WMLS TOPIC #2

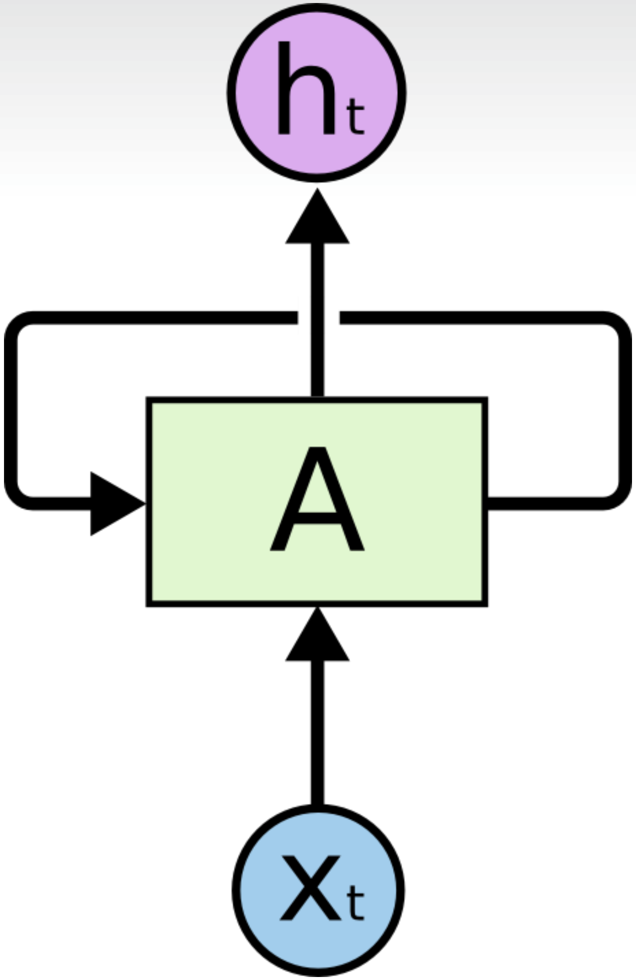
Long Short Term Memory

Dec 31, 2019

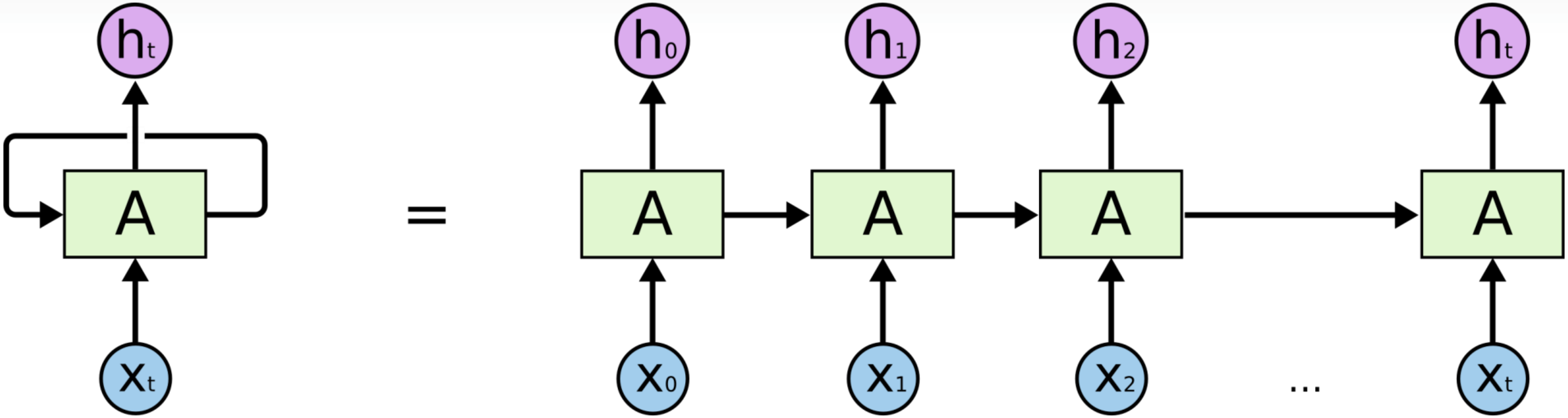
TEAM B
이충섭
이원빈
이종수

CONTENTS

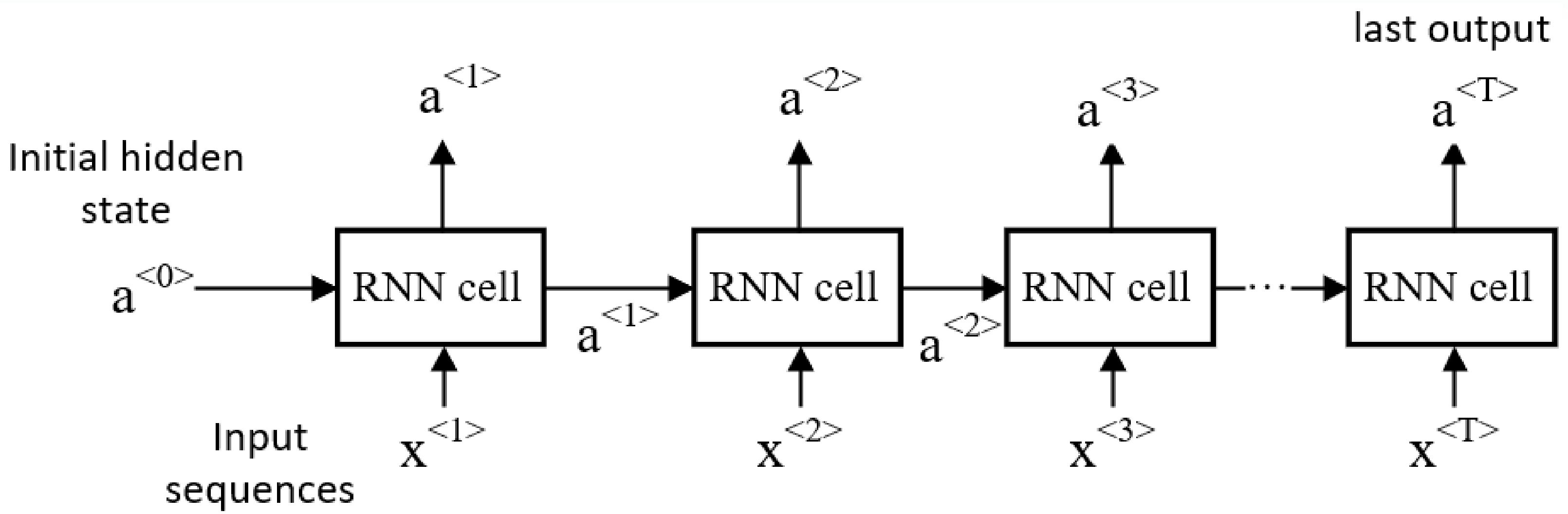
RNN



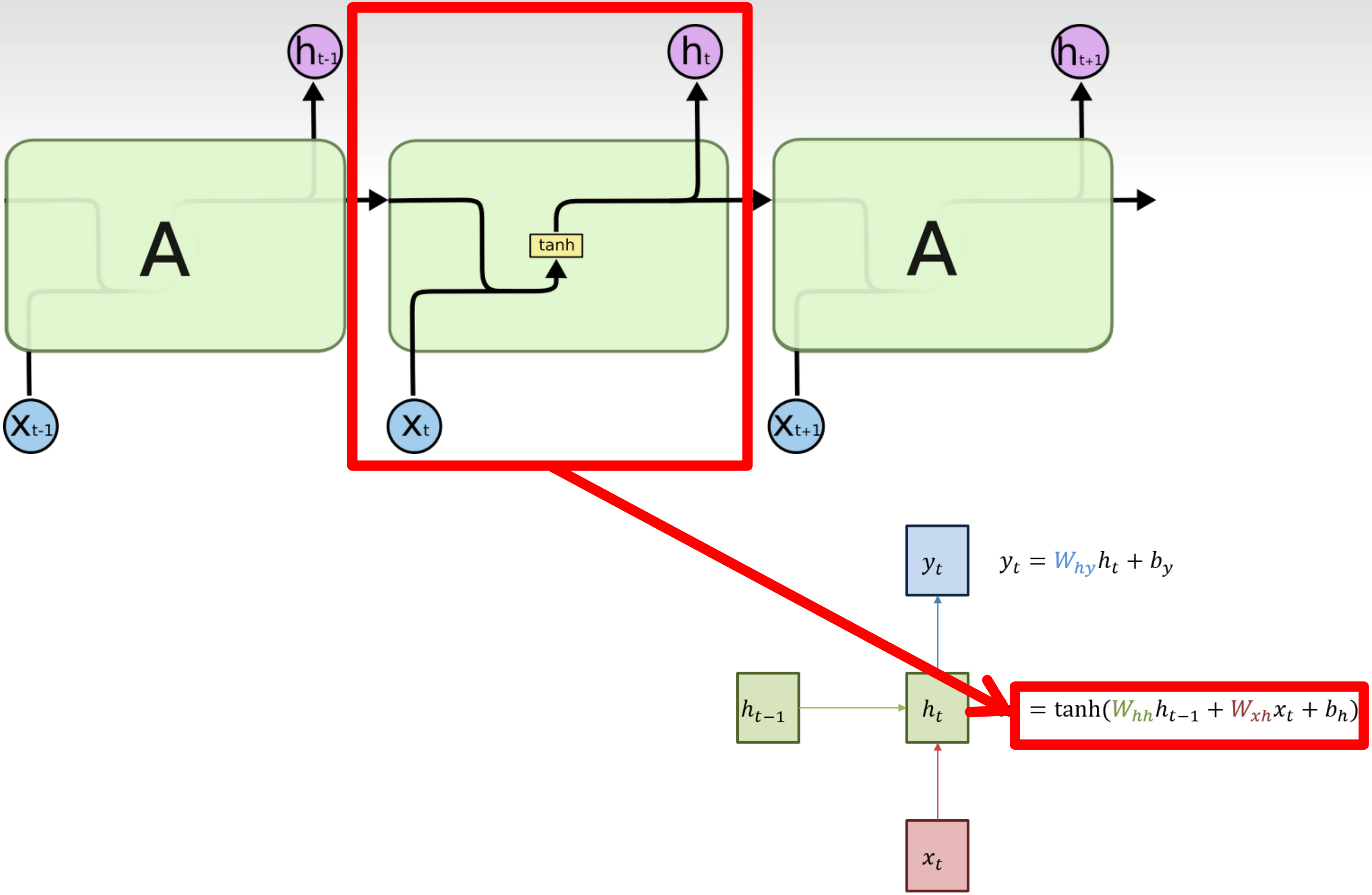
RNN



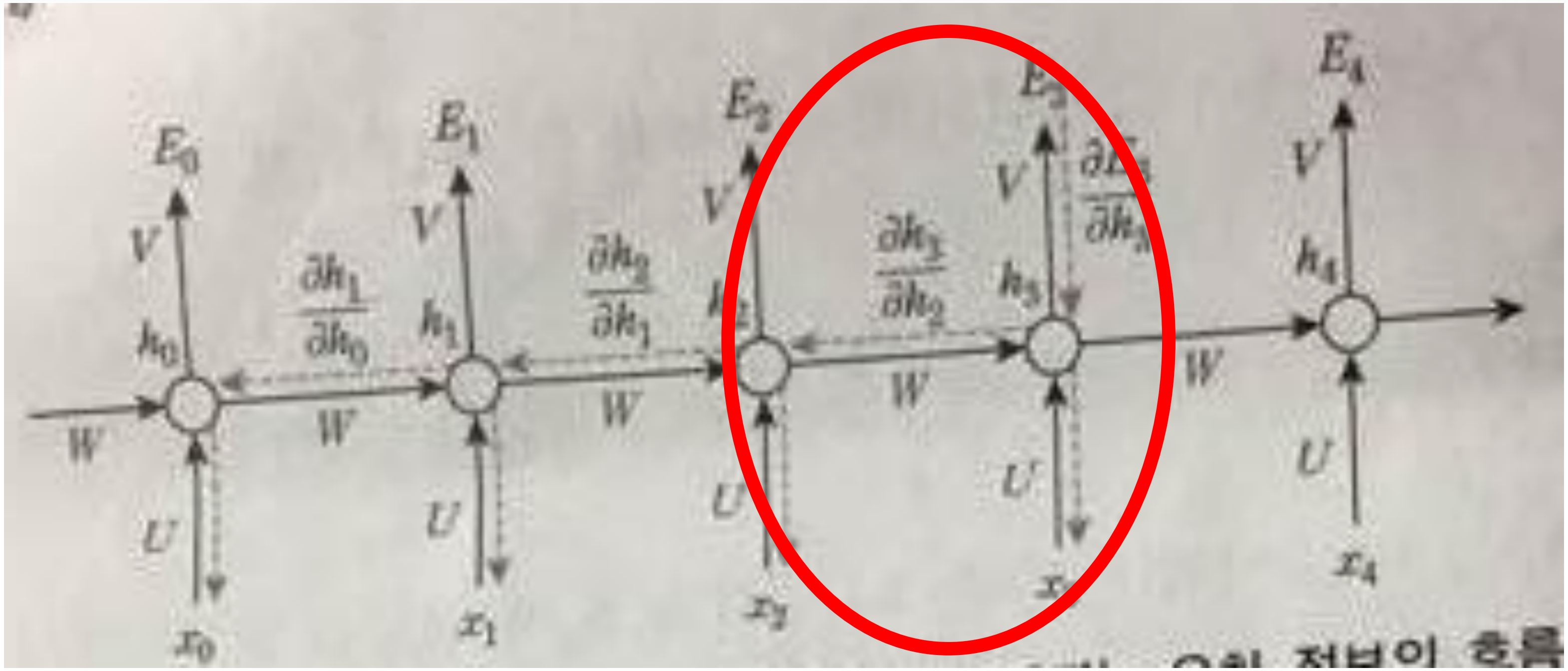
Mechanism of RNN



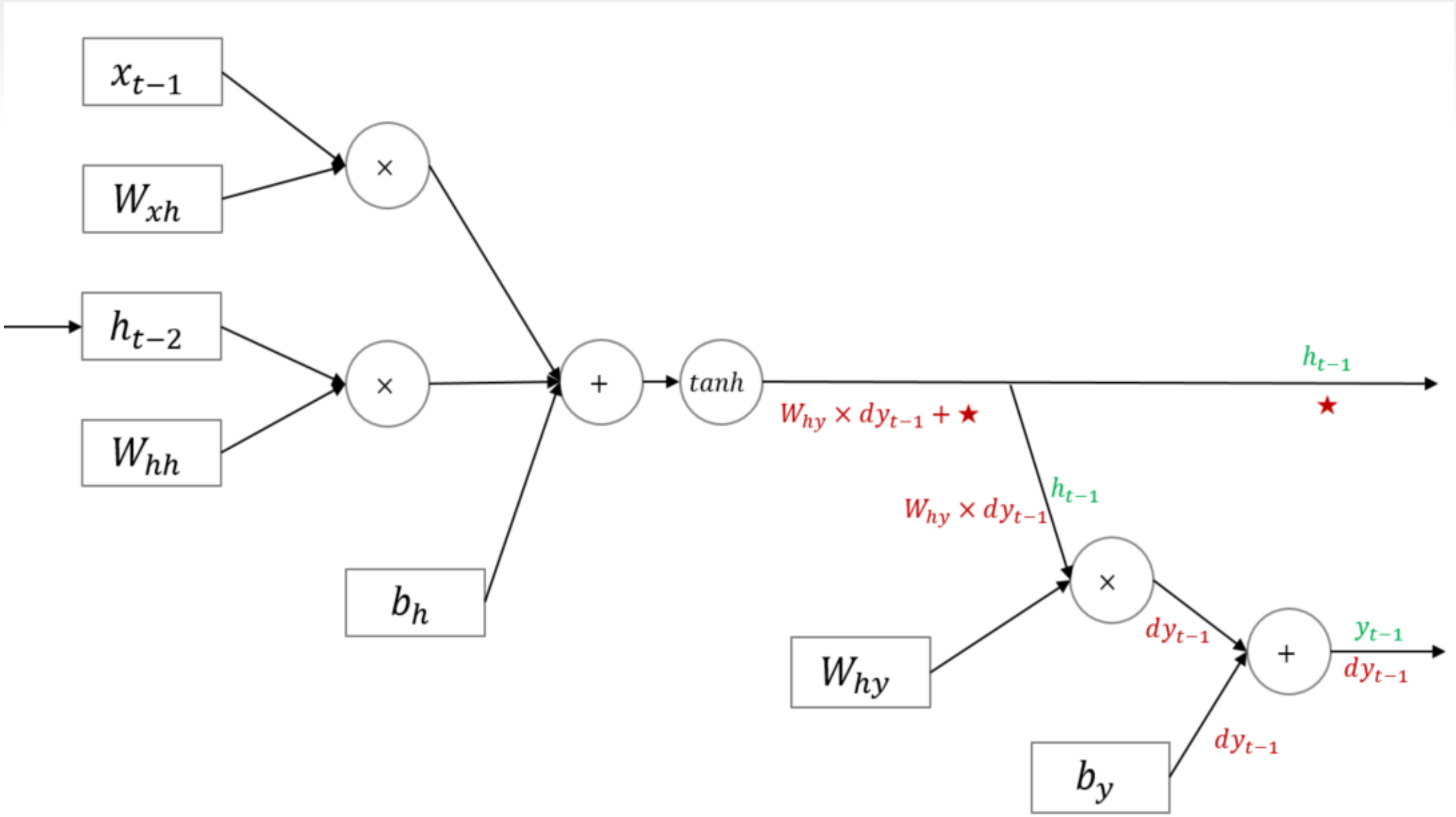
