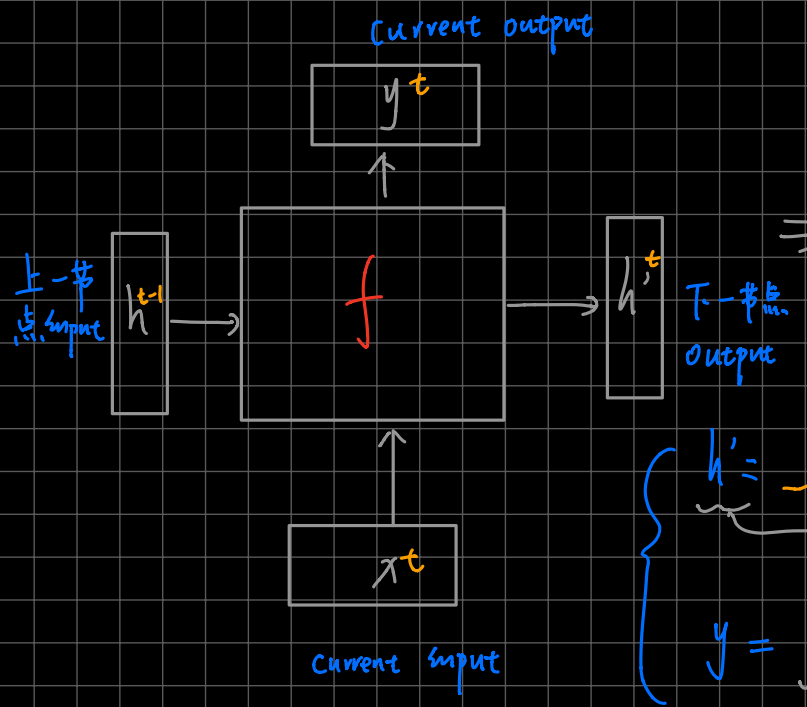
Forward Propogation

I. Naive RNN:



$\Rightarrow$ $h', y = f(h, x)$

输出 作用函数 输入

上一轮中间输出

词向量输入

$$\begin{cases} h' = \sigma(W^h \cdot h^{t-1} + W^x \cdot x^t) \\ y = \sigma(W^o \cdot h') \end{cases}$$

tanh / Relu

vectors，同维度大小

$\Rightarrow$ 或写成：

$$h' = g(W_{hh} h^{t-1} + W_{hx} x^t + b_a)$$

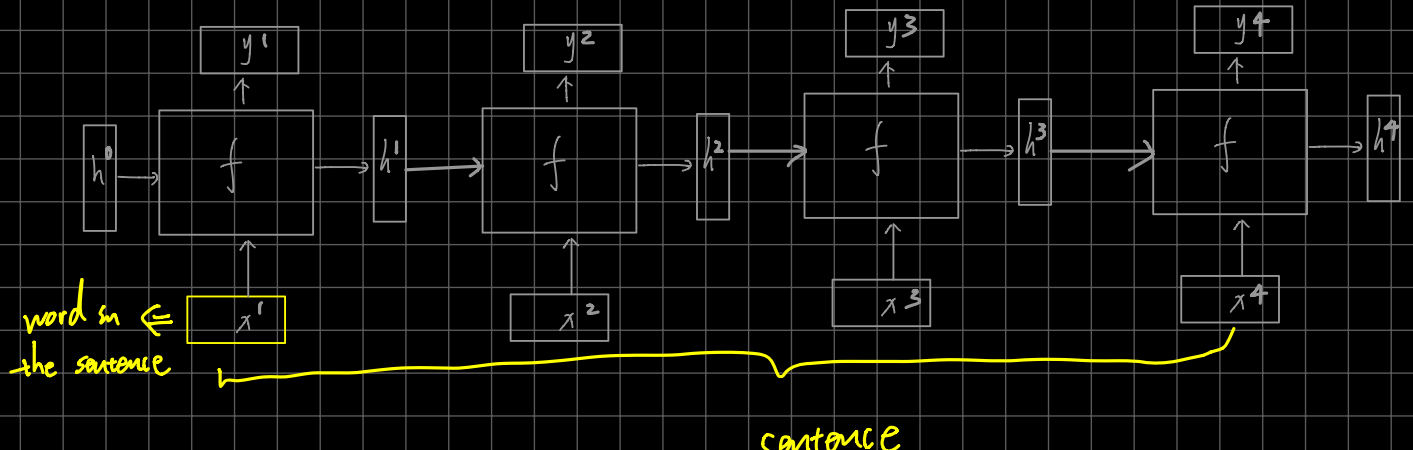
$$\begin{bmatrix} W^h & W^x \end{bmatrix} \begin{bmatrix} h^{t-1} \\ x^t \end{bmatrix}$$

