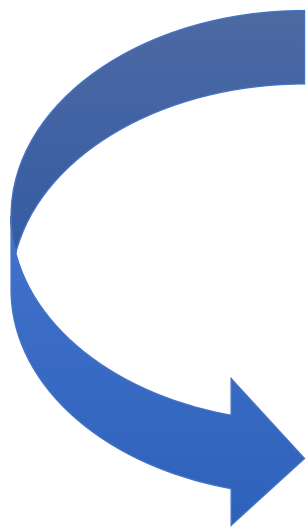
$$\arg \max_{\theta} (\log P(w_t = w | \mathbf{w}_1^{t-1}, \theta))$$

- 神经语言模型的参数 θ :
 - 词向量矩阵 R 和偏差向量 b_v
 - 权重: A, b_a, B, b_b
- 梯度下降算法更新参数 (η 是学习率) : $\theta \leftarrow \theta - \eta \frac{\partial L_t}{\partial \theta}$

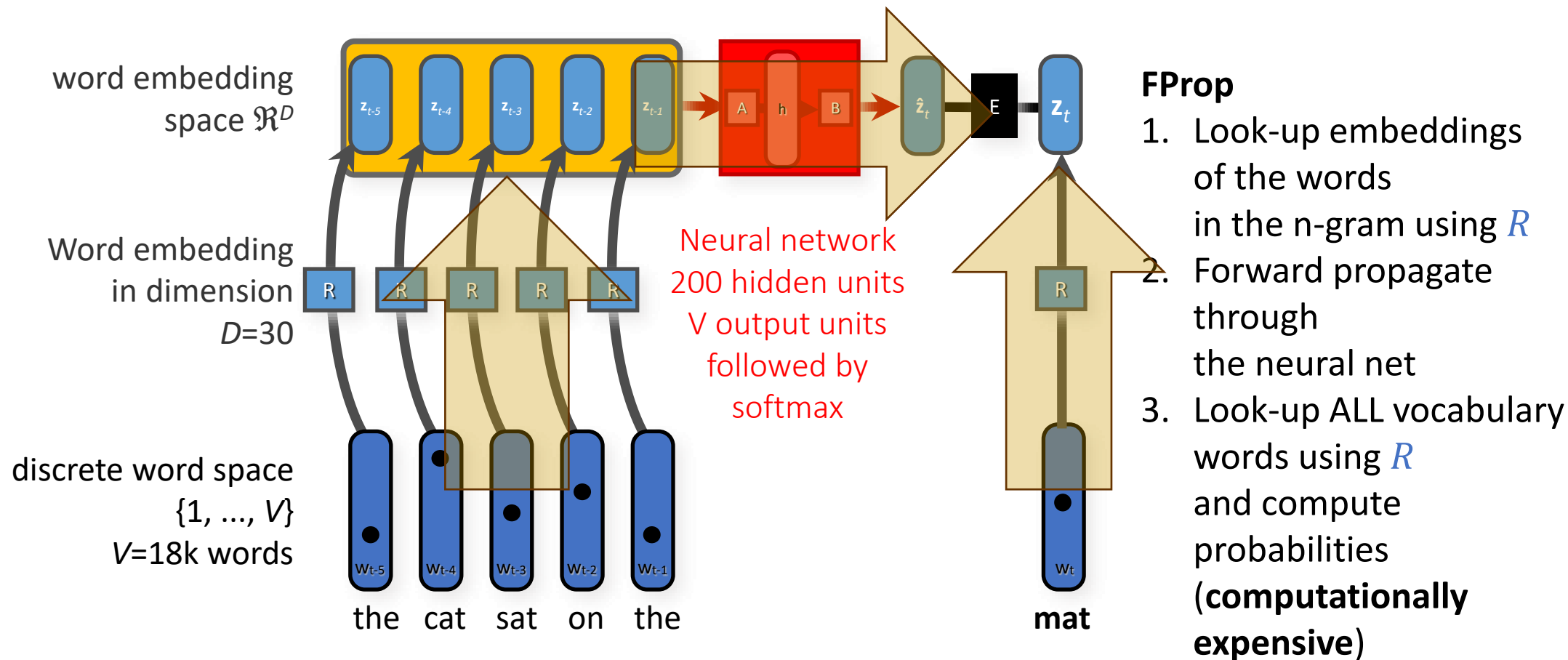
神经网络语言模型的学习过程

随机选择一个小批量
(例如, 1000个连续字)。

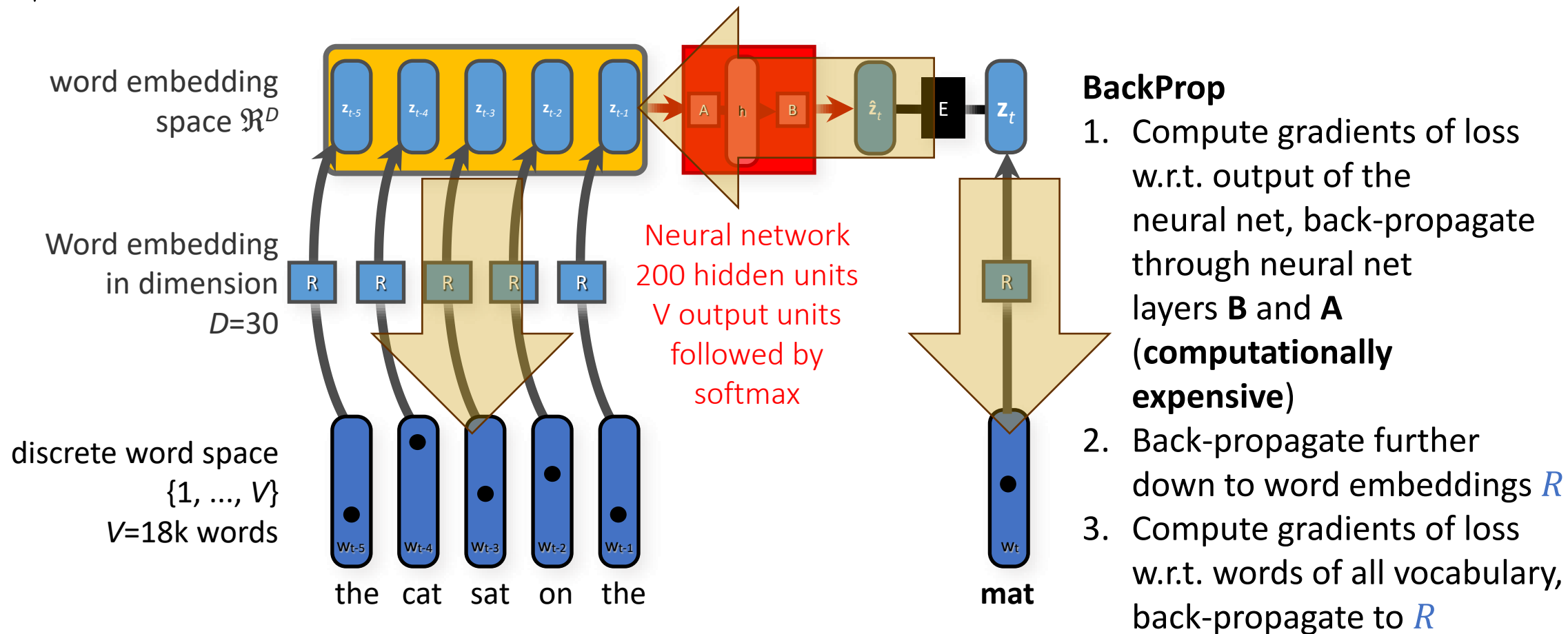
1. 通过词嵌入和模型进行正向传播
2. 估计词的可能性 (损失)。
3. 反向传播损失
4. 梯度步骤来更新模型



神经网络语言模型的前向过程



神经网络语言模型的反向传播过程



神经网络语言模型的特点

神经网络语言模型的特点

- 神经网络语言模型是否需要平滑？
- 神经网络语言模型如何处理未见过的词？
- 神经网络语言模型如何捕捉长期的n-gram依赖关系？

神经网络语言模型是否需要平滑？

- 在n-gram 语言模型中，我们需要平滑来处理未见过的n-grams

$$P(w_t | w_{t-n+1} \cdots w_{t-1}) = \frac{\text{count}(w_{t-n+1} \cdots w_t)}{\text{count}(w_{t-n+1} \cdots w_{t-1})}$$

← 未见过的 N-gram


- 两种情况：
 - 整个 $w_{t-n+1} \cdots w_{t-1}$ 在训练集中未出现，但是每个单独的词 $w_{t-n+1} \cdots w_{t-1}$ 在训练集中都出现了
 - $w_{t-n+1} \cdots w_{t-1}$ 中的一个或多个没有出现在训练集中

神经网络语言模型如何处理未登录N-grams

- 滑动窗口中的 N-gram $w_{t-n+1} \cdots w_{t-1}$ 没有见过
- 我们是否需要平滑?
- 不需要

词语用向量表示

- \mathbf{z}_{t-n+1}^{t-1} 是从 $w_{t-n+1} \cdots w_{t-1}$ 中的每个单字的词向量得到。
- 由于每个词都出现在训练数据中，我们可以获得它们的向量并计算出 \mathbf{z}_{t-n+1}^{t-1}



$$\mathbf{h} = \sigma(\mathbf{A}\mathbf{z}_{t-n+1}^{t-1} + \mathbf{b}_a)$$

$$\hat{\mathbf{z}}_t = \mathbf{B}\mathbf{h} + \mathbf{b}_b$$

$$s_\theta(v) = \hat{\mathbf{z}}_t^T \mathbf{z}_v + b_v$$

$$P(w_t | \mathbf{w}_{t-n+1}^{t-1}) = \frac{e^{s_\theta(w_t)}}{\sum_v e^{s_\theta(v)}}$$

神经网络语言模型是如何处理未登录词

- 如果 $w_{t-n+1} \cdots w_{t-1}$ 中包括一个或多个未见过的词怎么办？
- 该模型将无法找到它的词向量
- 但我们不希望模型崩溃
- 一般地，人们用一个特殊的词 “<UNK>” 来表示未见过的词。
- 我 买 了 一 串 葡 萄  未见过的词
- 我 买 了 一 串 <UNK>