Mechanism of RNN



Back Propagation of RNN



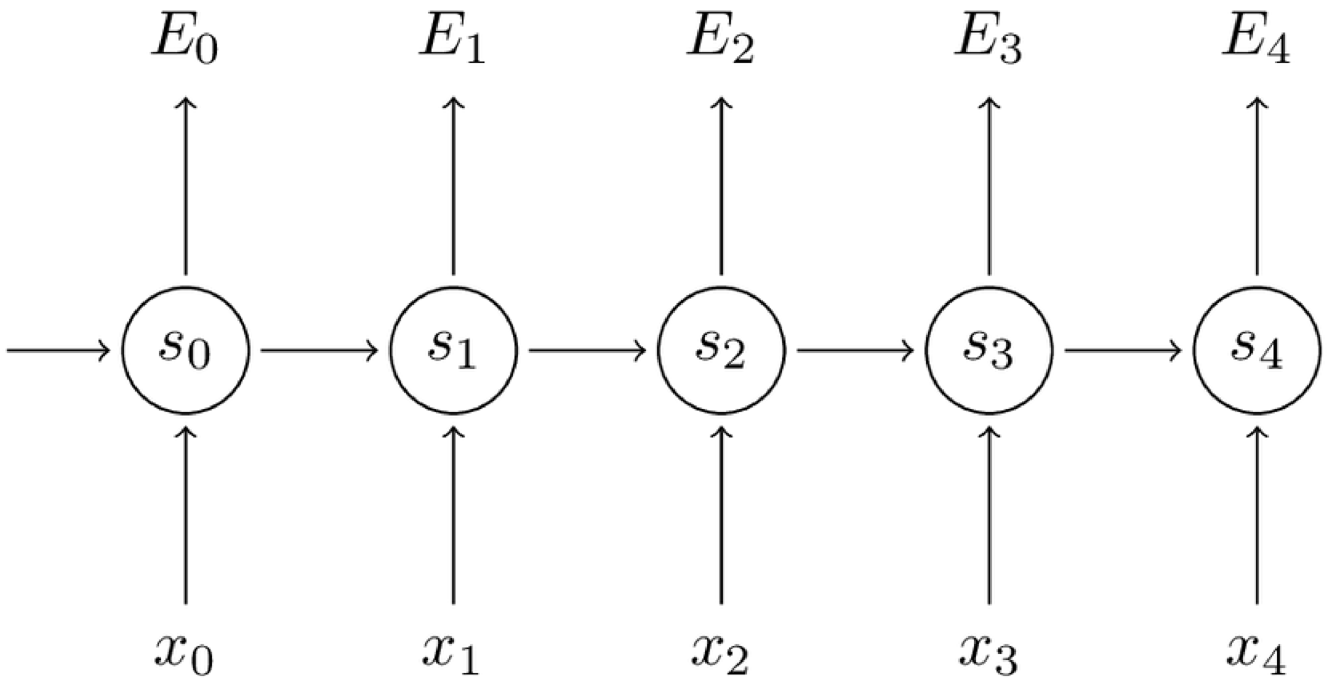
Back Propagation of RNN



Back Propagation of RNN

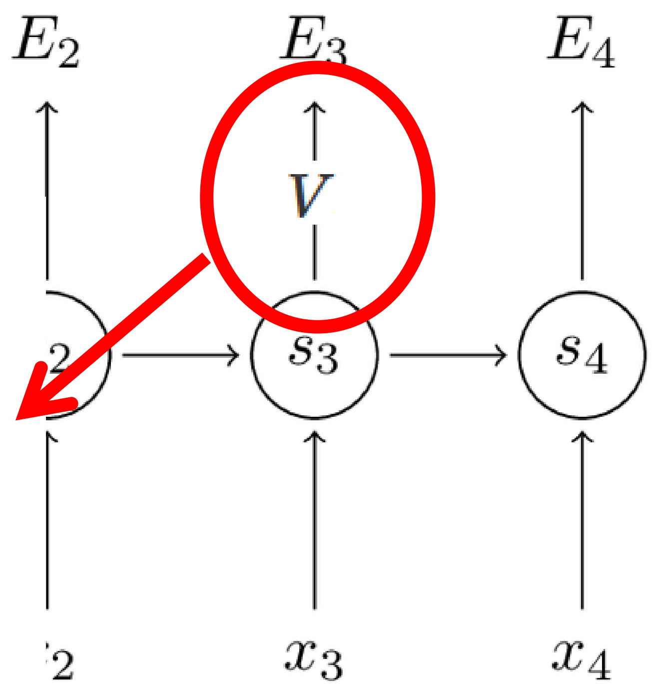
$$s_t = \tanh(Ux_t + Ws_{t-1})$$
$$\hat{y}_t = softmax(Vs_t)$$

$$E(y_t, \hat{y}_t) = -y_t \log \hat{y}_t$$
$$E(y, \hat{y}) = -\sum_t E_t(y_t, \hat{y}_t)$$
$$= -\sum_t -y_t \log \hat{y}_t$$