100 / 10000

σ: softmax 作用：分类

维度映射 (或写成 $= \sigma(W_{yh} h' + b_y)$)

△ 不论 input / output 序列有多长，我们只需要 1 个 f 函数.



II LSTM:

为何提出LSTM:

解决RNN的梯度消失问题，在更长的序列中有更好的表现。
Gradient Vanish

然而LSTM并不能解决梯度爆炸问题。⇒ 梯度剪裁算法
Gradient explosion　　Gradient Clipping Algorithm.

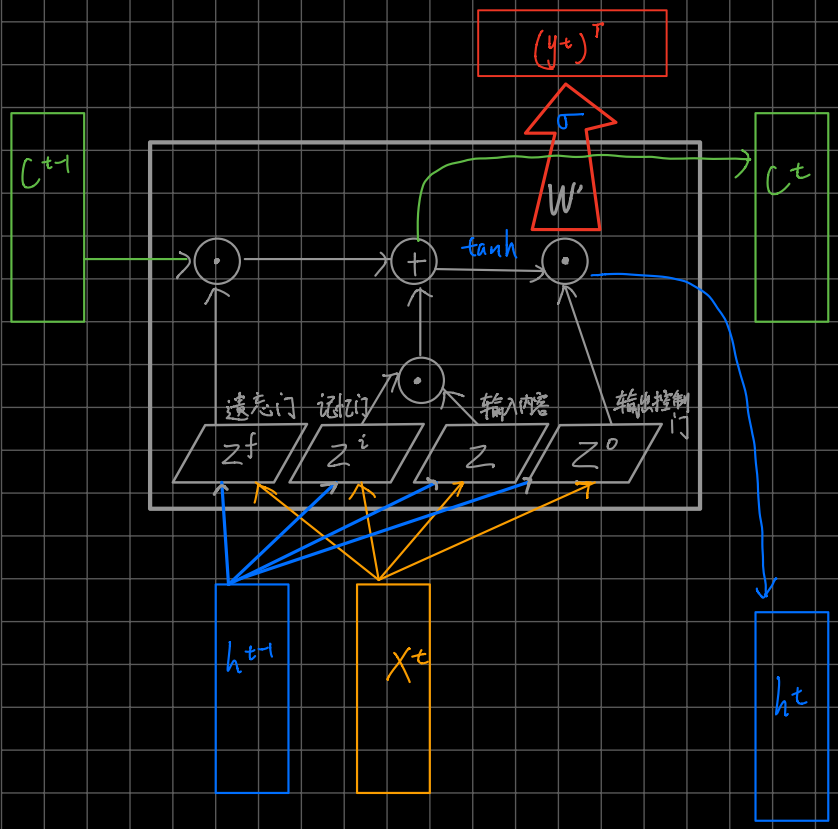
Gradient clipping
- 梯度上限剪裁：梯度的某个维度大于某个上限⇒剪裁为上限
- 梯度同比收缩：梯度(L_2范数)大于上限⇒

