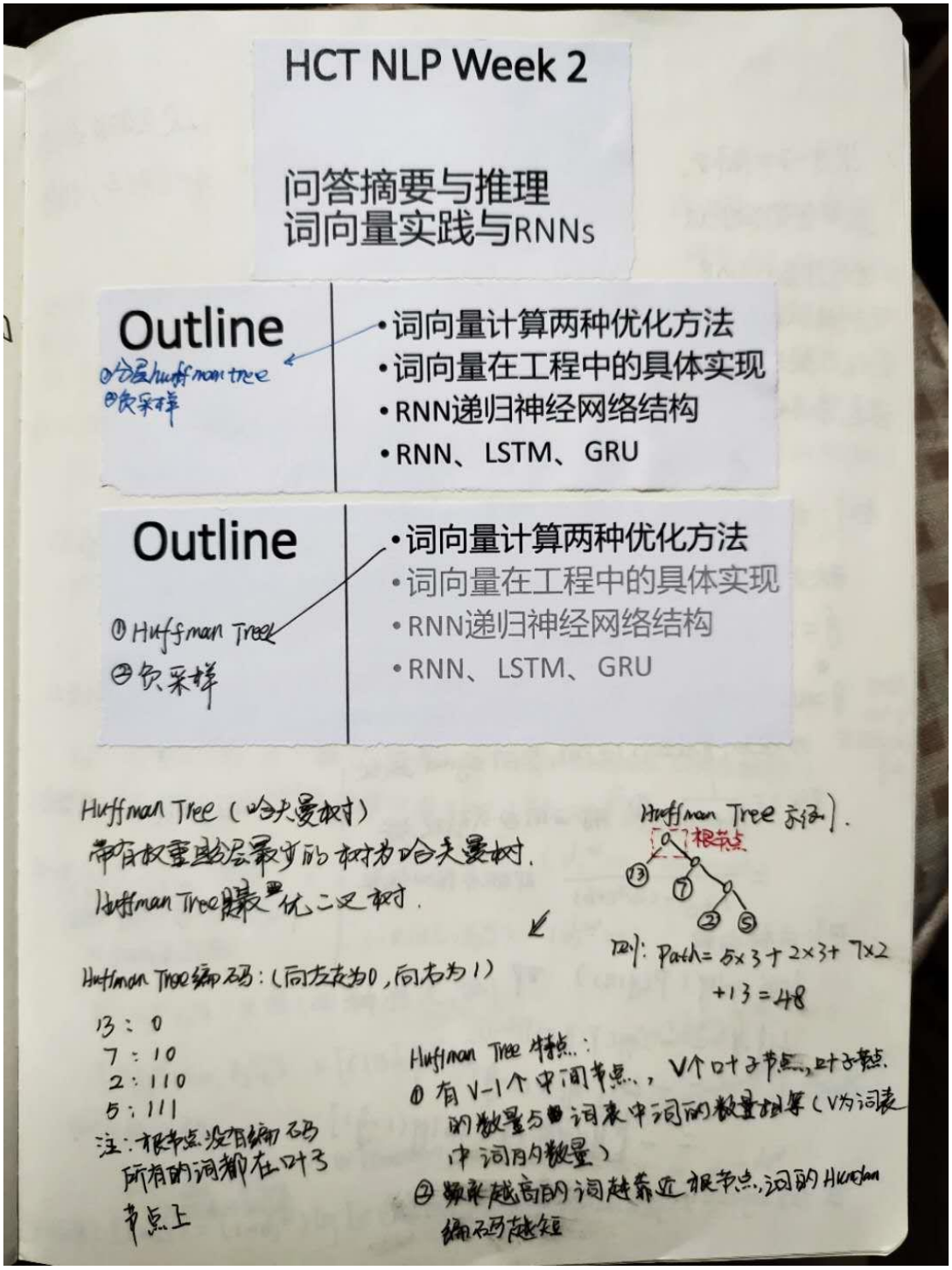
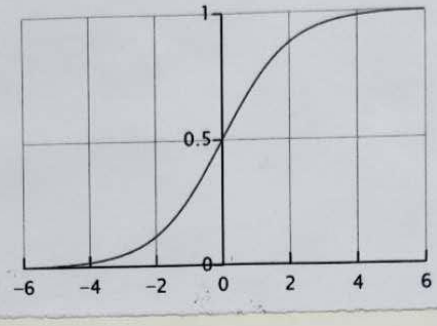
第二章 问答摘要与推理-词向量实践



