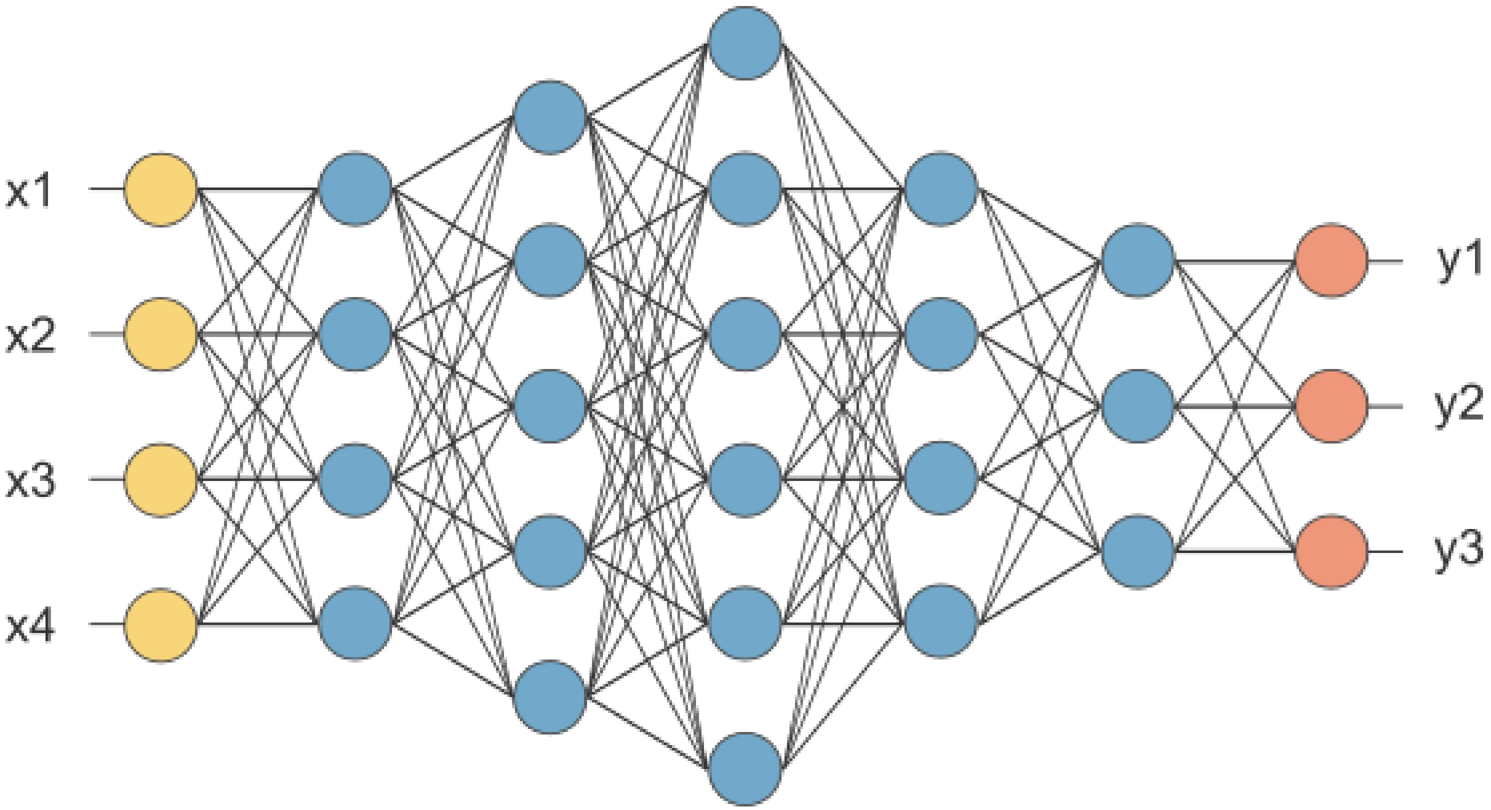
AI - FOUNDATION AND APPLICATION

Instructor:
Assoc. Prof. Dr. Truong Ngoc Son

Chapter 5
Recurrent Neural Network

Outline

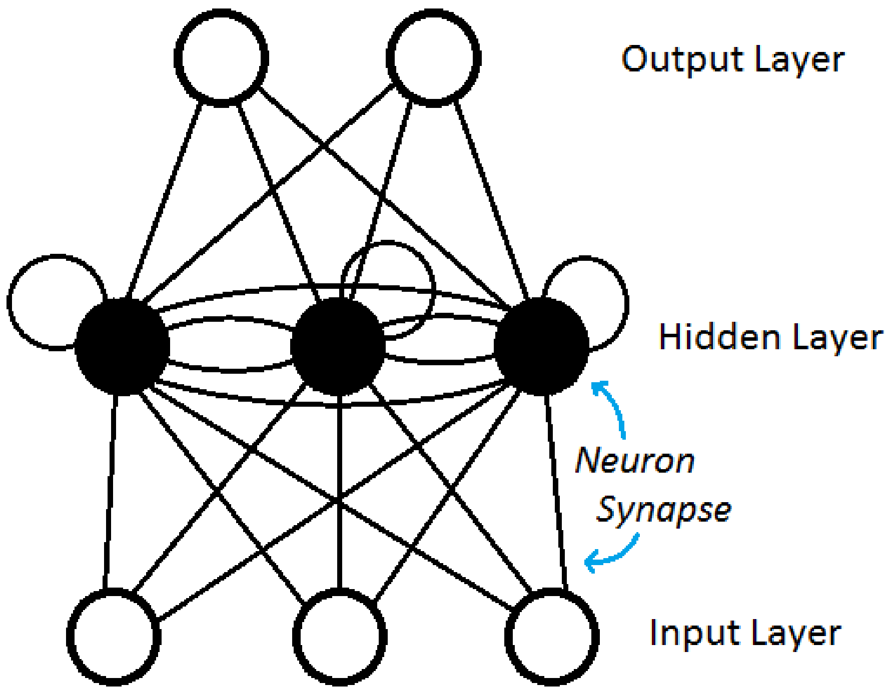
Feed Forward Neural Network



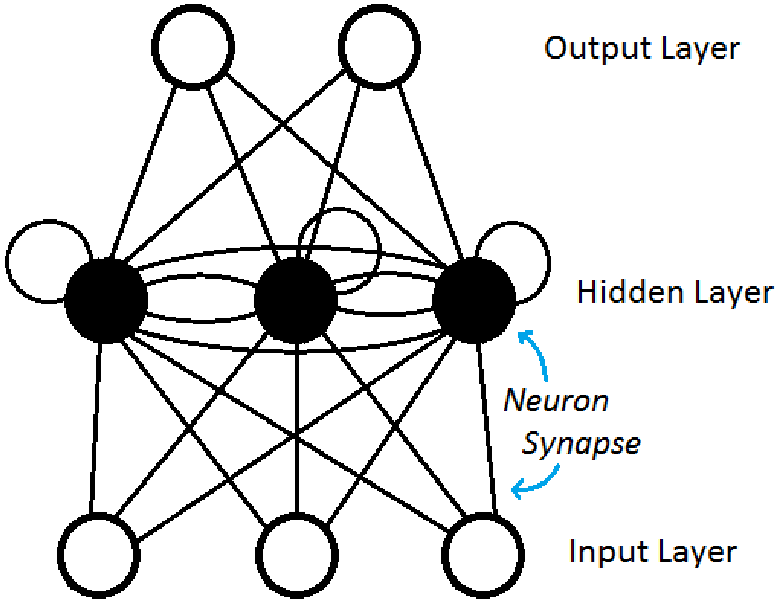
This is our fully connected network. If x_1 ... x_n, n is very large and growing, this network would become too large. We now will input one x_i at a time, and re-use the same edge weights.

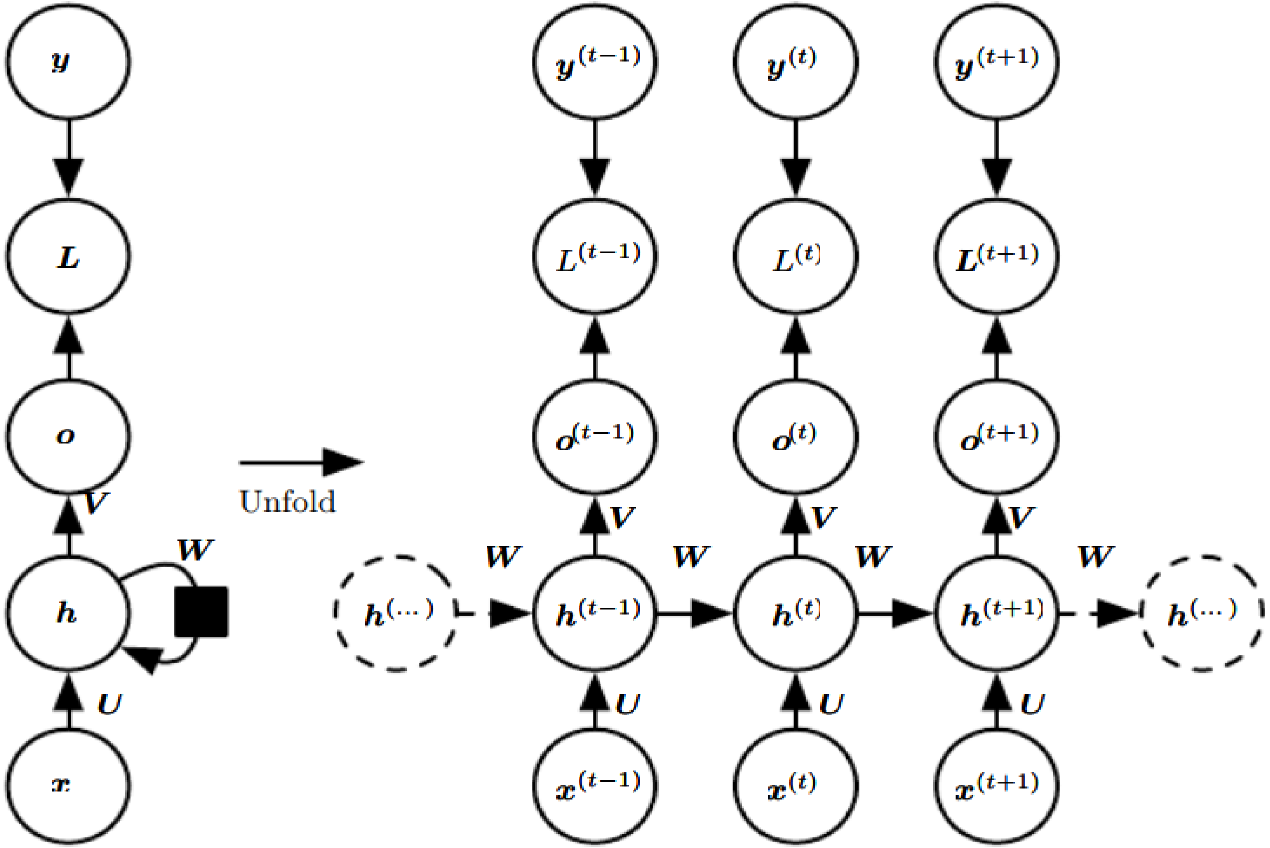
Sequence model – Recurrent Neural Network