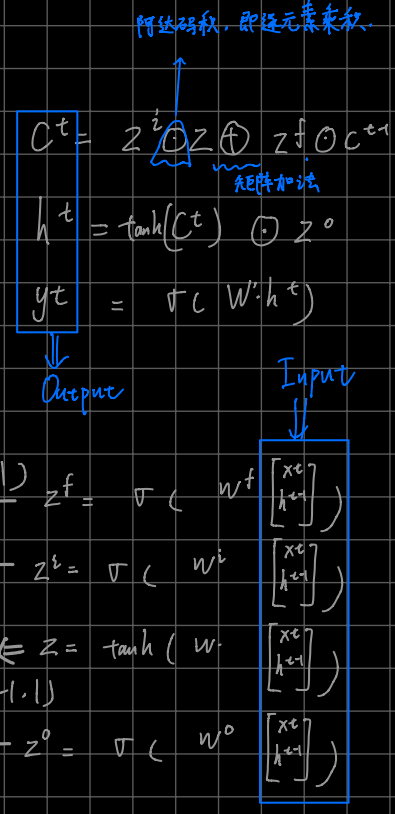
$$梯度 = \frac{梯度}{范数}$$

△ 在RNN及其变种(LSTM.GRU)网络中常出现梯度爆炸问题 ⇒ 梯度随步长累积。

△ 组成RNN单元(LSTM.GRU...)的组合($2^4, 2^2, 2^0$等)最好理解为对探索的一种约束，而非一种工程意义上的设计(e.g遗忘门.记忆门等)。