神经网络语言模型如何捕捉长距离N-gram的依赖关系

- 使用滑动窗口来选择历史词汇
 - 使用较大的窗口尺寸来捕捉较长的距离信息
- 然而，只考虑到固定大小的历史词

神经网络语言模型的增强

神经网络语言模型的增强

- 可以添加哪些额外的信息来增强神经网络语言模型？
- 我们能否改进前馈神经网络语言模型的结构或算法？
- 我们可以增强 A

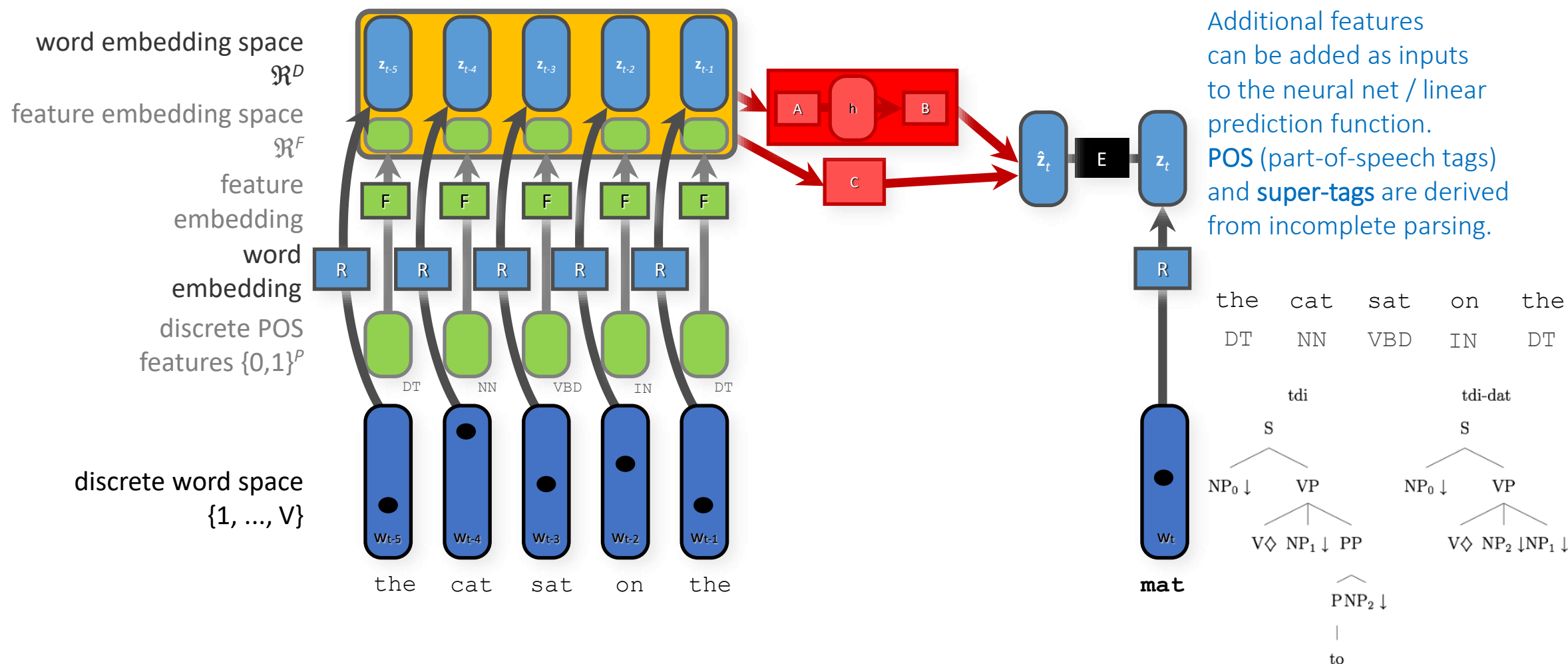
$$\mathbf{h} = \sigma(\mathbf{A}\mathbf{z}_{t-n+1}^{t-1} + \mathbf{b}_a)$$

$$\hat{\mathbf{z}}_t = \mathbf{B}\mathbf{h} + \mathbf{b}_b$$

$$s_{\theta}(v) = \hat{\mathbf{z}}_t^T \mathbf{z}_v + b_v$$

$$P(w_t \mid \mathbf{w}_{t-n+1}^{t-1}) = \frac{e^{s_{\theta}(w_t)}}{\sum_v e^{s_{\theta}(v)}}$$

将语言特征添加到神经网络语言模型中

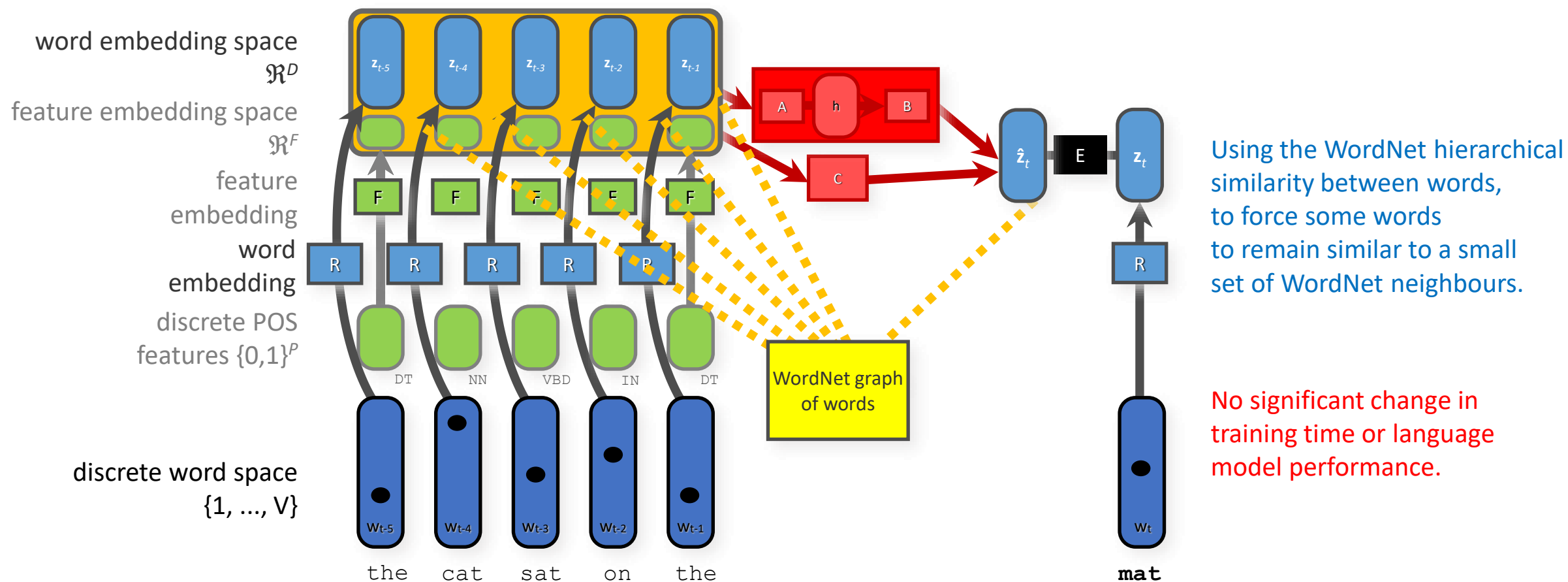


[Mirowski, Chopra, Balakrishnan and Bangalore (2010)

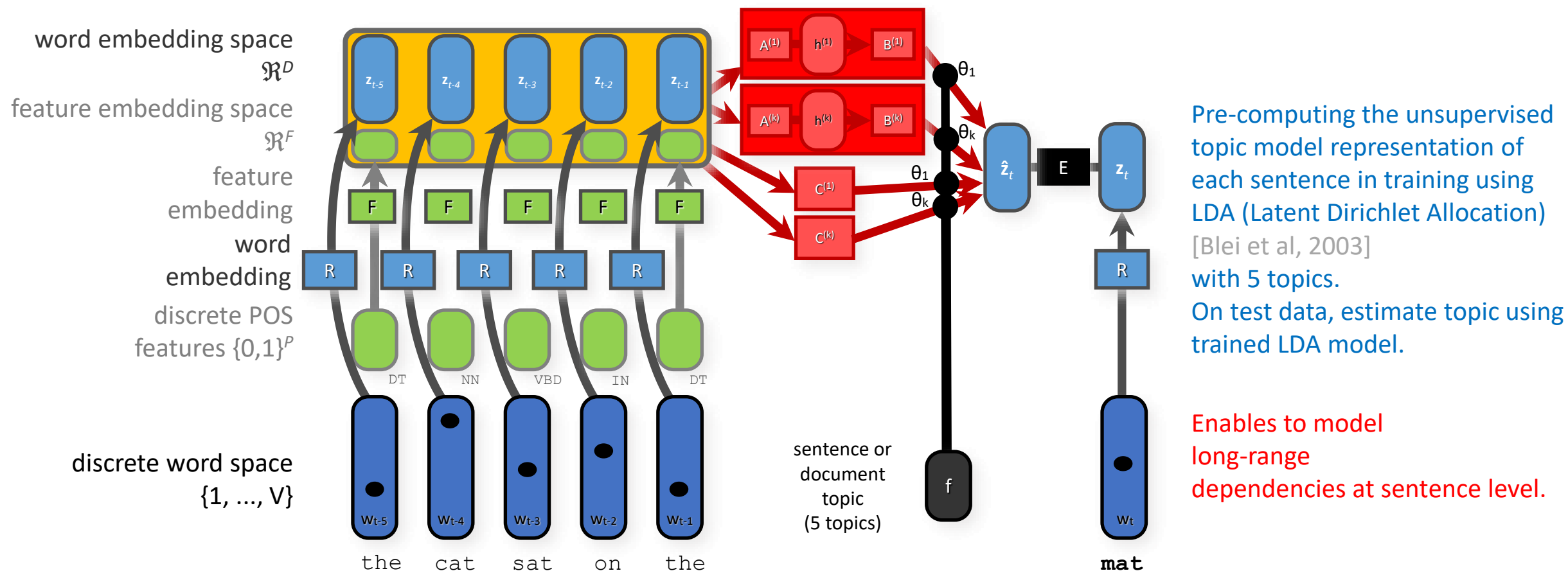
“Feature-rich continuous language models for speech recognition”, *SLT*;