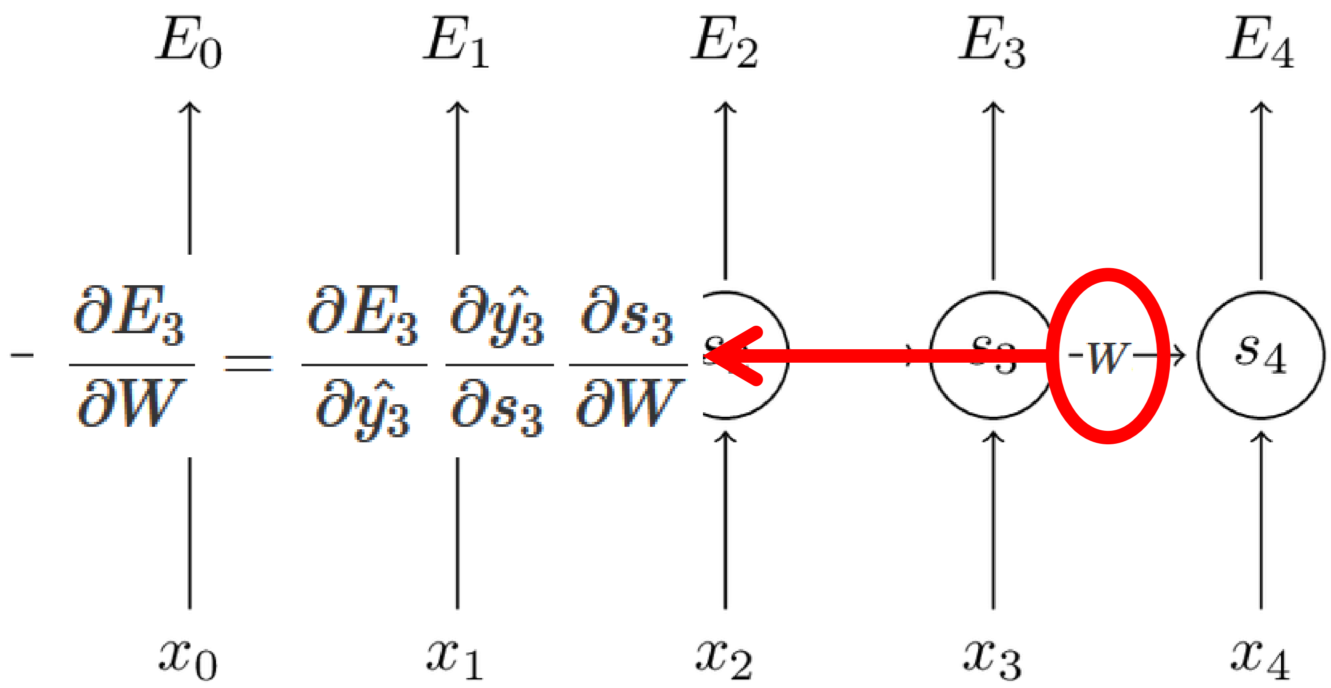


Back Propagation of RNN

$$
\begin{aligned}
\frac{\partial E_3}{\partial V} &= \frac{\partial E_3}{\partial \hat{y_3}} \frac{\partial \hat{y_3}}{\partial V} \\
&= \frac{\partial E_3}{\partial \hat{y_3}} \frac{\partial \hat{y_3}}{\partial z_3} \frac{\partial z_3}{\partial V} \\
&= (\hat{y_3} - y_3) \otimes s_3
\end{aligned}
$$



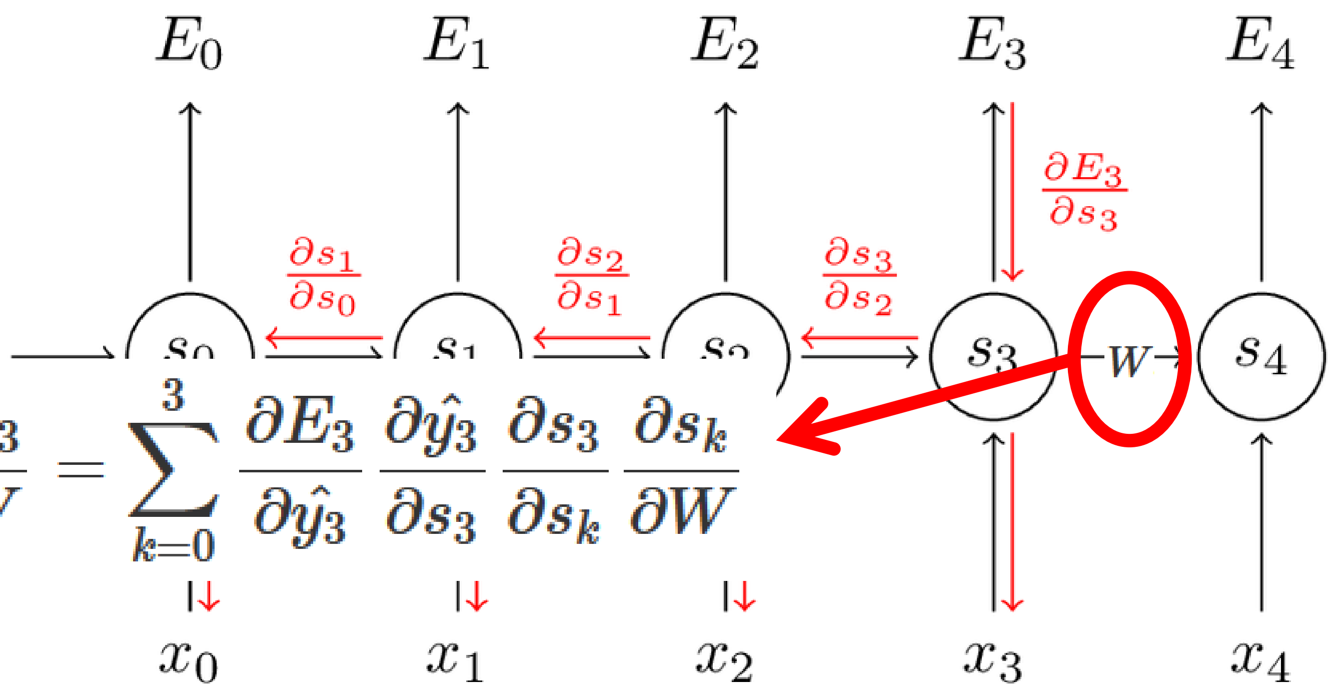
Back Propagation of RNN



$$\frac{\partial E_3}{\partial W} = \frac{\partial E_3}{\partial \hat{y}_3}\frac{\partial \hat{y}_3}{\partial s_3}\frac{\partial s_3}{\partial W}$$

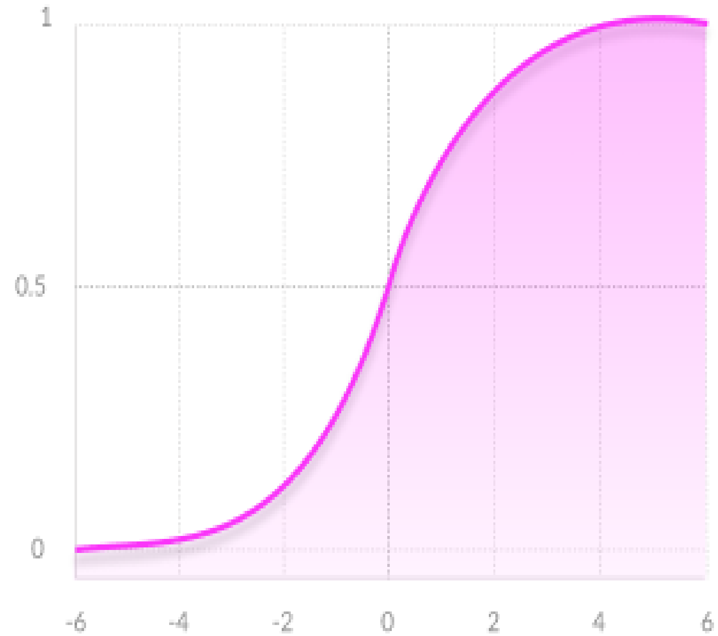
$$s_t = \tanh(Ux_t + Ws_{t-1})$$

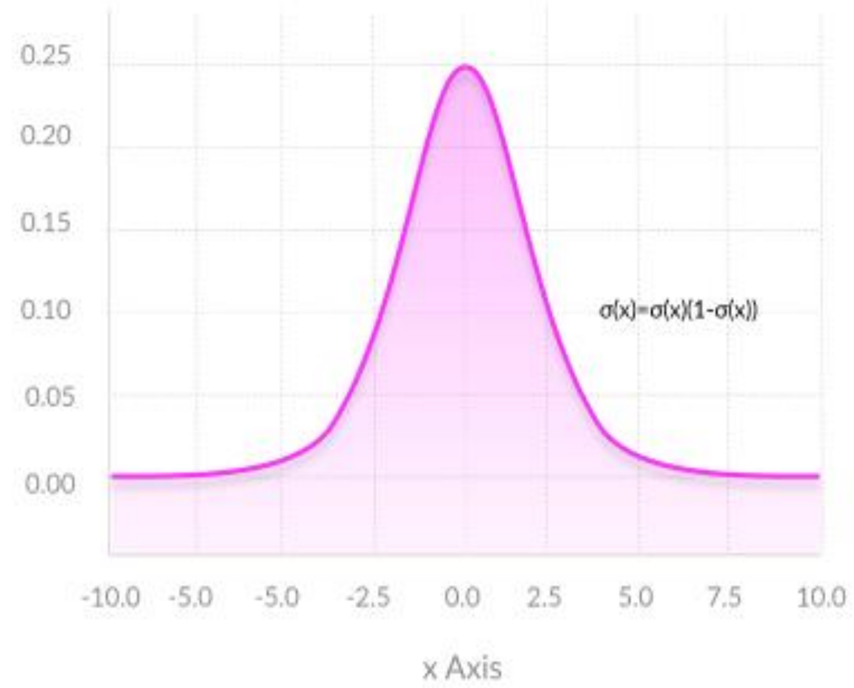
Back Propagation of RNN



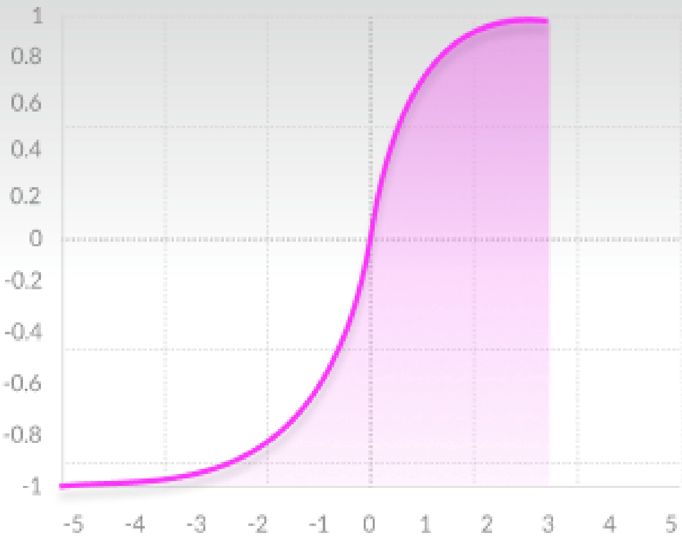
$$\frac{\partial E_3}{\partial W} = \sum_{k=0}^{3} \frac{\partial E_3}{\partial \hat{y}_3} \frac{\partial \hat{y}_3}{\partial s_3} \frac{\partial s_3}{\partial s_k} \frac{\partial s_k}{\partial W}$$