⇒ 因为RNN单元的作用由权重来决定，权重是通过端到端的学习来调整的，因此RNN单元类型不同只能说是对搜索的限制不同，并非是具有特定的工程意义。





C^t: cell state 相当于RNN中 h^t，主要用于保存先前结点数据，改变很慢，有利于长期记忆

h^t: hidden state 在不同节点下有很大区别，有利于短期记忆

LSTM内部处理三阶段：

I. 遗忘阶段 ⇒ 通过遗忘门 Z^f 控制，忘记重要

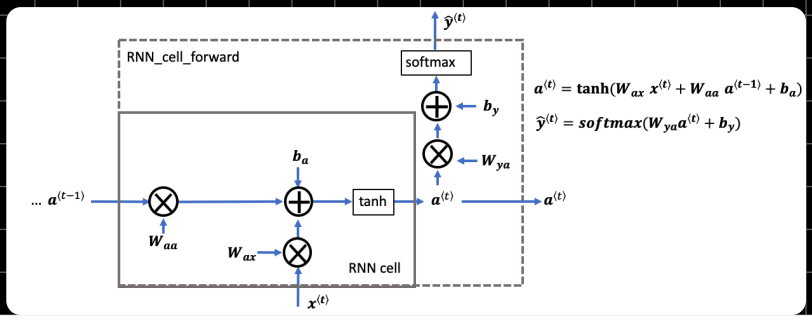
II. 选择记忆阶段 ⇒ 通过选择记忆门 Z^i 控制，记住重要的

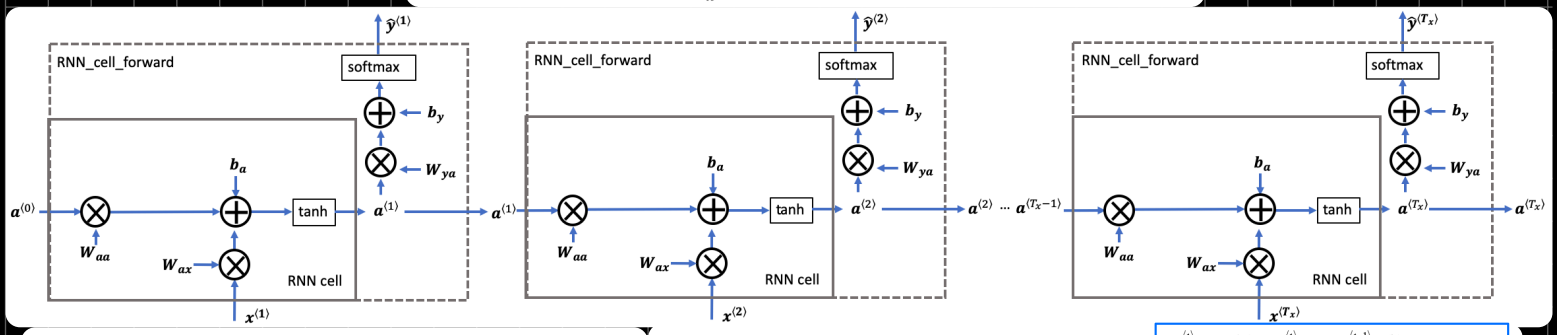
III. 输出阶段 ⇒ 通过输出控制门 Z^o 控制，决定哪些会被当成输出

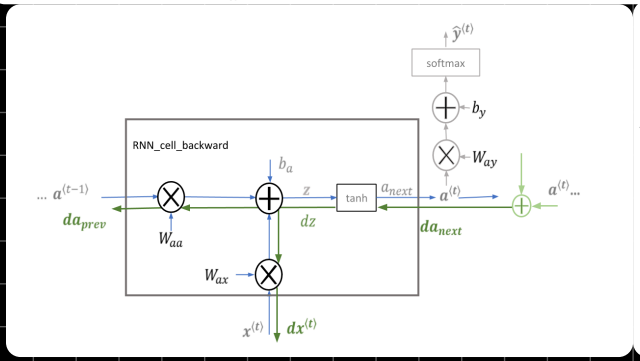
△ LSTM 相比 RNN 引入参数变多，输入、输出多3个传输状态，使训练难度增大许多 ⇒ GRU模型，参数更少，更容易训练

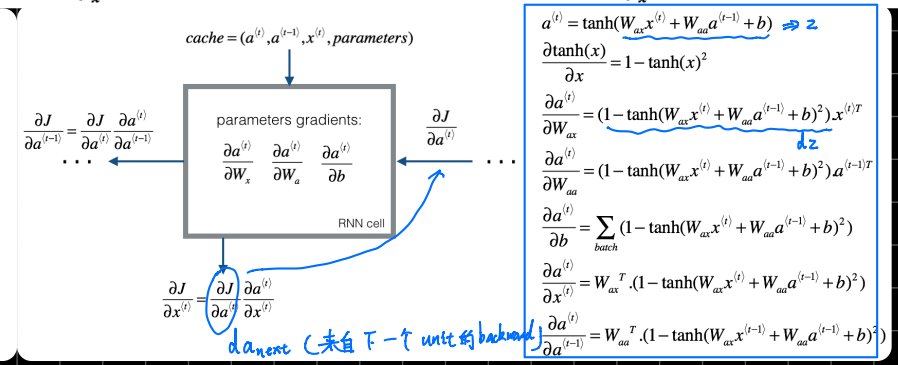
Backward Propagation

I. RNN









$$\frac{\partial J}{\partial W_{ax}} = \frac{\partial J}{\partial a^{\langle t \rangle}} \cdot \frac{\partial a^{\langle t \rangle}}{\partial W_{ax}}$$

$da_{next} \cdot (\cdots)$

$$a^{\langle t \rangle} = \tanh(W_{ax}x^{\langle t \rangle} + W_{aa}a^{\langle t-1 \rangle} + b_a) \tag{-}$$

$$\frac{\partial \tanh(x)}{\partial x} = 1 - \tanh^2(x) \tag{-}$$

$$dW_{ax} = (da_{next} * (1 - \tanh^2(W_{ax}x^{\langle t \rangle} + W_{aa}a^{\langle t-1 \rangle} + b_a)))x^{\langle t \rangle T} \tag{1}$$

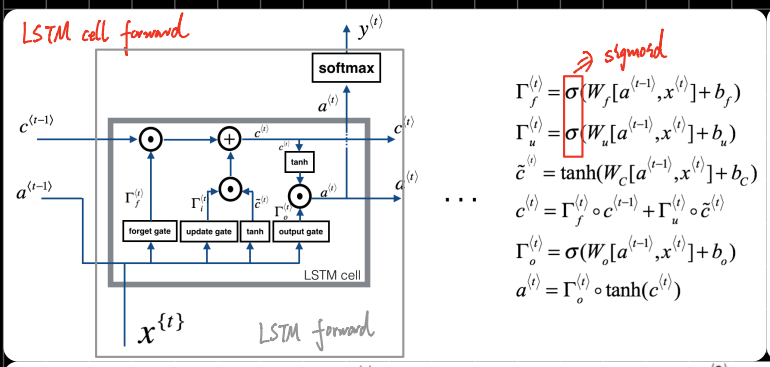
$$dW_{aa} = (da_{next} * (1 - \tanh^2(W_{ax}x^{\langle t \rangle} + W_{aa}a^{\langle t-1 \rangle} + b_a)))a^{\langle t-1 \rangle T} \tag{2}$$

