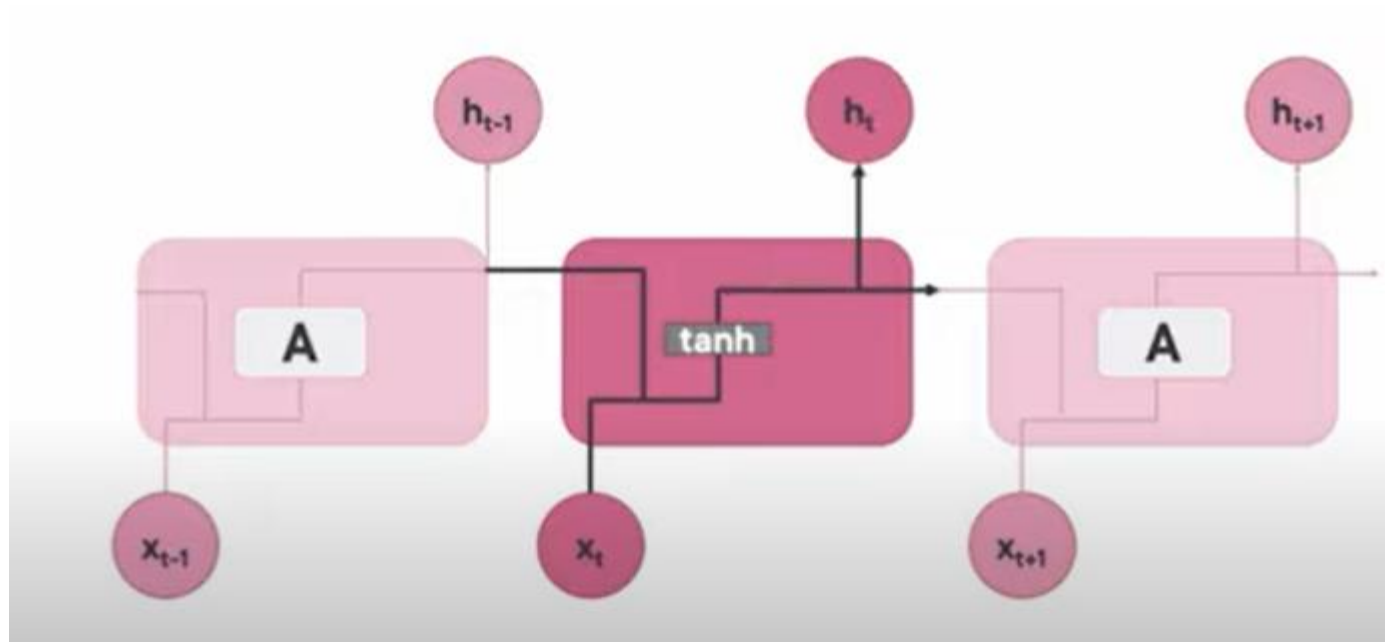


# LSTMs



# Long Short-Term Memory (LSTM)

- A type of RNN proposed by Hochreiter and Schmidhuber in 1997 as a solution to the vanishing gradients problem.
- On step  $t$ , there is a **hidden state**  $\mathbf{h}^{(t)}$  and a **cell state**  $\mathbf{c}^{(t)}$ 
  - Both are vectors length  $n$
  - The cell stores **long-term information**
  - The LSTM can **erase**, **write** and **read** information from the cell
- The selection of which information is erased/written/read is controlled by three corresponding **gates**
  - The gates are also vectors length  $n$
  - On each timestep, each element of the gates can be **open** (1), **closed** (0), or somewhere in-between.
  - The gates are **dynamic**: their value is computed based on the current context

# Long Short-Term Memory (LSTM)

We have a sequence of inputs  $\mathbf{x}^{(t)}$ , and we will compute a sequence of hidden states  $\mathbf{h}^{(t)}$  and cell states  $\mathbf{c}^{(t)}$ . On timestep  $t$ :

**Forget gate:** controls what is kept vs forgotten, from previous cell state

**Input gate:** controls what parts of the new cell content are written to cell

**Output gate:** controls what parts of cell are output to hidden state

**New cell content:** this is the new content to be written to the cell

**Cell state:** erase ("forget") some content from last cell state, and write ("input") some new cell content

**Hidden state:** read ("output") some content from the cell

**Sigmoid function:** all gate values are between 0 and 1

$$\mathbf{f}^{(t)} = \sigma \left( \mathbf{W}_f \mathbf{h}^{(t-1)} + \mathbf{U}_f \mathbf{x}^{(t)} + \mathbf{b}_f \right)$$

$$\mathbf{i}^{(t)} = \sigma \left( \mathbf{W}_i \mathbf{h}^{(t-1)} + \mathbf{U}_i \mathbf{x}^{(t)} + \mathbf{b}_i \right)$$

$$\mathbf{o}^{(t)} = \sigma \left( \mathbf{W}_o \mathbf{h}^{(t-1)} + \mathbf{U}_o \mathbf{x}^{(t)} + \mathbf{b}_o \right)$$

$$\tilde{\mathbf{c}}^{(t)} = \tanh \left( \mathbf{W}_c \mathbf{h}^{(t-1)} + \mathbf{U}_c \mathbf{x}^{(t)} + \mathbf{b}_c \right)$$

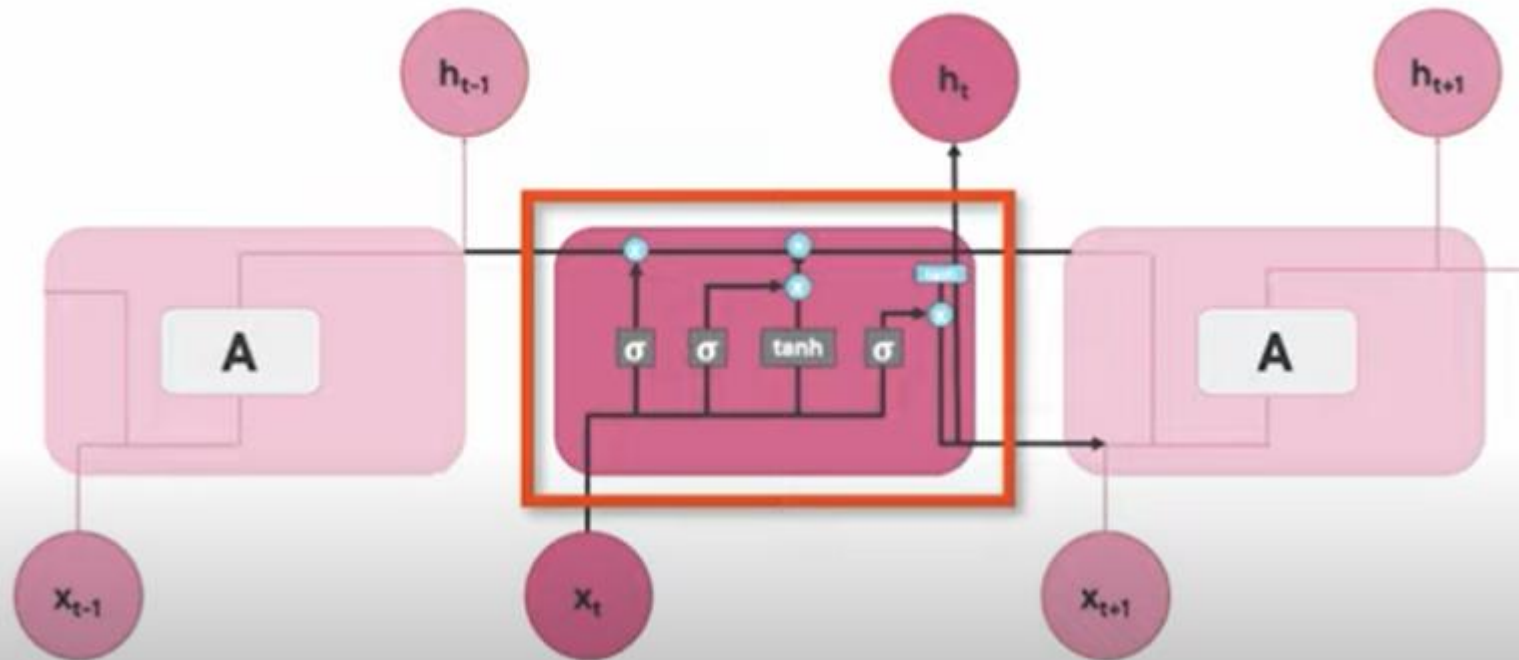
$$\mathbf{c}^{(t)} = \mathbf{f}^{(t)} \circ \mathbf{c}^{(t-1)} + \mathbf{i}^{(t)} \circ \tilde{\mathbf{c}}^{(t)}$$

