

# Recurrent Neural Networks (RNN)

Rowel Atienza  
[rowel@eee.upd.edu.ph](mailto:rowel@eee.upd.edu.ph)  
*University of the Philippines*  
*Updated: 3 Oct 2020*

# Recurrent Neural Network (RNN)

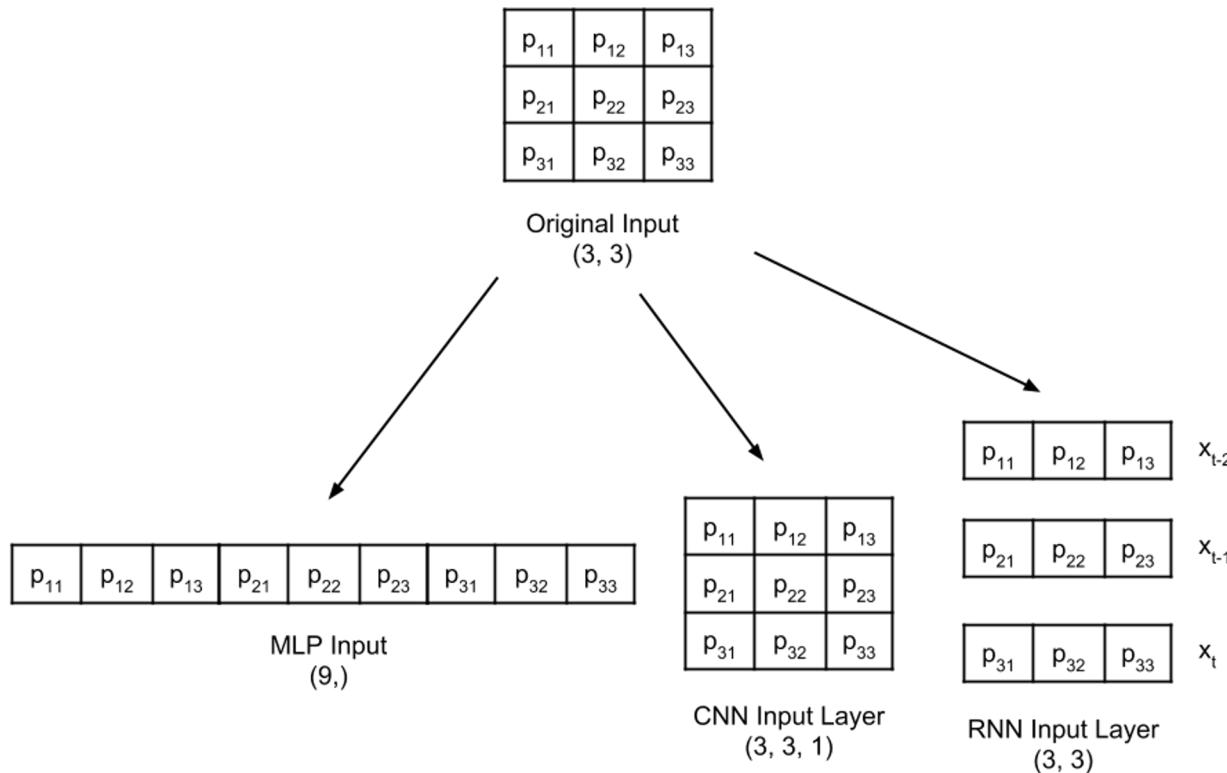
Specializes in processing **sequential data**:  $x_0, x_1, \dots x_t$

Sequence of words : the quick brown fox jumps over the lazy dog (NLP)

Sequence of signals : series of quantized audio signal for speech to text (ASR)

Sequence of pixels : 2D image can be converted into 1D array of pixels

# An image can be interpreted in 3 different ways

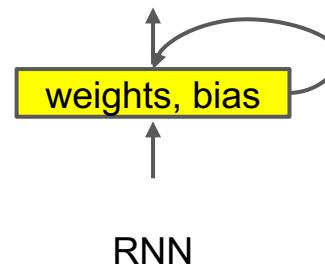
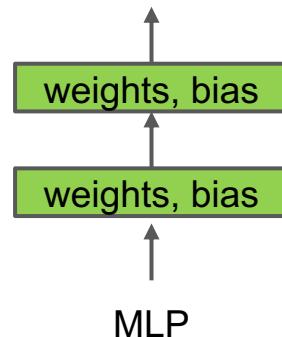
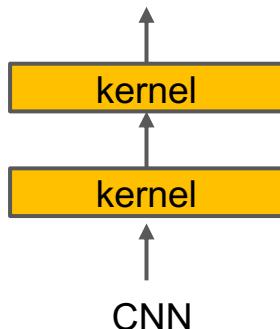


# Recurrent Neural Network

One difference between RNN and CNN and MLP is parameters to be learned are shared among inputs

$x_0, x_1, \dots x_t$  share the same set of parameters over the sequence

CNN and MLP share the same set of parameter over the entire input



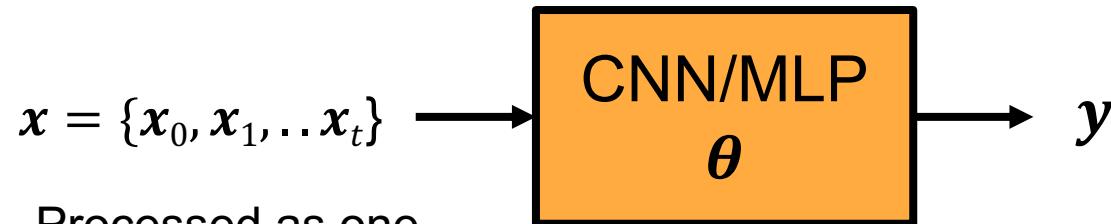
# Recurrent Neural Network

Specializes in processing sequential data:  $x_0, x_1, \dots x_t$

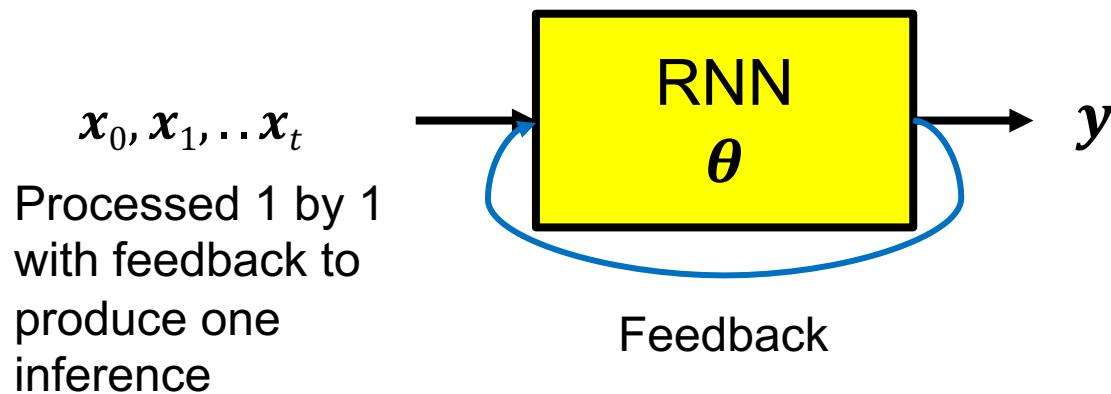
In MLP and CNN,  $x_0, x_1, \dots x_t$  are treated as **independent inputs** whose features needed to be learned

In RNN, sequential inputs are **dependent on one another**,  $x_0, x_1, \dots x_t$