Bangalore & Joshi (1999) “Supertagging: an approach to almost parsing”, *Computational Linguistics*]

约束词的表征



混合主题的语言模型

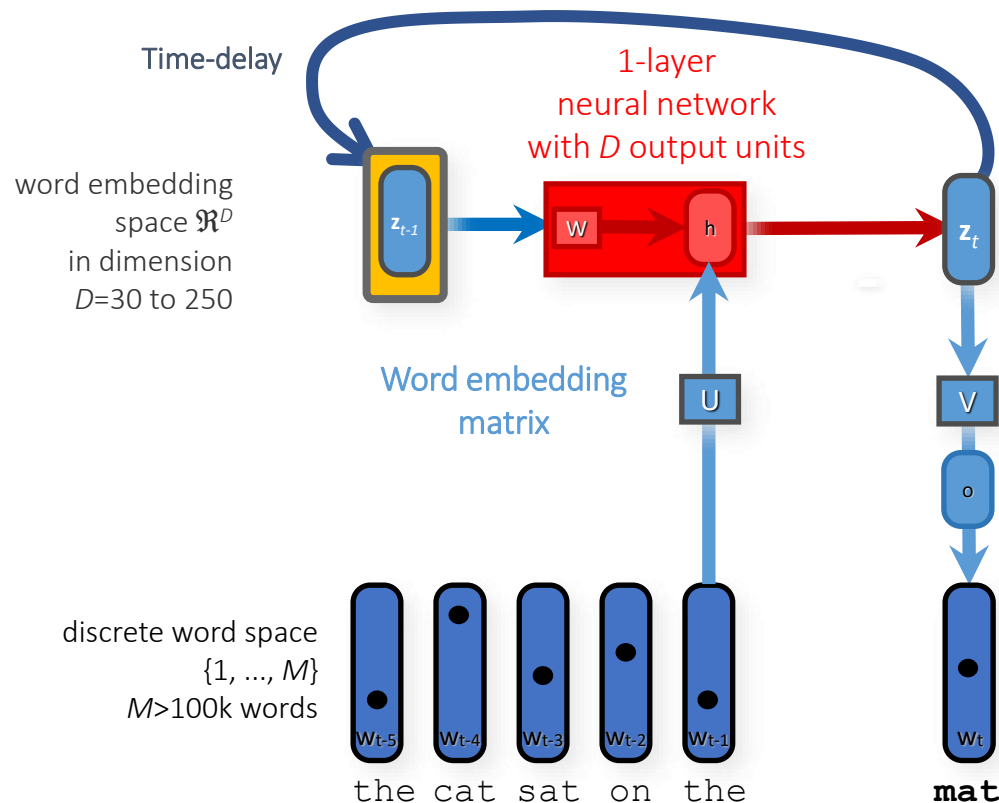


[Mirowski, Chopra, Balakrishnan and Bangalore (2010)
 "Feature-rich continuous language models for speech recognition", *SLT*;
 David Blei (2003) "Latent Dirichlet Allocation", *JMLR*]

进一步的改进？

- 之前介绍的神经LM是基于MLP的，它有一些优点和缺点
 - 结构简单
 - 高效的并行训练
 - 不能处理语言结构
 - 需要一个词汇量大小的输入维度
- MLP的一些技巧（特别是在NLP方面）。
 - 对于大多数任务，使用1-2个隐藏层
 - 最好作为一个集合模型使用
- 通过使用先进的架构可以获得进一步的改进：例如，循环神经网络（RNN）。

RNN 语言模型



Complexity: $D \times D + D \times D + D \times V$

$$\mathbf{z}_t = \sigma(\mathbf{W}\mathbf{z}_{t-1} + \mathbf{U}\mathbf{w}_t)$$

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

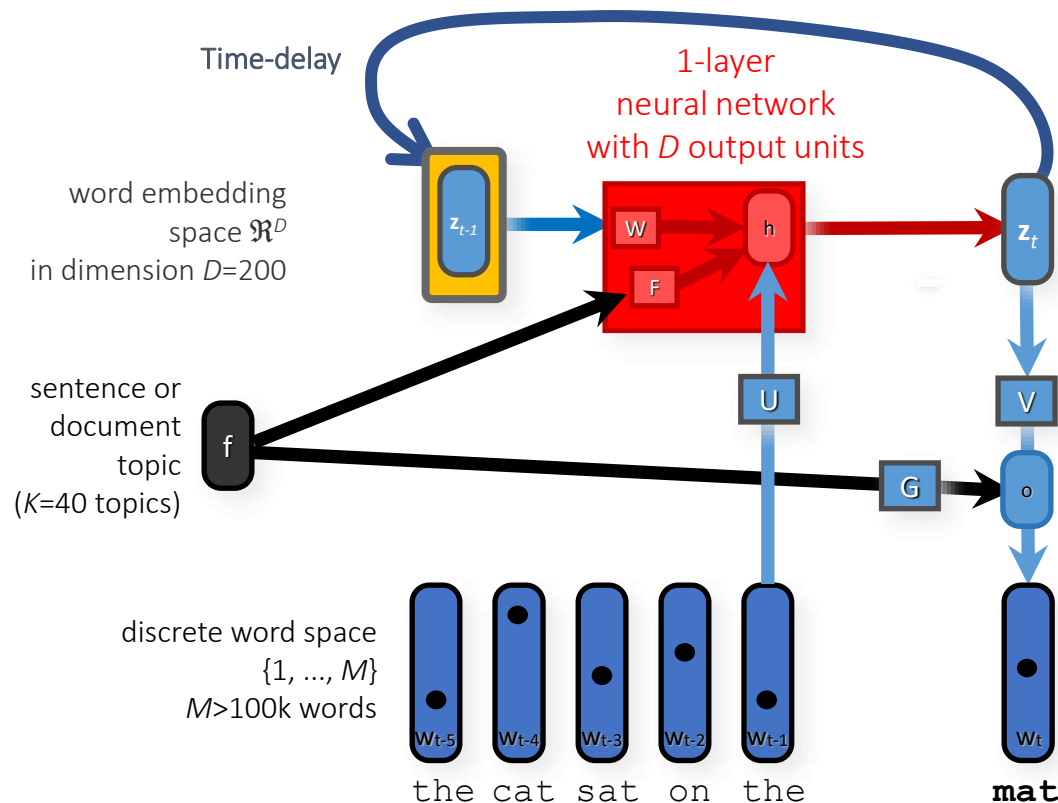
$$\mathbf{o} = \mathbf{V}\mathbf{z}_t$$

$$P(w_t | \mathbf{w}_{t-n+1}^{t-1}) = \mathbf{y}_t = \frac{e^{o(w)}}{\sum_v e^{o(v)}}$$

Handles longer word history
(~10 words) as well
as 10-gram feed-forward NNLM

Training algorithm: BPTT
Back-Propagation Through Time

依赖上下文的 RNN 语言模型



$$\mathbf{z}_t = \sigma(\mathbf{W}\mathbf{z}_{t-1} + \mathbf{U}\mathbf{w}_t + \mathbf{F}\mathbf{f}_t)$$

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

$$\mathbf{o} = \mathbf{V}\mathbf{z}_t + \mathbf{G}\mathbf{f}_t$$

$$P(w_t | \mathbf{w}_{t-n+1}^{t-1}) = \mathbf{y}_t = \frac{e^{o(w)}}{\sum_v e^{o(v)}}$$

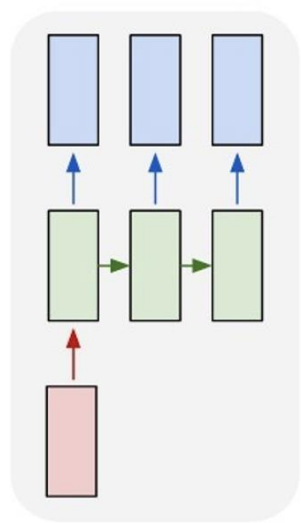
Compute topic model representation word-by-word on last 50 words using approximate LDA [Blei et al, 2003] with K topics. Enables to model long-range dependencies at sentence level.

