LSTM - NETWORKS

→ Solve the vanishing gradient issue of RNN.
→ Thus better suited for Lon-term dependancies.

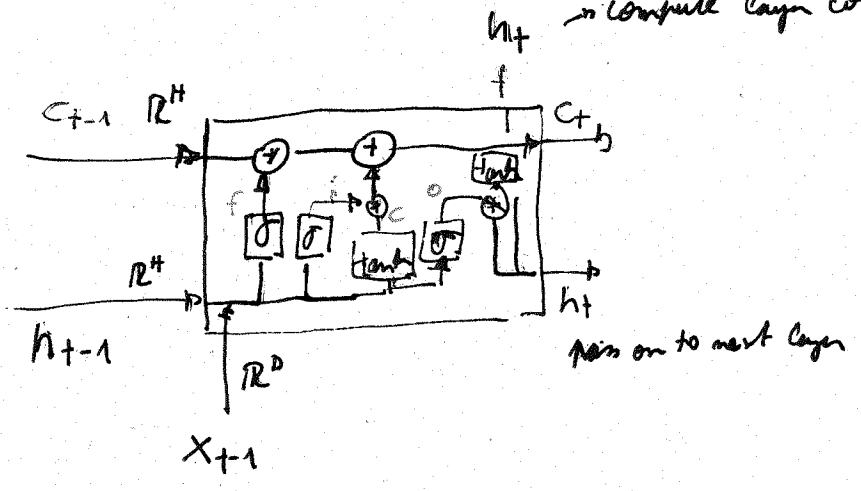
→ Weight matrices:

- W_f - forget gate
- W_i - input gate
- W_c - cell state
- W_o - output gate

This results in a input-to-hidden matrix and a hidden-to-hidden matrix.

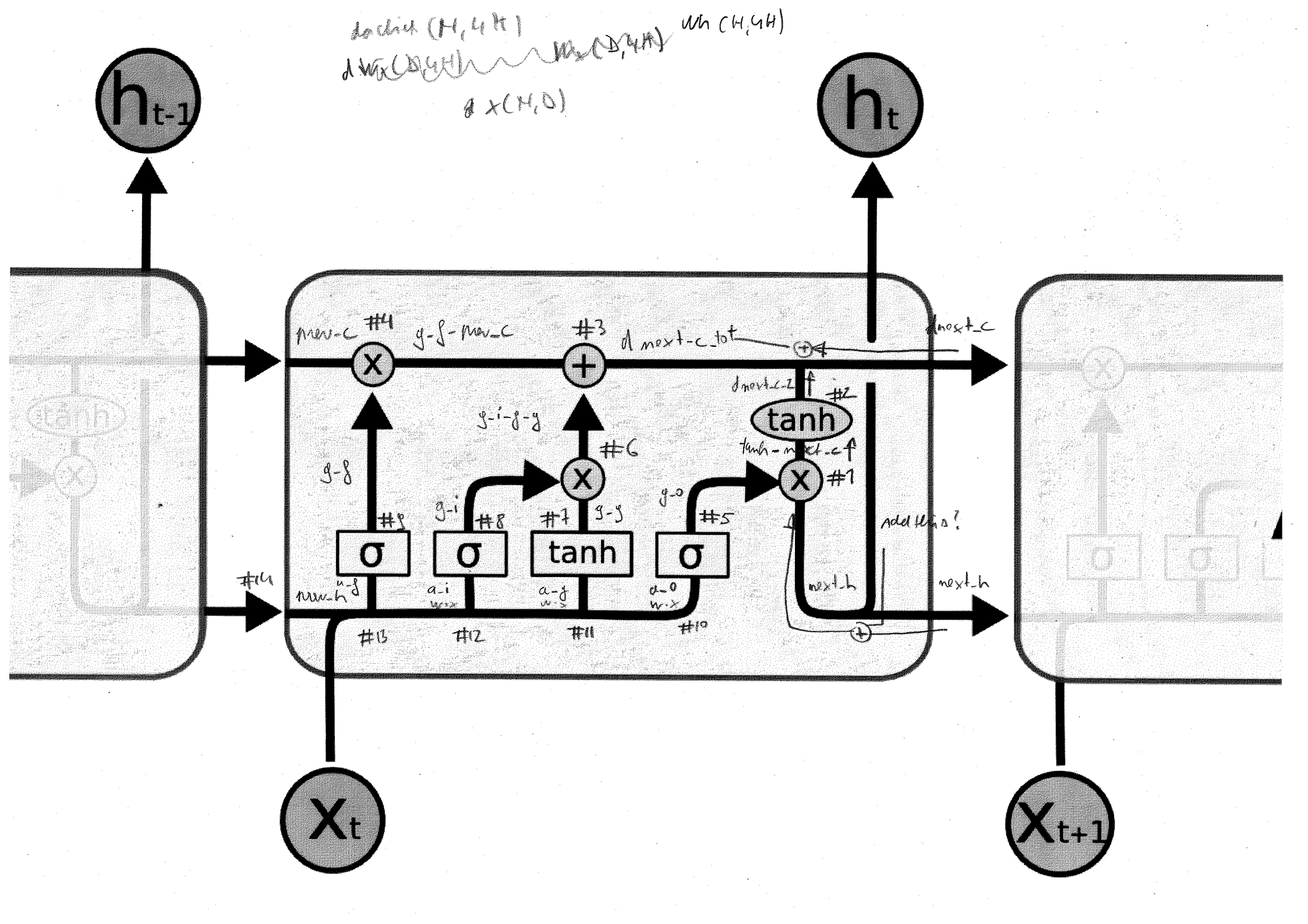
$W_x \in \mathbb{R}^{4H \times D}$
$W_h \in \mathbb{R}^{4H \times H}$

$$\underset{(4H,D)}{\begin{bmatrix} W_x \end{bmatrix}} \cdot \underset{(N,D)^{T}}{\begin{bmatrix} X_t \end{bmatrix}} \rightarrow \underset{(N,D)}{\begin{bmatrix} X_t \end{bmatrix}} \cdot \underset{(D,4H)}{\begin{bmatrix} W_x^T \end{bmatrix}} = \underset{(N,4H)}{\begin{bmatrix} A \end{bmatrix}}$$



LSTM Backwards

I get some loss at every layer and ~~from~~ a gradient a previous layer (through the upstream cell state loss and hidden loss). I need to compute the ~~loss~~ gradient with regards ~~dL/dx~~ to C_{t-1}, h_{t-1}, W_x, W_h. W_h is ~~dL/dx~~ used t-times and thus gradients add up. W_x is used only the first time thus it is computed only at the end.


h_{t-1}
h_t
X_t
X_{t+1}
tanh
σ
σ
tanh
σ
tanh
#1
#2
#3
#4
#5
#6
#7
#8
#9
#10
#11
#12
#13
#14
prev_c
g_f_prev_c
d next_c_tot
d next_c
d next_c_2
g_i_g_g
tanh_next_c
g_f
g_i
g_g
g_o
a_i
a_g
a_o
next_h
next_h