

## 第二章 问答摘要与推理-词向量实践

### HCT NLP Week 2

#### 问答摘要与推理 词向量实践与RNNs

##### Outline

① Huffman tree  
② 负采样

- 词向量计算两种优化方法
- 词向量在工程中的具体实现
- RNN递归神经网络结构
- RNN、LSTM、GRU

##### Outline

① Huffman Tree  
② 负采样

- 词向量计算两种优化方法
- 词向量在工程中的具体实现
- RNN递归神经网络结构
- RNN、LSTM、GRU

Huffman Tree (哈夫曼树)

带有权重且层数最少的树为哈夫曼树。

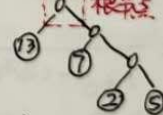
Huffman Tree 最优二叉树。

Huffman Tree 编码: (向左为0, 向右为1)

13: 0  
7: 10  
2: 110  
5: 111

注: 根节点没有编码  
所有的词都在叶子  
节点上

Huffman Tree 示例:



例:  $Path = 5 \times 3 + 2 \times 3 + 7 \times 2 + 13 = 48$

Huffman Tree 特点:

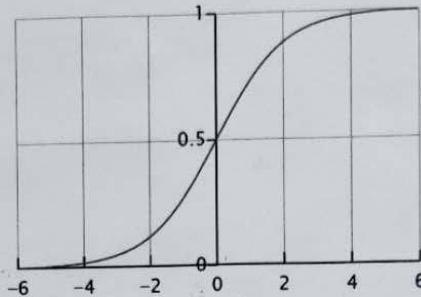
- ① 有  $V-1$  个中间节点,  $V$  个叶子节点, 叶子节点的数量与词表中词的数量相等 ( $V$  为词表中词的数量)
- ② 频率越高的词越靠近根节点, 词的 Huffman 编码越短

# Hierarchical Softmax

Logistic Regression 逻辑回归

LR 主要用 召回 和 预估/排序

1. LR 是一种解决 监督学习的方法
2. 进行 LR 的目的是 使训练数据的 标签与预测值的 误差最小化



例：垃圾邮件的分类：(二分类问题)

预测为垃圾邮件的概率：

$$\hat{y} = P(y=1|x) \quad 0 \leq \hat{y} \leq 1$$

$$P(y|x) = \begin{cases} \hat{y} & (y=1, y \text{ 为标签}) \\ 1-\hat{y} & (y=0) \end{cases}$$

线性组合的方法得  $\hat{y} = \sigma(W^T x + b)$   $W^T x = \sum_{i=1}^n w_i x_i = w_0 + w_1 x_1 + w_2 x_2 + \dots + w_n x_n$  (参数为机器学习的参数, 而不是深度学习的参数)

为控制  $\hat{y}$  在 [0, 1] 之间, 采用 sigmoid 函数

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

$$= \frac{1}{1+e^{-(W^T x + b)}} \quad \text{即概率预测函数}$$

例：目标函数：

$$\text{Loss} = -\log(P(y|x)) \quad \text{即}$$

$$L(y, \hat{y}) = -\log(P(y|x))$$

$$= -\log[\hat{y}^y * (1-\hat{y})^{(1-y)}]$$

$$= -[y \log \hat{y} + (1-y) \log (1-\hat{y})]$$

注：百度 Logistic Regression 可查到更多公式推导

因  $y$  的取值为 0 或 1

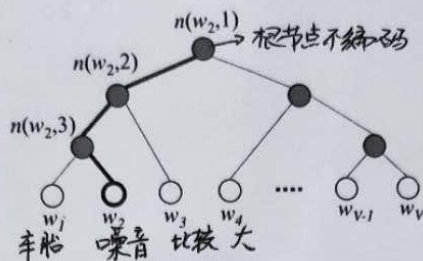
所以：

$$P(y|x) = \hat{y}^y * (1-\hat{y})^{(1-y)}$$

注：是乘不是加

# Hierarchical Softmax

分层 softmax



注: 图有个小问题:  
所有词并不在同一层

根据中心词预测  
周围词, 或根据周围  
词预测中心词

预测概率:

$$P(w | context(w)) = \prod_{j=2}^l P(d_j^w | x_w, \theta_{j-1}^w)$$

$\prod$  即每个点概率的乘积  
 $x_w$  输入 词向量  
 $\theta_{j-1}^w$  神经网络参数(权重)  
 $d_j^w$  根节点上没有词 (j从2开始取是因为不取根节点)

命名数说明:

$P^w$ : 从根节点到叶子节点的路径 (如  $P^{w_2}$ )

$l^w$ : 经过的节点数.