循环神经网络（RNN）

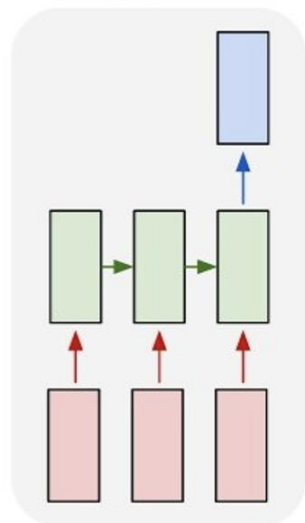
- 语言是一个在时间上展开的序列
- RNN有一个直接处理语言的顺序性的机制，允许它们处理语言的时间性，而不使用任意的固定大小的窗口
- RNN提供了一种新的方式来表示先前的语境，它允许模型的决定取决于过去数百个词的信息

循环神经网络（RNN）

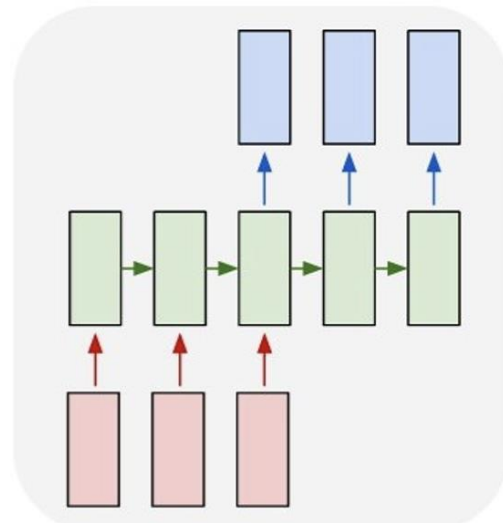
- 不同种类的RNN



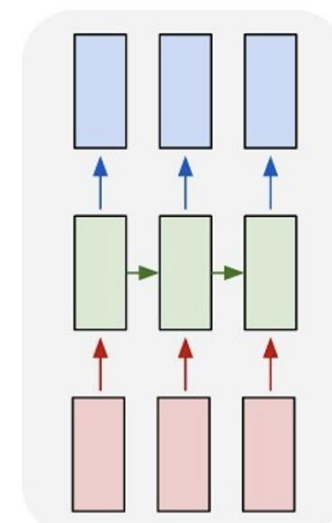
一对多



多对一



多对多



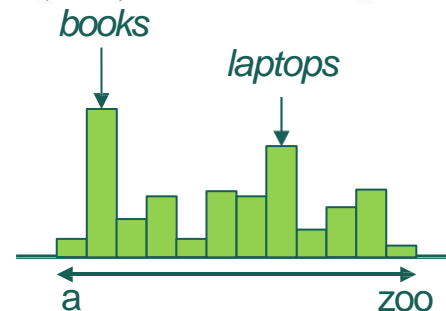
多对多

一个简单的RNN语言模型

输出分布

$$\hat{y}^{(t)} = \text{softmax} \left(U h^{(t)} + b_2 \right) \in \mathbb{R}^{|V|}$$

$$\hat{y}^{(4)} = P(x^{(5)} | \text{the students opened their})$$



隐向量

$$h^{(t)} = f(W_h h^{(t-1)} + W_e x^{(t)} + b_1)$$

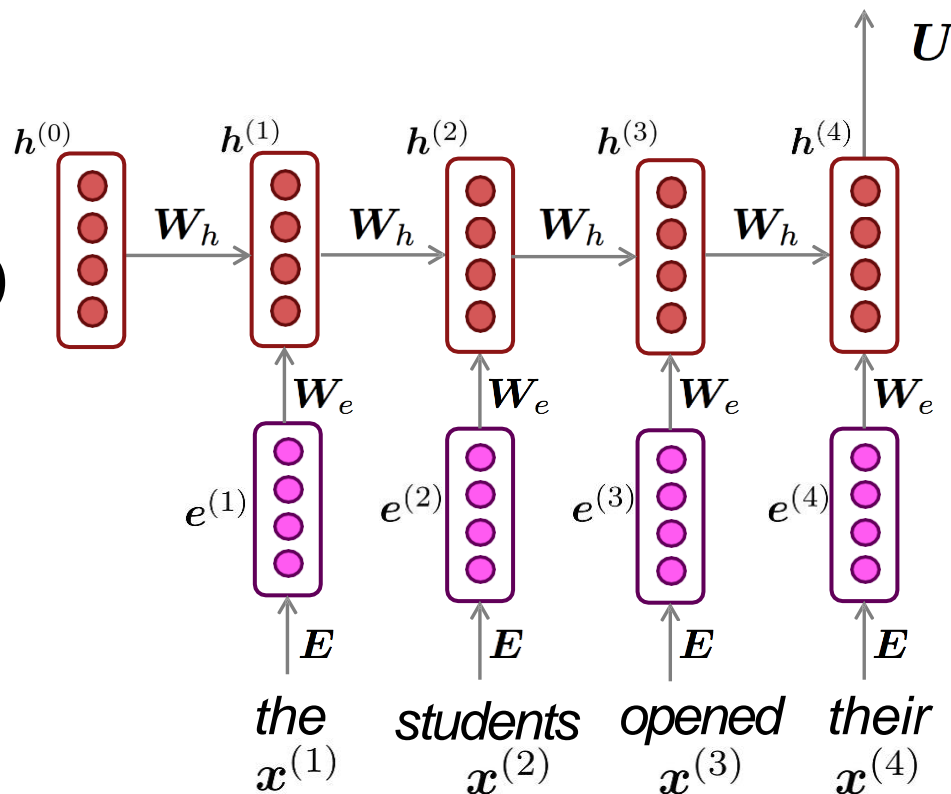
$h^{(0)}$ 是最初的隐向量

词向量

$$e^{(t)} = E x^{(t)}$$

词的 one-hot 向量

$$x^{(t)} \in \mathbb{R}^{|V|}$$



注意：这个输入序列可以很长

RNN 语言模型：优势和劣势

- RNN 优势（与使用滑动窗口的 NLM 相比）：
 - 可以处理任何长度的输入
 - 第 t 步的计算（理论上）可以使用前面许多步的信息
 - 在较长的输入背景下，模型大小不会增加
 - 在每个时间点上应用相同的权重，因此在输入的处理方式上是一致的
- RNN 劣势：
 - 循环计算很慢
 - 在实践中，很在许多步骤后获取先前的信息（后面会详细介绍）。

训练一个 RNN 语言模型

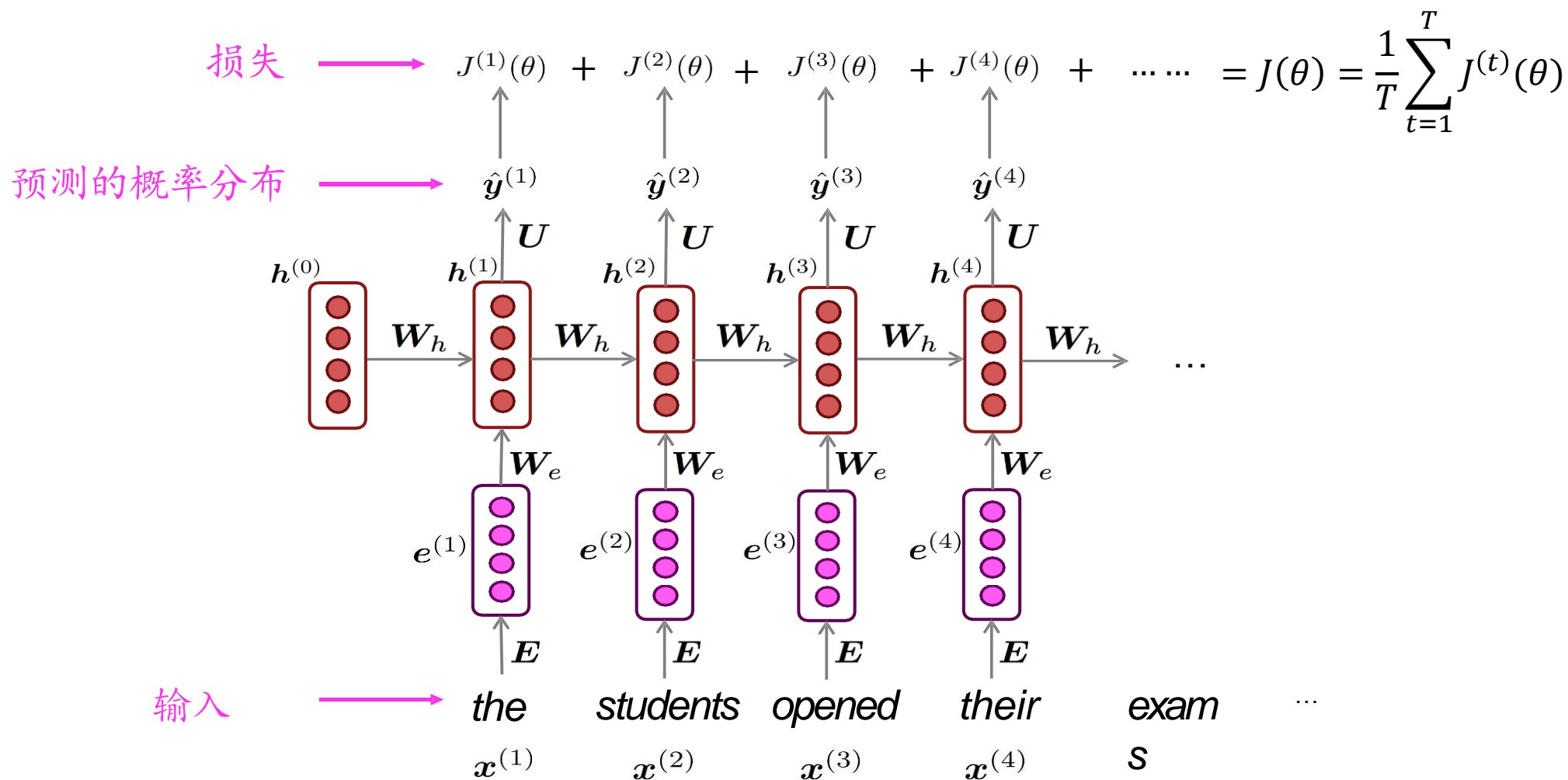
- 整个语料表示为词序列 $x^{(1)} \dots x^{(T)}$
- 把词序列送入 RNN 语言模型；对每个时刻 t 计算输出的分布 $\hat{y}^{(t)}$
 - 给定已经生成的词，预测当前单词的概率分布。
- 时刻 t 的损失是预测分布 $\hat{y}^{(t)}$ 和实际分布 y^t ($x^{(t+1)}$ 的 one-hot 表示) 的交叉熵：

$$J^{(t)}(\theta) = CE(\mathbf{y}^{(t)}, \hat{\mathbf{y}}^{(t)}) = - \sum_{w \in V} \mathbf{y}_w^{(t)} \log \hat{\mathbf{y}}_w^{(t)} = - \log \hat{\mathbf{y}}_{\mathbf{x}_{t+1}}^{(t)}$$

- 计算平局以得到整个训练集的损失：

$$J(\theta) = \frac{1}{T} \sum_{t=1}^T J^{(t)}(\theta) = \frac{1}{T} \sum_{t=1}^T - \log \hat{\mathbf{y}}_{\mathbf{x}_{t+1}}^{(t)}$$