For example:  
 $x$  is an image  
made of rows  
of pixels  $x_i$



Processed as one  
to produce one  
inference



Processed 1 by 1  
with feedback to  
produce one  
inference

Feedback

# Recurrent Neural Network - Unfolding

Consider a dynamical system with states:  $s_0, s_1, \dots, s_t$

$$s_t = f(s_{t-1}; \theta_{t-1})$$

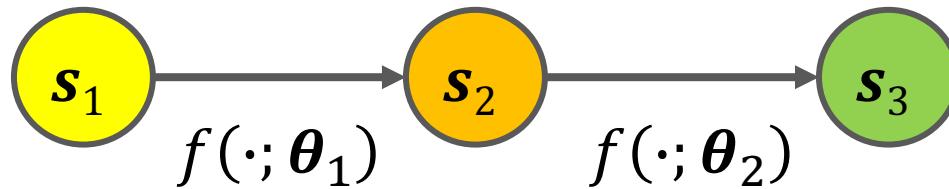
$\theta_{t-1}$  are the parameters at time  $t - 1$

For example:

$$s_3 = f(s_2; \theta_2) = f(f(s_1; \theta_1); \theta_2)$$

This process is called **unfolding**

# Recurrent Neural Network - Unfolding in Graphical Model



# Recurrent Neural Network - with external signal

The dynamical system can also have external input:  $x_t$

$$s_t = f(s_{t-1}, x_{t-1}; \theta_{t-1})$$

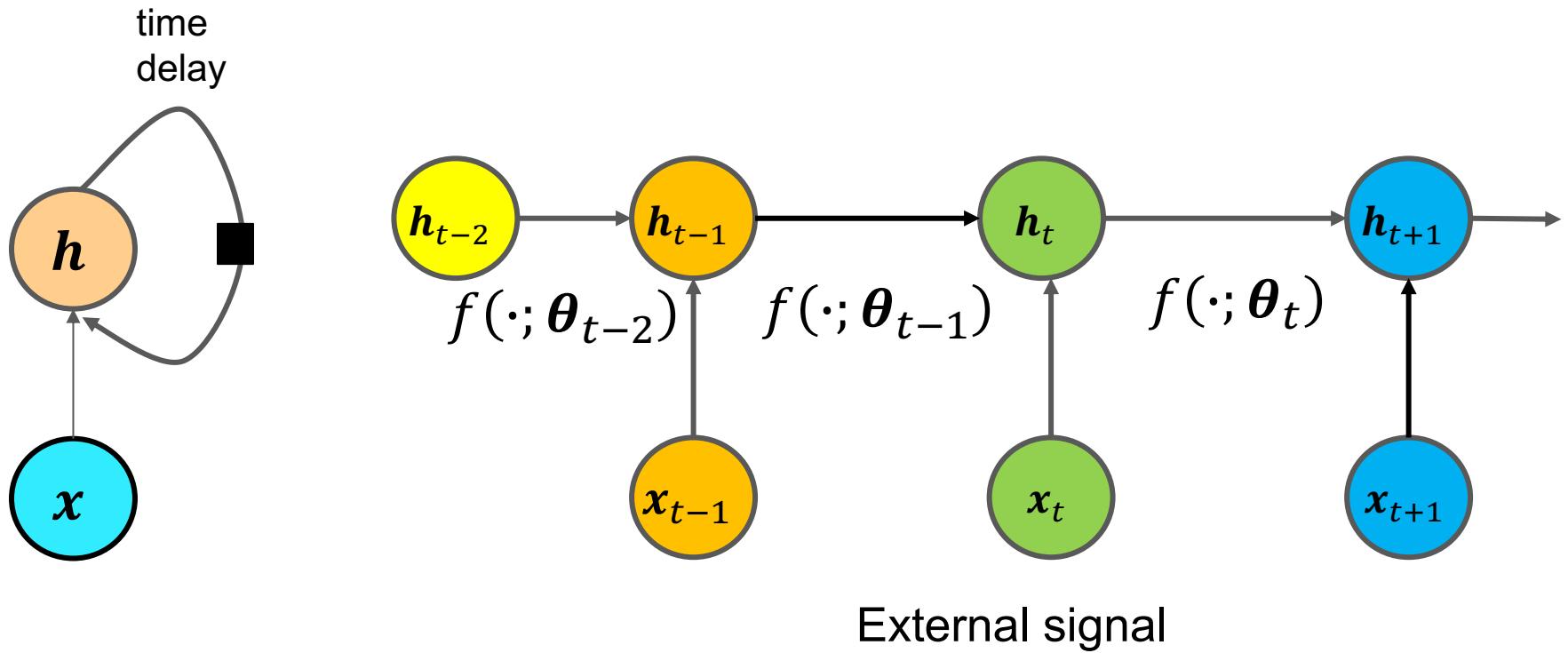
If we use a neural network to store the state, a hidden layer can be modeled as:

$$h_t = f(h_{t-1}, x_{t-1}; \theta_{t-1})$$

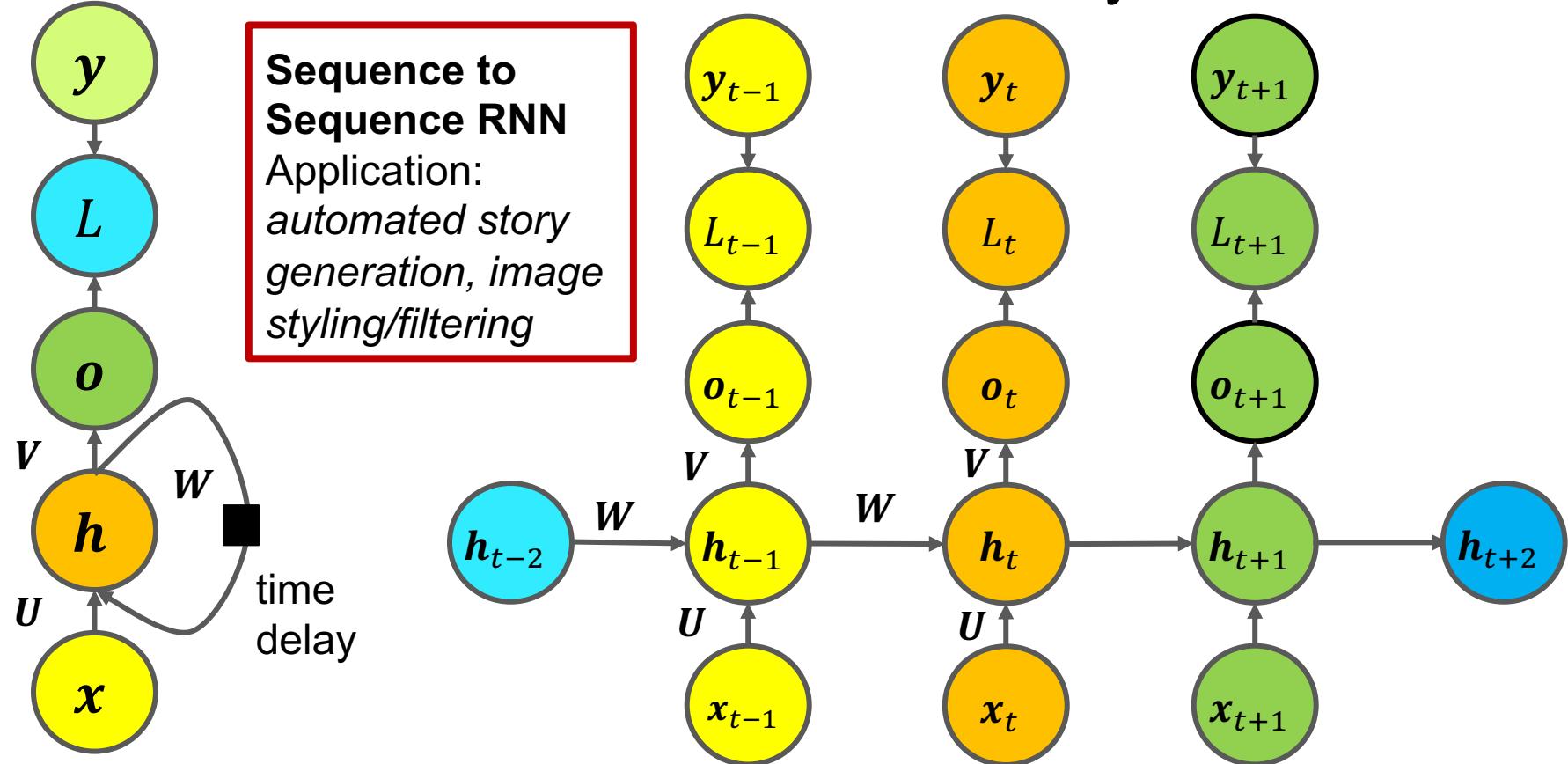
The output at time  $t$  is a function of previous output of hidden layer and the current input