Hierarchical Softmax

Logistic Regression 逻辑回归



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值与预测值的误差最小化

LR主要用于回归和预估/排序

例1：垃圾邮件的二分类：（二分类问题）

预测为垃圾邮件的概率：

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

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

特性加合的方法得出 $\hat{y}$

$\hat{y} = \sigma(W^T x + b)$ ② $W^T x = \sum_{i=1}^{n} w_i x_i = w_0 + w_1 x_1 + w_2 x_2 + \cdots + w_n x_n$

（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 sigmod 函数

$\sigma(z) = \frac{1}{1+e^{-z}}$ ③ 将z用②式替换，得：

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

因y的取值为0或者1，所以：

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

注：最终不是加

例1：目标函数：

$Loss = -\log(P(y|x))$ 即

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

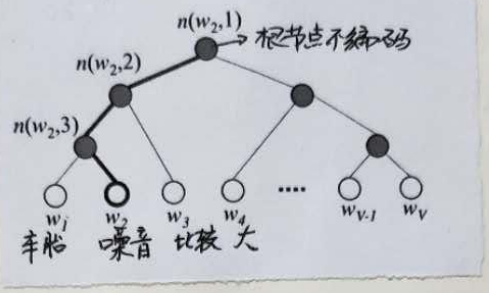
$= -\log[\hat{y}^y * (1-\hat{y})^{(1-y)}]$

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

注：Logistic Regression 可查看更多公式推导

Hierarchical Softmax

分层 softmax



注：两个小问题：
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：

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

即每个节点相乘的乘积

根节点上没有词（从2开始取是因为不取根节点）

x_w：输入词向量

θ_{j-1}^w：神经网络参数（权重）

公式参数说明：

p^w ：从根节点发，到达叶节点的路径（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 j 个节点的编码（不考虑根节点）

$\theta_1^w, \theta_2^w \dots$ ：指 p^w 路径中非叶节点，对应的参数向量

$p(d_j^w | x_w, \theta_{j-1}^w)$ 指：输出词的概率（二分类问题）

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

由于 d_j^w 只有0或者1两种情况，所以

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

则目标函数：

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

C为词表中的词

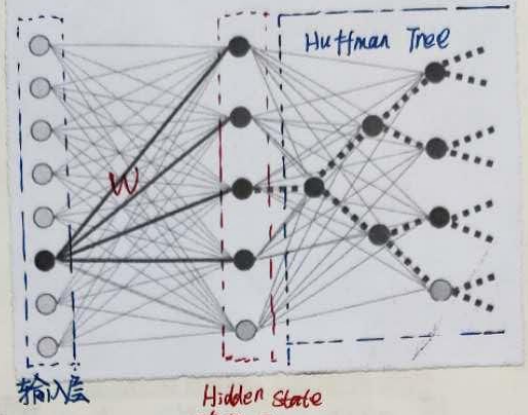
（loss函数）$L(w,j) = (1-d_j^w) \log [\sigma(x_w^T \theta_{j-1}^w)] + d_j^w \cdot \log [1-\sigma(x_w^T \theta_{j-1}^w)]$

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（比做一次正向传播就做一次反向传播的时间）

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中所有节点相连

利用 Huffman Tree进行向量计算优化的优点：

① 计算复杂度降低，复杂由 $O(n)$ 降为 $O(\log^n)$

② 词数越高与词越靠近根节点，搜索路径越短，计算更快（涉及到的分词越少，需要更新的参数越少）

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。

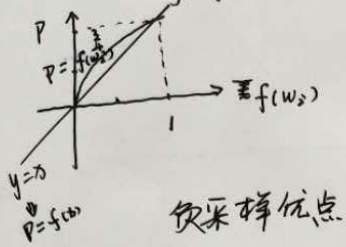
对于小规模数据集 选取 5～20个负样本。

哪些词被选做负样本是由词频决定的。

负样本选取概率：

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

其中 $f(w_i)$ 为 w_i 词的词频



为什么选用 $\frac{3}{4}$ 次幂：使低频词被选作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（更新的参数少）

☆ word2vec trick：高频词的亚采样（让低频词的被选为样本的概率高）

采样概率：$P(w_i) = 1 - \sqrt{\frac{sample}{freq(w_i)}}$

sample的值通常取：$10^{-5} \sim 10^{-3}$

$freq(w_i)$ 为 w_i 词的词频

在word2vec训练中采用以上方法可以提高 2~10 倍训练速度.