$$\mathbf{h}^{(t)} = \mathbf{o}^{(t)} \circ \tanh \mathbf{c}^{(t)}$$

Gates are applied using element-wise product

All these are vectors of same length  $n$

# LSTMs



Cell state: can maintain info in memory for longer period

**The cell state**, often denoted as "C," is the primary means by which LSTMs capture and store information over long sequences. It acts as a memory storage unit within the LSTM cell.

- **Hidden state**: It is a summary or representation of the information that the LSTM has processed up to that point in the sequence.

- The hidden state is also influenced by the input data and the cell state but is primarily designed **to capture the short-term or immediate information** needed for the current prediction.

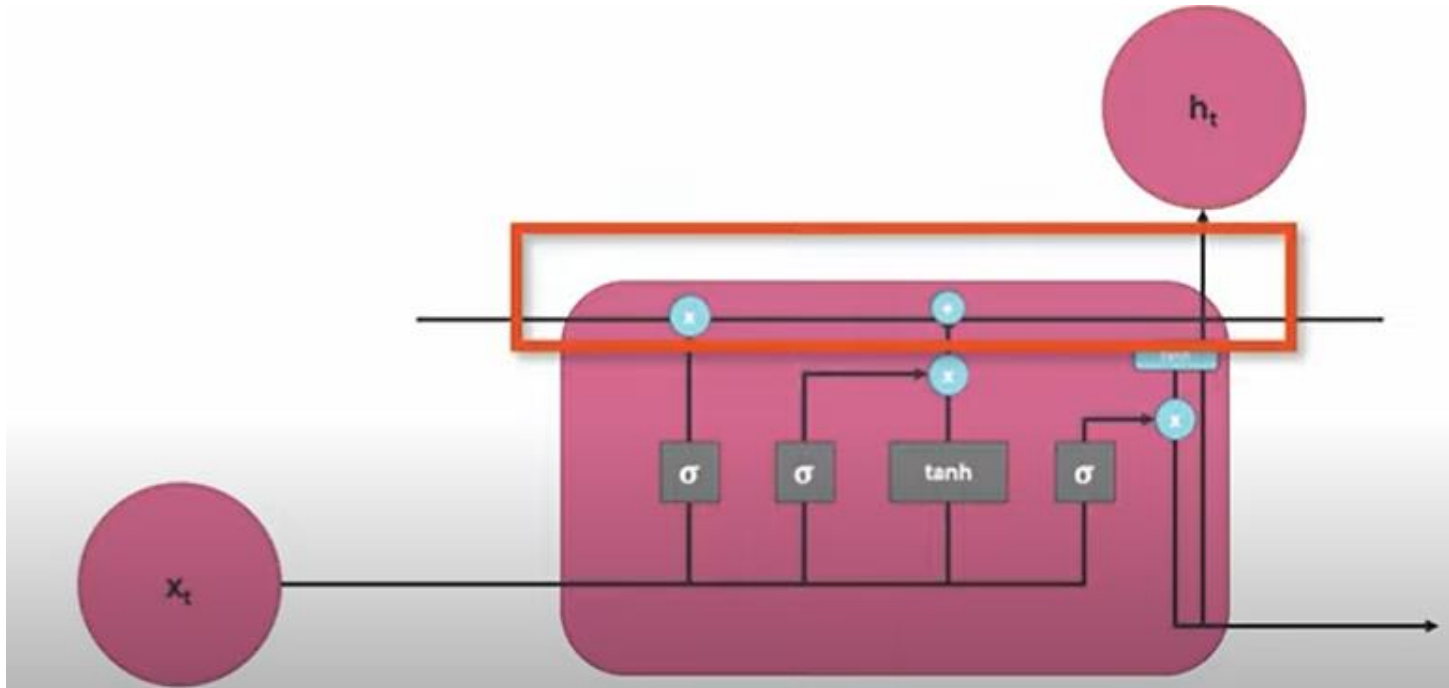
Cell State

Hidden State

Gates (Forget, Input, Output)



# LSTMs



The cell state, often denoted as "C," is the primary means by which LSTMs capture and store information over long sequences. It acts as a memory storage unit within the LSTM cell.

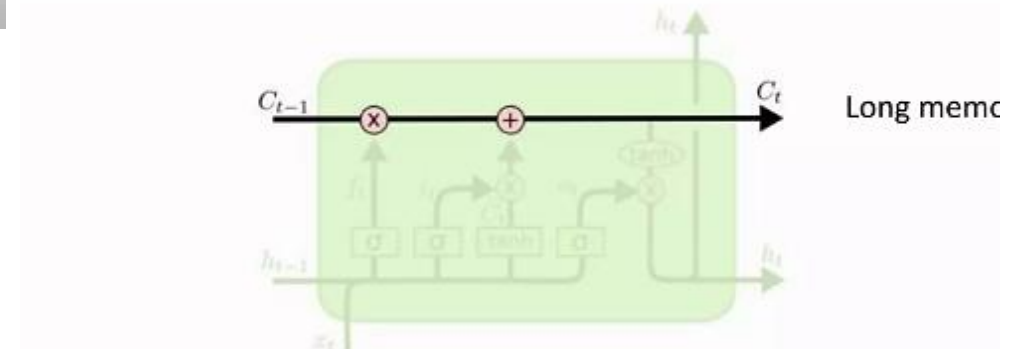
## Cell State

Transport highway to transfers information throughout the sequence chain

Information added or removed to the cell state via the gates

### Cell state:

- Like a conveyor belt
- It runs straight down the entire chain
- LSTM can remove or add information to the cell state



# LSTMs

- **Gates:**




- A way to optionally let information through
- They are composed out of:
  - A sigmoid neural net layer
  - A pointwise multiplication operation

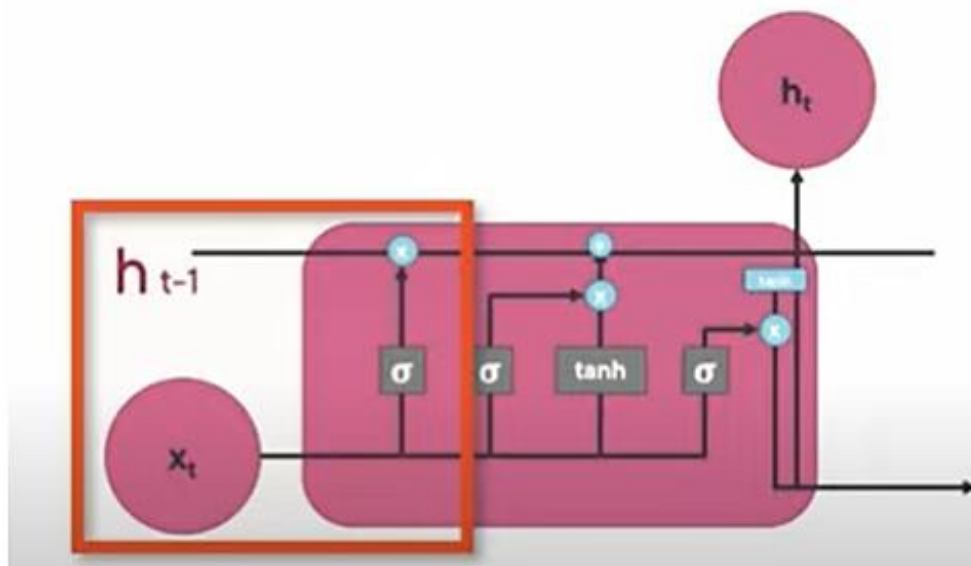
**Output** gate: Controls how much of the current cell state information should be passed to the next hidden state.