训练一个 RNN 语言模型



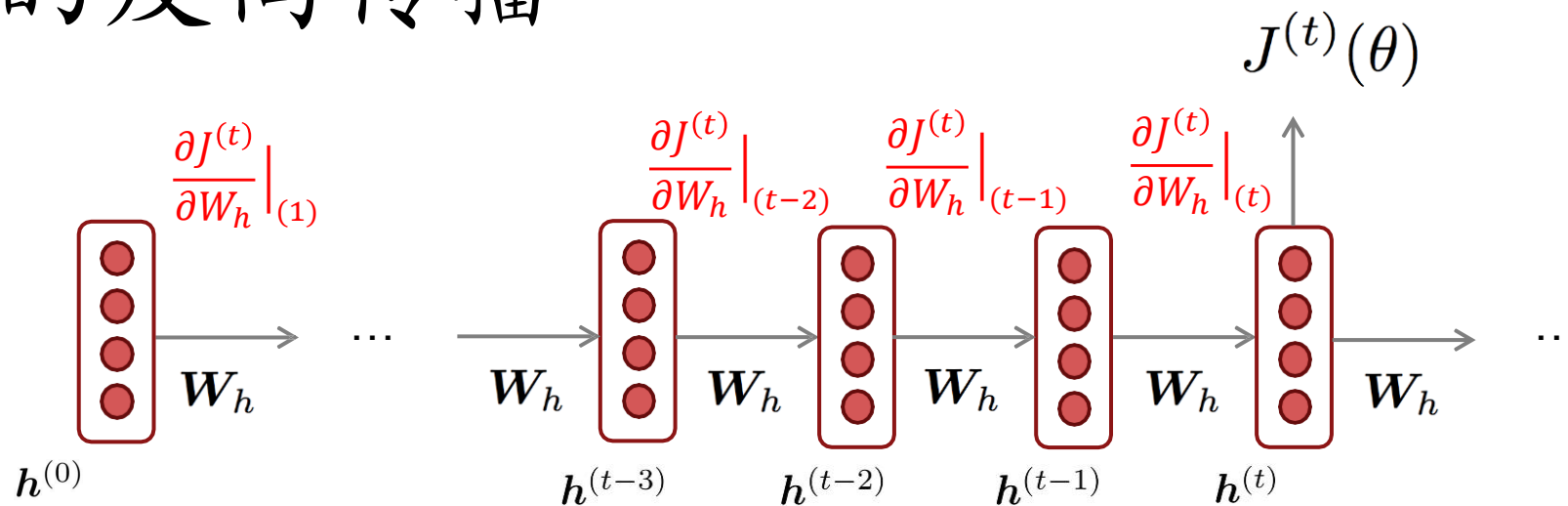
训练一个 RNN 语言模型

- 计算整个训练语料 $x^{(1)} \dots x^{(T)}$ 开销太大!

$$J(\theta) = \frac{1}{T} \sum_{t=1}^T J^{(t)}(\theta)$$

- 实践中，使用以句子或文档为单位的 $x^{(1)} \dots x^{(T)}$
- 回忆：随机梯度下降法允许我们计算小块数据的损失和梯度，并进行更新。
- 计算一个批次句子的损失 $J(\theta)$ ，计算梯度并更新权重。在新的批次的句子上重复。

RNN 的反向传播



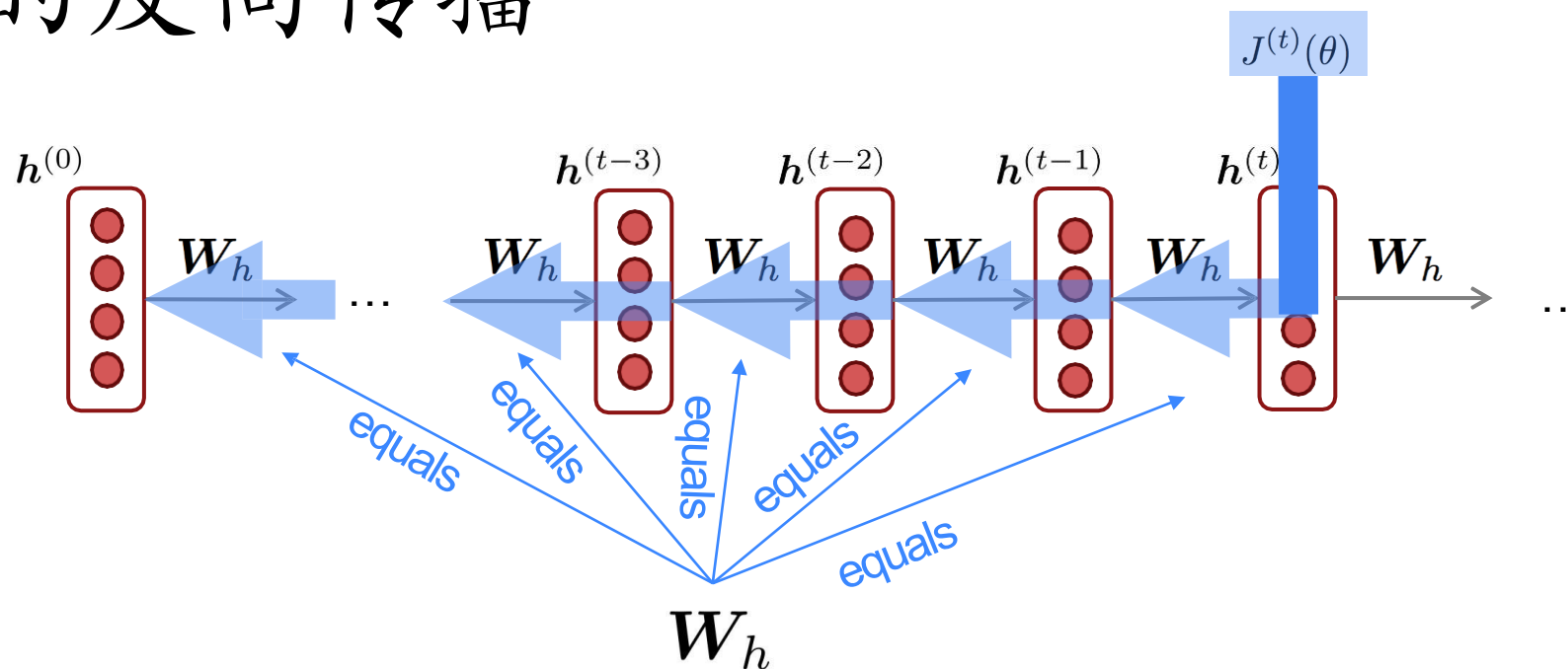
问题: $J(\theta)$ 对于重复使用的矩阵 W_h 的梯度是什么?

回答:
$$\frac{\partial J^{(t)}}{\partial W_h} = \sum_{i=1}^t \left[\frac{\partial J^{(t)}}{\partial W_h} \Big|_{(i)} \right]$$

$J^{(t)}$ 对于 W_h 在时刻 i 的偏导数

"对一个重复权重的梯度
是每次出现的梯度之和"

RNN 的反向传播



$$\frac{\partial J^{(t)}}{\partial W_h} = \sum_{i=1}^t \left. \frac{\partial J^{(t)}}{\partial W_h} \right|_{(i)}$$

如何计算?

答案是：在时间段内 $i = t, \dots, 0$ ，反向传播，边走边把梯度相加。

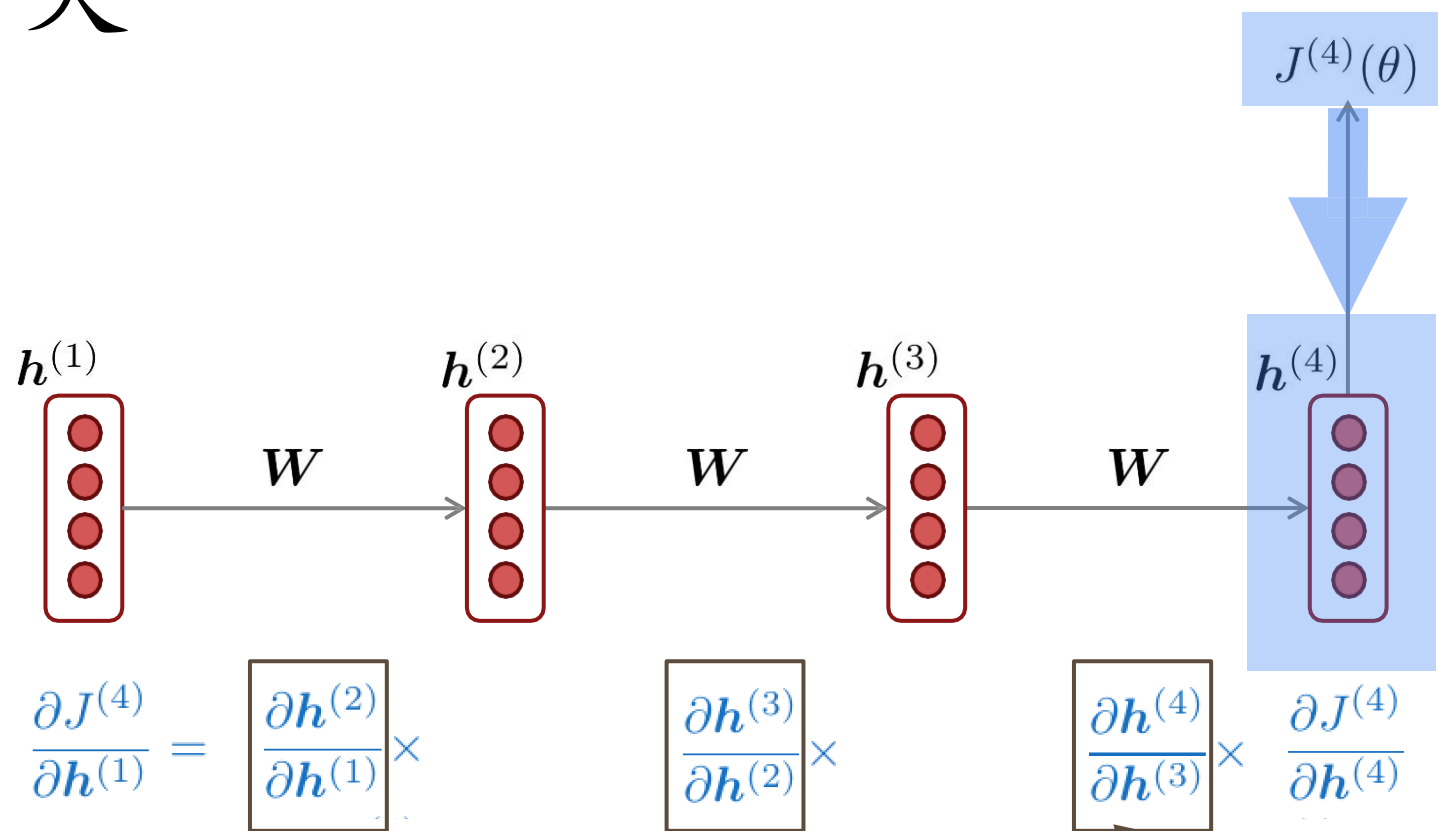
这种算法被称为“通过时间的反向传播”