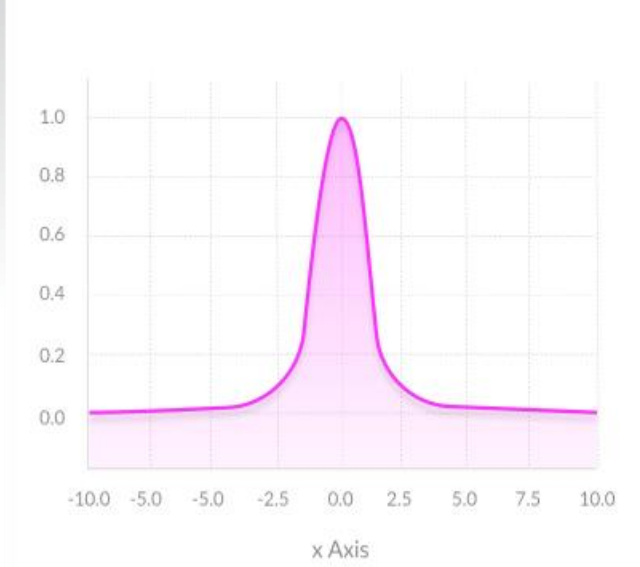
Activation Function

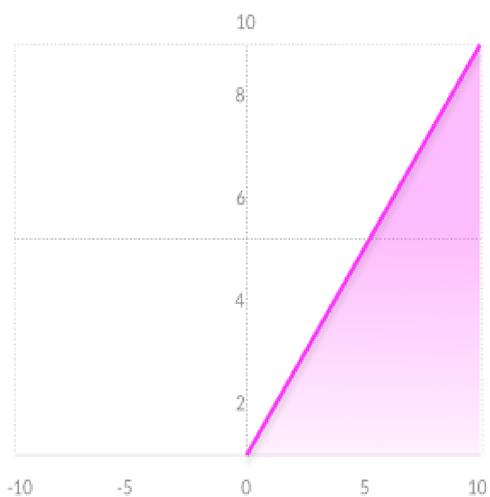


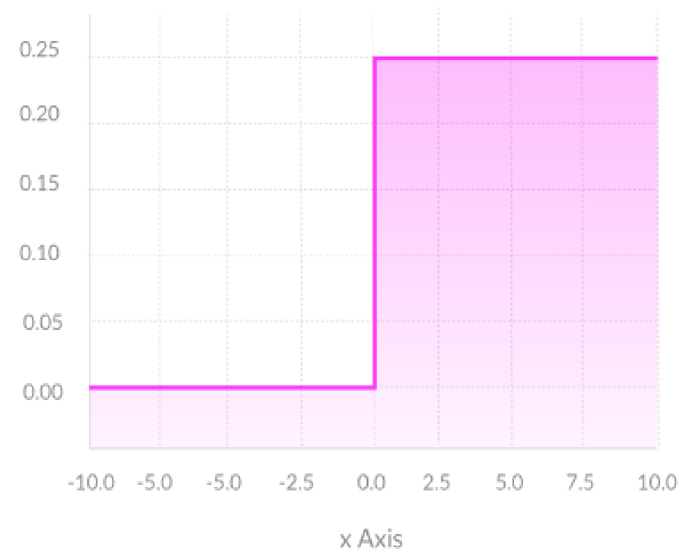


Activation Function

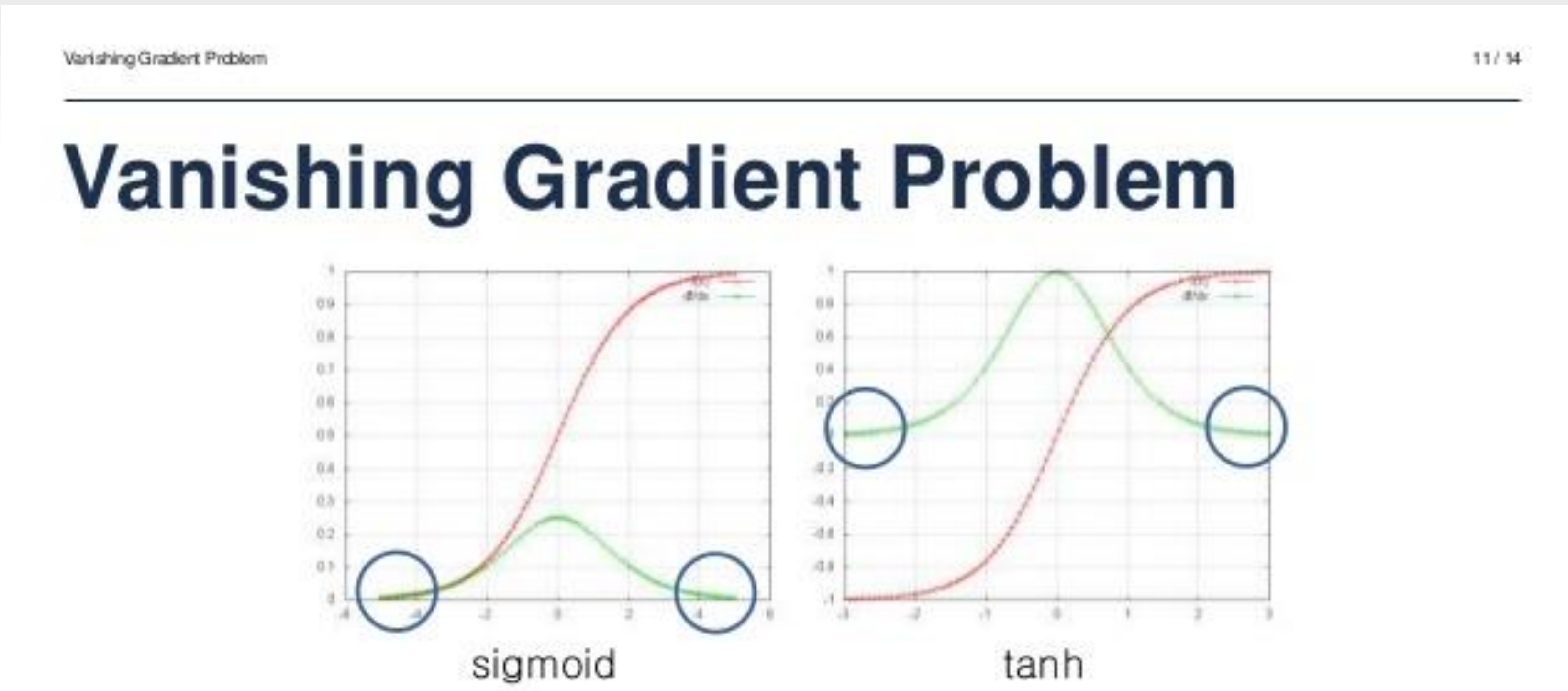




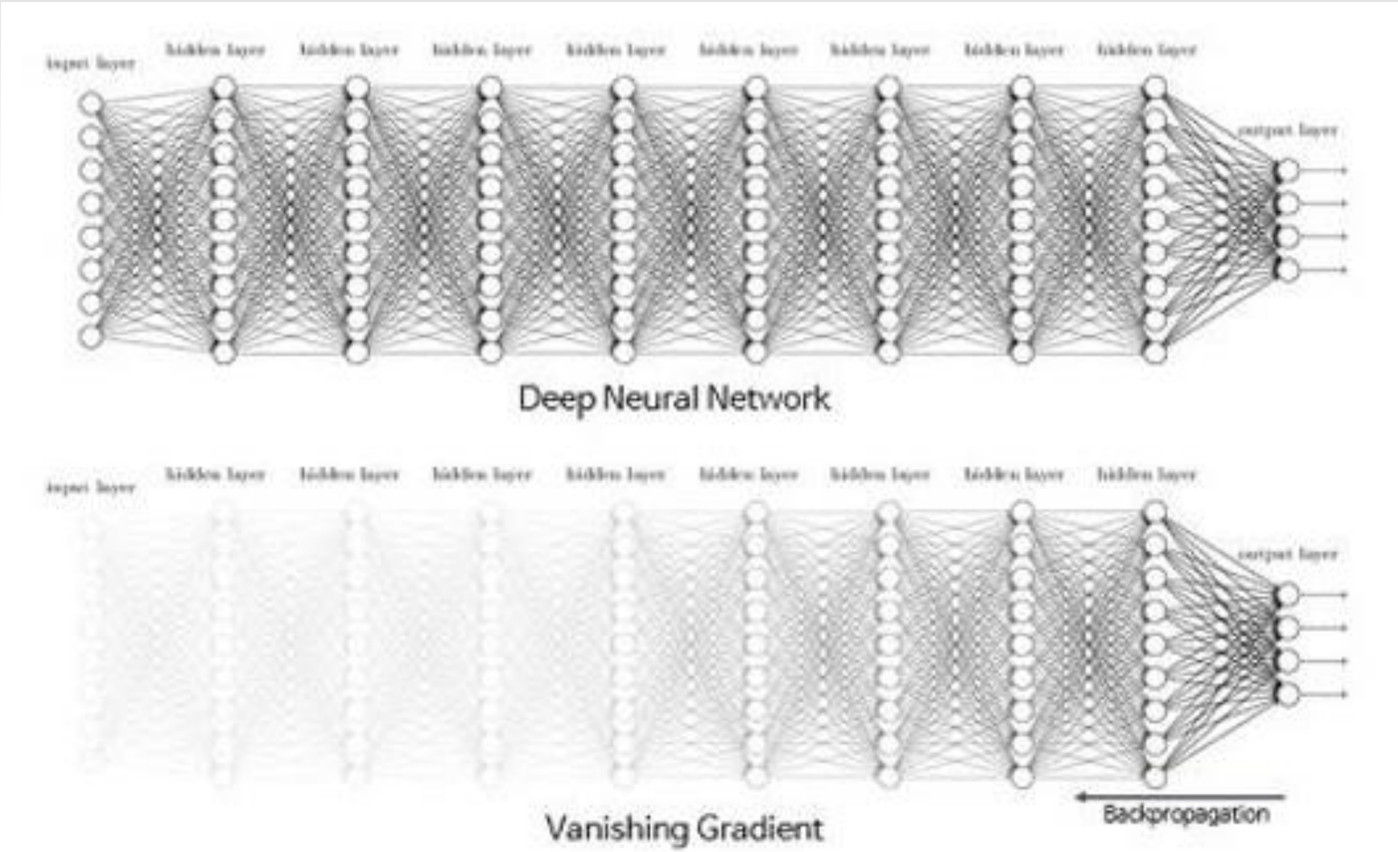




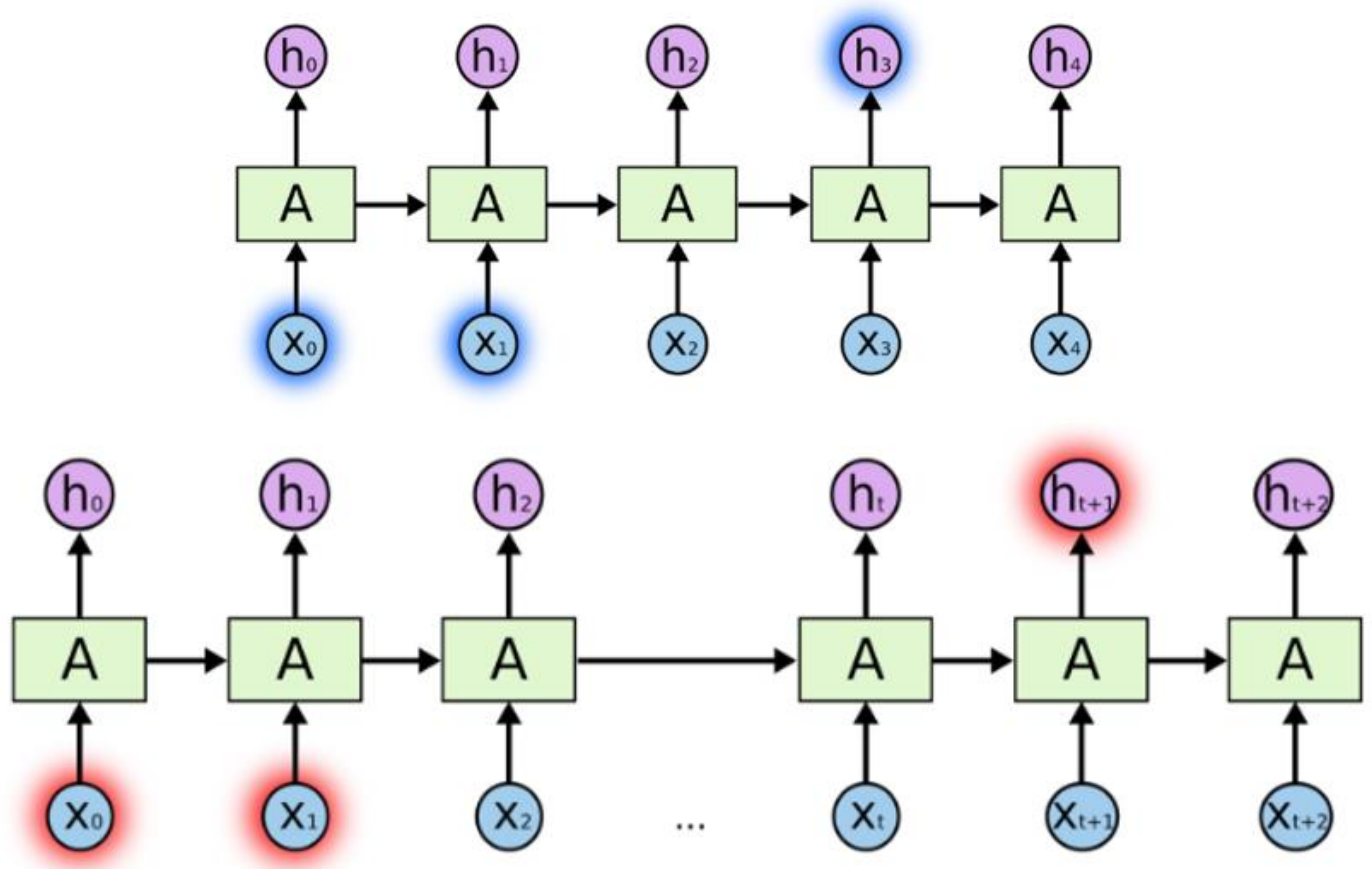