**Input:** Controls how much of the new information (candidate cell state) should be added to the current cell state.

**Forget:** Determines how much of the previous cell state should be "forgotten" (i.e., erased or retained).

- An LSTM has three of these gates, to protect and control the cell state:

- Forget gate layer  Keep gate
- Input gate layer  Write gate
- Output gate layer  Read gate

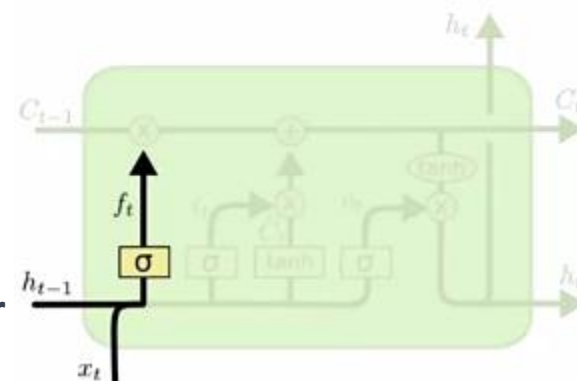


## Gates - Forget

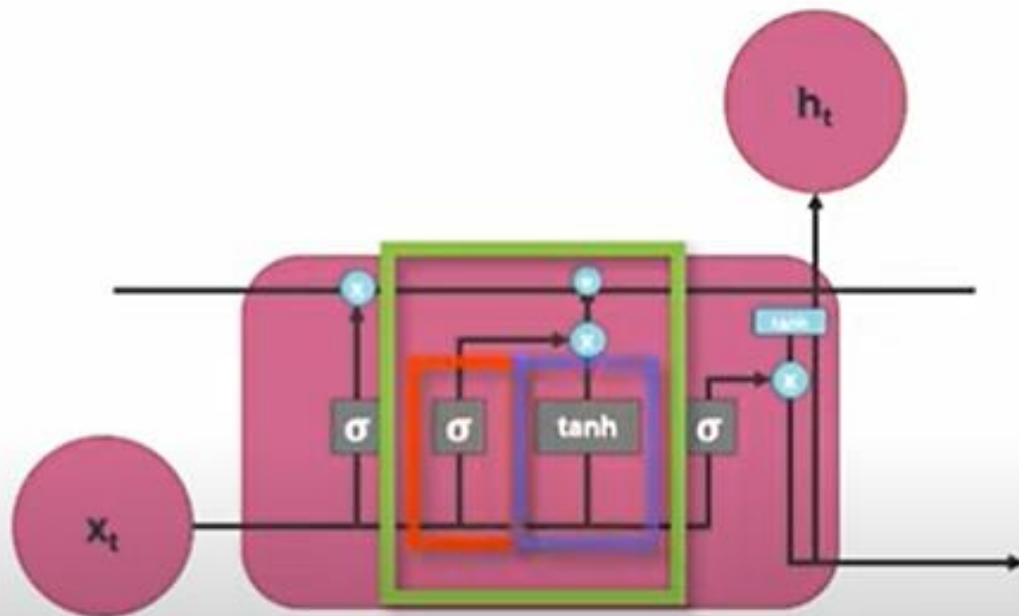
Decide what information to kept or thrown away

Values come out b/w 0 and 1. The closer to 0 means to forget, and closer to 1 means to keep

- **1** represents “**completely keep this**”
  - **0** represents “**completely get rid of this.**”
- The hidden state is the output of the LSTM cell at each time step. It is a summary or representation of the information that the LSTM has processed up to that point in the sequence.
  - The hidden state is also influenced by the input data and the cell state but is primarily designed **to capture the short-term or immediate information** needed for the current prediction.
- **Forget information:**
    - Decide what information throw away from the cell state
    - **Forget gate layer:**
      - Output a number between 0 and 1



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

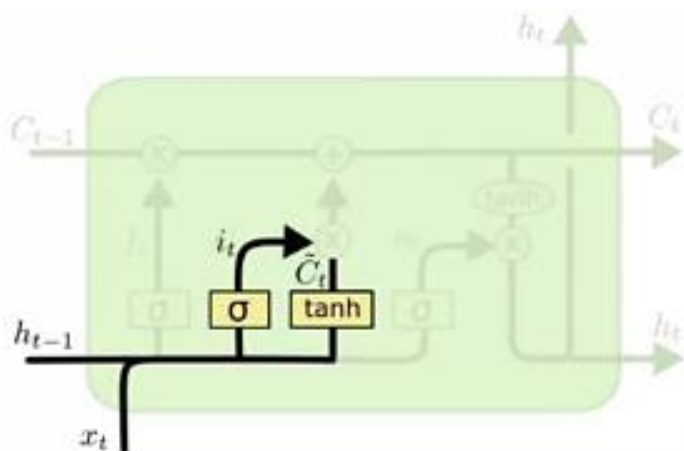


## Gates - Input

Has two parts:

- Sigmoid Layer (transform values b/w 0 & 1)
- Tanh Layer (squishing them between -1 and 1)

Sigmoid will decide which information to keep from tanh output



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

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

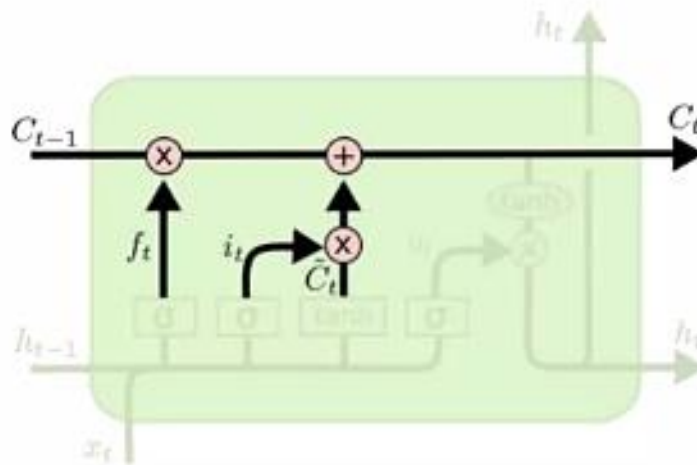
### • Add new information:

- Decide what new information store in the cell state
  - **Input gate layer:**
    - Decides which values we'll update
  - **Tanh layer:**
    - creates a vector of new candidate values,  $\tilde{C}_t$
- Candidate** values: new content to be written on the cell