[Werbos, P.G., 1988, Neural Networks 1, and others]

RNN的问题:梯度消失和梯度爆炸

- 在训练过程中，反向传播计算梯度来更新反复使用的权重矩阵。
- 如果权重矩阵的值很小
 - 梯度会变得非常小，因为它们被反复地乘以权重矩阵
 - 这将导致梯度消失的问题
- 如果权重矩阵中的值很大
 - 梯度可能会变得非常大，因为它们被反复地乘以权重矩阵
 - 就会导致梯度爆炸的问题

梯度消失



如果这些值很小会有哪些问题？

梯度消失问题：
当这些值很小的时候，梯度信号会随着进一步的反向传播变得越来越小

线性条件下梯度消失的简单证明

- RNN: $h^{(t)} = f(W_h h^{(t-1)} + W_e x^{(t)} + b_1)$

- 使用简单的线性激活函数 $f(x) = x$?

$$\begin{aligned}\frac{\partial h^{(t)}}{\partial h^{(t-1)}} &= \text{diag}\left(f'(W_h h^{(t-1)} + W_e x^{(t)} + b_1)\right) W_h && \text{链式法则} \\ &= I W_h = W_h\end{aligned}$$

- 在时刻 i , $J^{(i)}(\theta)$ 对于之前时刻 j 的隐向量 $h^{(j)}$ 的梯度计算方法为 (记 $l = i - j$)

$$\begin{aligned}\frac{\partial J^{(i)}(\theta)}{\partial h^{(j)}} &= \frac{\partial J^{(i)}(\theta)}{\partial h^{(i)}} \prod_{j < t \leq i} \frac{\partial h^{(t)}}{\partial h^{(t-1)}} && \text{链式法则} \\ &= \frac{\partial J^{(i)}(\theta)}{\partial h^{(i)}} \prod_{j < t \leq i} W_h = \frac{\partial J^{(i)}(\theta)}{\partial h^{(i)}} \boxed{W_h^l} && \left(\frac{\partial h^{(t)}}{\partial h^{(t-1)}} \text{ 的值}\right)\end{aligned}$$

Source: "On the difficulty of training recurrent neural networks", Pascanu et al, 2013.

<http://proceedings.mlr.press/v28/pascanu13.pdf>

(and supplemental materials), at <http://proceedings.mlr.press/v28/pascanu13-supp.pdf>

如果 W_h 很 "小", 那么当 l 变大时, 这个项就会出现指数级的问题。

线性条件下梯度消失的简单证明

- W_h^l 带来了什么问题？

- 如果 W_h 的特征值都小于1：

$$\lambda_1 \cdots \lambda_n < 1$$

$$q_1 \cdots q_n \text{ (特征值)}$$

- 我们可以用 W_h 的特征值把 $\frac{\partial J^{(i)}(\theta)}{\partial h^{(i)}} W_h^l$ 重写为：

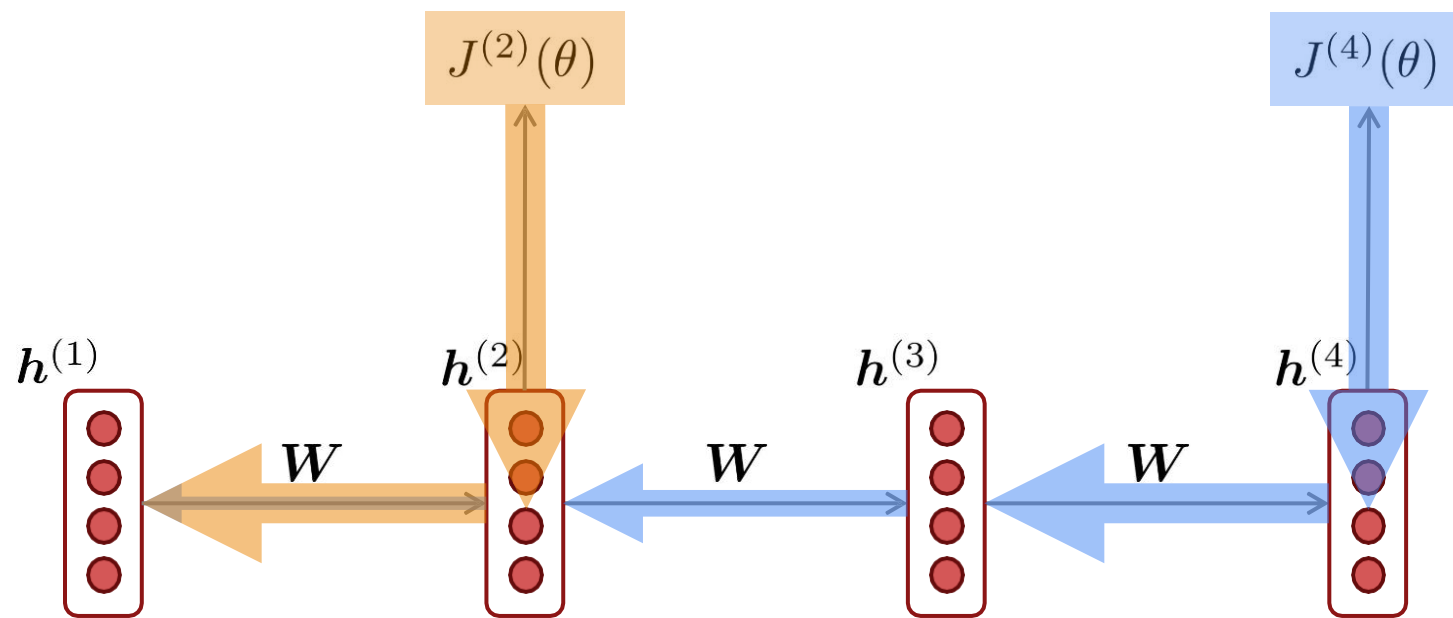
$$\frac{\partial J^{(i)}(\theta)}{\partial h^{(i)}} W_h^l = \sum_{i=1}^n c_i \lambda_i^l q_i \approx 0 \text{ (对于较大的 } l \text{)}$$