- Speech recognition
- Music generation
- Sentiment classification
- DNA sequence analysis
- Machine translate
- Video activity recognition
- Name entity recognition

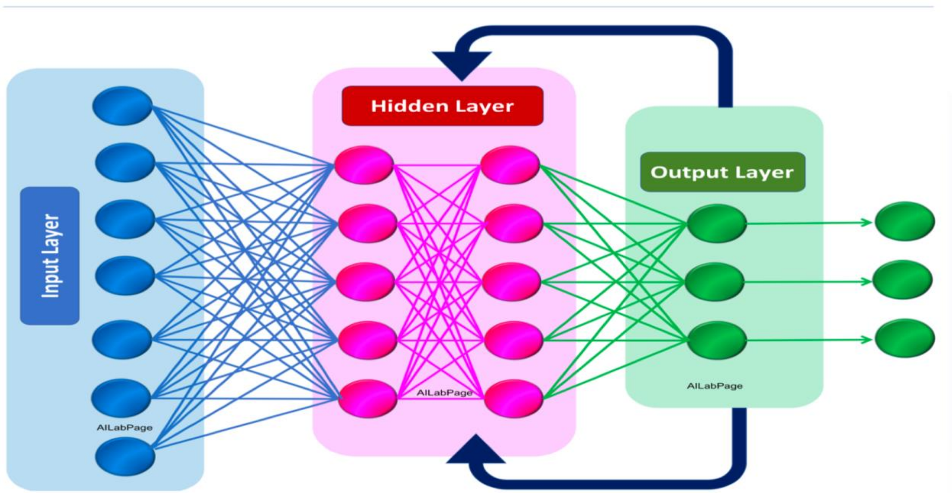


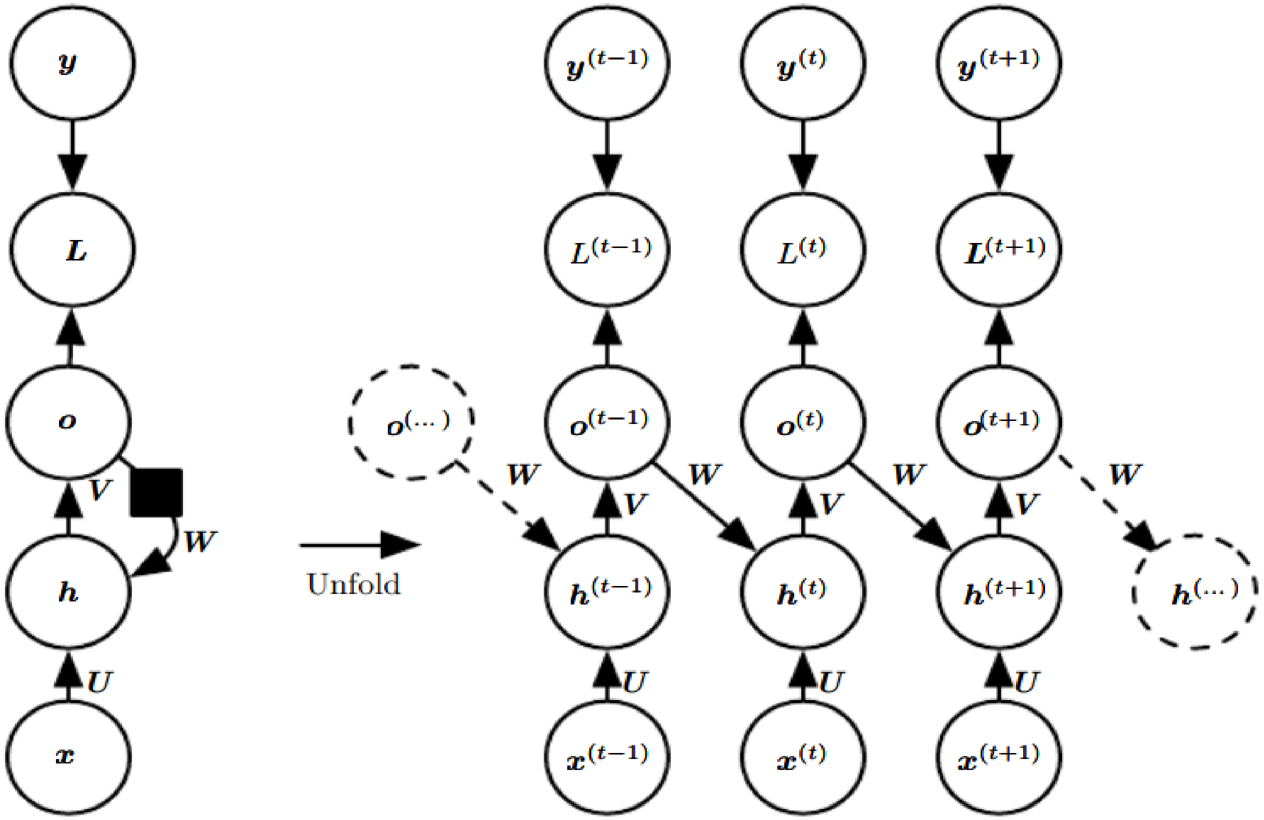
Recurrent Neural Network



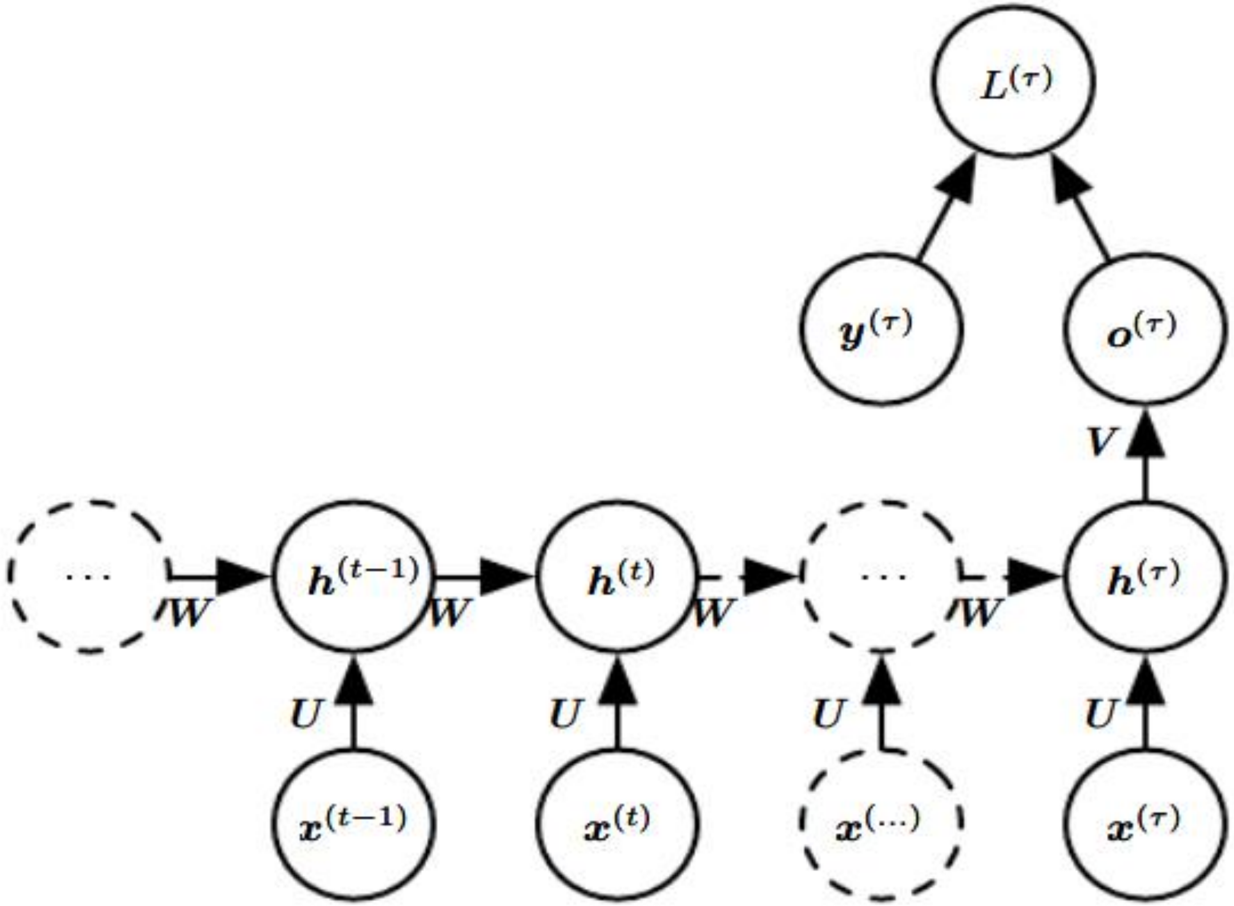


Recurrent Neural Network

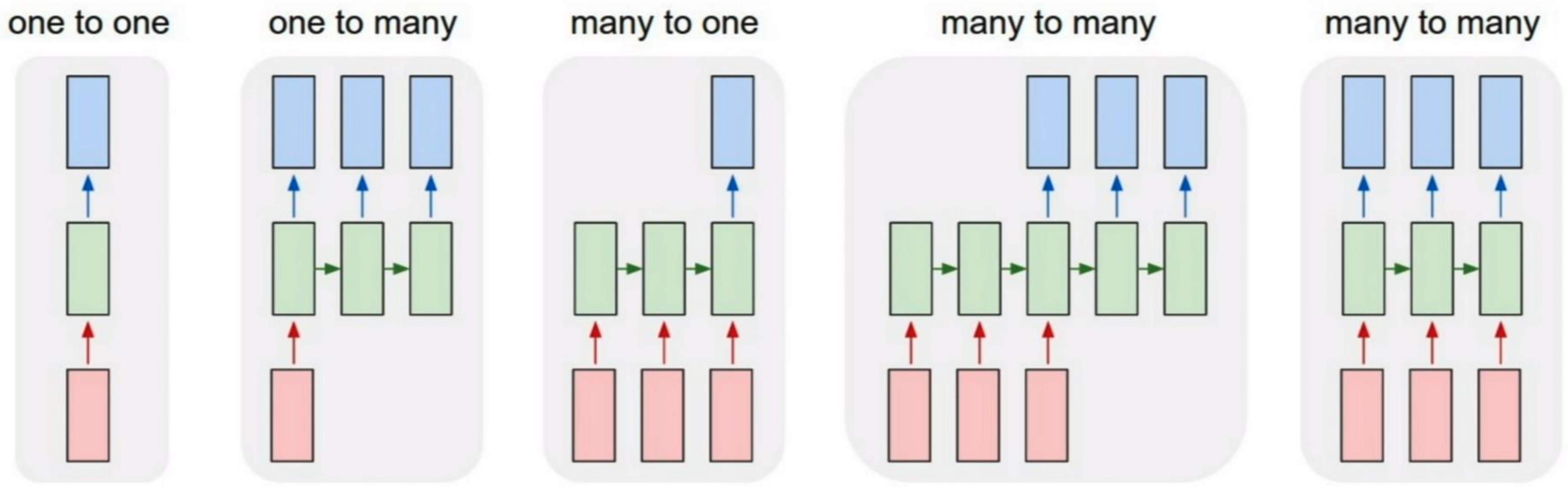




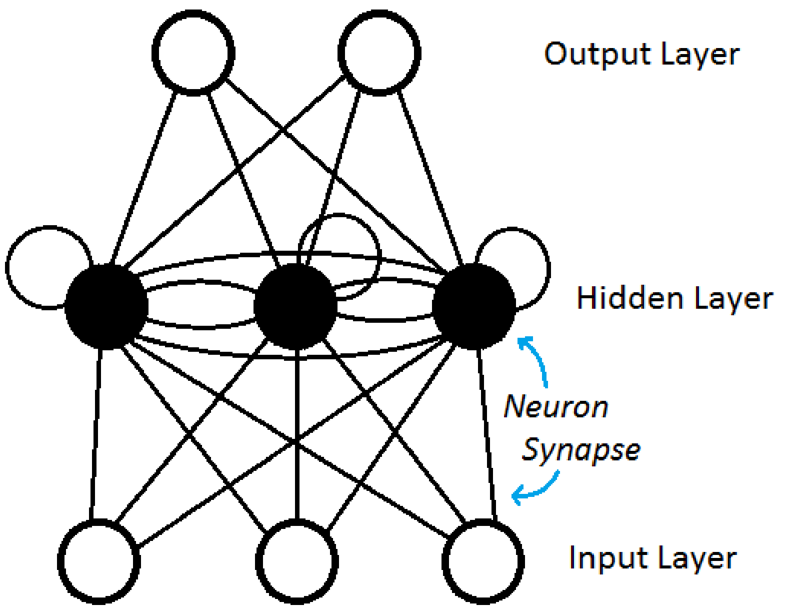
Recurrent Neural Network

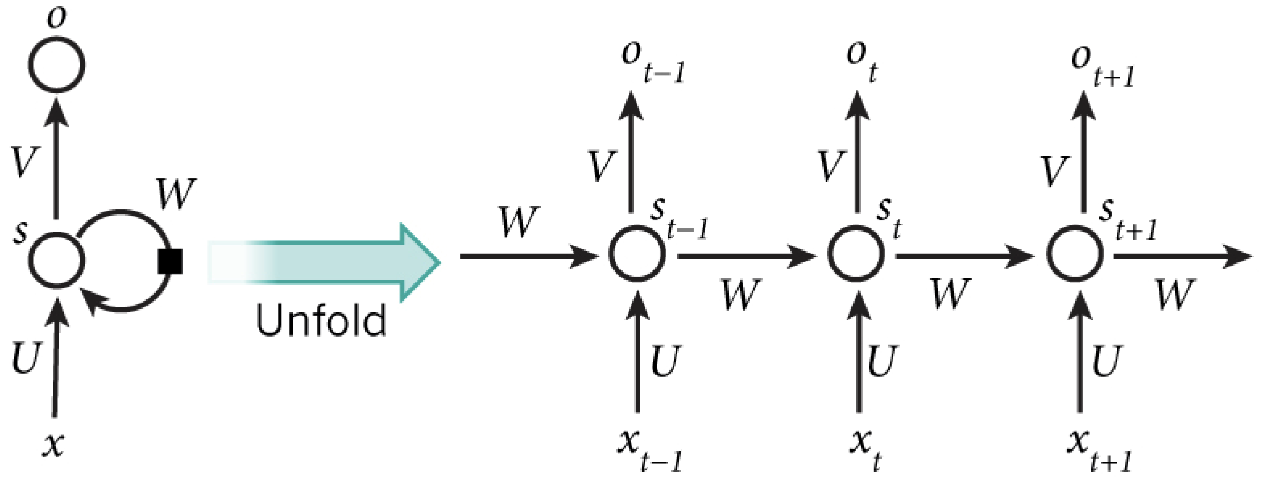