Problem (1)


Vanishing Gradient Problem
11 / 14
Vanishing Gradient Problem
sigmoid
tanh

Problem (1)

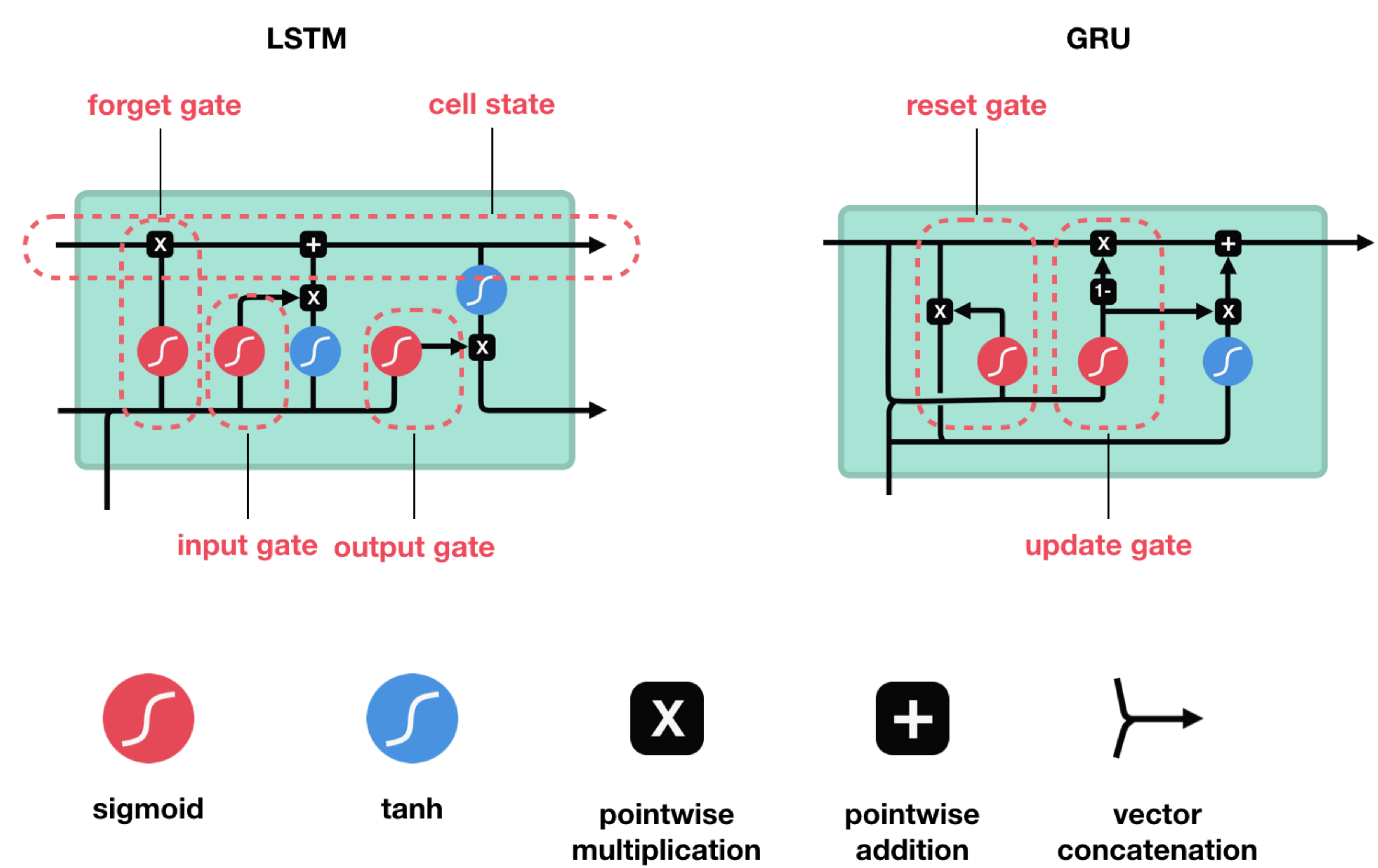


Problem (2)



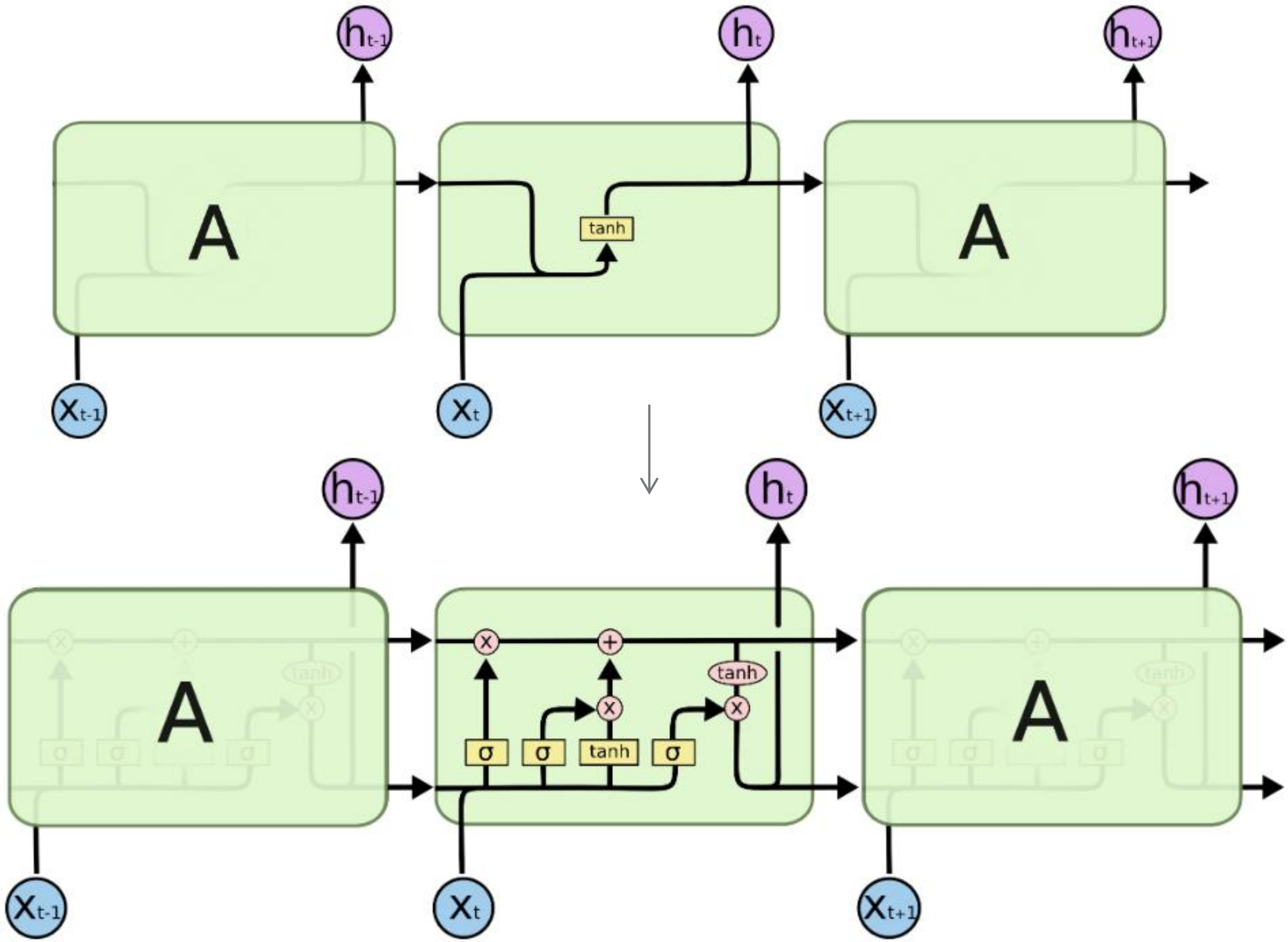
h0
h1
h2
h3
h4
A
A
A
A
A
X0
X1
X2
X3
X4
h0
h1
h2
ht
ht+1
ht+2
A
A
A
A
A
A
X0
X1
X2
...
Xt
Xt+1
Xt+2