$P_1^w, P_2^w, \dots$ : 指  $P^w$  路径上经过的各节点.

$d_2^w, d_3^w, \dots \in \{0, 1\}$ : 指  $P^w$  路径上第  $i$  个节点的编码 (不考虑根节点)

$\theta_1^w, \theta_2^w, \dots$ : 指  $P^w$  路径中非子叶节点对应的参数向量

$$P(d_j^w | x_w, \theta_{j-1}^w) = \begin{cases} \sigma(x_w^T \theta_{j-1}^w) & (d_j^w = 0, \text{编码为0}) \\ 1 - \sigma(x_w^T \theta_{j-1}^w) & (d_j^w = 1) \end{cases}$$

输入词的编码  
(=分类问题)

由于  $d_j^w$  只有 0 或者 1 两种情况, 所以

$$P(d_j^w | x_w, \theta_{j-1}^w) = [\sigma(x_w^T \theta_{j-1}^w)]^{(1-d_j^w)} \cdot [1 - \sigma(x_w^T \theta_{j-1}^w)]^{d_j^w}$$

训练目标函数:

$$L = \sum_{w \in c} \log \prod_{j=2}^{l^w} P(w | context(w)) = \sum_{w \in c} \log \prod_{j=2}^{l^w} \left\{ [\sigma(x_w^T \theta_{j-1}^w)]^{(1-d_j^w)} \cdot [1 - \sigma(x_w^T \theta_{j-1}^w)]^{d_j^w} \right\}$$

$c$  为词表中的词  
(loss函数)  $L(w, j) = (1 - d_j^w) \log [\sigma(x_w^T \theta_{j-1}^w)] + d_j^w \log [1 - \sigma(x_w^T \theta_{j-1}^w)]$

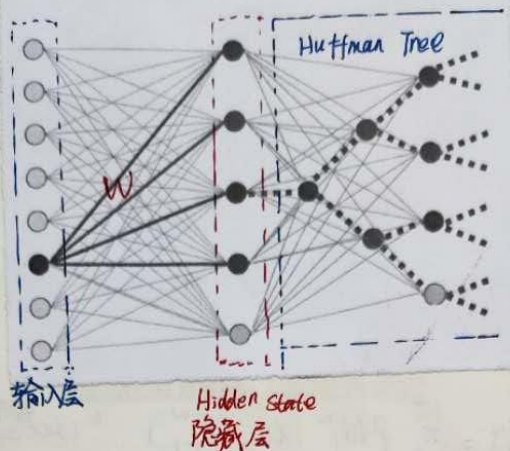


注:减少训练时间的一种方法:

累积一定次数的正向传播后,再统一做一次反向传播(比做一正向传播就做一次反向传播节约时间)

注:一次传播是指计算完一个 batch-size

## Hierarchical Softmax



Hidden state 中每个神经元与 Huffman Tree 中的每个节点都相连

利用 Huffman Tree 进行向量计算优化的优点:

- ① 计算复杂度降低, 复杂度由  $O(n)$  降为  $O(\log n)$
- ② 词频越高, 词越靠近根节点, 搜索路径越短, 计算更快, (词频越低的词越需要新的参数越多)