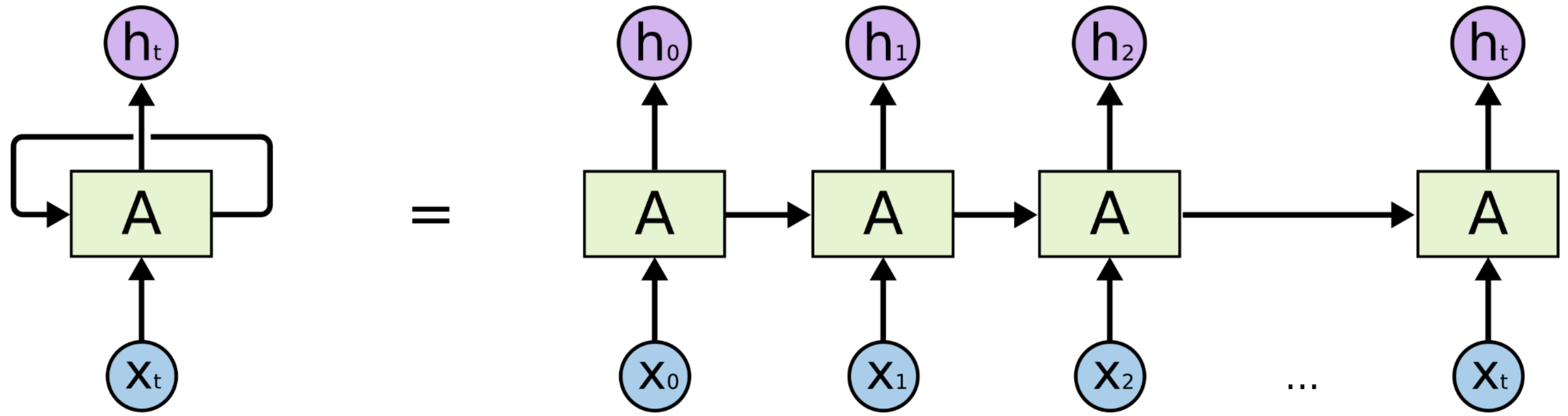
Types of Recurrent Neural Network



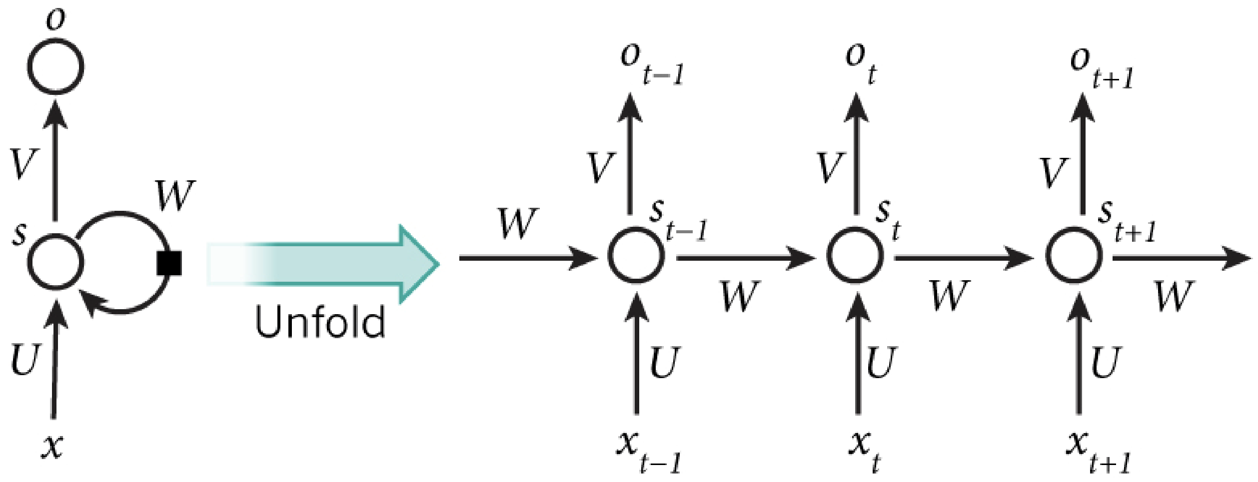
Recurrent Neural Network



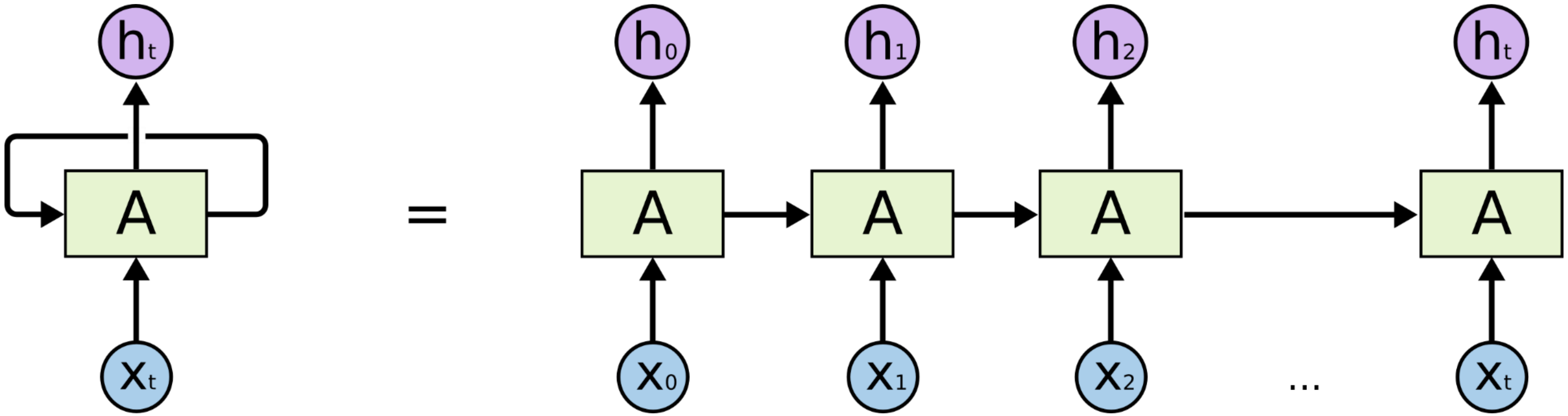




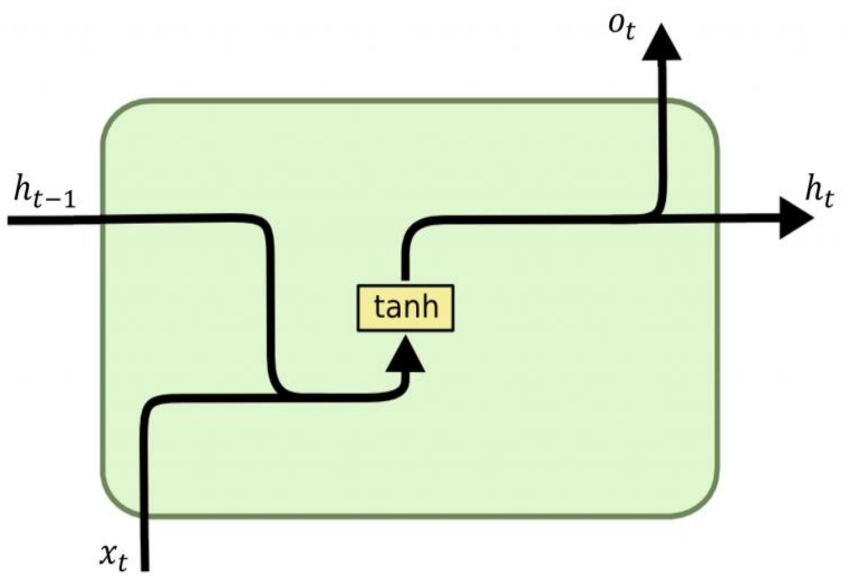
Recurrent Neural Network



$$a^{(t)} = b + Wh^{(t-1)} + Ux^{(t)}$$
$$h^{(t)} = tanh\left(a^{(t)}\right)$$
$$o^{(t)} = c + Vh^{(t)}$$
$$\hat{y}^{(t)} = softmax\left(o^{(t)}\right)$$



Recurrent layer representation



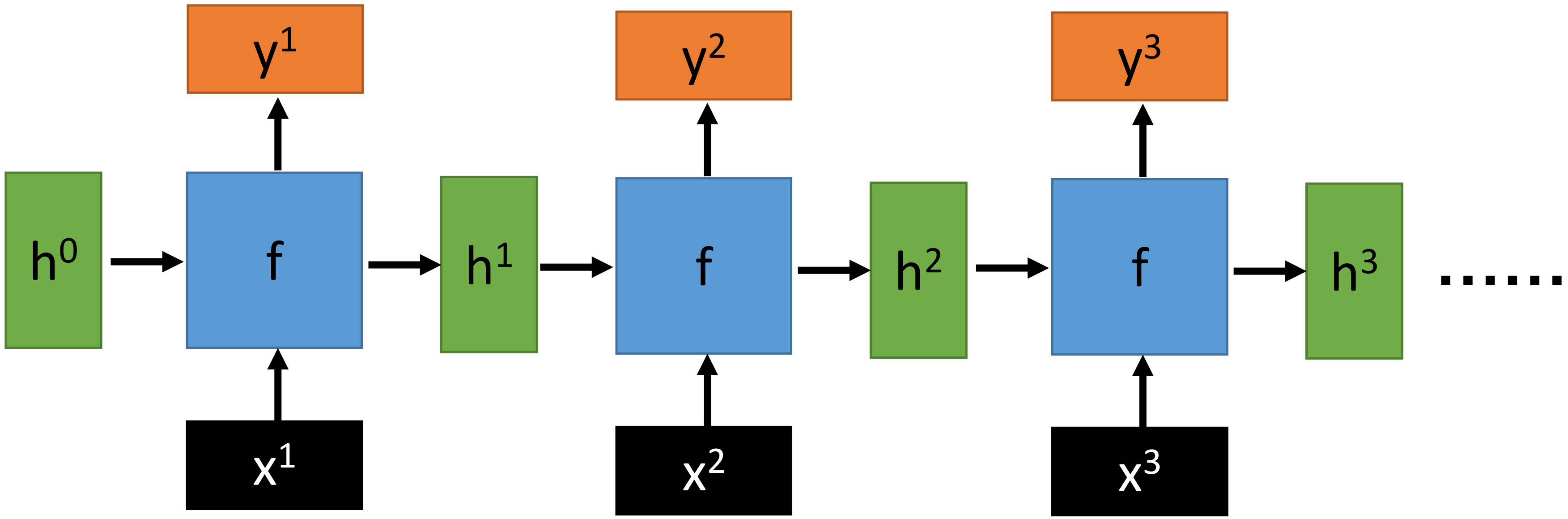
$$h_t = f(W_{hh}h_{t-1} + W_{xh}x_t + b_h)$$