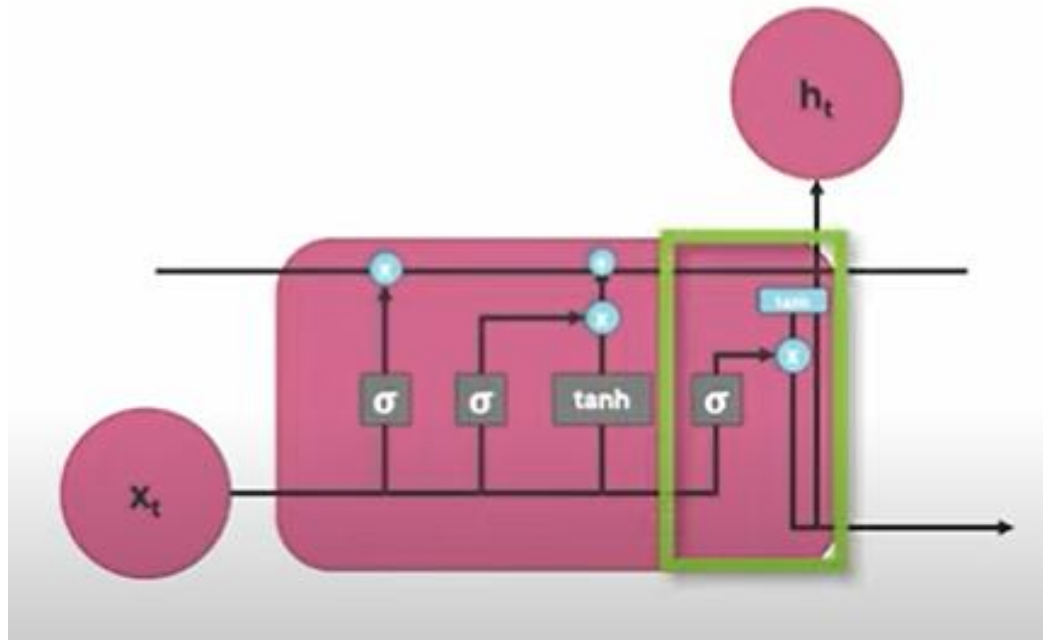


- **Update cell state:**

- Forgetting the things we decided to forget earlier:  $f_t * C_{t-1}$
- Adding information we decide to add:  $i_t * \tilde{C}_t$



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



## Gates - Output

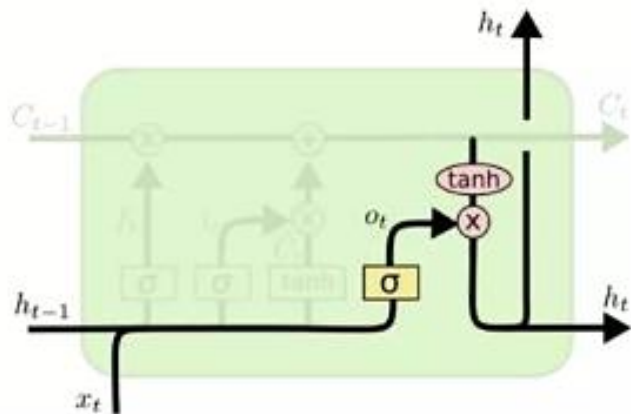
Decide what the next hidden state should be

First, pass information to Sigmoid

Then pass newly modified state to Tanh

Multiply outputs from Sigmoid and Tanh

New hidden state are then carried out to next timestep



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

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

Shadow state/ Short memory

### • Create output:

– Decide what we're going to output

– **Output gate layer:**

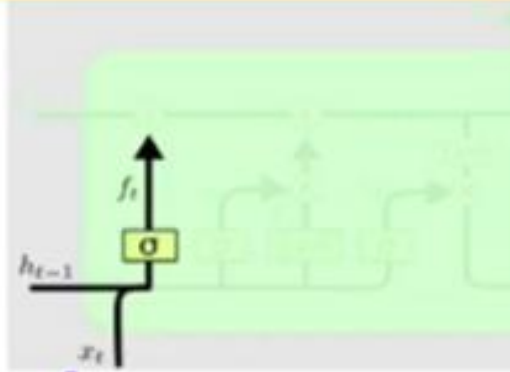
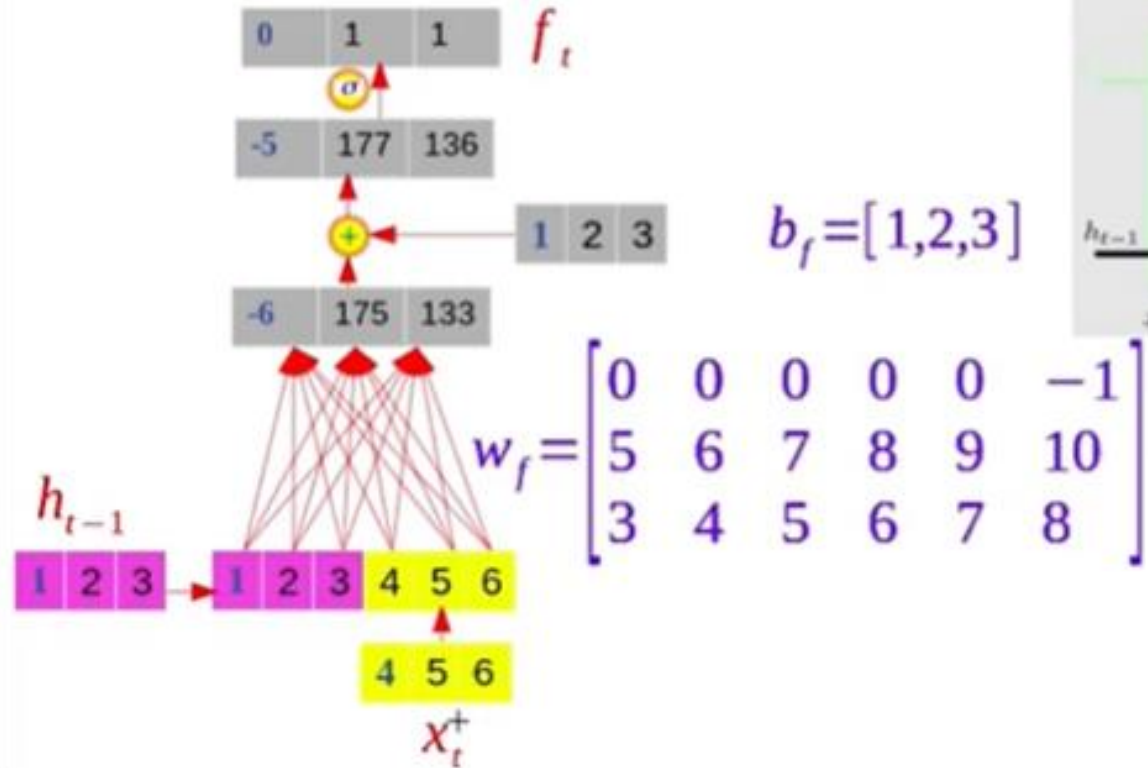
- Decides what parts of the cell state we're going to output

– **Tanh layer:**

- Push the values between -1 and +1

# Forget Gate

$C_{t-1}$   
5 5 5



- Suppose LSTM has 3 units then  $C_{t-1}$  and Hidden state will have a vector of size 3.
- Weight = units x seq length
- Here seq length is 6 ( $h_{t-1}, x_t$ )

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

$$w_f \cdot [h_{t-1}, x_t] = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & -1 \\ 5 & 6 & 7 & 8 & 9 & 10 \\ 3 & 4 & 5 & 6 & 7 & 8 \end{bmatrix} \cdot \begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \end{bmatrix} = \begin{bmatrix} -6 \\ 175 \\ 133 \end{bmatrix}$$