## Negative Sampling

每次让一个训练样本仅仅更新一小部分的权重

什么是负样本: 计算过程中输入的词为正样本, 其余所有词都为负样本。

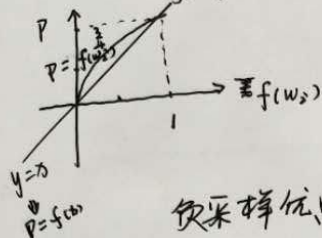
对于大规模数据集 选取 2~5 个负样本。

对于小规模数据集 选取 5~20 个负样本。

哪些词被选做负样本是由词频决定的。

负采样选取概率:

$$P(w_i) = \frac{f(w_i)^{\frac{3}{4}}}{\sum_{j=0}^n f(w_j)^{\frac{3}{4}}}$$



其中  $f(w_i)$  为  $w_i$  的词频

为什么选用  $\frac{3}{4}$  次幂: 使低频词被选做负采样词的概率比高频词的概率高 (往往低频词更重要)

负采样优点: 提高训练速度 (更新参数少)

★ word2vec trick: 高频词的亚采样 (让低频词的被选为样本的概率高)  
sample 的值通常取:  $10^{-5} \sim 10^{-3}$

采样概率:  $P(w_i) = 1 - \sqrt[\text{sample}]{\text{freq}(w_i)}$   $\text{freq}(w_i)$  为  $w_i$  的词频

在 word2vec 训练中使用上述方法可以提速 2-10 倍训练速度。

## Outline

JuliaPython

- 词向量计算两种优化方法
- 词向量在工程中的具体实现
- RNN 递归神经网络结构
- RNN、LSTM、GRU

利用 FastText 处理 oov (out of vocabulary)

FastText 会将每个词进行切分,

四种方式处理 oov

如: 详细在 JuliaPython

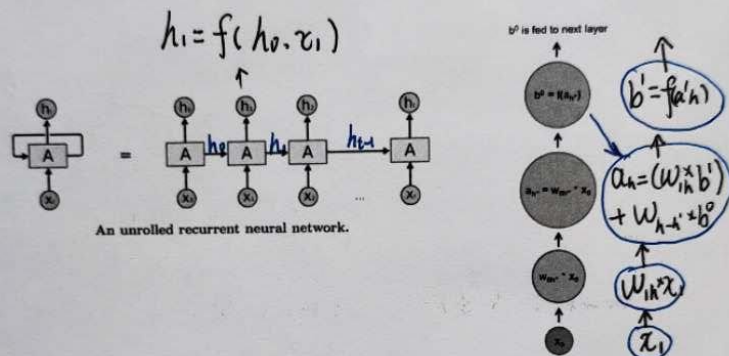
1. 不进行预训练: `tf.keras.Embedding (trainable=False)` 计算量比较大
2. 预训练:  $W \times V$  或 FastText
3. Tencent AILab
4. ELMo