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

见jupyter

利用Fasttext避免OOV（out of Vocabulary）

Fasttext会将每个词进行切分，如：详细 → 详细, 细节

四种方式避免OOV

1. 不进行预训练，tf.keras.Embedding(trainable=False) 计算可比较大
2. 预训练：采用 W2V 或 Fasttext
3. Tencent AILab
4. ELMO

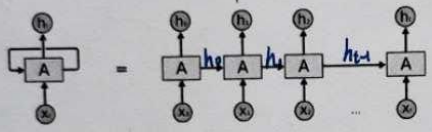
Outline

- 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

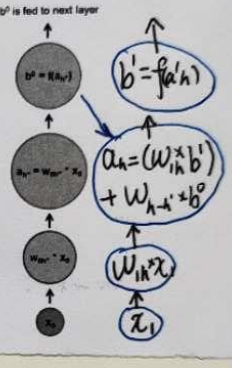
RNN

Recurrent Neural Network 递归神经网络

$h_1 = f(h_0, x_1)$

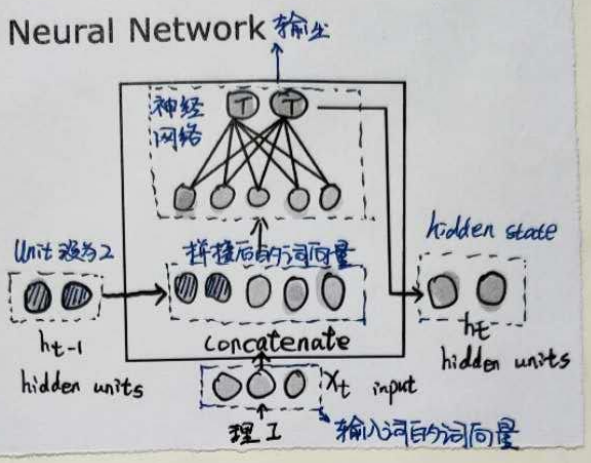


An unrolled recurrent neural network.

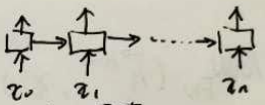


RNN

Recurrent Neural Network



RNN结构使得具有长距离依赖：

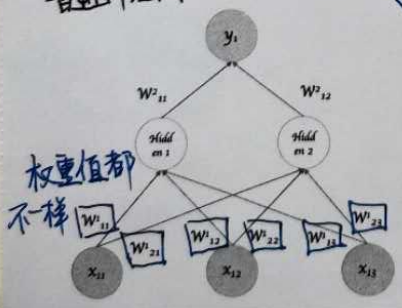


RNN只能串行计算，不能并行计算，不利于提速。
（某些所谓的RNN并行计算并不是真正的并行计算）

RNN

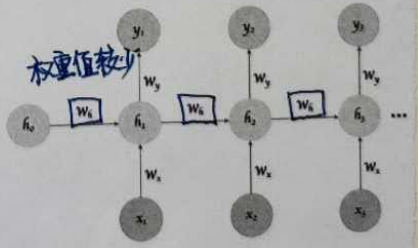
parameter sharing

普通神经网络



权重值都不一样

RNN



RNN相对于普通神经网络的优点：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：$\overrightarrow{h}^{(t)} = RNN_{FW}(\overrightarrow{h}^{(t-1)}, x^t)$ （FW：Forward）

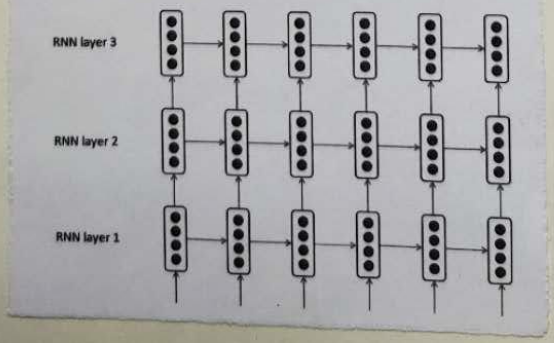
2. 反向：$\overleftarrow{h}^{(t)} = RNN_{BW}(\overleftarrow{h}^{(t+1)}, x^t)$ （BW：Backword）