$$o_t = SoftMax(W_{ho}h_t + b_o)$$

How does RNN reduce complexity?

- Given function f: h',y=f(h,x)

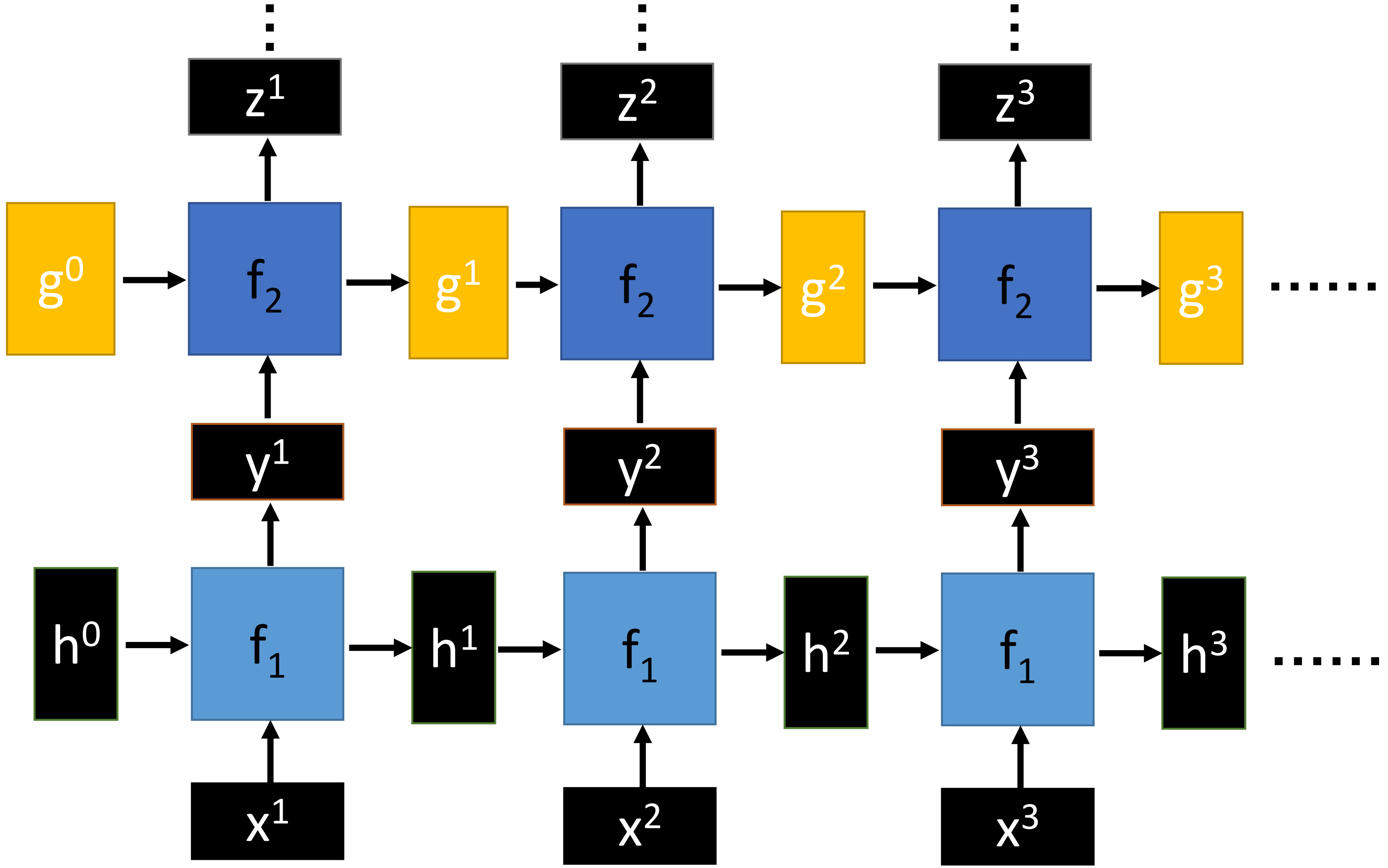
h and h' are vectors with the same dimension



No matter how long the input/output sequence is, we only need one function f. If f's are different, then it becomes a feedforward NN. This may be treated as another compression from fully connected network.

Deep RNN

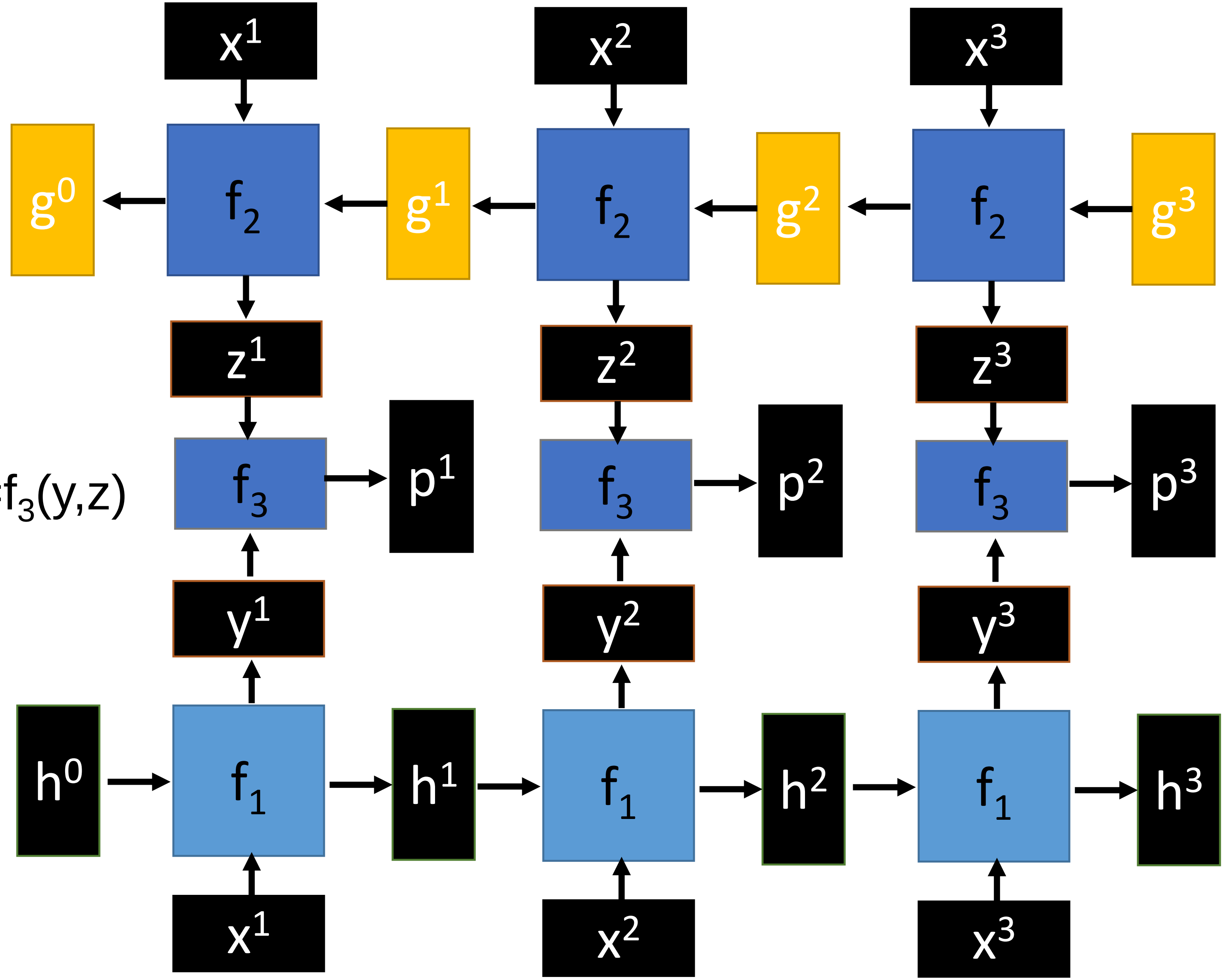
$h', y = f_1(h, x),\ g', z = f_2(g, y)$ …