# Outline

- 词向量计算两种优化方法
- 词向量在工程中的具体实现
- RNN递归神经网络结构
- RNN、LSTM、GRU

## RNN

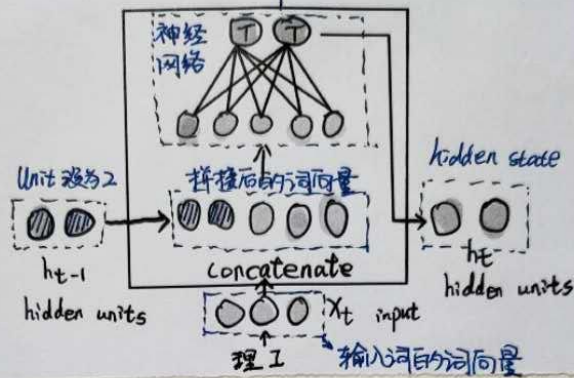
Recurrent Neural Network 递归神经网络





## RNN

Recurrent Neural Network 输出

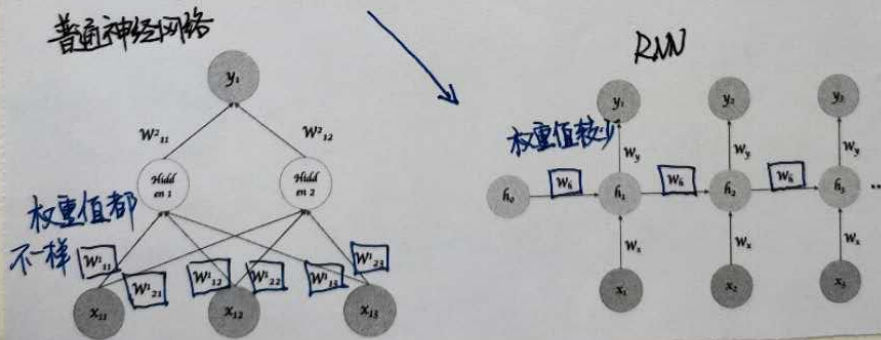


RNN结构使得其具有长距离依赖:  $\begin{matrix} \uparrow & \uparrow & \dots & \uparrow \\ \square & \square & & \square \\ \downarrow & \downarrow & & \downarrow \\ z_0 & z_1 & & z_n \end{matrix}$

RNN只能串行计算, 不能并行计算, 不利于提速。  
(某些所谓的 RNN 并行计算并不是真正的并行计算)

## RNN

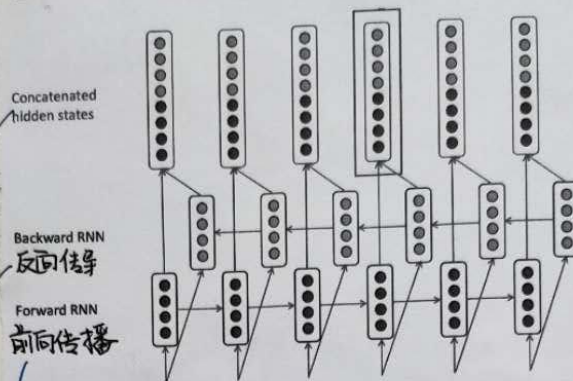
parameter sharing



RNN相对于普通神经网络的优势: 传统神经网络权重不共享  
↑  
要计算的权重少

## RNN

### Bidirectional RNNs



hidden state :

1. 前向:  $\vec{h}^{(t)} = \text{RNN}_{\text{FW}}(\vec{h}^{(t-1)}, x^t)$  (FW: Forward)

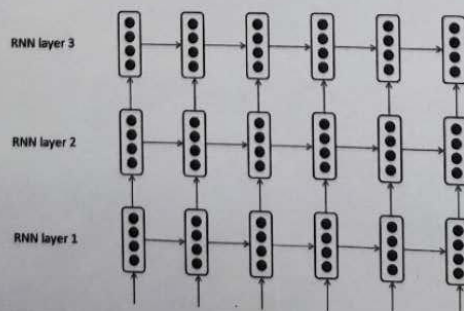
2. 反向:  $\overleftarrow{h}^{(t)} = \text{RNN}_{\text{BW}}(\overleftarrow{h}^{(t+1)}, x^t)$  (BW: Backward)

3. 拼接:  $h^t = [\vec{h}^{(t)}, \overleftarrow{h}^{(t)}]$

注: 双向RNN, 前向和反向是同时开始训练计算的

## RNN

### Deep RNNs



Neural machine translation  
在NMT (机器翻译)  
中用RNN较多。

encoder中一般2~4层  
decoder中一般4层

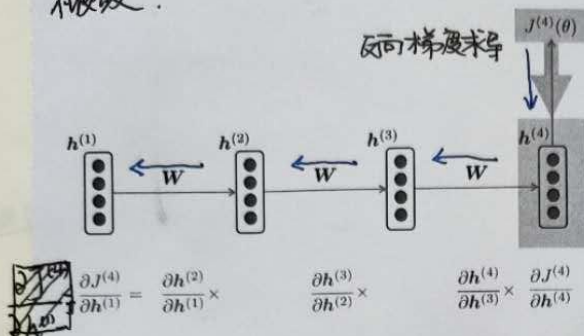
文本摘要一般用单层,  
单向或双向都行。  
常用RNN, GRU, LSTM