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

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。

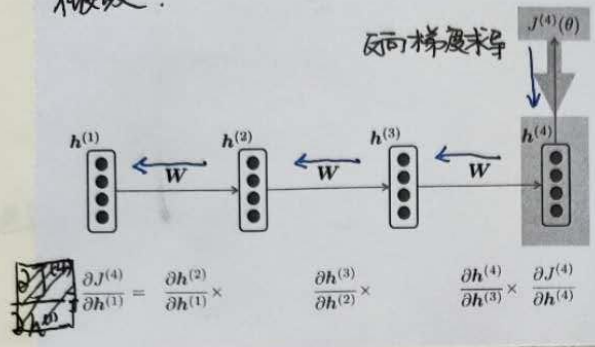
encoder中一般2～4层

decoder中一般4层

文本摘要一般用单层，单向或双向都行。通常用RNN、GRU、LSTM

Exploding and Vanishing Gradient Problem

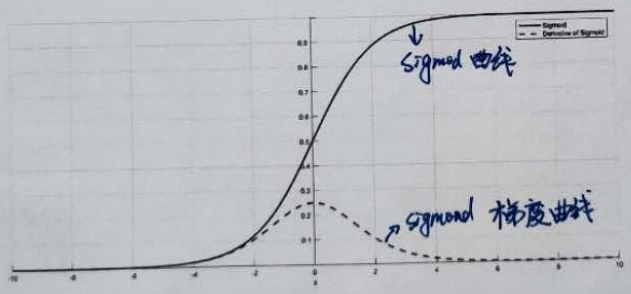
↓
不收敛.



$$\frac{\partial J^{(4)}}{\partial h^{(1)}} = \frac{\partial h^{(2)}}{\partial h^{(1)}} \times \frac{\partial h^{(3)}}{\partial h^{(2)}} \times \frac{\partial h^{(4)}}{\partial h^{(3)}} \times \frac{\partial J^{(4)}}{\partial h^{(4)}}$$

为什么会出现梯度爆炸和梯度消失：由于反向梯度求导的方式（连乘），单激活函数（$\sigma(wx+b)$）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：

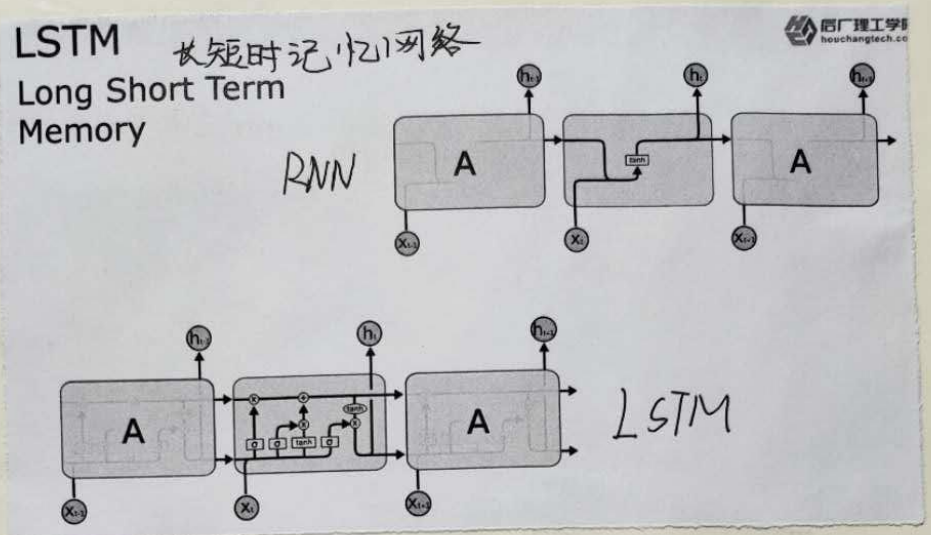
- Identity Initialization：更换激活函数
- LSTM：更换神经网络
- Residual Networks：残差网络（跳过中间某些）
- Batch Normalization：进行规一化操作