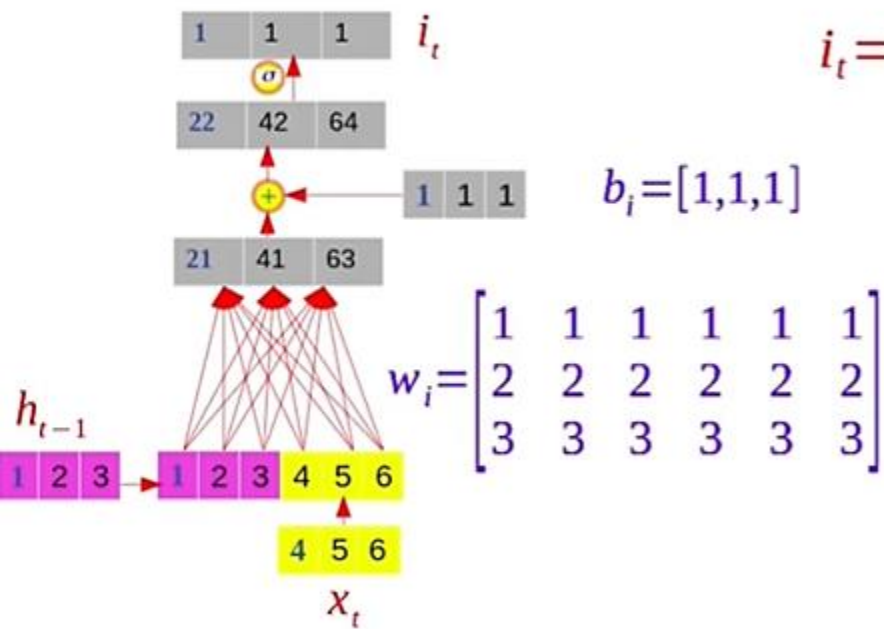
$C_{t-1}$   
5 5 5

0 5 5  $C_{t-1} * f_t$



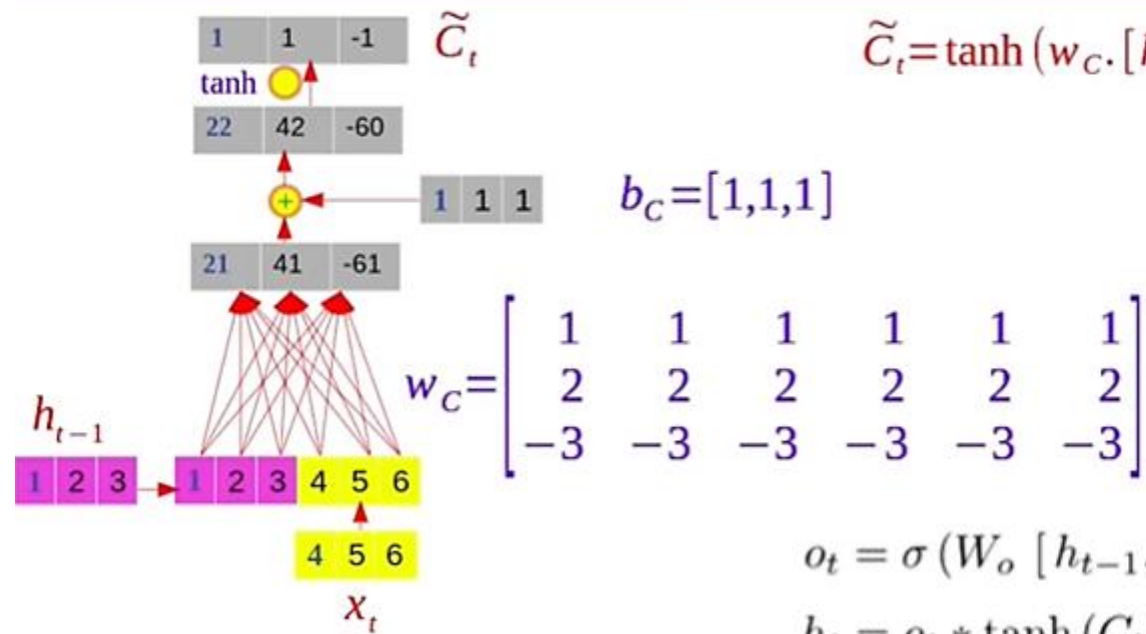
$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

## Input Gate – Sigmoid Layer



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

## Input Gate – Tanh Layer



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

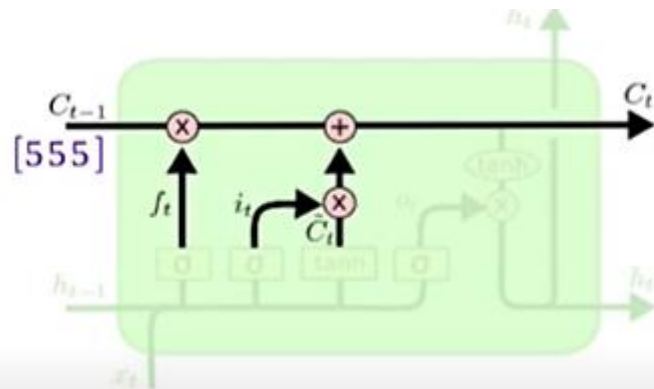
$$o_t = \sigma(W_o [h_{t-1}, x_t] + b_o)$$

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

result of  $\tanh(C_t)$  is  $[0.76, 0.9999, 0.9993]$

Suppose  $O_t = [0, 0.5, 1]$

$h_t = [(0*0.76), (0.5*0.99), (1*0.99)] = [0, 0.495, 0.99]$



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

$$C_t = \begin{bmatrix} 0 \\ 1 \\ 1 \end{bmatrix} * \begin{bmatrix} 5 \\ 5 \\ 5 \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} * \begin{bmatrix} 1 \\ 1 \\ -1 \end{bmatrix} = \begin{bmatrix} 1 \\ 6 \\ 4 \end{bmatrix}$$

## Gates:

$$o_t = \sigma(W_o h_{t-1} + U_o x_t + b_o)$$

$$i_t = \sigma(W_i h_{t-1} + U_i x_t + b_i)$$

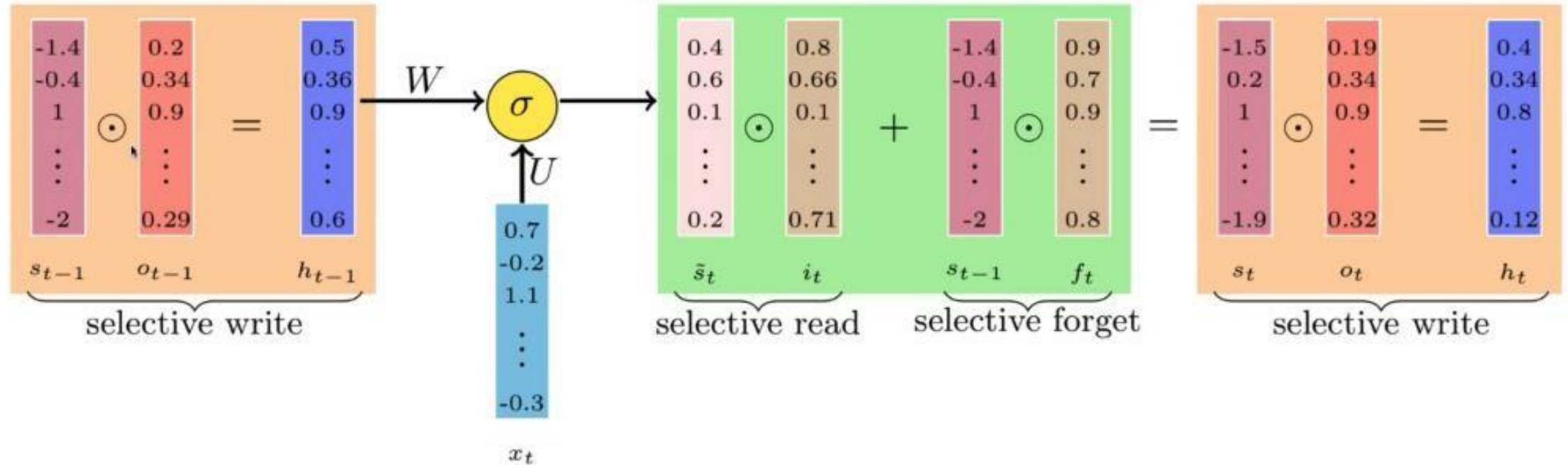
$$f_t = \sigma(W_f h_{t-1} + U_f x_t + b_f)$$