Bidirectional RNN

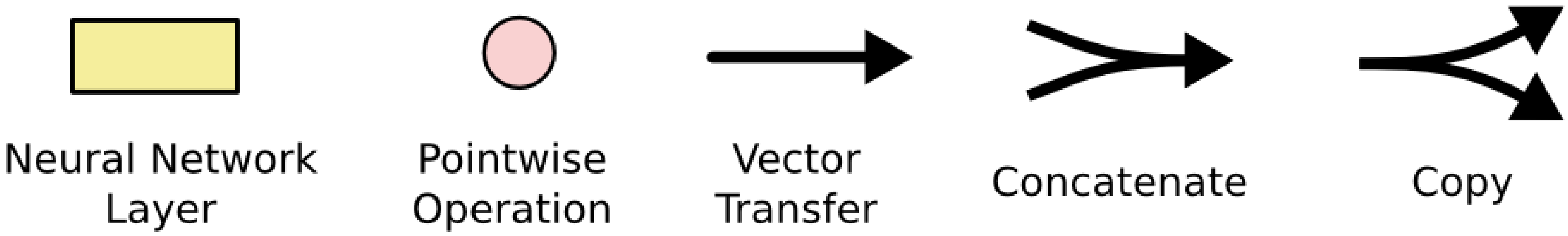
$y,h = f_1(x,h)$ $z,g = f_2(g,x)$

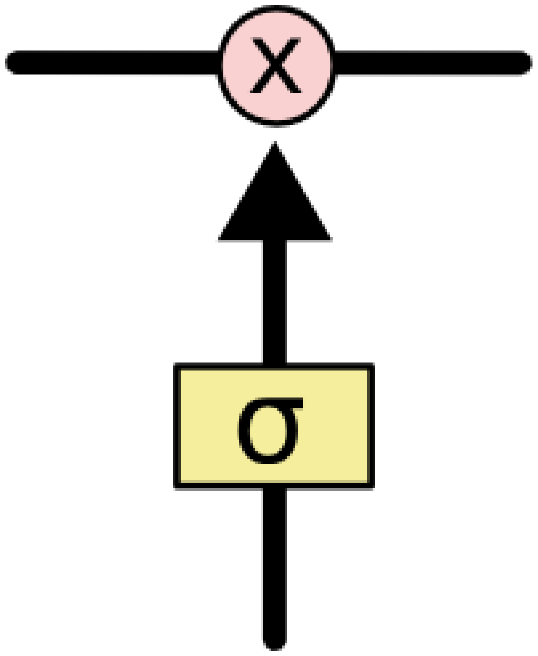
$p = f_3(y,z)$



Problems with naive RNN

- When dealing with a time series, it tends to forget old information. When there is a distant relationship of unknown length, we wish to have a “memory” to it.
- Vanishing gradient problem.





The sigmoid layer outputs numbers between 0-1 determine how much
each component should be let through. Pink X gate is point-wise multiplication.

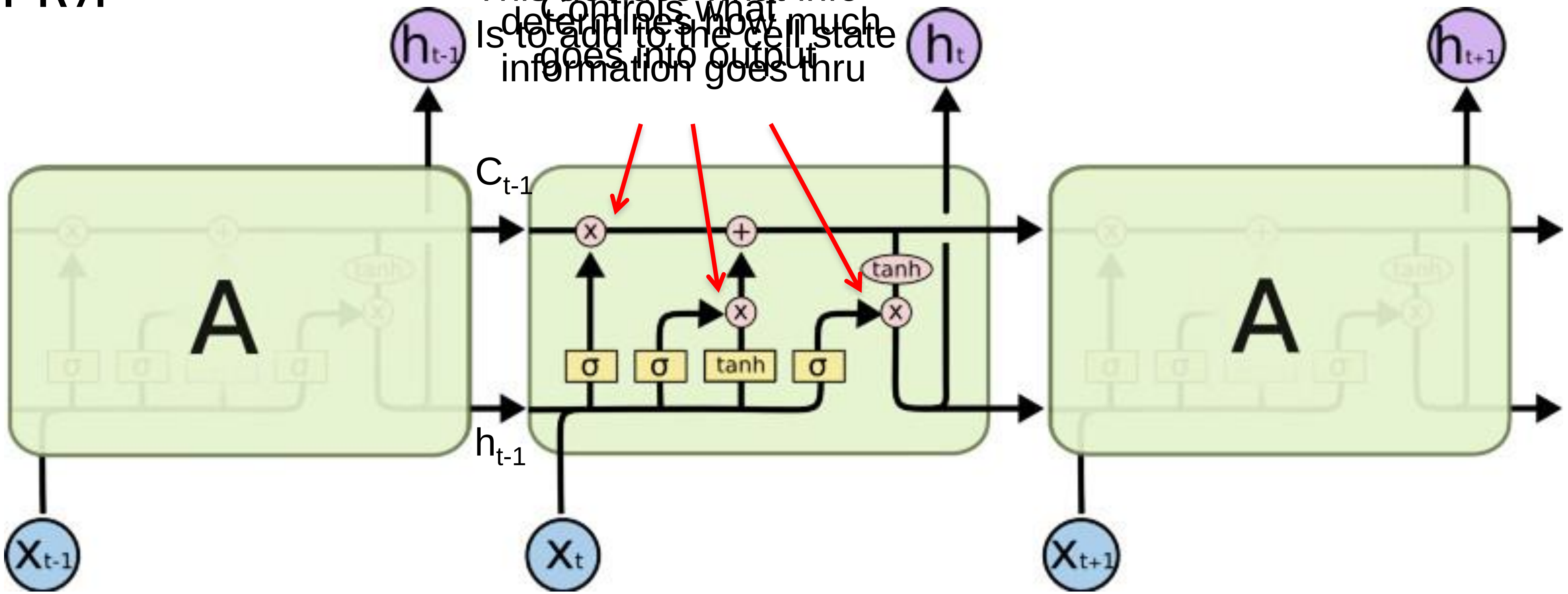
LSTM

Output gate

This sigmoid gate determines how much information goes thru

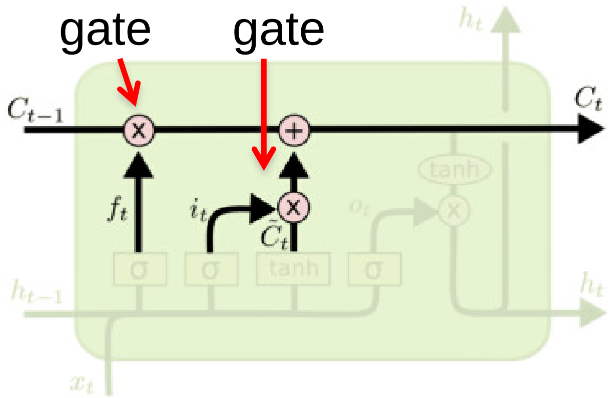
This decides what info Is to add to the cell state

Controls what goes into output



Forget gate

input gate



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

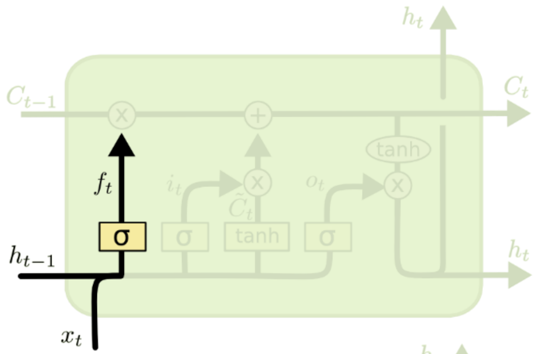
The core idea is this cell state C_t, it is changed slowly, with only minor linear interactions. It is very easy for information to flow along it unchanged.

Why sigmoid or tanh:

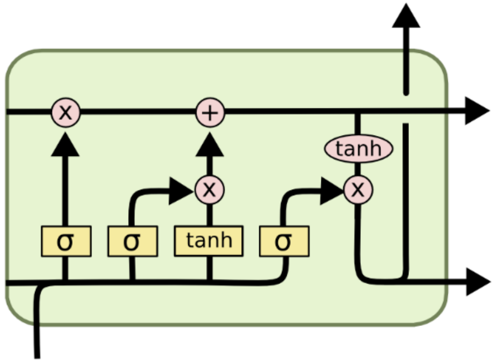
Sigmoid: 0,1 gating as switch.

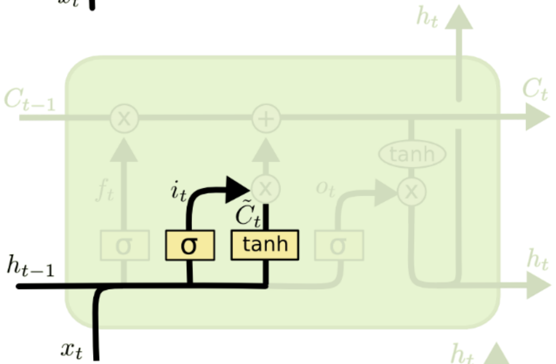
Vanishing gradient problem in LSTM is handled already.

ReLU replaces tanh ok?