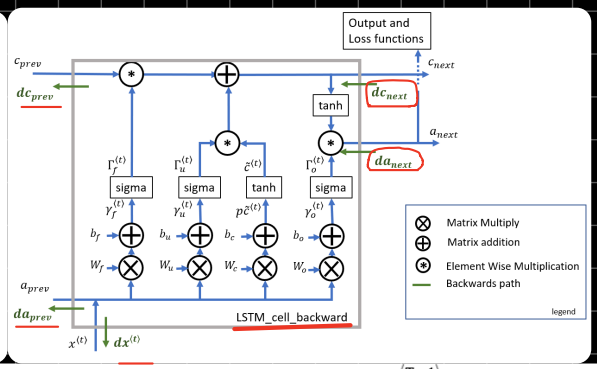
$$db_a = \sum_{batch}(da_{next} * (1 - \tanh^2(W_{ax}x^{\langle t \rangle} + W_{aa}a^{\langle t-1 \rangle} + b_a))) \tag{3}$$

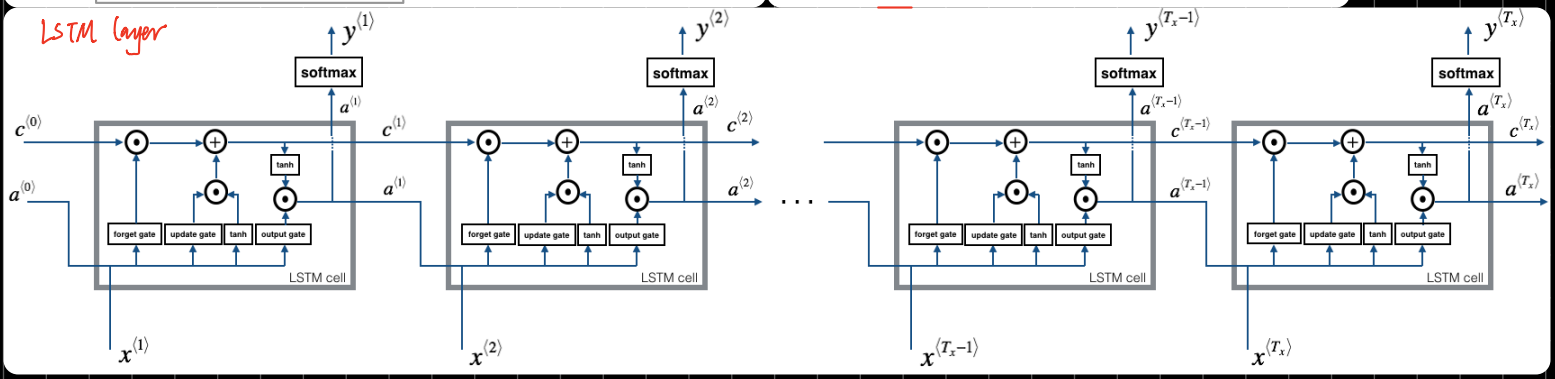
$$dx^{\langle t \rangle} = W_{ax}{}^T(da_{next} * (1 - \tanh^2(W_{ax}x^{\langle t \rangle} + W_{aa}a^{\langle t-1 \rangle} + b_a))) \tag{4}$$

$$da_{prev} = W_{aa}{}^T(da_{next} * (1 - \tanh^2(W_{ax}x^{\langle t \rangle} + W_{aa}a^{\langle t-1 \rangle} + b_a))) \tag{5}$$

II. LSTM







LSTM Backward: 分 0. I. II. III 共4步计算，输入为 da_{next}, dC_{next}, 输出为: $\begin{bmatrix} da_{prev} \\ dc_{prev} \\ dx_t \end{bmatrix}$

3.2.2 gate derivatives

Note the location of the gate derivatives ($\gamma..$) between the dense layer and the activation function (see graphic above). This is convenient for computing parameter derivatives in the next step.

$$d\gamma_o^{\langle t\rangle} = da_{next} * \tanh(c_{next}) * \Gamma_o^{\langle t\rangle} * \left(1 - \Gamma_o^{\langle t\rangle}\right) \tag{7}$$

$$dp\tilde{c}^{\langle t\rangle} = \left(dc_{next} * \Gamma_u^{\langle t\rangle} + \Gamma_o^{\langle t\rangle} * (1 - \tanh^2(c_{next})) * \Gamma_u^{\langle t\rangle} * da_{next}\right) * \left(1 - \left(\tilde{c}^{\langle t\rangle}\right)^2\right) \tag{8}$$