## States:

$$\tilde{s}_t = \sigma(W h_{t-1} + U x_t + b) \text{ Candidate state}$$

$$s_t = f_t \odot s_{t-1} + i_t \odot \tilde{s}_t \text{ Cell state}$$

$$h_t = o_t \odot \sigma(s_t)$$

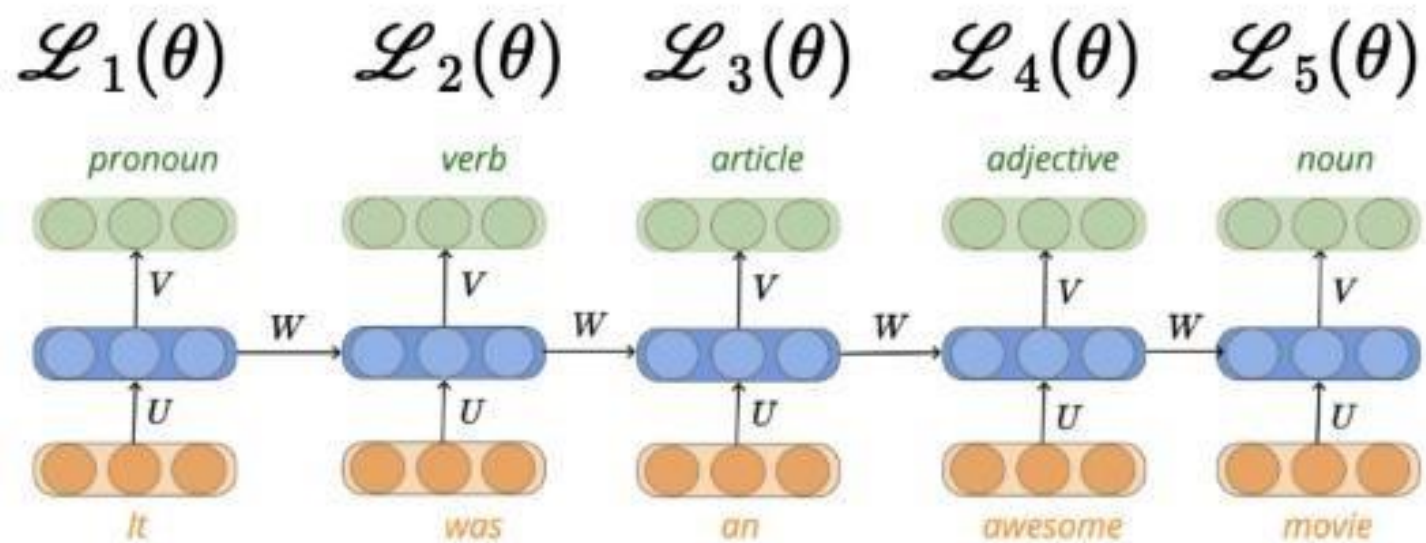
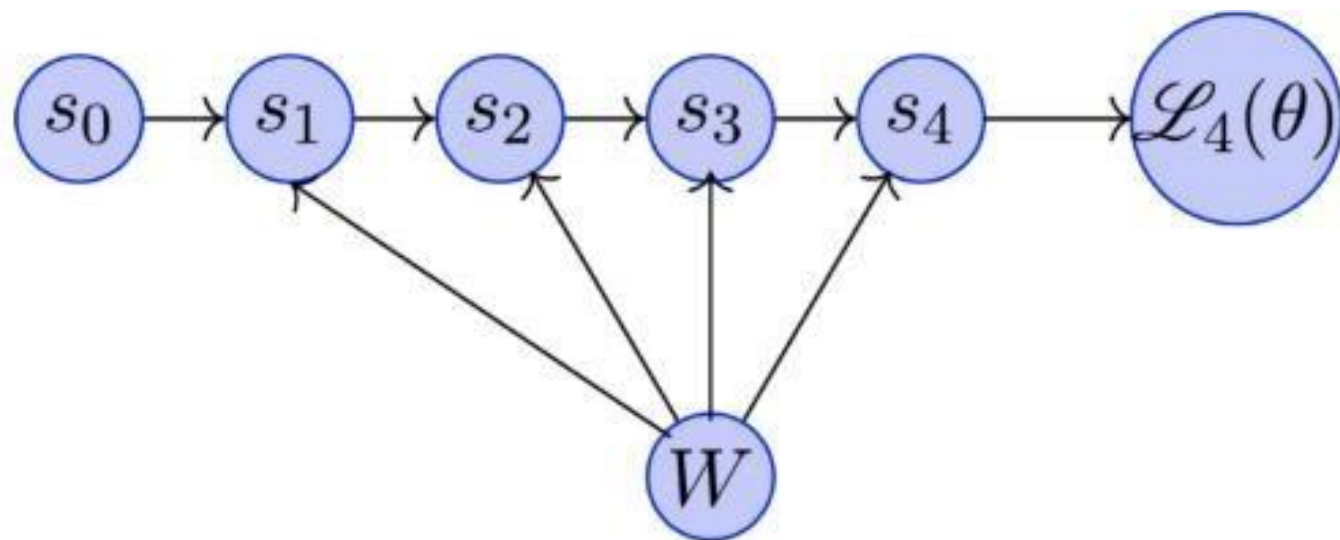






If the state at time  $t - 1$  did not contribute much to the state at time  $t$  (i.e., if  $\|f_t\| \rightarrow 0$  and  $\|o_{t-1}\| \rightarrow 0$ ) then during backpropagation the gradients flowing into  $s_{t-1}$  will vanish

When you don't want certain information to pass through, during backprop u will experience vanishing gradient but that is not an issues since u decided not to pass the information further.



✓ In general, the gradient of  $\mathcal{L}_t(\theta)$  w.r.t.  $\theta_i$  vanishes when the gradients flowing through **each and every path** from  $L_t(\theta)$  to  $\theta_i$  vanish.

✓ On the other hand, the gradient of  $\mathcal{L}_t(\theta)$  w.r.t.  $\theta_i$  explodes when the gradient flowing through **at least one path** explodes.

# How LSTM Solves Vanishing Gradient Problem

$$o_k = \sigma(W_o h_{k-1} + U_o x_k + b_o)$$

$$i_k = \sigma(W_i h_{k-1} + U_i x_k + b_i)$$

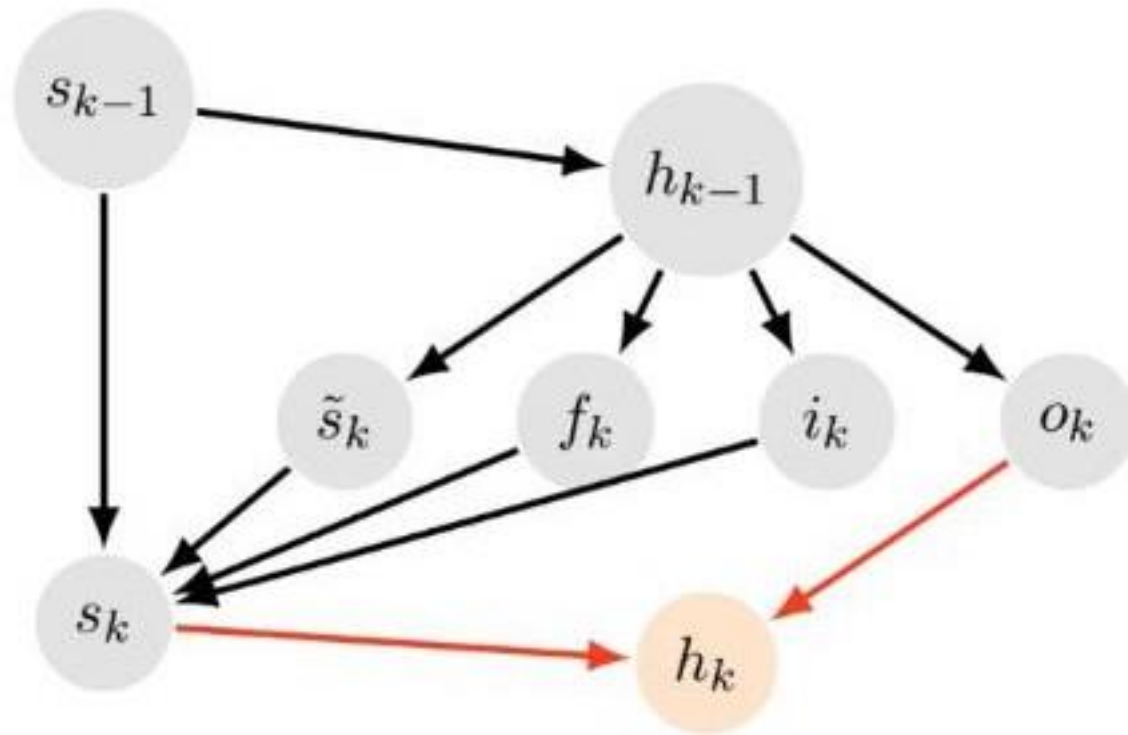
$$f_k = \sigma(W_f h_{k-1} + U_f x_k + b_f)$$