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

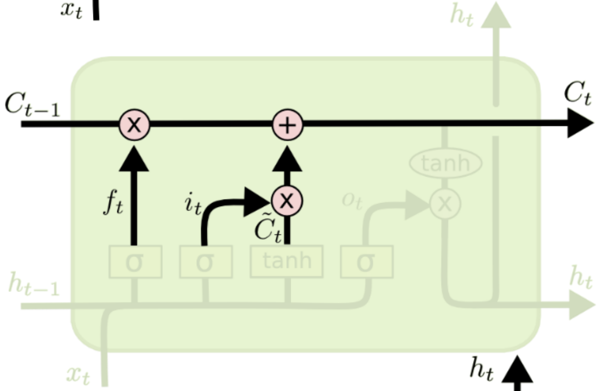




$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$

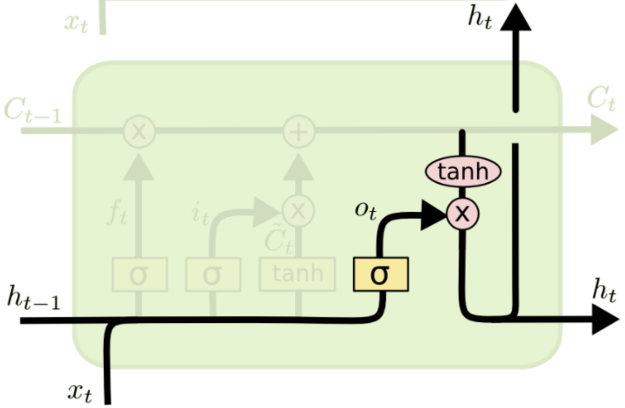
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \; + \; b_C)$$

i_t decides what component is to be updated.
C'$_t$ provides change contents



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

Updating the cell state

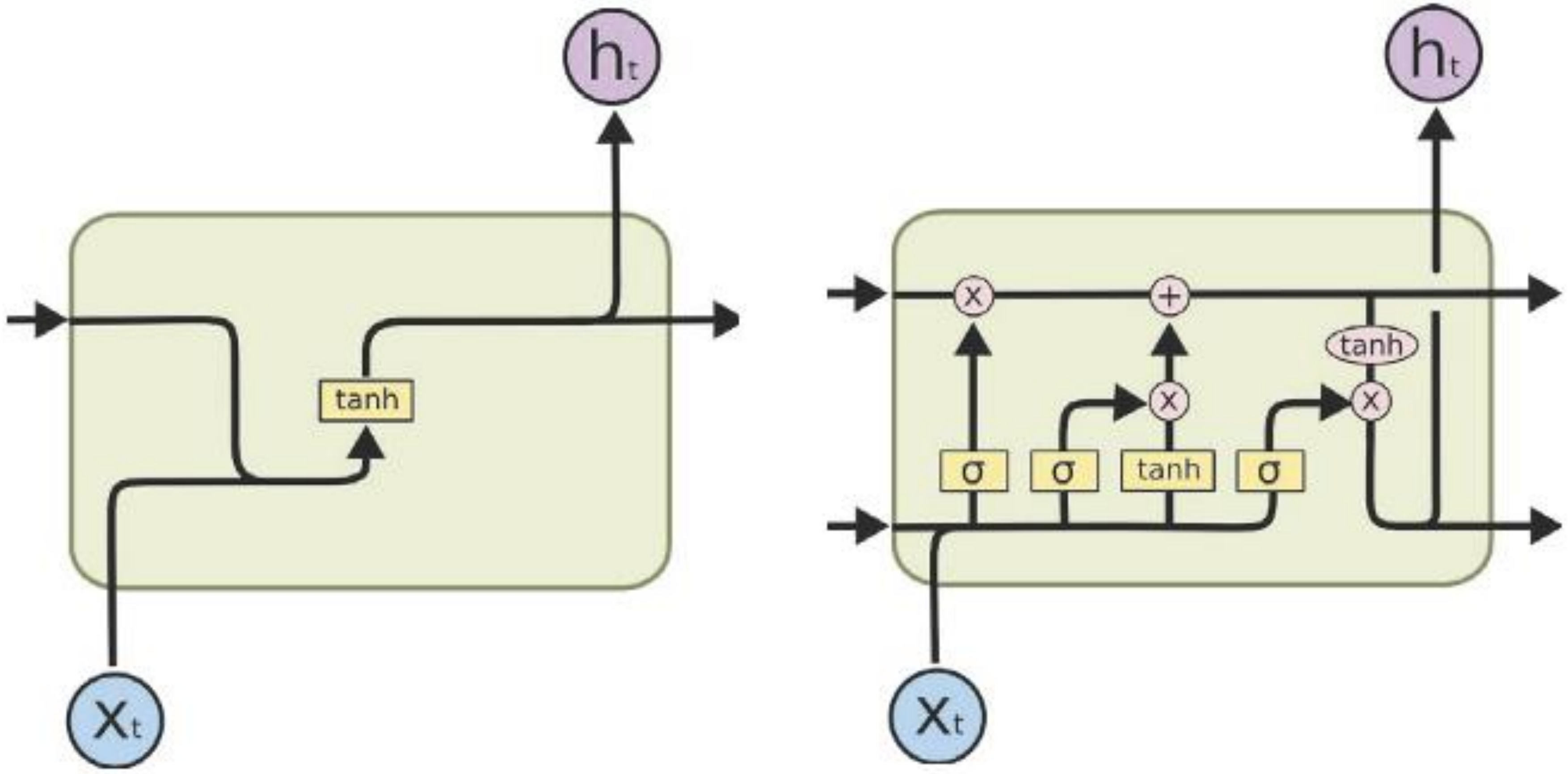


$$o_t = \sigma\left(W_o \; [h_{t-1}, x_t] \; + \; b_o\right)$$

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

Decide what part of the cell state to output

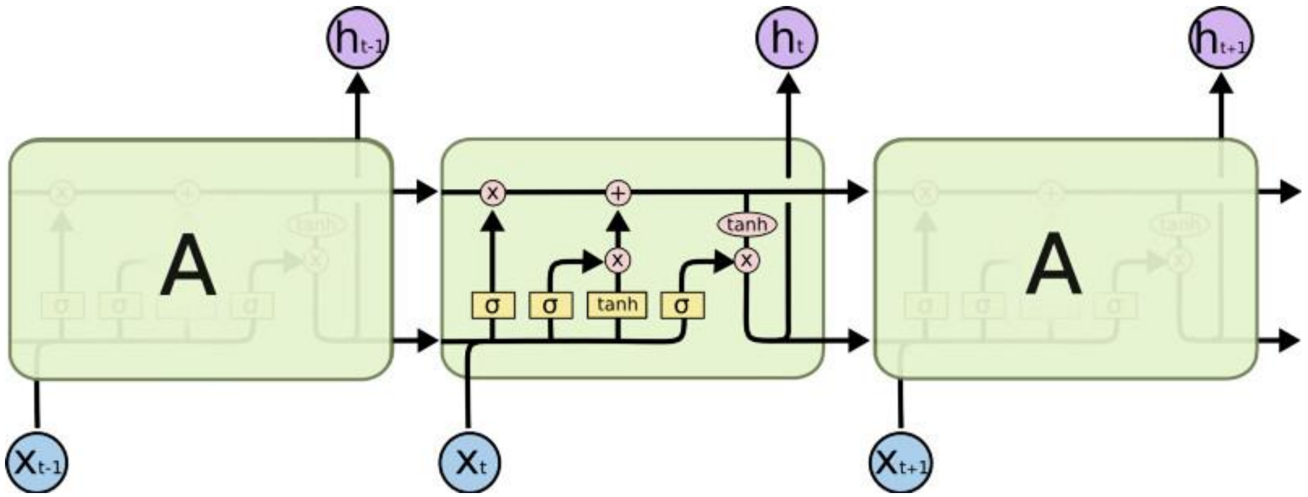
RNN vs LSTM



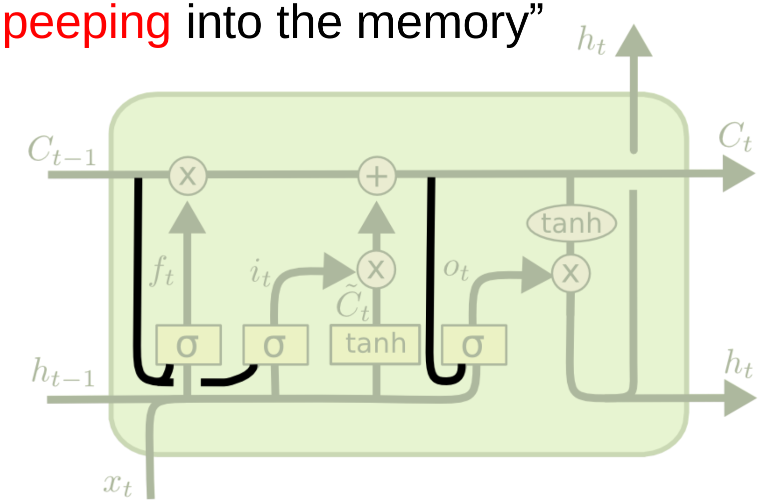
(a) RNN

(b) LSTM

Peephole LSTM

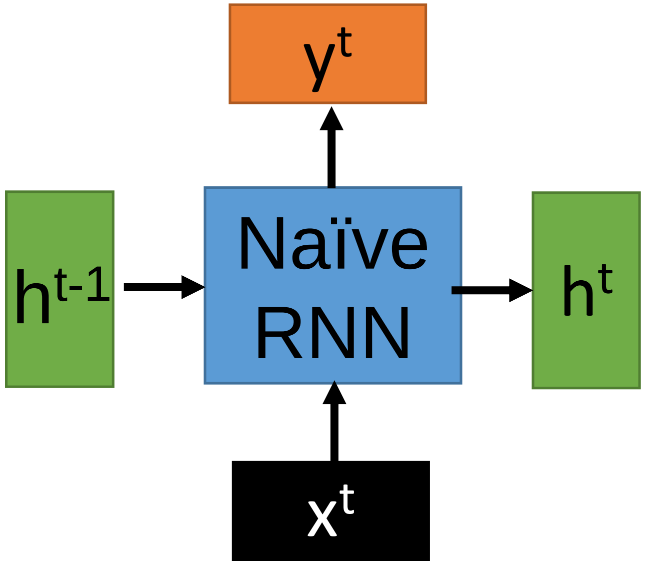


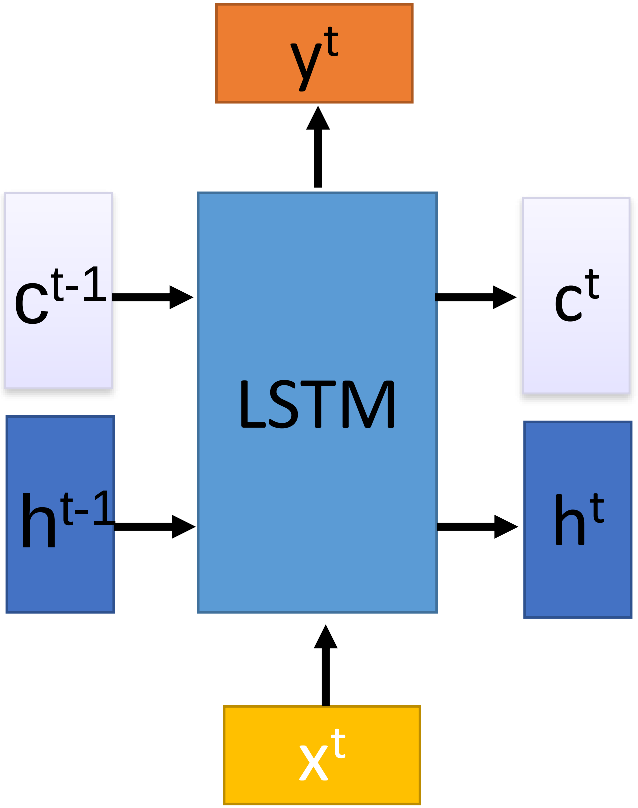
Allows “peeping into the memory”