当 l 增长时，会趋向于0并造成梯度消失问题

- 如果激活函数 σ 是非线性的

- 可以使用类似的方式证明。 只是对于某些取决于维度和 σ 的 γ ，需要 $\lambda_i < \gamma$

梯度消失的问题



远处的梯度信号会丢失，因为它比近处的梯度信号小得多。

所以，模型权重的更新只考虑到近处的影响，而不是长期的影响。

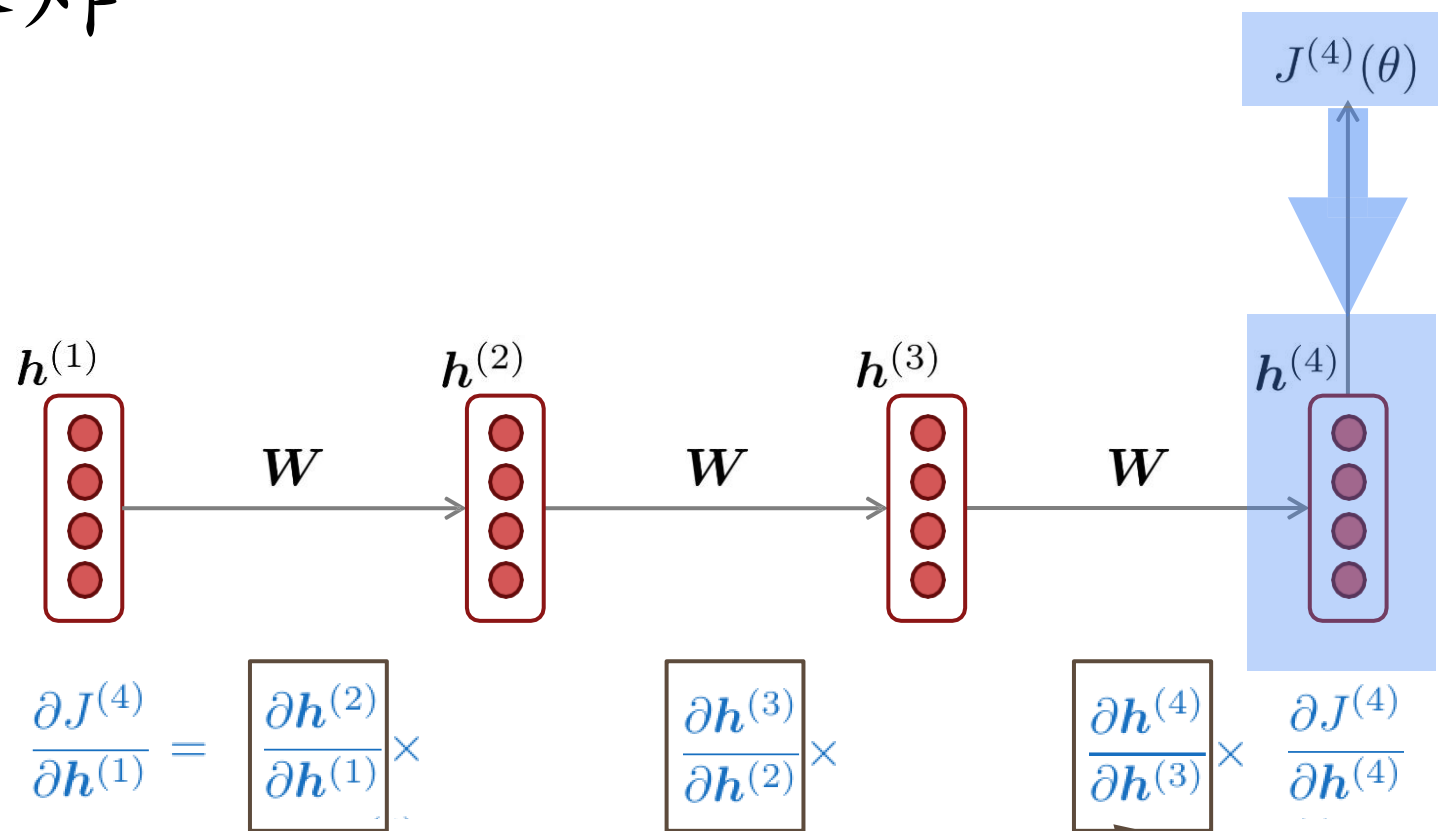
梯度消失问题对于RNN语言模型的影响

- 语言模型任务:

*When she tried to print her **tickets**, she found that the printer was out of toner. She went to the stationery store to buy more toner. It was very overpriced. After installing the toner into the printer, she finally printed her ____?*

- 为了从这个例子中学到信息，RNN语言模型需要对第七个词“tickets”和最后的目标词“tickets”之间的依赖关系建模
- 如果梯度消失，那么模型难以学习到这种依赖关系
 - 所以，模型无法在测试时预测类似的长距离依赖关系

梯度爆炸



如果这些值很大会有什么问题？

爆炸性梯度问题
当这些值很大时，梯度信号随着进一步的反向传播而变得越来越大

梯度爆炸的问题

- 如果梯度变得太大，那么SGD每步的更新就会变得太大：

$$\theta^{new} = \theta^{old} - \overset{\text{学习率}}{\alpha} \underbrace{\nabla_{\theta} J(\theta)}_{\text{梯度}}$$

- 这可能会导致模型难以收敛
- 在最坏的情况下，这将导致你的网络中出现Inf或NaN（然后你必须从较早的检查点重新开始训练）

梯度剪裁：爆炸性梯度的解决方案

- 梯度剪裁：如果梯度的标准值大于某个阈值，在应用SGD更新之前将其缩减。

Algorithm 1 Pseudo-code for norm clipping

$$\begin{aligned} \hat{\mathbf{g}} &\leftarrow \frac{\partial \mathcal{E}}{\partial \theta} \\ \text{if } \|\hat{\mathbf{g}}\| &\geq \text{threshold} \text{ then} \\ &\quad \hat{\mathbf{g}} \leftarrow \frac{\text{threshold}}{\|\hat{\mathbf{g}}\|} \hat{\mathbf{g}} \\ \text{end if} \end{aligned}$$

- 在同一方向上走一步，但步子要小一些
- 在实践中，剪辑梯度是很重要的

如何解决梯度消失？

- 主要问题是，RNN很难学会保存先前时刻的信息。
- 在一个普通的RNN中，隐藏状态不断被重写

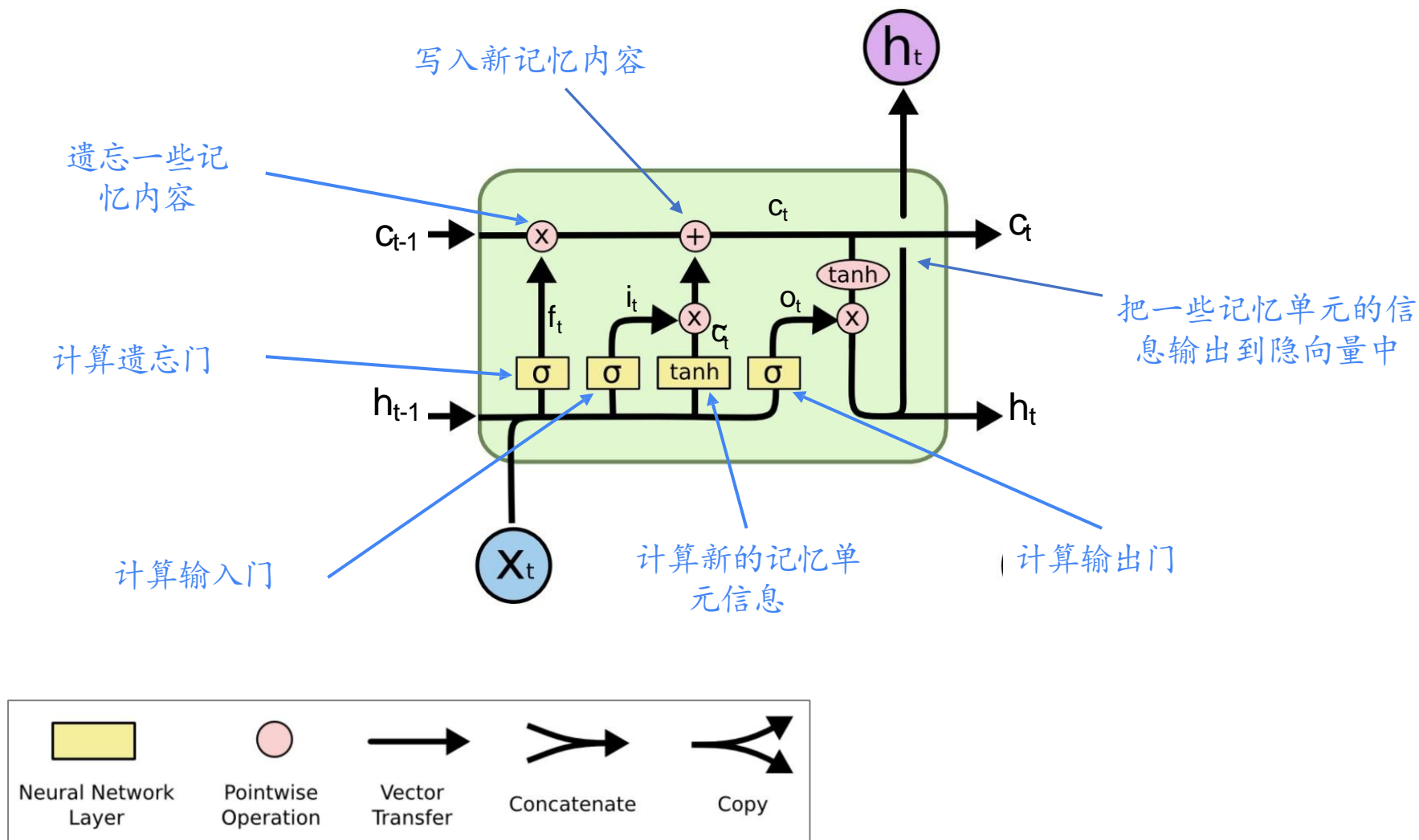
$$h^{(t)} = \sigma(W_h h^{(t-1)} + W_x x^{(l)} + b_1)$$

- 一个有独立内存的RNN来存储历史信息
 - LSTMs
- 进一步在模型中创造更多的直接和线性的传递连接
 - 注意力，等等。

长短时记忆网络(LSTM)

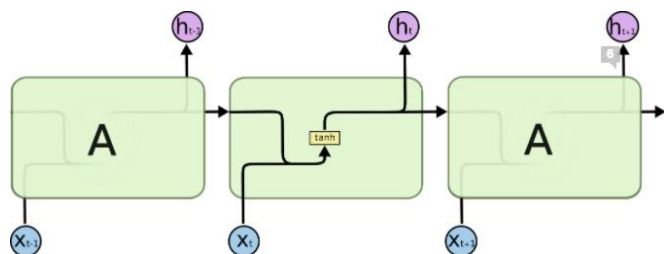
- 在时刻 t ，使用隐向量 h'' 和记忆单元 c''
 - 两者都是长度为 n 的向量
 - 记忆单元存储长期信息(记忆的概念)
 - LSTM可以从记忆单元中读取、擦除和写入信息（使用和更新记忆）。
 - 记忆单元在概念上变得相当像计算机中的RAM
- 使用三个相应的门控制哪些信息被擦除、写入、读取。
 - 这些门也是长度为 n 的向量
 - 在每个时刻，门的每个元素都可以打开（1），关闭（0），或介于两者之间。
 - 门是动态的：它们的值是根据当前环境计算出来的

长短时记忆网络 (LSTM)



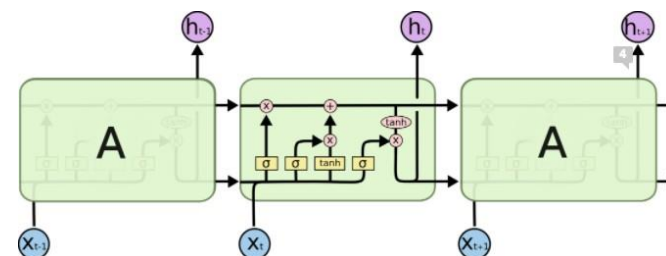
RNN vs. LSTM

- LSTM 引入记忆单元 c 来建模长距离的依赖关系



$$h_t = \tanh(W_{xh}x_t + W_{hh}h_{t-1} + b_h)$$

RNN



$$\begin{aligned} f_t &= \sigma(W_{xf}x_t + W_{hf}h_{t-1} + b_f) \\ i_t &= \sigma(W_{xi}x_t + W_{hi}h_{t-1} + b_i) \\ \tilde{c}_t &= \tanh(W_{xc}x_t + W_{hc}h_{t-1} + b_c) \\ o_t &= \sigma(W_{xo}x_t + W_{ho}h_{t-1} + b_o) \\ c_t &= c_{t-1} \odot f_t + \tilde{c}_t \odot i_t \\ h_t &= \tanh(c_t) \odot o_t \end{aligned}$$