## Exploding and Vanishing Gradient Problem

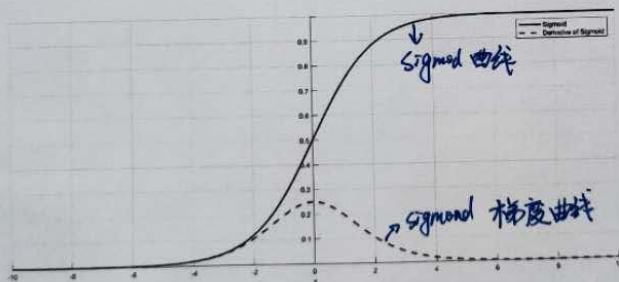
不收敛.

反向梯度求导



为什么会发现梯度爆炸和梯度消失: 由于反向梯度求导的方式(连乘), 单激函数  $\sigma(w \cdot x + b)$  的  $\sigma$  的值较大或较小的时候会出现梯度爆炸或梯度消失。

## Exploding and Vanishing Gradient Problem



Exploding  
Gradient

解决方法:

gradient clipping

利用截断(设置阈值)  
解决梯度爆炸问题

Vanishing  
Gradient

解决方法:

Identity Initialization

更换激活函数

LSTM

更换神经网络

Residual Networks

残差网络(跳过中间某些层)

Batch Normalization

进行规范化操作

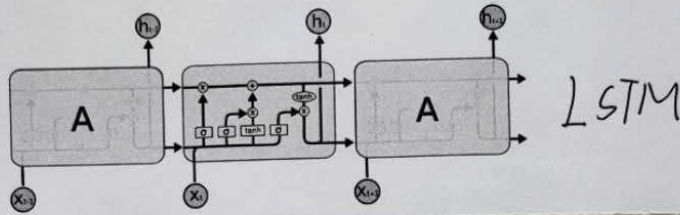
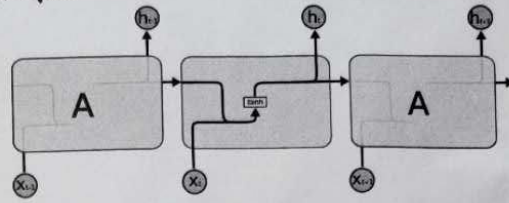
## Outline

- 词向量计算两种优化方法
- 词向量在工程中的具体实现
- RNN递归神经网络结构
- RNN、LSTM、GRU

# LSTM 长短时记忆网络

## Long Short Term Memory

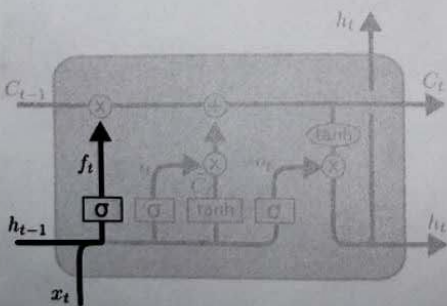
RNN



运算符号说明:

- $\square$  : 神经网络
- $\circ$  : 一个操作
- $\rightarrow$  : 向量传播方向
- $\searrow$  : concat
- $\swarrow$  : copy

## LSTM



遗忘门:

$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f)$$

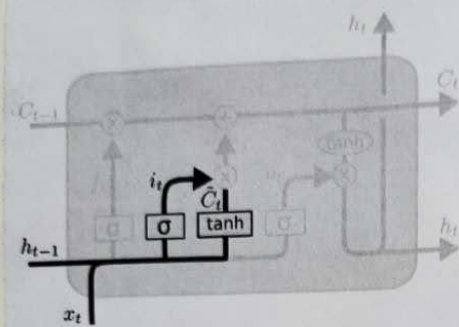
神经网络      偏置权重  
权重

$$C_t \neq h_t$$

$$C_{t-1} \neq h_{t-1} \neq x_t$$



## LSTM



输入门:  $\sigma$  起筛选作用, 即保留多少信息  
偏置权重

$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)$$

$$\tilde{C}_t = \tanh(W_c \cdot [h_{t-1}, x_t] + b_c)$$

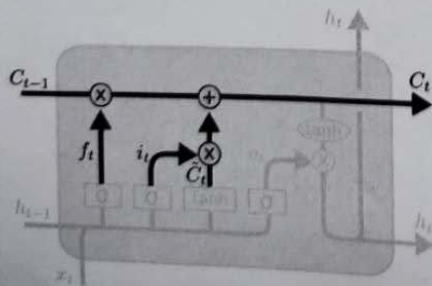
- ① 取值在 (0, 1) 之间, 起到选取保留多少信息的作用
- ② 避免梯度消失
- ③ 收敛比 Sigmoid 慢

注:

Sigmoid 通常作两门, 用于决定遗忘掉信息的多少

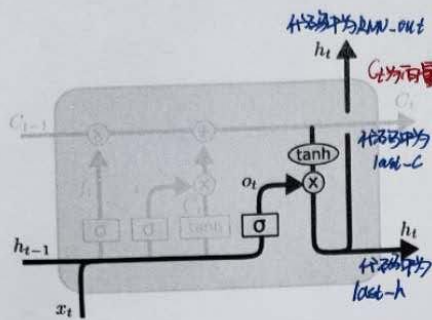
tanh 用作正常信息的记录, 对信息进行压缩, 压缩到 [-1, 1] 之间

## LSTM



$$C_t = f_t \times C_{t-1} + i_t \times \tilde{C}_t$$

## LSTM



输出为:

$$O_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o)$$

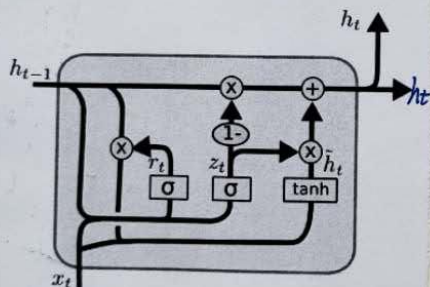
$$h_t = O_t \times \tanh(C_t)$$

注: LSTM 模型最终有 3 个输出:

2 个  $h_t$ , 1 个  $C_t$ . 所以代码中应写 3 个变量接收返回的结果

$$h_t, h_c, C_t = \text{lstm}()$$

## GRU Gated Recurrent Unit



GRU 为 LSTM 的一个变种.

$$r_t = \sigma(W_r \cdot [h_{t-1}, x_t])$$

$$z_t = \sigma(W_z \cdot [h_{t-1}, x_t])$$

$$\tilde{h}_t = \tanh(W \cdot [r_t \times h_{t-1}, x_t])$$

$$h_t = (1 - z_t) h_{t-1} + z_t \times \tilde{h}_t$$

LSTM 与 GRU 相比:

- 1) LSTM 有 3 个输入 ( $C_{t-1}$ ,  $h_{t-1}$ ,  $x_t$ ), 2 个输出 ( $C_t$ ,  $h_t$ ), 3 个门
- 2) GRU 有 2 个输入 ( $h_{t-1}$ ,  $x_t$ ), 1 个输出 ( $h_t$ ), 2 个门 (没有遗忘门)
- 3) GRU 运算速度更快 (因为参数少)
- 4) LSTM 比 GRU 所需的数据集大

- ① GRU 和 LSTM 从准确率上讲, 区别不大, 但 GRU 网络结构简单, 计算量少一些
- ② 通常可以先用 LSTM 模型计算, 再用 GRU 模型检测: LSTM  $\rightarrow$  GRU

附：

- (1) 腾讯 800 万中文词的 NLP 数据集开源: <https://zhuanlan.zhihu.com/p/47133426>
- (2) 腾讯 Allib: <https://ai.tencent.com/ailab/nlp/embedding.html>
- (3) pycharm 中安装 Conda、pytorch 环境: <https://pytorch.org/get-started/locally/>
- (4) 负采样示例: <http://mccormickml.com/2017/01/11/word2vec-tutorial-part-2-negative-sampling/>