$$f_t = \sigma\left(W_f \cdot [\boldsymbol{C_{t-1}}, h_{t-1}, x_t] \ + \ b_f\right)$$
$$i_t = \sigma\left(W_i \cdot [\boldsymbol{C_{t-1}}, h_{t-1}, x_t] \ + \ b_i\right)$$
$$o_t = \sigma\left(W_o \cdot [\boldsymbol{C_t}, h_{t-1}, x_t] \ + \ b_o\right)$$

Naïve RNN vs LSTM

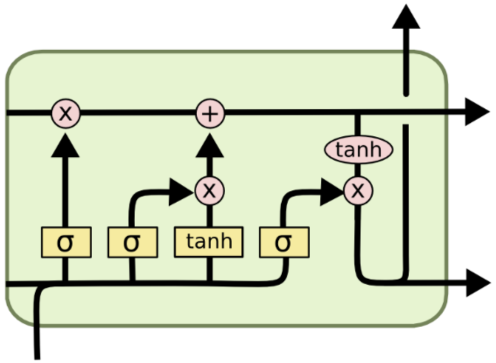




c changes slowly ➡ c^t is c^{t-1} added by something

h changes faster ➡ h^t and h^{t-1} can be very different

These 4 matrix computation should be done concurrently.



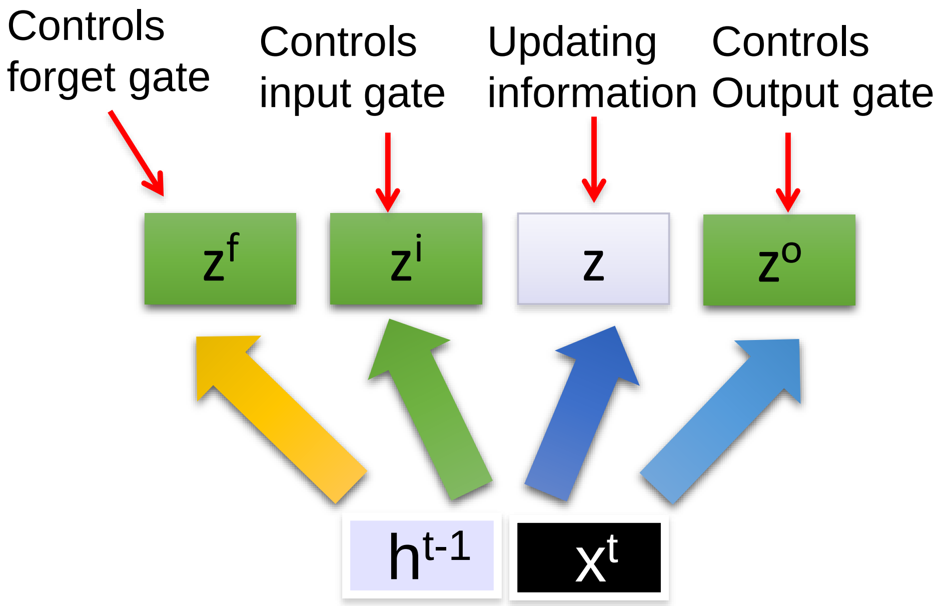
c^{t-1}

$$z = tanh(W \begin{bmatrix} x^t \\ h^{t-1} \end{bmatrix})$$

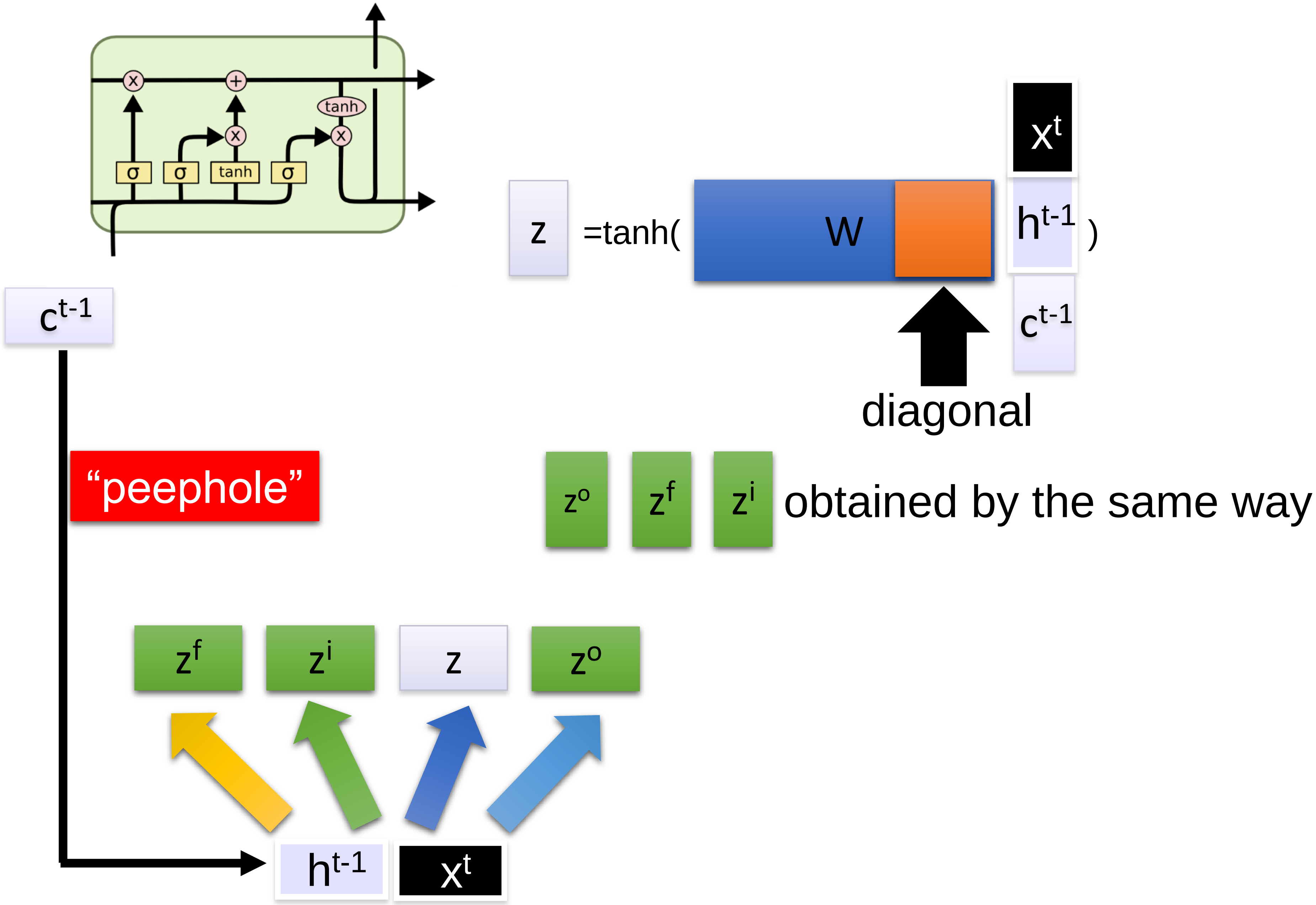
$$z^i = \sigma(W^i \begin{bmatrix} x^t \\ h^{t-1} \end{bmatrix})$$

$$z^f = \sigma(W^f \begin{bmatrix} x^t \\ h^{t-1} \end{bmatrix})$$

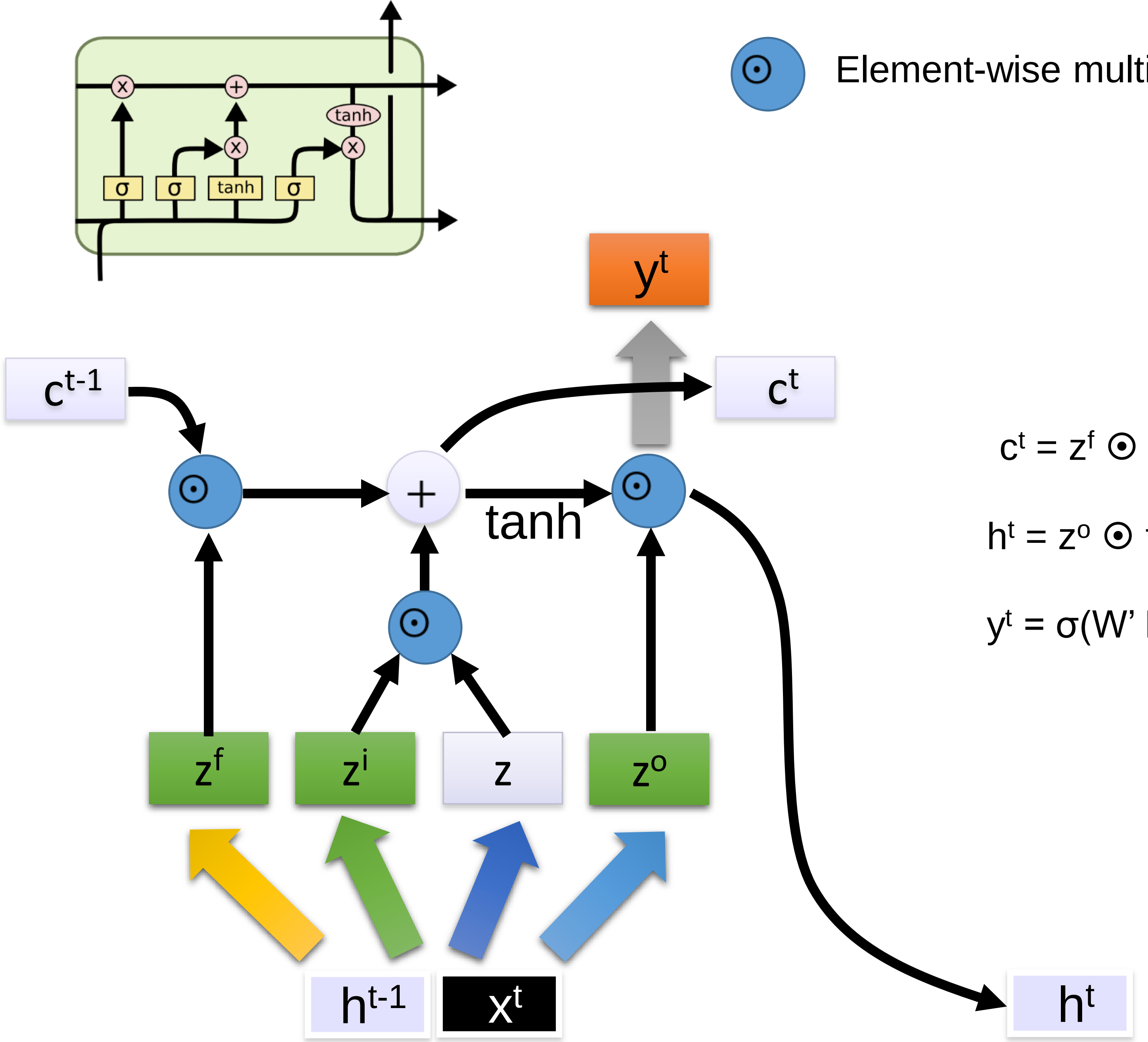
$$z^o = \sigma(W^o \begin{bmatrix} x^t \\ h^{t-1} \end{bmatrix})$$