$h_t$  encodes (lossy) all previous inputs  $x_0, x_1, \dots, x_{t-1}$

# Unfolding of RNN (Graphical Model)



# RNN - Recurrence between hidden layers



# RNN - Recurrence between hidden layers

$$\boldsymbol{a}_t = \boldsymbol{b} + \boldsymbol{W}\boldsymbol{h}_{t-1} + \boldsymbol{U}\boldsymbol{x}_t$$

$$\boldsymbol{h}_t = \tanh(\boldsymbol{a}_t)$$

$$\boldsymbol{o}_t = \boldsymbol{c} + \boldsymbol{V}\boldsymbol{h}_t$$

$$L_t = -\log p(\boldsymbol{y}_t | \{\boldsymbol{x}_1, \boldsymbol{x}_2, \dots, \boldsymbol{x}_t\})$$

$$\boldsymbol{y}_t = \text{softmax}(\boldsymbol{o}_t)$$

$\boldsymbol{W}$ ,  $\boldsymbol{U}$  and  $\boldsymbol{V}$  are parameter matrices

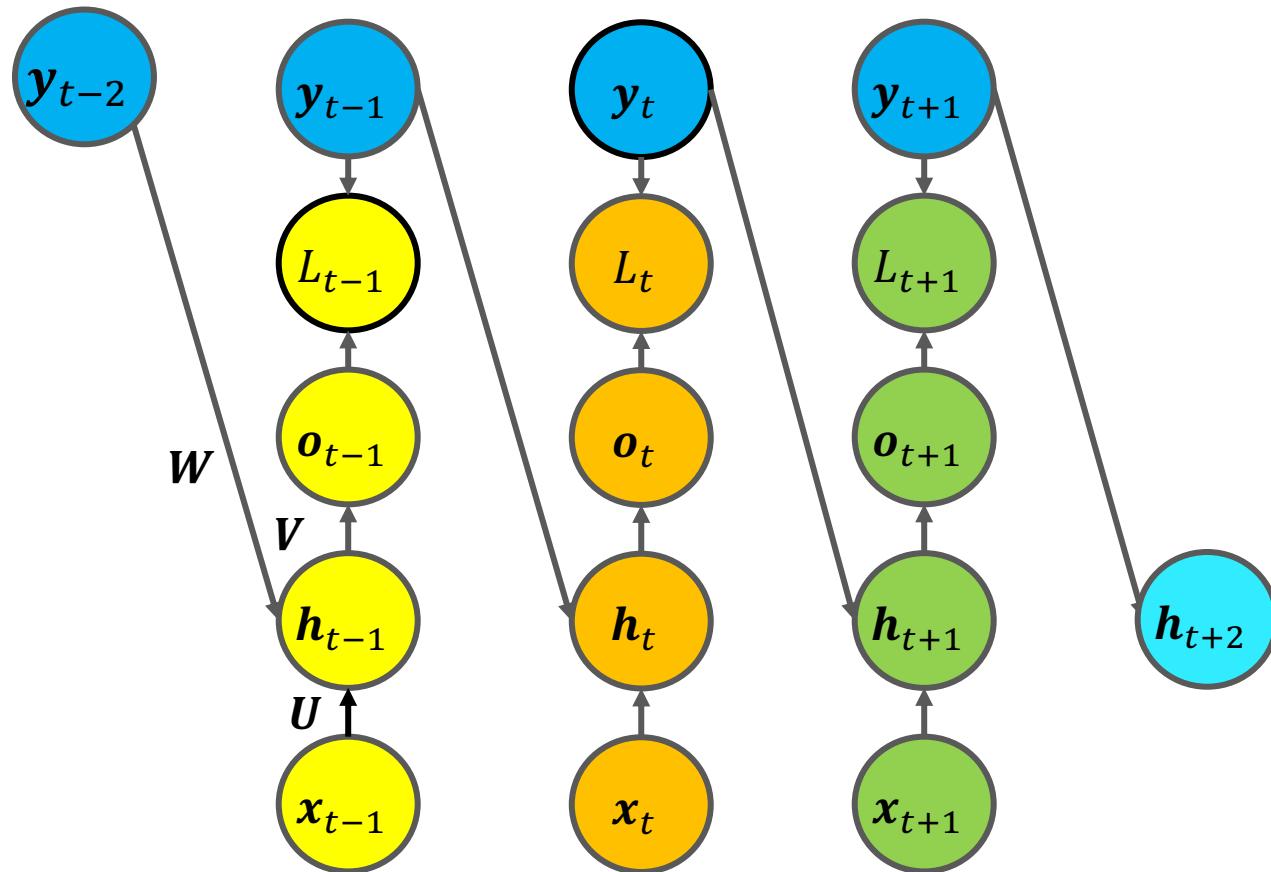
$\boldsymbol{b}$  and  $\boldsymbol{c}$  are biases

RNN maximizes the conditional probability to minimize  $L$

$$L_t = -\log p(y_t | \{x_1, x_2, \dots, x_t\})$$

By nature, RNN is not parallelizable

# RNN - Teacher forcing (Parallelizable RNN)

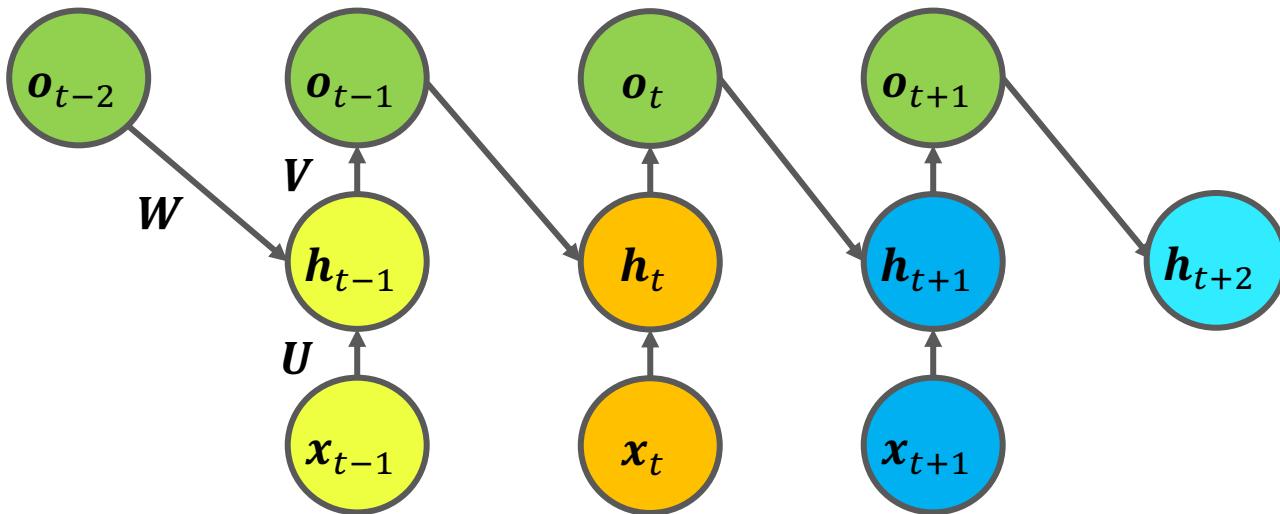


Teacher forcing maximizes the conditional probability to minimize  $L$

$$L = -\log p(y_t, y_{t-1} | \{x_1, x_2, \dots, x_t\})$$

$$L = -\log p(y_t | \{x_1, x_2, \dots, x_t, y_{t-1}\}) - \log p(y_{t-1} | \{x_1, x_2, \dots, x_t\})$$