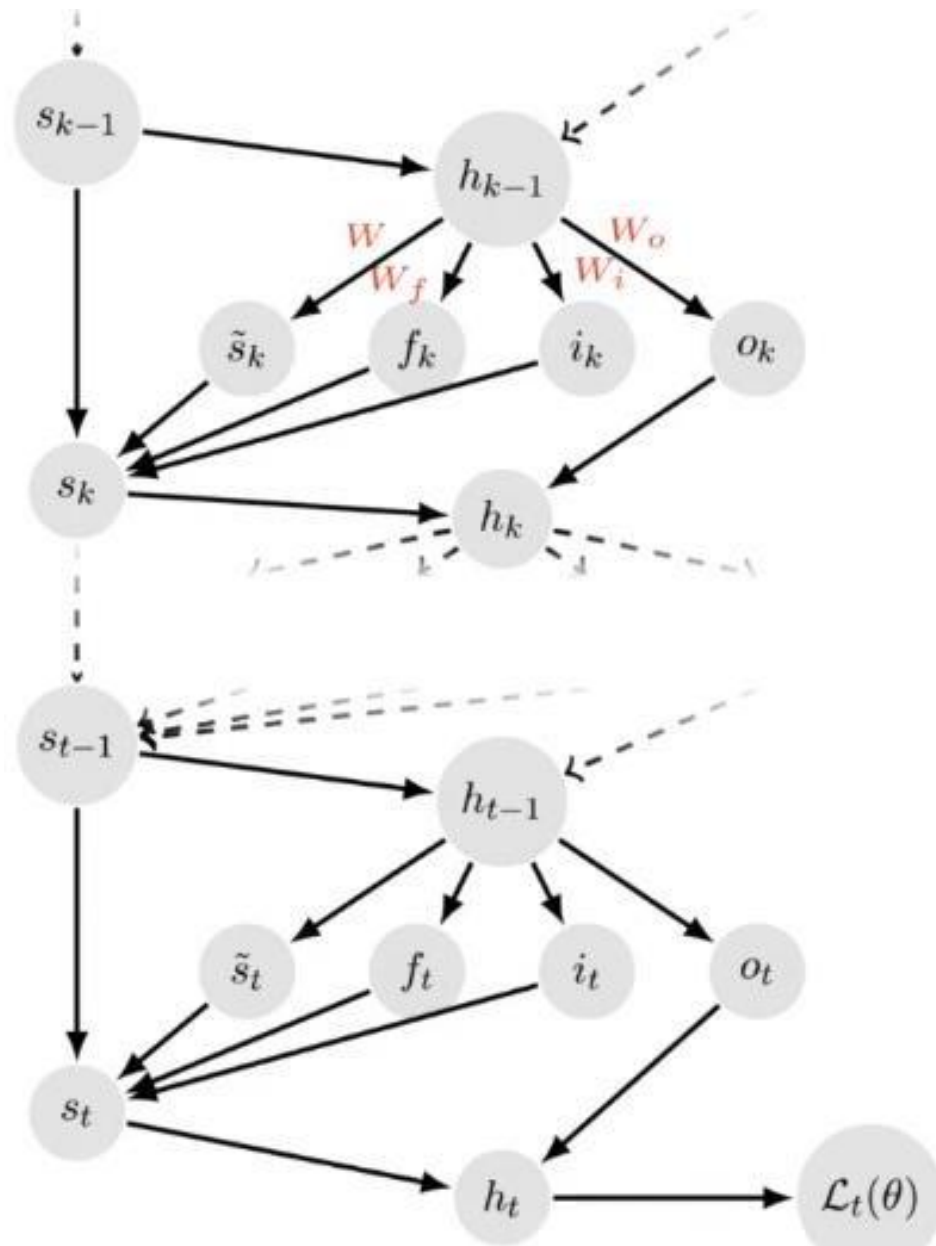
Candidate state  $\tilde{s}_k = \sigma(W h_{k-1} + U x_k + b)$

Cell state  $s_k = f_k \odot s_{k-1} + i_k \odot \tilde{s}_k$

$$h_k = o_k \odot \sigma(s_k)$$

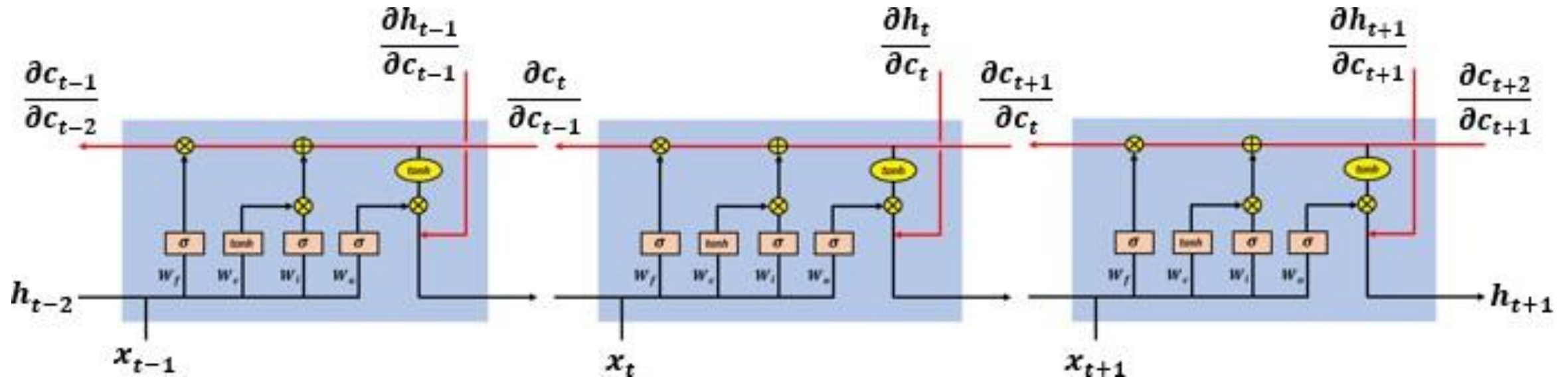






<https://medium.com/@prvnk10/how-lstms-solve-the-problem-of-vanishing-gradients-ea88f08c78ca>

## Back propagating through time for gradient computation



As in RNNs, the error term gradient is given by the following sum of T gradients:

$$\frac{\partial E}{\partial W} = \sum_{t=1}^T \frac{\partial E_t}{\partial W} \quad (3)$$

For the complete error gradient to vanish, all of these T sub gradients need to vanish. If we think of (3) as a series of functions, then by definition, this series converges to zero if the sequence of its partial sums tends to zero,

$$\sum_{t=1}^T \frac{\partial E_t}{\partial W} \rightarrow 0$$

so if the series of partial sums

$$(S_1, S_2, S_3, \dots)$$

where

$$S_n = \sum_{t=1}^n \frac{\partial E_t}{\partial W}$$

tends to zero

$$\frac{\partial E}{\partial W} = \sum_{t=1}^T \frac{\partial E_t}{\partial W} \quad (3)$$

**So if we want (3) not to vanish, our network needs to increase the likelihood that at least some of these sub gradients will not vanish**, in other words, make the series of sub gradients in (3) not converge to zero.