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



运算符号说明：

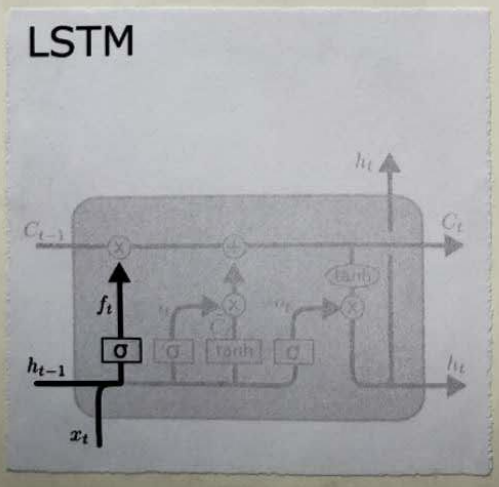
□ ：神经网络

○ ：一个操作

→ ：向量传播方向

↘ ：Concat

↙ ：copy



遗忘门：

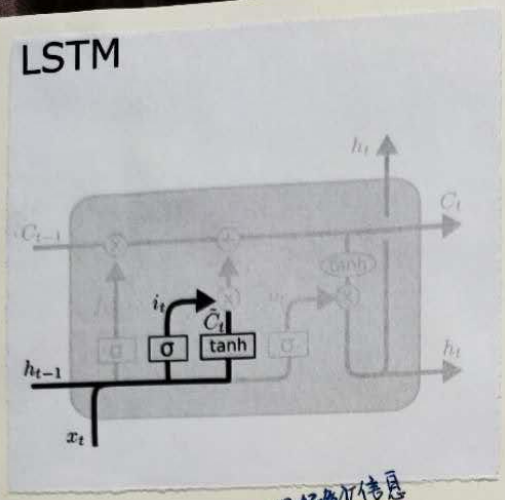
$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f)$

神经网络权重　　偏置权重

$C_t \neq h_t$

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

LSTM



注：
Sigmoid通常作门，用于决定遗忘/保留信息的多少

tanh用作正常信息的记录，对信息进行压缩，压缩到[-1, 1]之间

输入门：

Sigmoid：起到类似作用，即保留多少信息

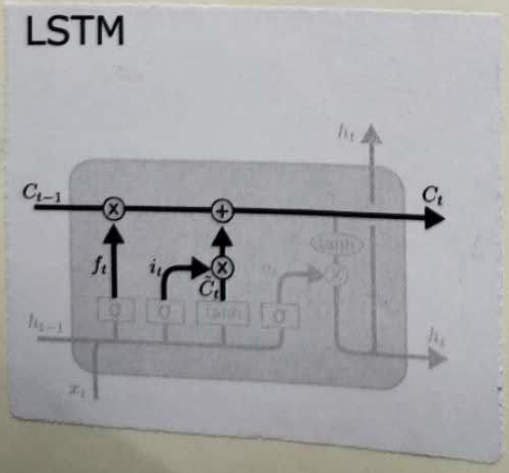
偏置权重

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

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

① 便取值在(0,1)之间，起到类似保留多少信息的作用
② 避免梯度消失
③ 收敛比Sigmoid慢

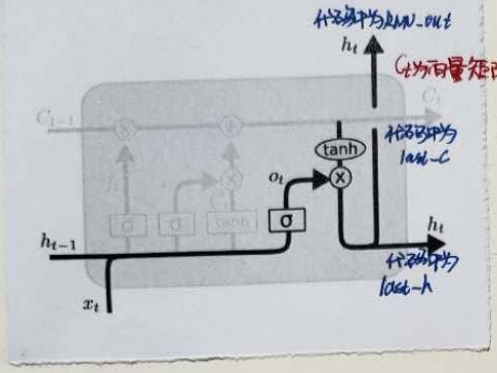
LSTM



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

①
②

LSTM



输出门：

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

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

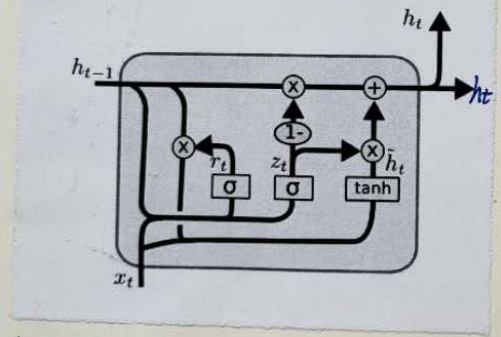
注：LSTM 模型最终有3个输出：

2个 h_t，1个 C_t。四则 代码中应对

3个变量接收调用返回

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

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：

3个输出：h_0, h_t, C_t

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）

2个输出：h_t, h_t

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 → 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/