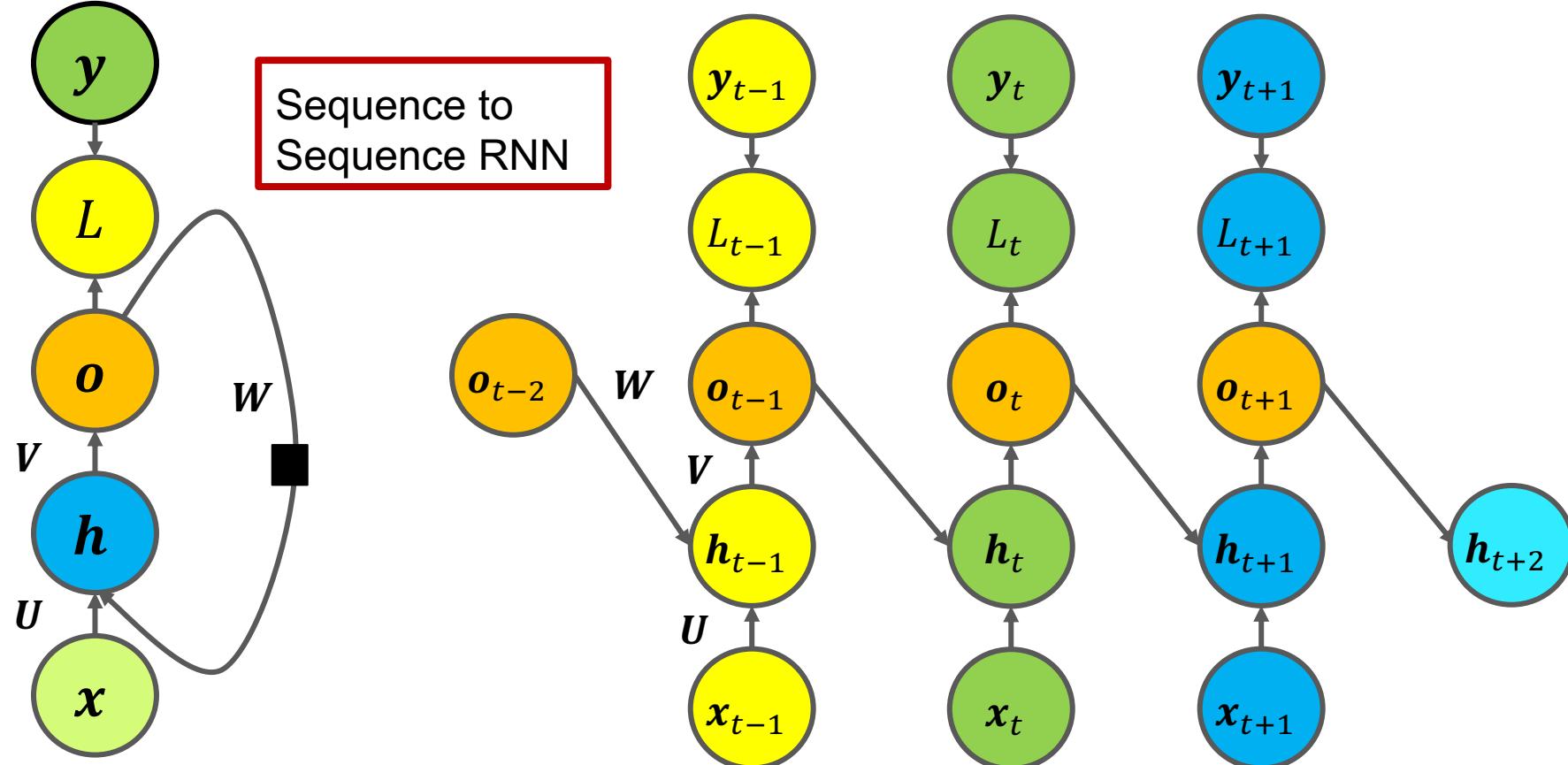
# RNN - Teacher forcing (testing)



Since we have no access to training outputs, we use our predicted outputs

# RNN - Recurrence between output & hidden layers

Can be parallelized!  $\mathbf{o}_t$  can be estimated by  $y_t$  during training



# RNN - Recurrence between output & hidden layers

$$\mathbf{a}_t = \mathbf{b} + \mathbf{W}\mathbf{h}_t + \mathbf{U}\mathbf{x}_t$$

$$\mathbf{h}_t = \tanh(\mathbf{a}_t)$$

$$\mathbf{o}_t = \mathbf{c} + \mathbf{V}\mathbf{h}_t$$

$$L = -\log p(\mathbf{y}_t | \{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_t\})$$

$$\mathbf{y}_t = \text{softmax}(\mathbf{o}_t)$$

$\mathbf{W}$ ,  $\mathbf{U}$  and  $\mathbf{V}$  are parameter matrices

$\mathbf{b}$  and  $\mathbf{c}$  are biases

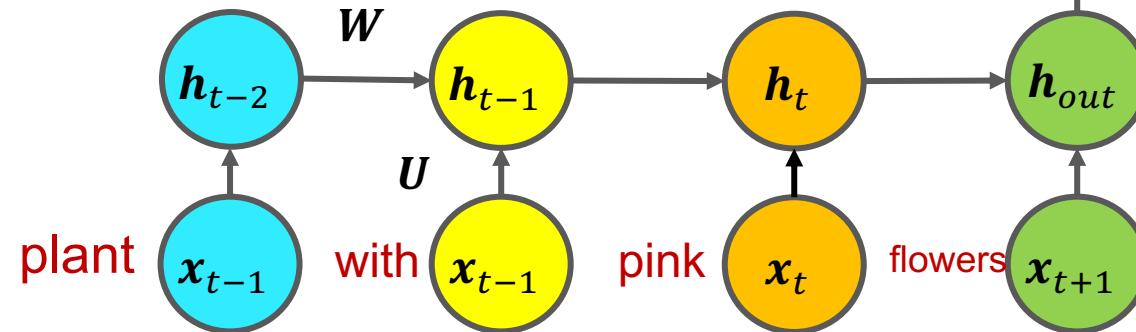
Exercise: Modify the equation for the preceding figure