LSTM/ GRU

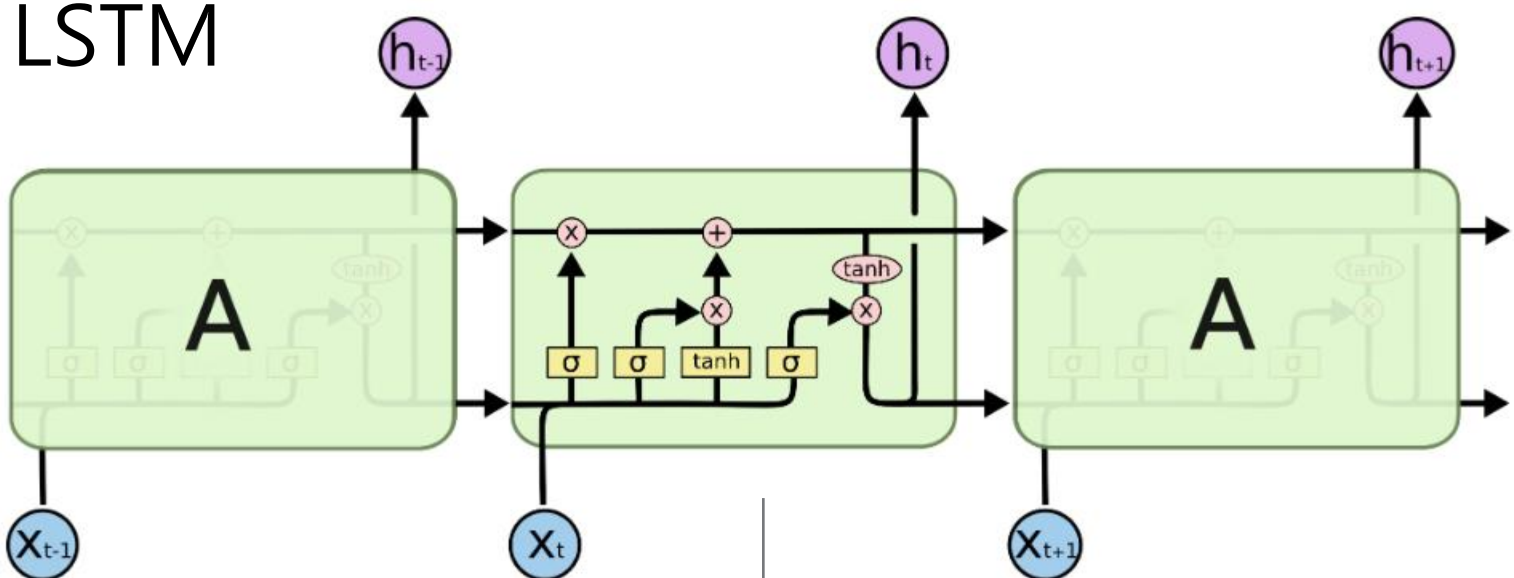


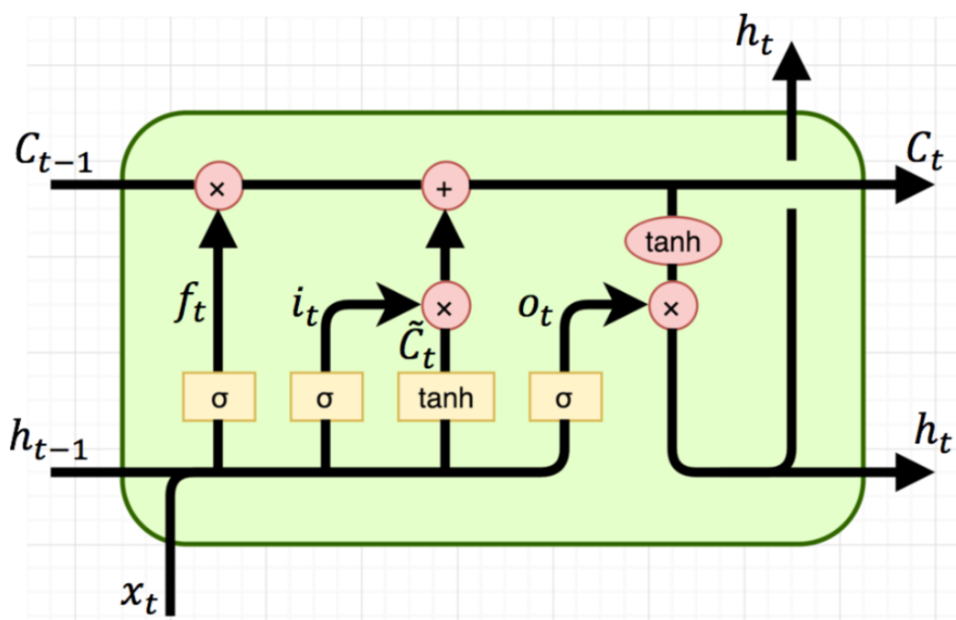
LSTM
GRU
forget gate
cell state
reset gate
input gate
output gate
update gate
1-
sigmoid
tanh
pointwise multiplication
pointwise addition
vector concatenation

RNN changes into LSTM



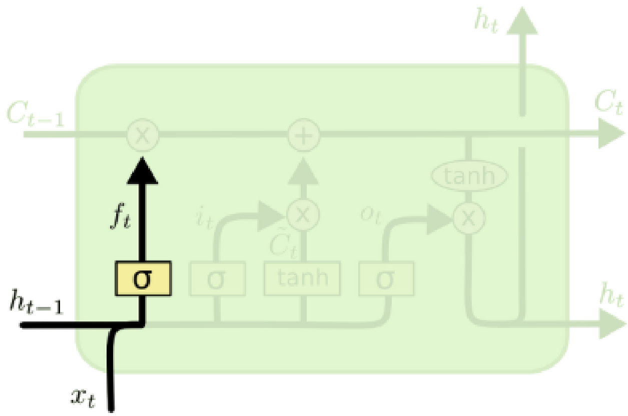
LSTM





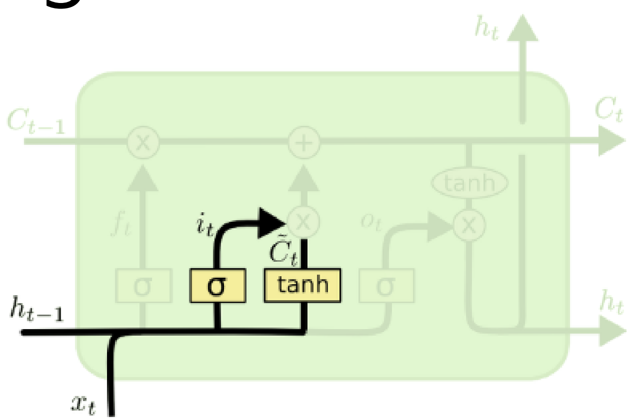
(a) Long Short-Term Memory

Forget gate



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

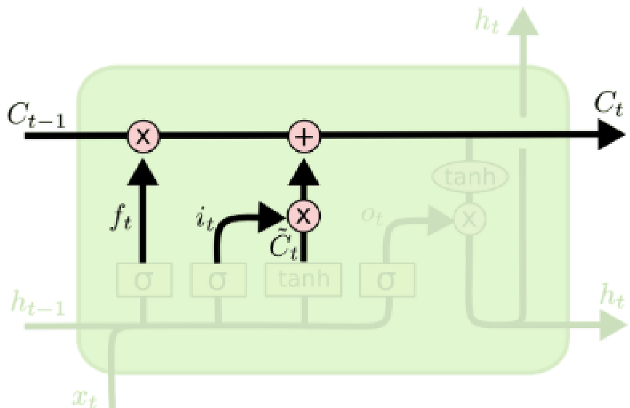
Input gate



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

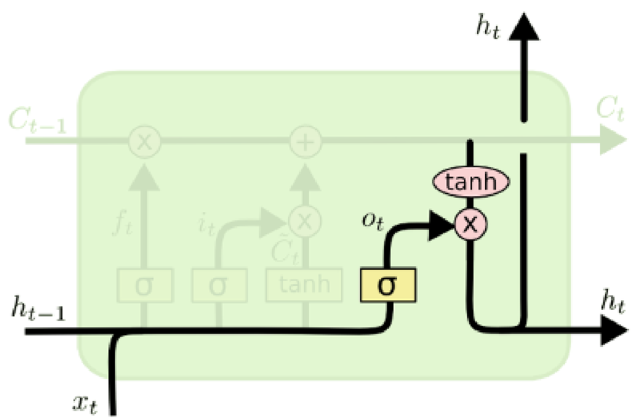
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \ + \ b_C)$$

Long term memory



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

Output gate and Short term memory



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

$$h_t = o_t * \tanh\left(C_t\right)$$

LSTM
- Algorithm

입력 : 서열 데이터 $(\boldsymbol{x}_1, \boldsymbol{x}_2, \cdots, \boldsymbol{x}_T)$

가중치 $W_c, U_c, W_i, U_i, W_f, U_f, W_o, U_o, V_0$

편차항 $\boldsymbol{b}_c, \boldsymbol{b}_i, \boldsymbol{b}_f, \boldsymbol{b}_o$

출력 : LSTM의 출력 $(\boldsymbol{h}_1, \boldsymbol{h}_2, \cdots, \boldsymbol{h}_T)$

1. $\boldsymbol{h}_0 \leftarrow 0$
2. $\boldsymbol{c}_0 \leftarrow 0$

초기화(랜덤 : 0)

3. for $t = 1$ to T 행렬원소의 개수만큼 반복
4. $\quad \boldsymbol{i}_t \leftarrow \sigma(U_i \boldsymbol{x}_t + W_i \boldsymbol{h}_{t-1} + \boldsymbol{b}_i)$ 입력비율
5. $\quad \boldsymbol{a}_t \leftarrow \tanh(U_c \boldsymbol{x}_t + W_c \boldsymbol{h}_{t-1} + \boldsymbol{b}_c)$ 상태값 재정비
6. $\quad \boldsymbol{f}_t \leftarrow \sigma(U_f \boldsymbol{x}_t + W_f \boldsymbol{h}_{t-1} + \boldsymbol{b}_f)$ 망각비율
7. $\quad \boldsymbol{c}_t \leftarrow \boldsymbol{i}_t \circ \boldsymbol{a}_t + \boldsymbol{f}_t \circ \boldsymbol{c}_{t-1}$ 상태셀 정립
8. $\quad \boldsymbol{o}_t \leftarrow \sigma(U_o \boldsymbol{x}_t + W_o \boldsymbol{h}_{t-1} + V_o \boldsymbol{c}_{t-1} + \boldsymbol{b}_o)$ 출력 비율.
9. $\quad \boldsymbol{h}_t \leftarrow \boldsymbol{o}_t \circ \tanh(\boldsymbol{c}_t)$ 출력도출

LSTM BPTT-Mathmetical Proof

$$\frac{\partial E}{\partial c_t^k} = \frac{\partial E}{\partial h_t^k}\frac{\partial h_t^k}{\partial c_t^k} = \frac{\partial E}{\partial h_t^k} o_t^k (1 - \tanh^2(c_t^k))$$

$$\frac{\partial E}{\partial i_t^k} = \frac{\partial E}{\partial c_t^k}\frac{\partial c_t^k}{\partial i_t^k} = \frac{\partial E}{\partial c_t^k} a_t^k$$

$$\frac{\partial E}{\partial f_t^k} = \frac{\partial E}{\partial c_t^k}\frac{\partial c_t^k}{\partial f_t^k} = \frac{\partial E}{\partial c_t^k} c_{t-1}^k$$