Information flow of LSTM



Information flow of LSTM



$\odot$ Element-wise multiply

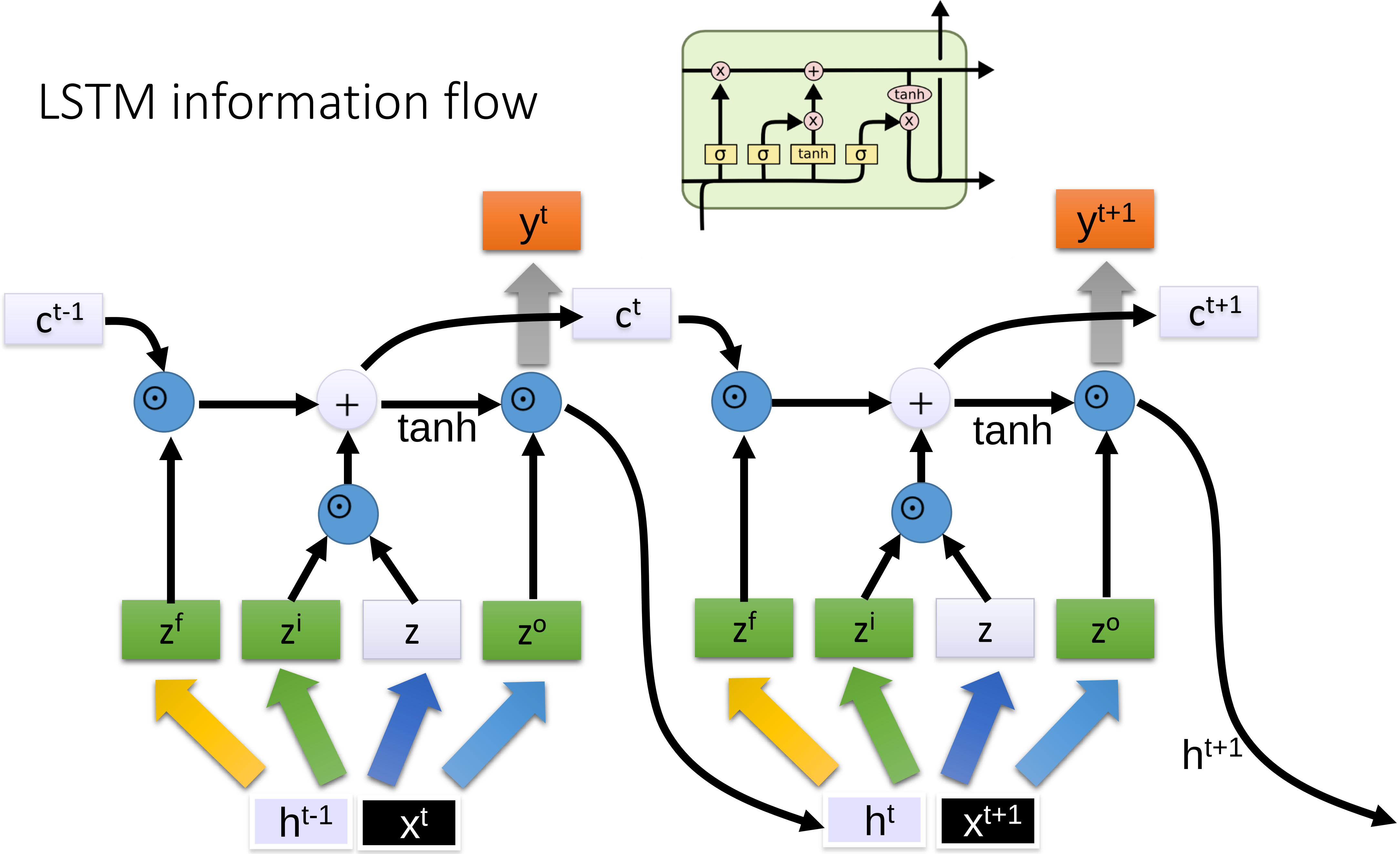
$c^t = z^f \odot c^{t-1} + z^i \odot z$

$h^t = z^o \odot \tanh(c^t)$

$y^t = \sigma(W' h^t)$

Information flow of LSTM

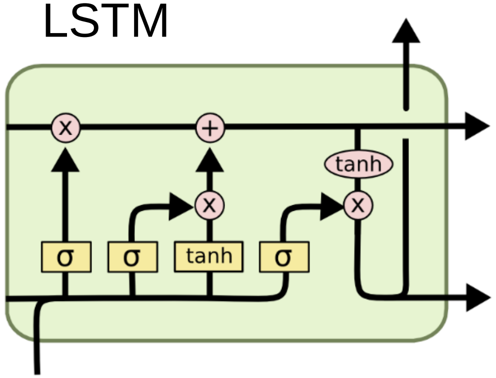
LSTM information flow

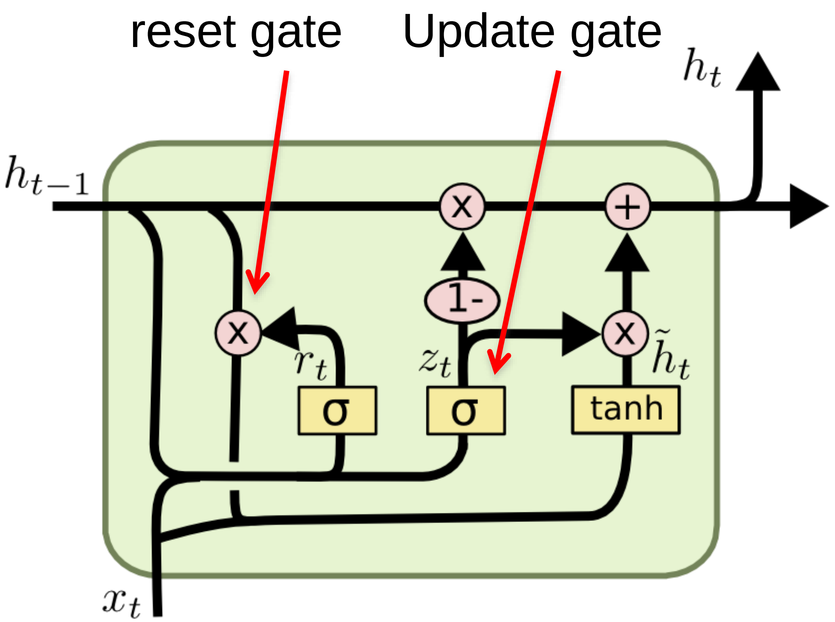


Information flow of LSTM

GRU – gated recurrent unit

(more compression)





$$z_t = \sigma\left(W_z \cdot [h_{t-1}, x_t]\right)$$

$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$

$$\tilde{h}_t = \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right)$$

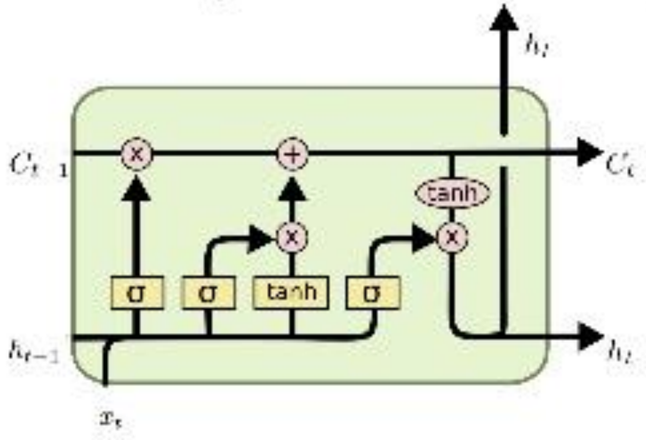
$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

It combines the forget and input into a single update gate.
It also merges the cell state and hidden state. This is simpler than LSTM. There are many other variants too.

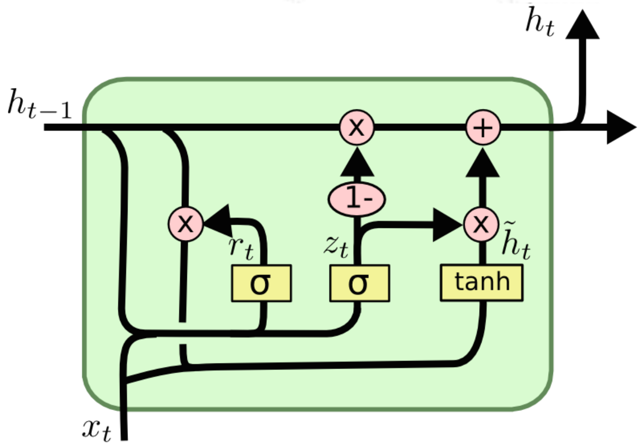
X,*: element-wise multiply

LSTM and GRU

- LSTM [Hochreiter&Schmidhuber97]



- GRU [Cho+14]



GRUs also takes x_t and h_{t-1} as inputs. They perform some calculations and then pass along h_t. What makes them different from LSTMs is that GRUs don't need the cell layer to pass values along. The calculations within each iteration insure that the h_t values being passed along either retain a high amount of old information or are jump-started with a high amount of new information.

PYTHON CODE