LSTM

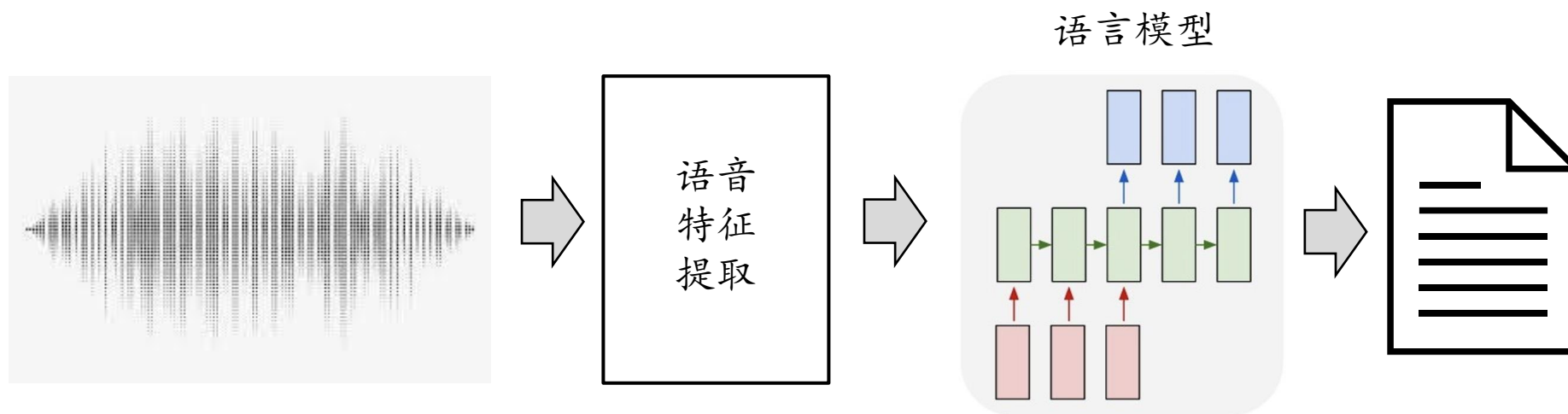
LSTM如何解决梯度消失问题

- LSTM结构使RNN更容易在许多时间段内保存信息
 - 例如，如果一个单元维度的遗忘门设置为1，而输入门设置为0，那么该单元的信息就会被无限期地保留下来。
 - 相比之下，RNN要学习一个循环权重矩阵 W_h 来保存隐藏状态的信息。
- 然而，在长距离依赖的模型中，还有其它方法可以创造更直接和线性的传递连接

神经网络语言模型的应用

语音识别

- 语音特征提取器被用来对输入的音频进行编码
- LM用于从语音特征中生成流畅的文本



拼写自动纠正

- LM用于检测潜在的错词
 - 错误的词通常出现的概率很低
- LM也被用来建议正确的词
 - 正确的词通常呈现高概率

I just happened to write something worng

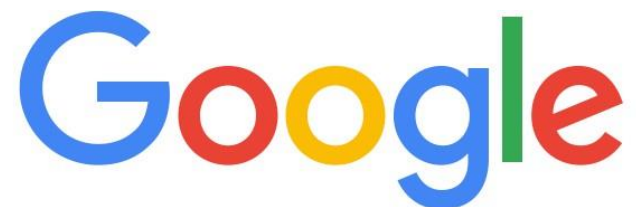
• SPELLING


~~worng~~ →

wrong

搜索引擎

- LM预测用户更有可能用于搜索的下一个词



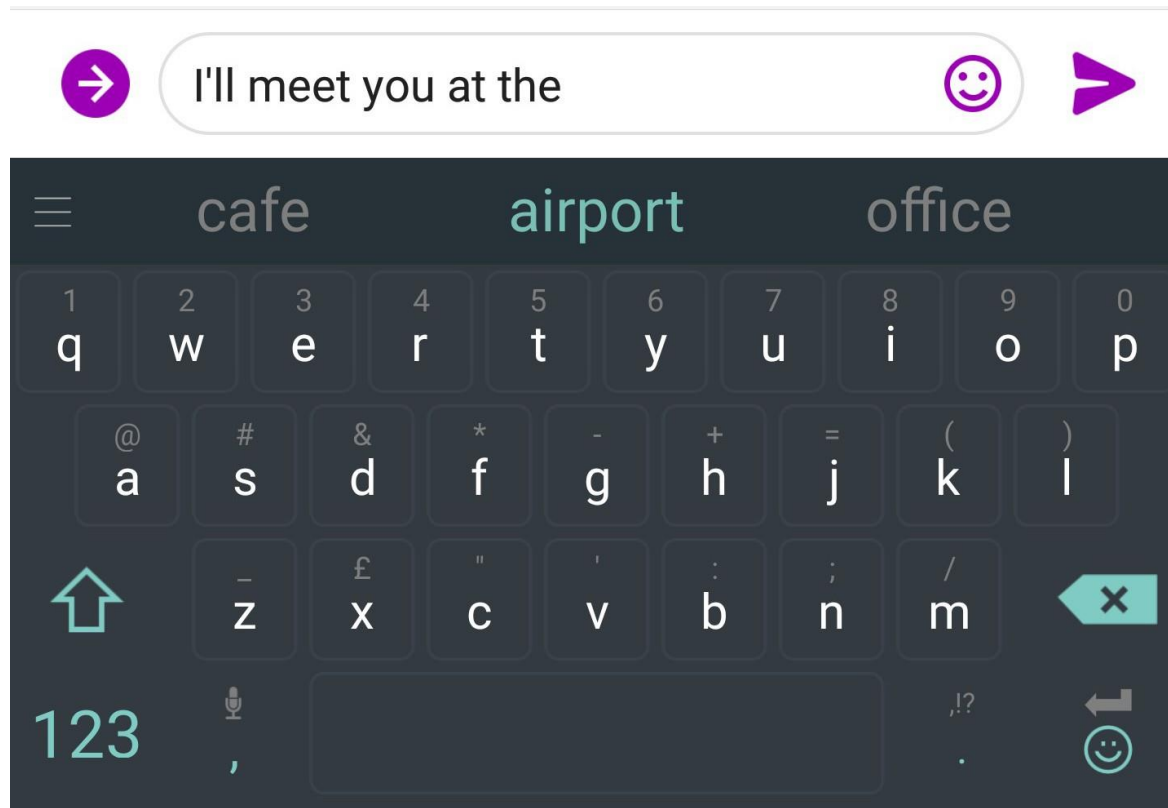
what is the | 

what is the **weather**
what is the **meaning of life**
what is the **dark web**
what is the **xfi**
what is the **doomsday clock**
what is the **weather today**
what is the **keto diet**
what is the **american dream**
what is the **speed of light**
what is the **bill of rights**

Google Search I'm Feeling Lucky

文本自动补全

- LM预测用户在特定环境下更可能使用的词汇



对话机器人

- LM对用户输入进行编码，然后生成相应的输出。



How are you today, ChatGPT?



As a language model trained by OpenAI, I don't have the ability to experience emotions like a human would. I am designed to assist with providing information and answering questions to the best of my ability. Is there something specific you would like to know?



What is the Capital of the Klingon homeworld, Qo'noS?

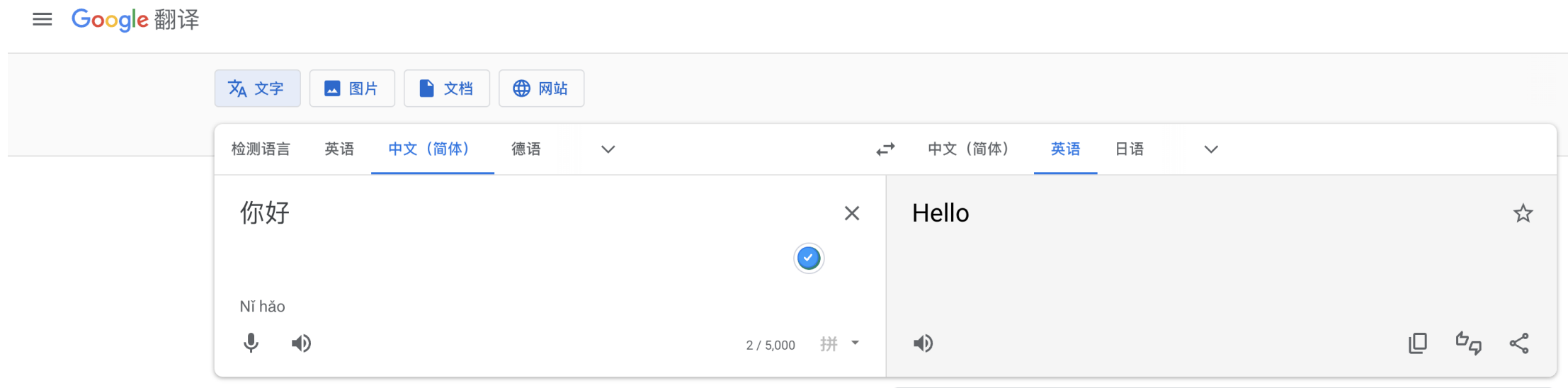


In the fictional Star Trek universe, the Klingon homeworld is called Qo'noS and its capital is called First City. It is the political and cultural center of the Klingon Empire and the location of the Klingon High Council. It is a city of great size and importance in Klingon society.



机器翻译

- 源语言的语言模型对输入文本进行编码；
- 目标语言的语言模型生成输出翻译。



信息检索



2023年春节是什么时候?



News



Images



Videos



Books



Maps



Shopping



Flights

About 47,700,000 results (0.66 seconds)

Chinese New Year / Date (2023)

Sun, Jan 22, 2023