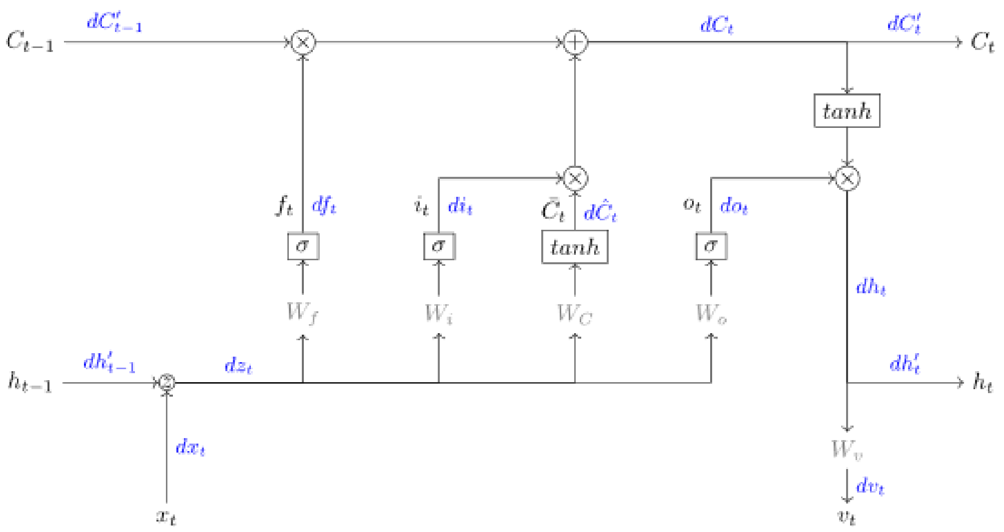
$$\frac{\partial E}{\partial a_t^k} = \frac{\partial E}{\partial c_t^k}\frac{\partial c_t^k}{\partial a_t^k} = \frac{\partial E}{\partial c_t^k} i_t^k$$

$$\frac{\partial E}{\partial o_t^k} = \frac{\partial E}{\partial h_t^k}\frac{\partial h_t^k}{\partial o_t^k} = \frac{\partial E}{\partial h_t^k} tanh(c_t^k)$$

$$\frac{\partial E}{\partial c_{t-1}^k} = \frac{\partial E}{\partial c_t^k}\frac{\partial c_t^k}{\partial c_{t-1}^k} = \frac{\partial E}{\partial c_t^k}\frac{\partial\left(i_t^k a_t^k + f_t^k c_{t-1}^k\right)}{\partial c_{t-1}^k} = \frac{\partial E}{\partial c_t^k} f_t^k$$

$$\Rightarrow \quad \frac{\partial E}{\partial c_{t-p}^k} = \frac{\partial E}{\partial c_t^k} \prod_{n=t-p}^{t} f_n^k$$

LSTM



C_{t-1}
dC'_{t-1}
dC_t
dC'_t
C_t
$tanh$
f_t
df_t
i_t
di_t
$\tilde{C}_t$
$d\hat{C}_t$
o_t
do_t
σ
σ
$tanh$
σ
W_f
W_i
W_C
W_o
dh_t
h_{t-1}
dh'_{t-1}
dz_t
dh'_t
h_t
dx_t
W_v
dv_t
x_t
v_t

LSTM
-Activation Function

Activation Functions and Derivatives

Sigmoid

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

$$\frac{d\sigma(x)}{dx} = \sigma(x) \cdot (1 - \sigma(x))$$

Tanh

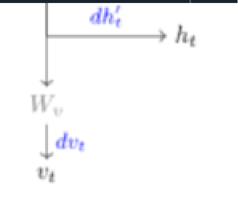
$$\frac{d\tanh(x)}{dx} = 1 - \tanh^2(x)$$

```python
def sigmoid(x):
    return 1 / (1 + np.exp(-x))

def dsigmoid(y):
    return y * (1 - y)

def tanh(x):
    return np.tanh(x)

def dtanh(y):
    return 1 - y * y
```

LSTM
-set hypermaraters

```
import numpy as np
import matplotlib.pyplot as plt
import signal

data = open('input.txt', 'r').read()
chars = list(set(data))   # set >> 중복 허용 x 순서 x  list 생성
data_size= len(data)      # data 길이
X_size = len(chars)       # X_size = list의 길이 ( 중복 x 순서 x)

print("data has %d characters, %d unique" % (data_size, X_size))

char_to_idx = {ch:i for i,ch in enumerate(chars)}  # key , value dic 생성
idx_to_char = {i:ch for i,ch in enumerate(chars)}  # key , value dic 생성

hidden_size = 100     # 출력 사이즈 100
Timesteps = 25        # 시퀀스 25개 , label 글자 25개
learning_rate = 0.01
weight_sd = 0.1 # Standard deviation of weights for initialization
z_size = hidden_size + X_size # Size of concatenate(H, X) vector
```

```python
class Param :
    def __init__(self, name, value) :
        self.name = name
        self.v = value # parameter value
        self.d = np.zeros_like(value) # derivative
        self.m = np.zeros_like(value) # momentum for Adagrad

class Parameters :
    def __init__(self) :  # randn 가우시안 표준 정규 분포를 따르는 난수 생성
                          # 초기 weight & bias 추가된 상태
        self.W_f = Param('W_f',    # forget gate Weight
                         np.random.randn(hidden_size, z_size) * weight_sd + 0.5)
        self.b_f = Param('b_f',    # forget gate bias
                         np.zeros((hidden_size, 1)))
        self.W_i = Param('W_i',    # input gate weight
                         np.random.randn(hidden_size, z_size) * weight_sd + 0.5)
        self.b_i = Param('b_i',    # input gate bias
                         np.zeros((hidden_size, 1)))
        self.W_C = Param('W_C',    # Cell weight
                         np.random.randn(hidden_size, z_size) * weight_sd)
        self.b_C = Param('b_C',    # Cell weight
                         np.zeros((hidden_size, 1)))
        self.W_o = Param('W_o',    # output gate weight
                         np.random.randn(hidden_size, z_size) * weight_sd + 0.5)
        self.b_o = Param('b_o',    # output gate bias
                         np.zeros((hidden_size, 1)))

        #For final layer to predict the next character

        # V = logits = (W_v * h_t) + b_v
        self.W_v = Param('W_v',
                         np.random.randn(X_size, hidden_size) * weight_sd)
        self.b_v = Param('b_v',
                         np.zeros((X_size, 1)))

    def all(self):
        return [self.W_f, self.W_i, self.W_C, self.W_o, self.W_v,
               self.b_f, self.b_i, self.b_C, self.b_o, self.b_v]

paramaters = Parameters()
```



LSTM
- set parameters & notation of each gates in code

Concatenation of h_{t-1} and x_t

$$z = [h_{t-1}, x_t]$$

LSTM functions

$$f_t = \sigma(W_f \cdot z + b_f)$$
$$i_t = \sigma(W_i \cdot z + b_i)$$
$$\bar{C}_t = tanh(W_C \cdot z + b_C)$$
$$C_t = f_t * C_{t-1} + i_t * \bar{C}_t$$
$$o_t = \sigma(W_o \cdot z + b_t)$$
$$h_t = o_t * tanh(C_t)$$

Logits

$$v_t = W_v \cdot h_t + b_v$$

Softmax

$$\hat{y}_t = \text{softmax}(v_t)$$

Forward propagation

```
def forward (x, h_prev, C_prev, p = Parameters) :
    assert x.shape == (X_size,1)  # 조건
    assert h_prev.shape == (hidden_size,1) # 조건 H_t-1
    assert C_prev.shape == (hidden_size,1) # 조건 H_t-1

    z = np.row_stack((h_prev, x)) # 첫 번째 축 따라 배열 쌓기 >>> h_prev 위에 x 쌓기
    f = sigmoid(np.dot(p.W_f.v, z) +  p.b_f.v)  # forget gate
    i = sigmoid(np.dot(p.W_i.v, z) + p.b_i.v)   # input gate
    C_bar = tanh(np.dot(p.W_C.v, z + p.b_C.v)   # C_prev

    C = f * C_prev + i * C_bar
    o = sigmoid(np.dot(p.W_o.v, z + p.b_o.v))
    h = o * tanh(C)

    v = np.dot((p.W_v.v, h) + p.b_v.v)  # logits
    y = np.exp(v) / np.sum(np.exp(v))   # Softmax    y^

    return z, f, i, C_bar, C, o ,h ,v , y
```

Concatenation of h_{t-1} and x_t

$$z = [h_{t-1}, x_t]$$

LSTM functions

$$f_t = \sigma(W_f \cdot z + b_f)$$
$$i_t = \sigma(W_i \cdot z + b_i)$$
$$\bar{C}_t = tanh(W_C \cdot z + b_C)$$
$$C_t = f_t * C_{t-1} + i_t * \bar{C}_t$$
$$o_t = \sigma(W_o \cdot z + b_t)$$
$$h_t = o_t * tanh(C_t)$$

Logits

$$v_t = W_v \cdot h_t + b_v$$

Softmax

$$\hat{y}_t = \text{softmax}(v_t)$$

LSTM BPTT with derivative parameters

Model parameter gradients

$$dW_v = dv_t \cdot h_t^T$$
$$db_v = dv_t$$

$$dW_f = df'_t \cdot z^T$$
$$db_f = df'_t$$

$$dW_i = di'_t \cdot z^T$$
$$db_i = di'_t$$

$$dW_C = d\bar{C}'_t \cdot z^T$$
$$db_C = d\bar{C}'_t$$

$$dW_o = do'_t \cdot z^T$$
$$db_o = do'_t$$

Loss

$$L_k = -\sum_{t=k}^{T} \sum_j y_{t,j} log\hat{y_{t,j}}$$
$$L = L_1$$

Gradients

$$dv_t = \hat{y_t} - y_t$$
$$dh_t = dh'_t + W_y^T \cdot dv_t$$
$$do_t = dh_t * \tanh(C_t)$$
$$dC_t = dC'_t + dh_t * o_t * (1 - \tanh^2(C_t))$$
$$d\bar{C}_t = dC_t * i_t$$
$$di_t = dC_t * \bar{C}_t$$
$$df_t = dC_t * C_{t-1}$$

$$df'_t = f_t * (1 - f_t) * df_t$$
$$di'_t = i_t * (1 - i_t) * di_t$$
$$d\bar{C}'_{t-1} = (1 - \bar{C}_t^2) * d\bar{C}_t$$
$$do'_t = o_t * (1 - o_t) * do_t$$
$$dz_t = W_f^T \cdot df'_t + W_i^T \cdot di_t + W_C^T \cdot d\bar{C}_t + W_o^T \cdot do_t$$

$$[dh'_{t-1}, dx_t] = dz_t$$
$$dC'_t = f_t * dC_t$$

```python
def backward ( target, dh_next, dC_next, C_prev,
               z, f, i, C_bar, C, o, h, v, y,
               p = parameters) :
                 assert z.shape == (X_size + hiddent_size, 1)
                 assert v.shape == (X_size, 1)
                 assert y.shape == (X_size, 1)
                 for param in pdh_next, dC_next, C_prev, f, i, c_bar, C, o, h] :
                      assert param.shape == (hidden_size, 1)
                 dv = np.copy(y)
                 dv[target] -= 1

                 p.W_v.d += np.dot(dv, h.T)
                 p.b_v.d += dv

                 dh = np.dot(p.W_v.v.T, dv)
                 dh += dh_next
                 do = dh * tanh(C)
                 do dsigmoid(O) * do

                 p.W_o.d  += np.dot(do, z.T)
                 p.b_o.d += do

                 dC = np.copy(dC_next)
                 dC += dh * o * dtanh(tanh(C))
                 dC_bar = dC * i
                 dC_bar = dtanh(C_bar) * dC_bar
                 p.W_C.d += np.dot(dC_bar, z.T)
                 p.b_C.d += dC_bar

                 di = dC * C_bar
                 di = dsigmoid(i) * di
                 p.W_i.d += np.dot(di, z.T)
                 p.b_i.d += di

                 df = dC * C_prev
                 df = dsigmoid(f) * df
                 p.W_f.d += np.dot(df, z.T)
                 p.b_f.d += df

                 dz = (np.dot(p.W_f.v.T, df) + np.dot(p.W_i.v.T, di) + np.dot(p.W_C.v.T, dC_bar) + np.dot(p.W_o.v.T, do))
                 dh_prev = dz[:H_size, :]
                 dC_prev = f * dC

                 return dh_prev, dC_prev
```