# RNN - Recurrence between hidden layers; 1 output

## Sequence to Vector RNN

Applicable to problems generating one vector output given a sequence of inputs

Example: Generate an image of a **plant** with pink flowers



$y_{out}$

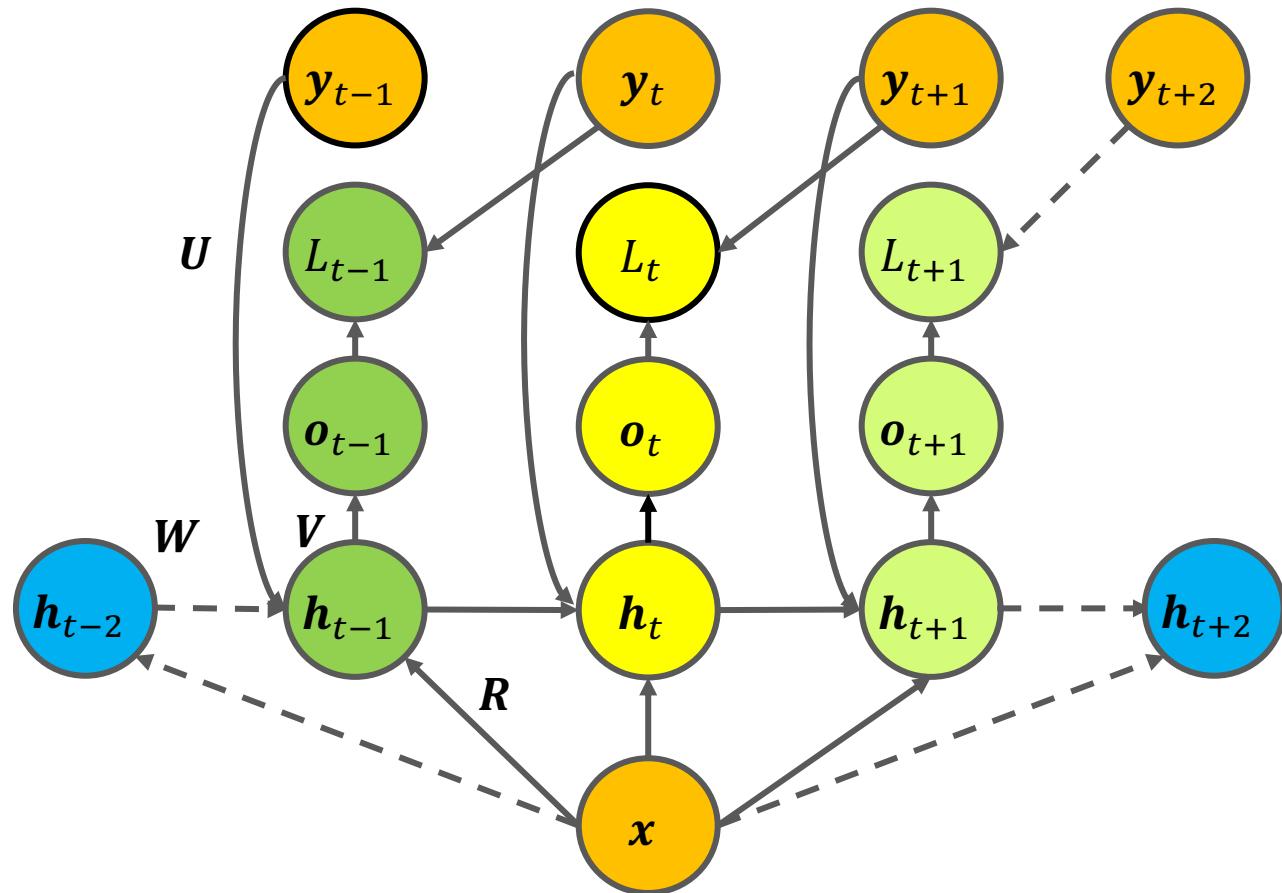
*Exercise:*  
Write the equations  
for this network

# Vector to Sequence RNN

$y_0$   $y_1$        $y_2$        $y_1$   
A ripe calamansi fruit



$x$



# Vector to Sequence

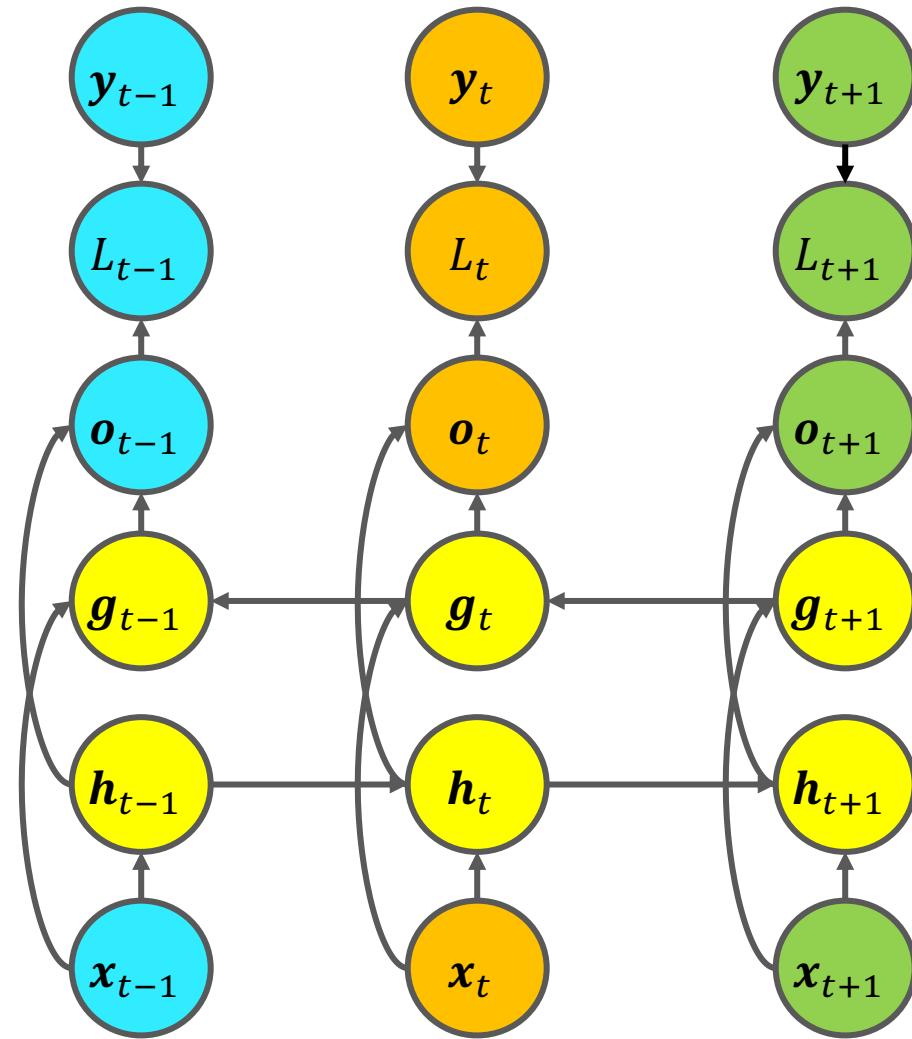
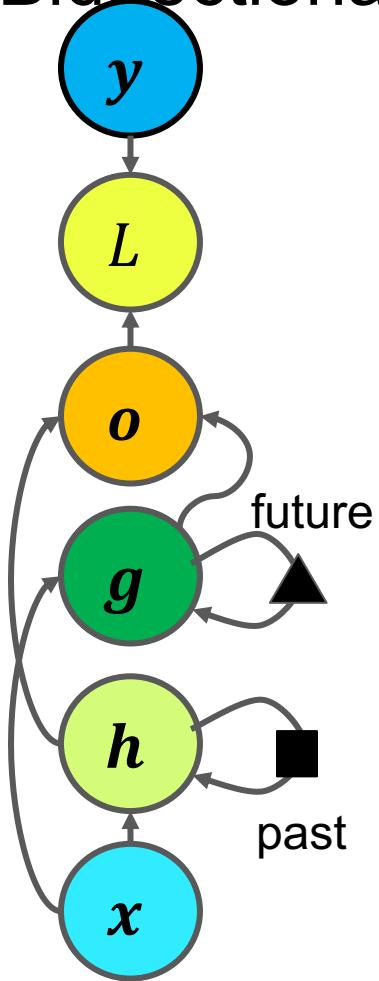
Applicable to problems converting fixed data to a variable length sequence

Image (fixed) → Caption (variable)

See <http://visualqa.org>

for good problems to solve.