0.

$$d\gamma_u^{\langle t\rangle} = \left(dc_{next} * \tilde{c}^{\langle t\rangle} + \Gamma_o^{\langle t\rangle} * (1 - \tanh^2(c_{next})) * \tilde{c}^{\langle t\rangle} * da_{next}\right) * \Gamma_u^{\langle t\rangle} * \left(1 - \Gamma_u^{\langle t\rangle}\right) \tag{9}$$

$$d\gamma_f^{\langle t\rangle} = \left(dc_{next} * c_{prev} + \Gamma_o^{\langle t\rangle} * (1 - \tanh^2(c_{next})) * c_{prev} * da_{next}\right) * \Gamma_f^{\langle t\rangle} * \left(1 - \Gamma_f^{\langle t\rangle}\right) \tag{10}$$

3.2.3 parameter derivatives

e.g.: $\frac{\partial J}{\partial W_f} = \frac{\partial J}{\partial a} \cdot \frac{\partial a}{\partial W_f}$

$$dW_f = d\gamma_f^{\langle t\rangle} \begin{bmatrix} a_{prev} \\ x_t \end{bmatrix}^T \tag{11}$$

$$dW_u = d\gamma_u^{\langle t\rangle} \begin{bmatrix} a_{prev} \\ x_t \end{bmatrix}^T \tag{12}$$

$$dW_c = dp\tilde{c}^{\langle t\rangle} \begin{bmatrix} a_{prev} \\ x_t \end{bmatrix}^T \tag{13}$$

$$dW_o = d\gamma_o^{\langle t\rangle} \begin{bmatrix} a_{prev} \\ x_t \end{bmatrix}^T \tag{14}$$

$\begin{bmatrix} a^{\langle t-1\rangle} \\ x_t \end{bmatrix}^T$

I

To calculate db_f, db_u, db_c, db_o you just need to sum across all 'm' examples (axis= 1) on $d\gamma_f^{\langle t\rangle}, d\gamma_u^{\langle t\rangle}, dp\tilde{c}^{\langle t\rangle}, d\gamma_o^{\langle t\rangle}$ respectively. Note that you should have the `keepdims = True` option.

sum up ⇒ II

$$db_f = \sum_{batch} d\gamma_f^{\langle t\rangle} \tag{15}$$

$$db_u = \sum_{batch} d\gamma_u^{\langle t\rangle} \tag{16}$$

$$db_c = \sum_{batch} d\gamma_c^{\langle t\rangle} \tag{17}$$

$$db_o = \sum_{batch} d\gamma_o^{\langle t\rangle} \tag{18}$$

Finally, you will compute the derivative with respect to the previous hidden state, previous memory state, and input.

III. 最终要backward的

$$da_{prev} = W_f^T d\gamma_f^{\langle t\rangle} + W_u^T d\gamma_u^{\langle t\rangle} + W_c^T dp\tilde{c}^{\langle t\rangle} + W_o^T d\gamma_o^{\langle t\rangle} \tag{19}$$

Here, to account for concatenation, the weights for equations 19 are the first n_a, (i.e. $W_f = W_f[:, :n_a]$ etc...)

$$dc_{prev} = dc_{next} * \Gamma_f^{\langle t\rangle} + \Gamma_o^{\langle t\rangle} * (1 - \tanh^2(c_{next})) * \Gamma_f^{\langle t\rangle} * da_{next} \tag{20}$$

$$dx^{\langle t\rangle} = W_f^T d\gamma_f^{\langle t\rangle} + W_u^T d\gamma_u^{\langle t\rangle} + W_c^T dp\tilde{c}^{\langle t\rangle} + W_o^T d\gamma_o^{\langle t\rangle} \tag{21}$$

where the weights for equation 21 are from n_a to the end, (i.e. $W_f = W_f[:, n_a:]$ etc...)

△ Note: 在RNN, LSTM 的 Backward 过程中，计算出的 gradients 均要储存于 cache 之中，等到所有该层的 units 均通过 Backward 计算之后，再通过: I. Gradient Clipping (此对RNN类非常重要)

II. update parameters.

实例: ① 恐龙名字生成 project

② Jazz音乐生成 project