The gradient of the error for some time step  $k$  has the form:

$$\begin{aligned}\frac{\partial E_k}{\partial W} &= \frac{\partial E_k}{\partial h_k} \frac{\partial h_k}{\partial c_k} \dots \frac{\partial c_2}{\partial c_1} \frac{\partial c_1}{\partial W} \\ &= \frac{\partial E_k}{\partial h_k} \frac{\partial h_k}{\partial c_k} \left( \prod_{t=2}^k \frac{\partial c_t}{\partial c_{t-1}} \right) \frac{\partial c_1}{\partial W} \quad (4)\end{aligned}$$

the following product causes the gradients to vanish:

$$\prod_{t=2}^k \frac{\partial c_t}{\partial c_{t-1}}$$

$$c_t = c_{t-1} \otimes f_t \oplus \tilde{c}_t \otimes i_t \quad (5)$$

$$\begin{aligned} \frac{\partial c_t}{\partial c_{t-1}} &= \frac{\partial}{\partial c_{t-1}} [c_{t-1} \otimes f_t \oplus \tilde{c}_t \otimes i_t] \\ &= \frac{\partial}{\partial c_{t-1}} [c_{t-1} \otimes f_t] + \frac{\partial}{\partial c_{t-1}} [\tilde{c}_t \otimes i_t] \\ &= \frac{\partial f_t}{\partial c_{t-1}} \cdot c_{t-1} + \frac{\partial c_{t-1}}{\partial c_{t-1}} \cdot f_t + \frac{\partial i_t}{\partial c_{t-1}} \cdot \tilde{c}_t + \frac{\partial \tilde{c}_t}{\partial c_{t-1}} \cdot i_t \end{aligned}$$

$$\frac{\partial c_t}{\partial c_{t-1}} = \sigma'(W_f \cdot [h_{t-1}, x_t]) \cdot W_f \cdot o_{t-1} \otimes \tanh'(c_{t-1}) \cdot c_{t-1}$$

$$+ f_t$$

$$+ \sigma'(W_i \cdot [h_{t-1}, x_t]) \cdot W_i \cdot o_{t-1} \otimes \tanh'(c_{t-1}) \cdot \tilde{c}_t$$

$$+ \sigma'(W_c \cdot [h_{t-1}, x_t]) \cdot W_c \cdot o_{t-1} \otimes \tanh'(c_{t-1}) \cdot i_t$$

$$A_t = \sigma'(W_f \cdot [h_{t-1}, x_t]) \cdot W_f \cdot o_{t-1} \otimes \tanh'(c_{t-1}) \cdot c_{t-1}$$

$$B_t = f_t$$

$$C_t = \sigma'(W_i \cdot [h_{t-1}, x_t]) \cdot W_i \cdot o_{t-1} \otimes \tanh'(c_{t-1}) \cdot \tilde{c}_t$$

$$D_t = \sigma'(W_c \cdot [h_{t-1}, x_t]) \cdot W_c \cdot o_{t-1} \otimes \tanh'(c_{t-1}) \cdot i_t$$

$$\frac{\partial c_t}{\partial c_{t-1}} = A_t + B_t + C_t + D_t \quad (6)$$

$$\frac{\partial E_k}{\partial W} = \frac{\partial E_k}{\partial h_k} \frac{\partial h_k}{\partial c_k} \left( \prod_{t=2}^k [A_t + B_t + C_t + D_t] \right) \frac{\partial c_1}{\partial W}$$

## Preventing the error gradients from vanishing

Notice that the gradient contains the forget gate's vector of activations, which allows the network to better control the gradients values, at each time step, using suitable parameter updates of the forget gate.

$$\frac{\partial c_t}{\partial c_{t-1}} = \sigma'(W_f \cdot [h_{t-1}, x_t]) \cdot W_f \cdot o_{t-1} \otimes \tanh'(c_{t-1}) \cdot c_{t-1}$$

$$+ f_t$$

$$+ \sigma'(W_i \cdot [h_{t-1}, x_t]) \cdot W_i \cdot o_{t-1} \otimes \tanh'(c_{t-1}) \cdot \tilde{c}_t$$

$$+ \sigma'(W_c \cdot [h_{t-1}, x_t]) \cdot W_c \cdot o_{t-1} \otimes \tanh'(c_{t-1}) \cdot i_t$$



Say that for some time step  $k < T$ , we have that:

$$\sum_{t=1}^k \frac{\partial E_t}{\partial W} \rightarrow 0$$

Then for the gradient not to vanish, we can find a suitable parameter update of the forget gate at time step  $k+1$  such that:

$$\frac{\partial E_{k+1}}{\partial W} \nrightarrow 0$$

**It is the presence of the forget gate's vector of activations in the gradient term along with additive structure which allows the LSTM to find such a parameter update at any time step, and this yields:**

$$\sum_{t=1}^{k+1} \frac{\partial E_t}{\partial W} \nrightarrow 0$$

Another important property to notice is that the cell state gradient is an additive function

$$A_t \approx \overrightarrow{0.1}, B_t = f_t \approx \overrightarrow{0.7}, C_t \approx \overrightarrow{0.1}, D_t \approx \overrightarrow{0.1}$$

$$\begin{aligned}\prod_{t=2}^k [A_t + B_t + C_t + D_t] &\approx \prod_{t=2}^k [\overrightarrow{0.1} + \overrightarrow{0.7} + \overrightarrow{0.1} + \overrightarrow{0.1}] \\ &\approx \prod_{t=2}^k \overrightarrow{1} \nrightarrow 0\end{aligned}$$

$$\frac{\partial E}{\partial W} = \sum_{t=1}^T \frac{\partial E_t}{\partial W} \quad (3)$$

In **RNNs**, the sum in (3) is made from expressions with a similar behavior that are likely to all be in  $[0,1]$  which causes vanishing gradients.

In **LSTMs**, however, the presence of the forget gate, along with the additive property of the cell state gradients, enables the network to update the parameter in such a way that the different sub gradients in (3) do not necessarily agree

$$\sum_{t=1}^T \frac{\partial E_t}{\partial W} \nrightarrow 0$$

• The LSTM's cell state evolves as:

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

Here,  $f_t$  is the forget gate,  $i_t$  is the input gate, and  $\tilde{C}_t$  is the candidate cell state.

- The crucial difference is that the **cell state updates additively**, meaning that information can flow through many time steps **without being multiplied by small values** (like in RNNs). This additive nature helps preserve the gradient flow during backpropagation.

# Summary

- We have seen that RNNs suffer from vanishing gradients and caused by long series of multiplications of small values, diminishing the gradients and causing the learning process to become degenerate. In a analogous way, RNNs suffer from exploding gradients affected from large gradient values and hampering the learning process.
- LSTMs solve the problem using a unique additive gradient structure that includes direct access to the forget gate's activations, enabling the network to encourage desired behavior from the error gradient using frequent gates update on every time step of the learning process.

## Is vanishing/exploding gradient just a RNN problem?

- No! It can be a problem for all neural architectures (including feed-forward and convolutional), especially deep ones.
  - Due to chain rule / choice of nonlinearity function, gradient can become vanishingly small as it backpropagates
  - Thus lower layers are learnt very slowly (hard to train)
  - Solution: lots of new deep feedforward/convolutional architectures that add more direct connections (thus allowing the gradient to flow)
- Conclusion: Though vanishing/exploding gradients are a general problem, RNNs are particularly unstable due to the repeated multiplication by the same weight matrix [Bengio et al, 1994]