# Bidirectional RNN



# Bidirectional RNN

Prediction is not only dependent on the past but also on the future

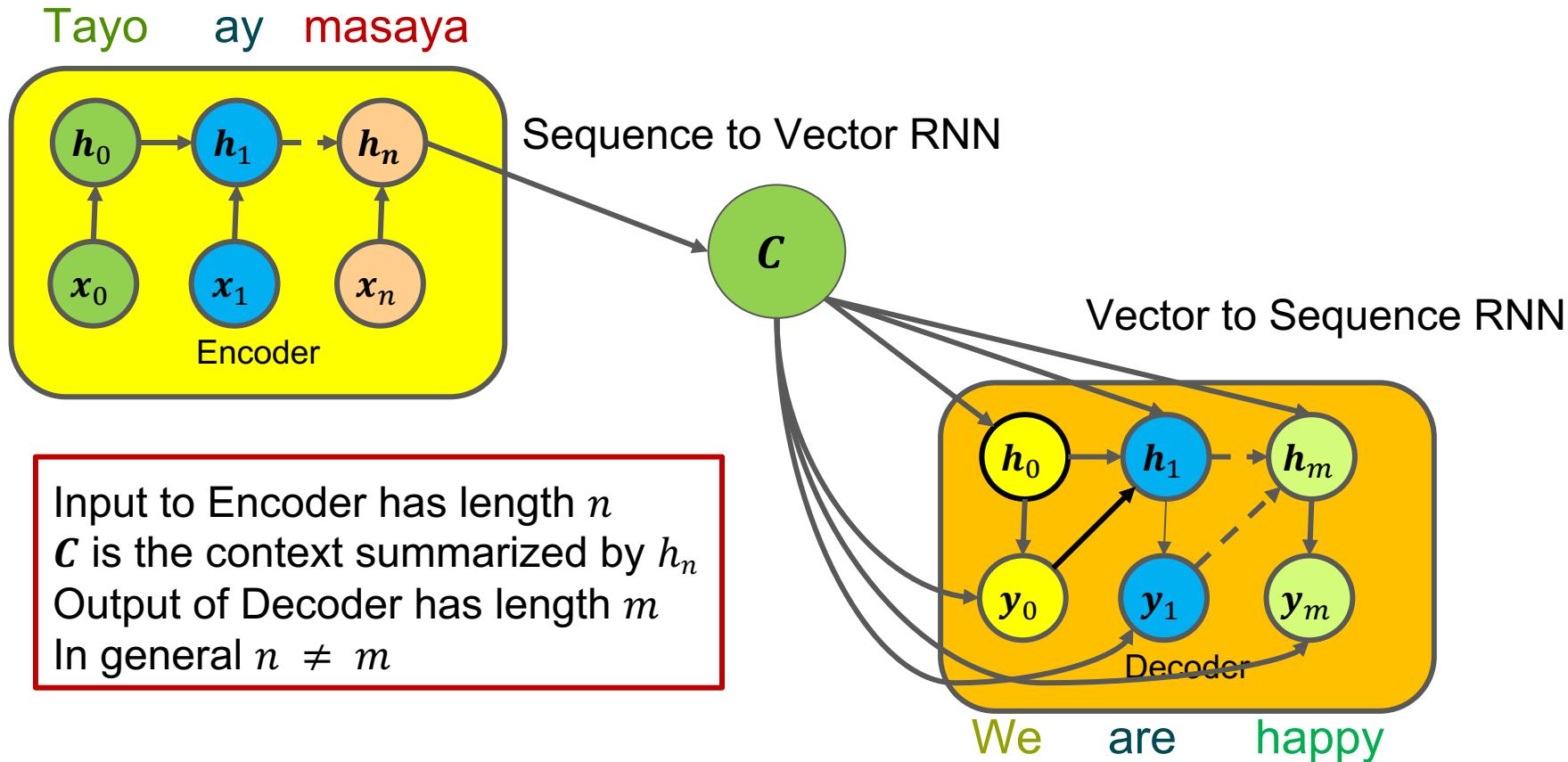
RNN prediction: *bring a golden* → **ring**

Even if RNN sees the next word is chicken, there is no way to correct ring

Bi RNN prediction: *bring a golden* → **spring** ← *chicken*

Applicable to problems where correction/disambiguation on the prediction is needed: Speech to text, Video understanding, NLU

# Encoder-Decoder Sequence-to-Sequence



# Encoder-Decoder Sequence-to-Sequence

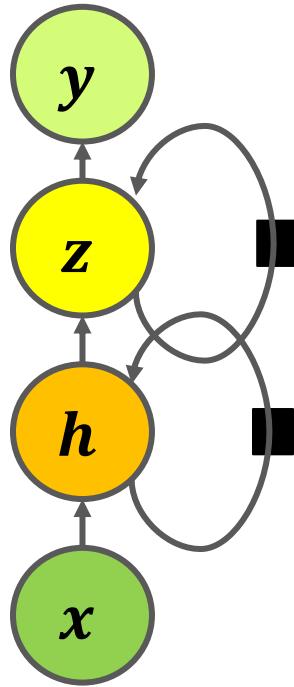
A vector,  $x$ , is encoded into context,  $C$ . The information in  $C$  is used to decode output vector,  $y$ .

Applicable in problems generating a sequence from a given sequence:

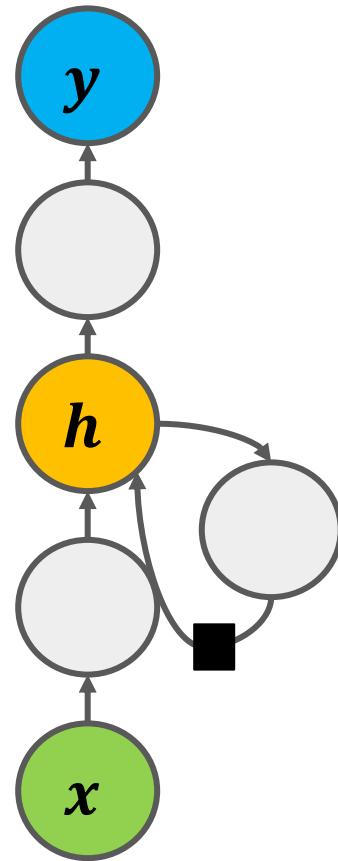
Language translation, speech to text, question-answering, etc

# Deep RNN

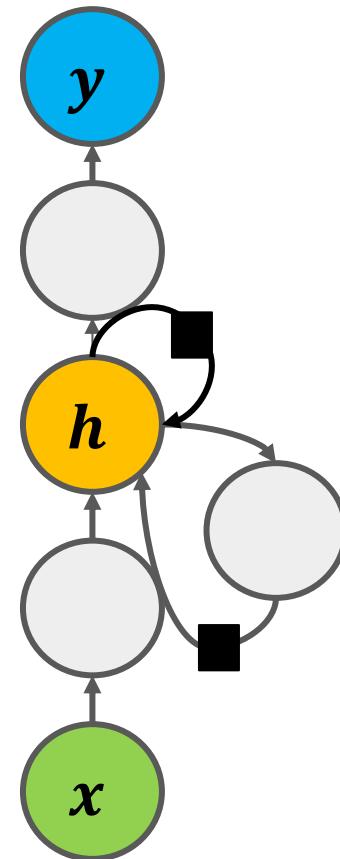
Deep Hidden Layer



MLP in input, hidden  
and output layers



Skip connection



# Problem with Long-Term Dependency

Consider:  $\mathbf{h}_t = \mathbf{W}\mathbf{h}_{t-1} = \mathbf{W}^t\mathbf{h}_0 = \mathbf{Q}^T\Lambda^t\mathbf{Q}\mathbf{h}_0$