- `target` is target character index y_t
- `dh_next` is dh'_t (size H x 1)
- `dC_next` is dC'_t (size H x 1)
- `C_prev` is C_{t-1} (size H x 1)
- df'_t, di'_t, $d\bar{C}'_t$, and do'_t are *also* assigned to `df`, `di`, `dC_bar`, and `do` in the **code**.
- *Returns* dh_t and dC_t

$$
\begin{aligned}
dv_t &= \hat{y}_t - y_t \\
dh_t &= dh'_t + W_y^T \cdot dv_t \\
do_t &= dh_t * \tanh(C_t) \\
dC_t &= dC'_t + dh_t * o_t * (1 - \tanh^2(C_t)) \\
d\bar{C}_t &= dC_t * i_t \\
di_t &= dC_t * \bar{C}_t \\
df_t &= dC_t * C_{t-1} \\
\\
df'_t &= f_t * (1 - f_t) * df_t \\
di'_t &= i_t * (1 - i_t) * di_t \\
d\bar{C}'_{t-1} &= (1 - \bar{C}_t^2) * d\bar{C}_t \\
do'_t &= o_t * (1 - o_t) * do_t \\
dz_t &= W_f^T \cdot df'_t \\
&\quad + W_i^T \cdot di_t \\
&\quad + W_C^T \cdot d\bar{C}_t \\
&\quad + W_o^T \cdot do_t
\end{aligned}
$$

$$
\begin{aligned}
dW_v &= dv_t \cdot h_t^T \\
db_v &= dv_t \\
\\
dW_f &= df'_t \cdot z^T \\
db_f &= df'_t \\
\\
dW_i &= di'_t \cdot z^T \\
db_i &= di'_t \\
\\
dW_C &= d\bar{C}'_t \cdot z^T \\
db_C &= d\bar{C}'_t \\
\\
dW_o &= do'_t \cdot z^T \\
db_o &= do'_t
\end{aligned}
$$

$$
\begin{aligned}
[dh'_{t-1}, dx_t] &= dz_t \\
dC'_t &= f_t * dC_t
\end{aligned}
$$

Gradient handling

```
############################################################ ###################################################

# forward backward pass

# Clear gradients before each backwards pss

def clear_gradients(params = parameters) :  # params = Wf , Wi, Wo, Bf, Bi , Bo, ...
    for p in params.all() :
        p.d.fill(0)   #  p.d > 미분 배열 공간 0으로 초기화

# exploding gradients를 완화시키기 위해 gradients 묶기

def clip_gradients( params = parameters ) : # params = Wf , Wi, Wo, Bf, Bi , Bo, ...
    for p inparams.all():
        np.clip(p.d, -1, 1, out = p.d) # np.clip(배열,최소값, 최대값, 배열 리턴) 을 사용해서 p.d 배열에 결과값 반환
         # -1,1 사이의 값만 남기고 나머지 0으로 변경 후 p.d에 저장

# forward pass 에서 value 값들을 계산하고 축적
# bptt에서 gradients 를 축적하고 exploding gradients를 막기 위해 묶음
```

Forward - Backward propagation

```
def forward_backward (inputs, targets, h_prev, C_prev):
    # inputs, target are list of integers with character indexes.
    # inputs, target 은 글자 인덱스들로 이루어진 정수 리스트
    # h_prev is the array of initial h at h-1 (size H x 1)
    # C_prev is the array of initial C at C-1 (size H x 1)
    global parameters # 전역변수

    # To store the values for each time step  # 딕셔너리 형태로 저장 > key & value
    x_s, z_s, f_s, i_s, = {}, {}, {}, {}
    C_bar_s, C_s, o_s, h_s = {}, {}, {}, {}
    v_s, y_s = {}, {}

    h_s[-1] = np.copy(h_prev)  # 1개 이전의 timestep value
    C_s[-1] = np.copy(C_prev)  # 1개 이전의 timestep value

    loss = 0
    # timestep에 따른 loop

    # input is list of integers with character indexs

    assert len(inputs) == T_steps # 조건 inputs의 길이 == timestep
    for t in range(len(inputs)):
        x_s[t] = np.zeros((X_size,1))
        x_s[t][inputs[t]] = 1      # input character
        # >>>>  X_S 배열 모두 0으로 초기화 후 해당 index에 해당하는 포인트만 1로 One-hot encoding

        #forward pass
        (z_s[t], f_s[t], i_s[t], C_bar_s[t], C_s[t], o_s[t], h_s[t], v_s[t], y_s[t]
         = forward(x_s[t], h_s[t-1], C_s[t-1]))
        # def forward (x, h_prev, C_prev, p = Parameters)
        # forward >>>> return z, f, i, C_bar, C, o ,h ,v , y

        loss += -np.log(   y_s[t] [targets[t], 0] ) # Loss for at t

    clear_gradients()   # gradient 초기화

    return loss, h_s[len(inputs) -1] , C_s(len(inputs) - 1)  # loss 값, 현재 h, C value 직전 value 반환
```

Sampling

```python
# 다음 글자를 샘플링

def sample(h_prev, C_prev, first_char_idx, sentence_length):
    x = np.zeros((X_size, 1))
    x[first_char_idx] = 1     # one - hot

    h = h_prev    # h-1
    C = C_prev    # C-1

    indexes = []    # list 생성

    for t in range(sentence_length):
        _, _, _, _, C, _, h, _, p = forward(x, h, C)
        # return z, f, i, C_bar, C, o ,h ,v , y

        # p  = y 값
        idx = np.random.choice(range(X_size), p=p.ravel())  # X_size 만큼 골라내기, p 평평한 배열로 만들기
        # p 배열중에서 X_size 만큼 골라내기
        x = np.zeros((X_size, 1))  # x 1차원 배열으로 초기화
        x[idx] = 1   # index에 해당하는 key 값 1로 변환
        indexes.append(idx) # indexs list에 idx 글자 붙이기

    return indexes # 문자열 반환
```

Update_status and parameters

```
def update_status (inputs, h_prev, C_prev) :
    #initialized later
    global plot_iter, plot_loss
    global smooth_loss

    # Get predictrions for 200 letters with current model

    sample_idx = sample(h_prev, C_prev, inputs[0], 200)  # sample(h_prev, C_prev, first_char_idx, sentence_length)
    txt = ''.join(idx_to_char[idx] for idx in sample_idx)

    # idx_to_char = {i:ch for i,ch in enumerate(chars)}  # key , value dic 생성
    # 인덱스 > 글자로 바꾸어 이어붙이기

    # Clear and plot
    plt.plot(pplot_iter, plot_loss)
    display.clear_output(wait=True)
    plt.show()

    # Print prediction and loss
    print( "---- \n %s \n -----" % (txt,) )
    print("iter %d, loss %f" % iteration, smooth_loss)

    ##################################### training ##################################
    ####################################### update parameters ######################

def update_parameters(params = parameters) :
    for p in params.all():
        p.m += p.d*p.d # Calculate sum of gradients
                       # p.d ==   self.d = np.zeros_like(value) # derivative
        p.v += - (learning_rate * p.d /  np.sqrt (p.m + 1e-8))  # adagrad 매개 변수 갱신할 때 1/sqrt(h) 곱해 조정
        # self.v = value #parameter value

####################################################################################################
```

Calulate numerical gradient

```
from random import uniform

# Calculate numerical gradient

def calc_numerical_gradient (param, idx, delta, inputs, targets, h_prev, C_prev) :
    old_val = param.v.flat[idx]   # paramete  idx값에 해당하는 value 를 평평하게 만듬

    # evaluate loss at  [x + delta] and [x - delta]
    param.v.flat[idx] = old_val + delta
    loss_plus_delta, _, _ = forward_backward(nputs, targets,
                                             h_prev, C_prev)
    param.v.flat[idx] = old_val - delta
    loss_mins_delta, _, _ = forward_backward(inputs, targets,
                                             h_prev, C_prev)
    # forward_backward return value >>
    # return loss, h_s[len(inputs) -1] , C_s(len(inputs) - 1)  # loss 값, 현재 h, C value 직전 value 반환
    param.v.flat[idx] = old_val # reset된 것!

    grad_numerical = (loss_plus_delta - loss_mins_delta) / (2 * delta)
    # Clip numerical error because analytical gradient is clipped
    [grad_numerical] = np.clip([grad_numerical], -1, 1)
    # grad 수치를 -1~1 사이 값들을 제외하고 0으로 만든 후 리스트로 저장
    return grad_numerical # grad 수치 반환
```

Gradient check

```python
def gradient_check (num_checks, delta, inputs, target, h_prev, C_prev) :
    global parameters

    # To calculate computed gradients
    _, _, _ = forward_backward(inputs, targets, h_prev, C_prev)
    # loss 직전 h, s state 반환
    for param in parameters.all():
        # make a copy because this will get modified
        d_copy = np.copy(param.d) # deriavative shape of parameters

        # Test num_checks times
        for i in range(num_checks) :
            # Pick a random index
            rnd_idx = int(uniform(0, param.v.size))

            grad_numerical = calc_numerical_gradient(param, idx,  delta, inputs, targets, h_prev, C_prev)
            grad_numerical = d_copy.flat[rnd_idx]

            err_sim = abs(grad_numerical + grad_analytical) + 1e-09      # abs = 절대값
            rel_error = abs(grad_analytical - grad_numerical) / err_sum  # 상대오차

            # If relative error is greater than 1e-06
            if rel_error > 1e-06:
                print('%s (%e, %e' >= %e
                      %(param.name, grad_numerical, grad_analytical, rel_error))

gradient_check(10, 1e-5, inputs, targets, g_h_prev, g_C_prev)
```

References

인공지능 : 튜링 테스트에서 딥러닝까지 - 이건명

https://ratsgo.github.io/natural%20language%20processing/2017/03/09/rnnlstm/

https://blog.varunajayasiri.com/numpy_lstm.html

https://wegonnamakeit.tistory.com/7

http://colah.github.io/posts/2015-08-Understanding-LSTMs/

https://ikaros0909.tistory.com/1?category=676839

https://aikorea.org/blog/rnn-tutorial-1/

https://wordbe.tistory.com/entry/RNN-LSTM이란