As  $t \rightarrow \infty$ , eigenvalues  $< 1.0$  will decay to zero while those  $> 1.0$  will explode in time

# Problem with Long-Term Dependency

How to deal with long-term dependency problems

Time-delay e.g.  $t \rightarrow t/2$

Skip connections/Remove connections

Leaky Unit:  $u_t = \alpha u_{t-1} + (1 - \alpha) v_t$  where  $\alpha \in [0.0, 1.0]$ ,  $u$  is running average

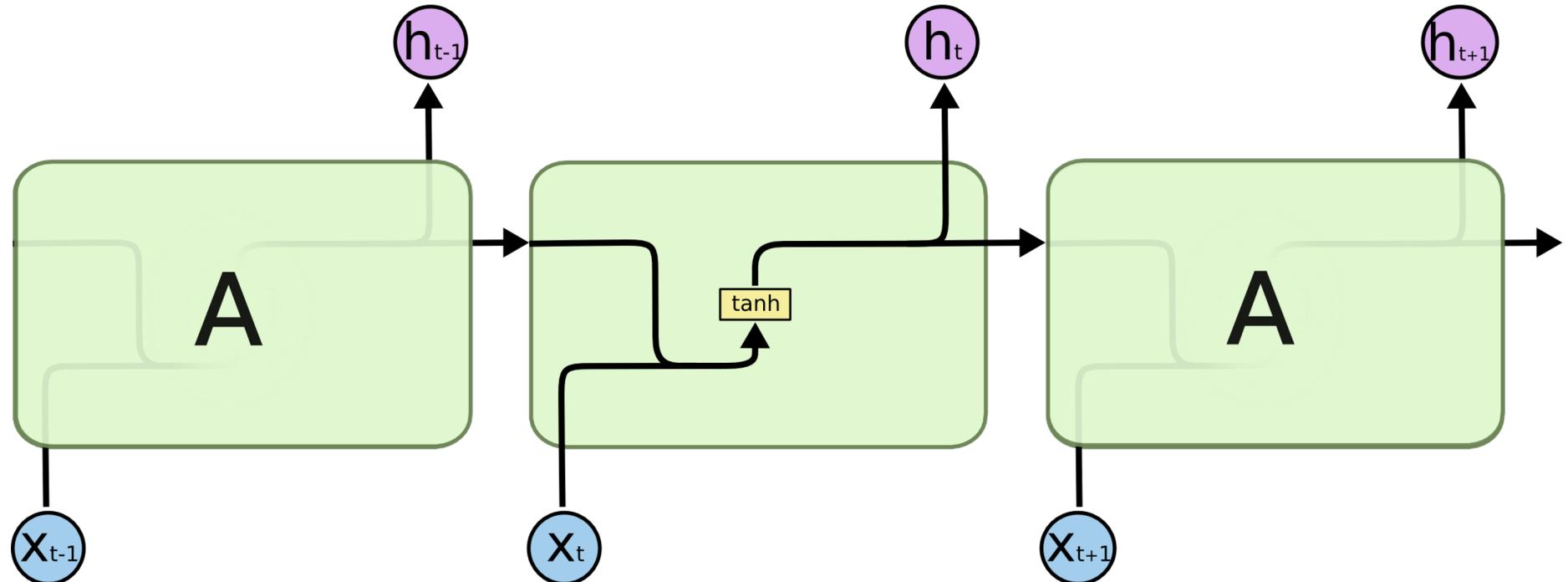
Gradient clipping

Use LSTM

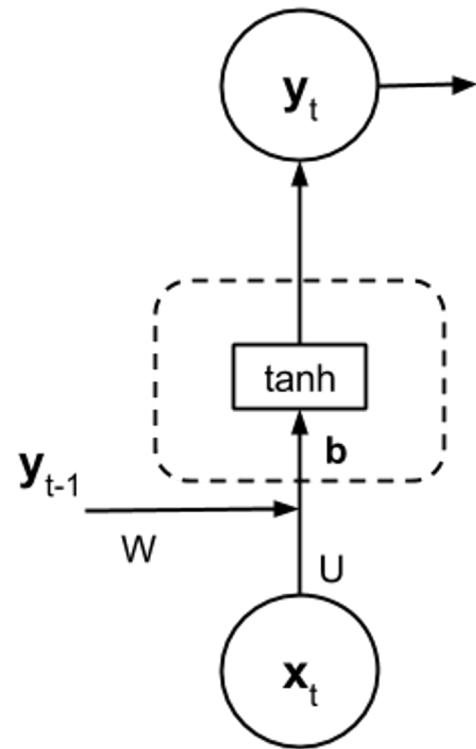
# Long Short-Term Memory (LSTM)

# RNN Cell [by Chris Olah]

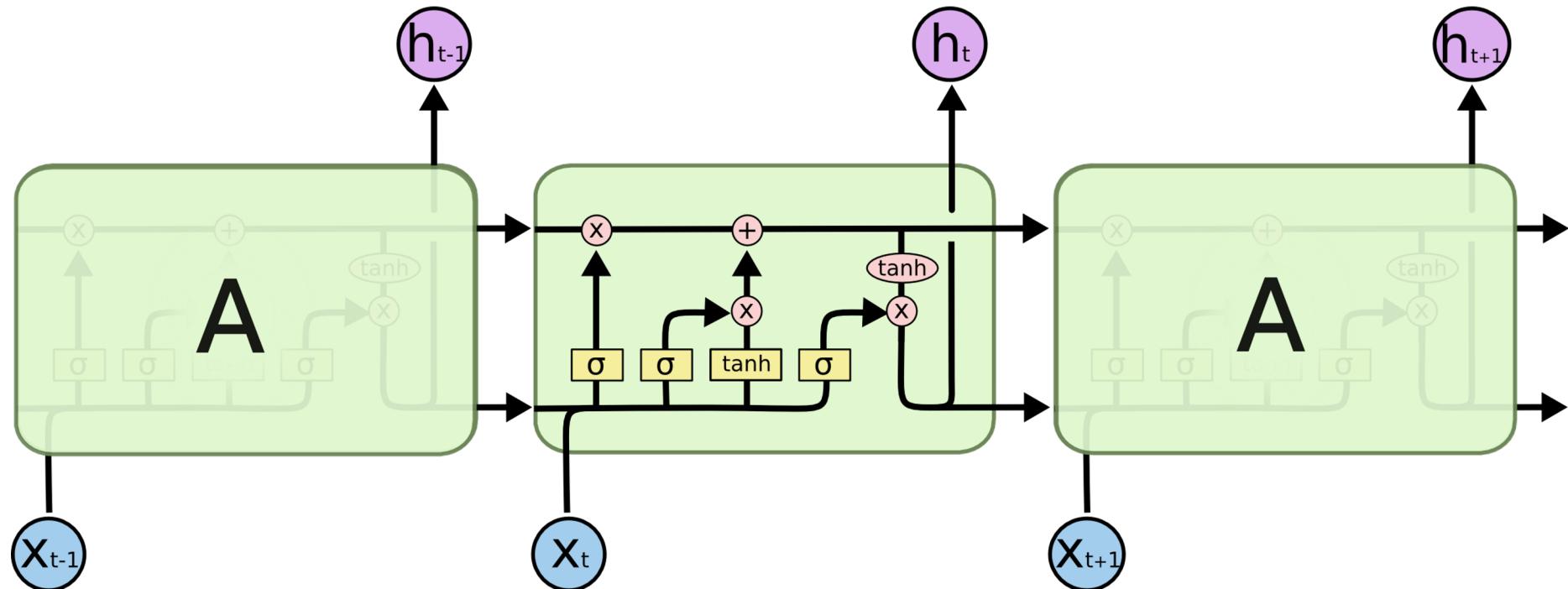
$$\begin{aligned} a_t &= b + Wh_{t-1} + Ux_t \\ h_t &= \tanh(a_t) \end{aligned}$$



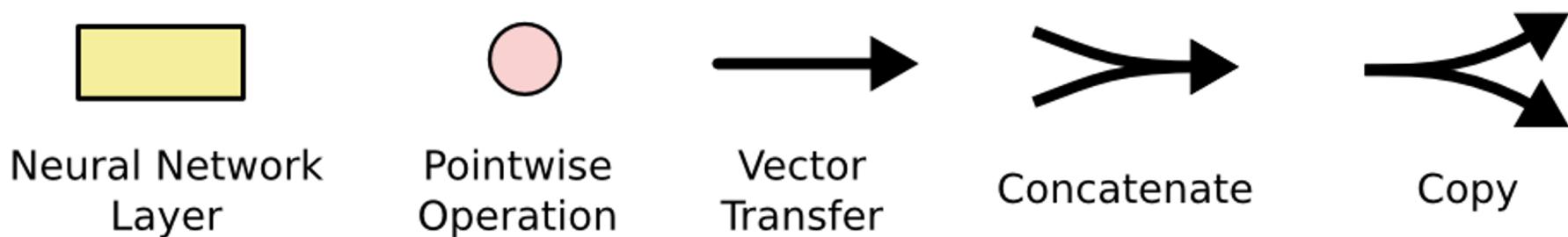
# SimpleRNN in Keras



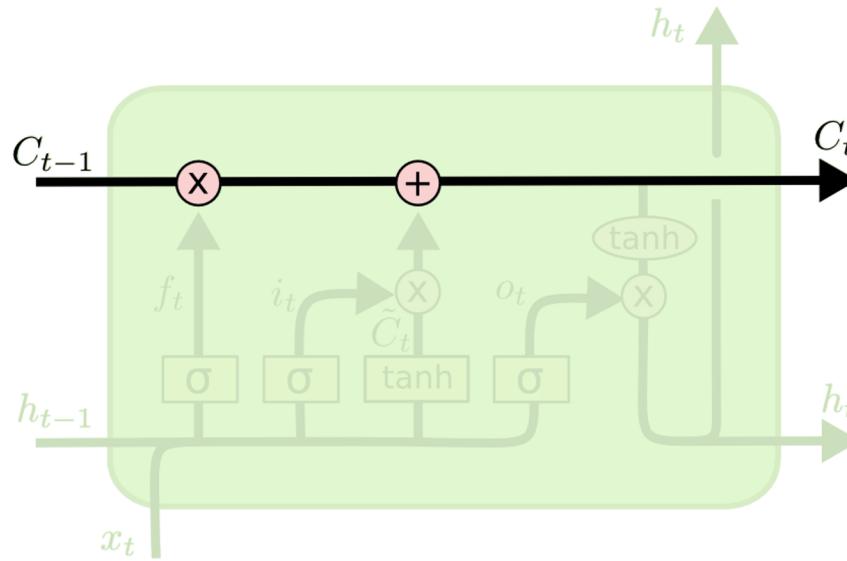
# LSTM Cell [by Chris Olah]



# LSTM Cell Operators [by Chris Olah]



# LSTM Cell State Sequence [by Chris Olah]



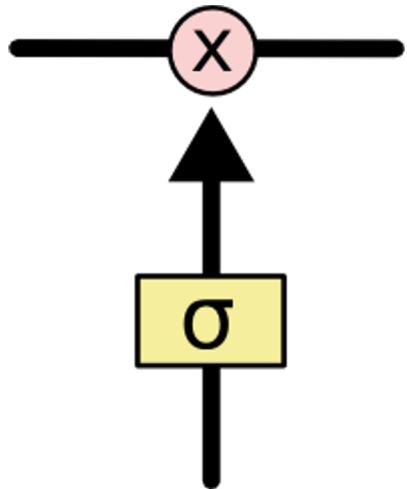
Cell states are like items in a conveyor belt

Cells states represent the summary of what the network knows so far. It is like the “context” of what we are talking about.

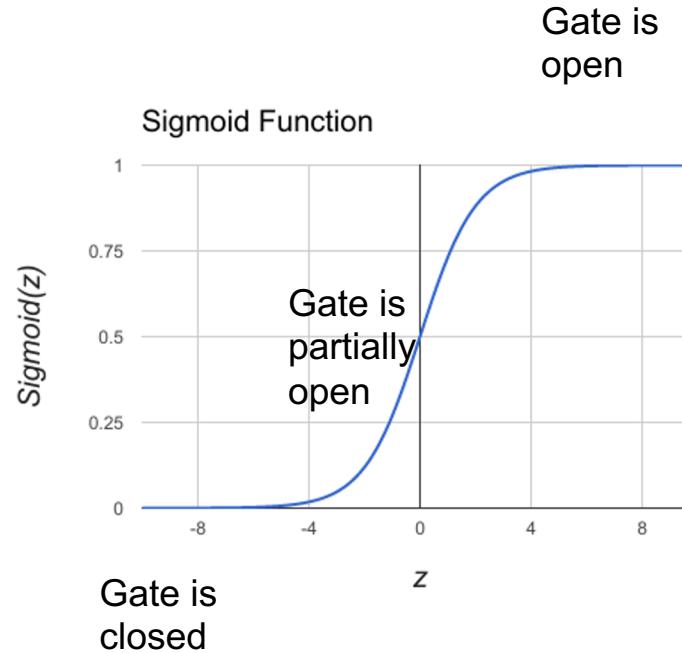
Example: History: “Maria likes summer.” Context: “A person’s opinion about season”

# LSTM Cell Sigmoid as Gate [by Chris Olah]

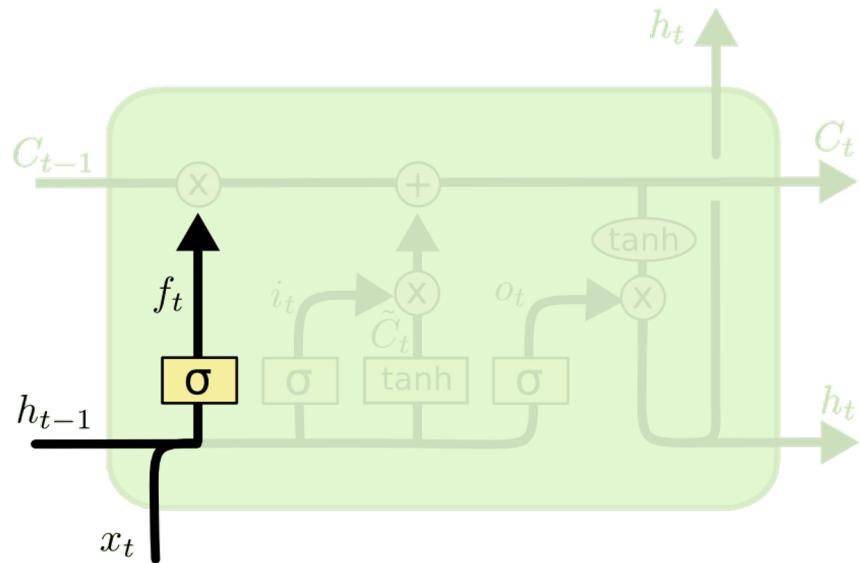
$$\sigma(z) = 1/(1+e^{-z})$$



Gate controls the amount of information to pass



# LSTM Cell Gated State [by Chris Olah]

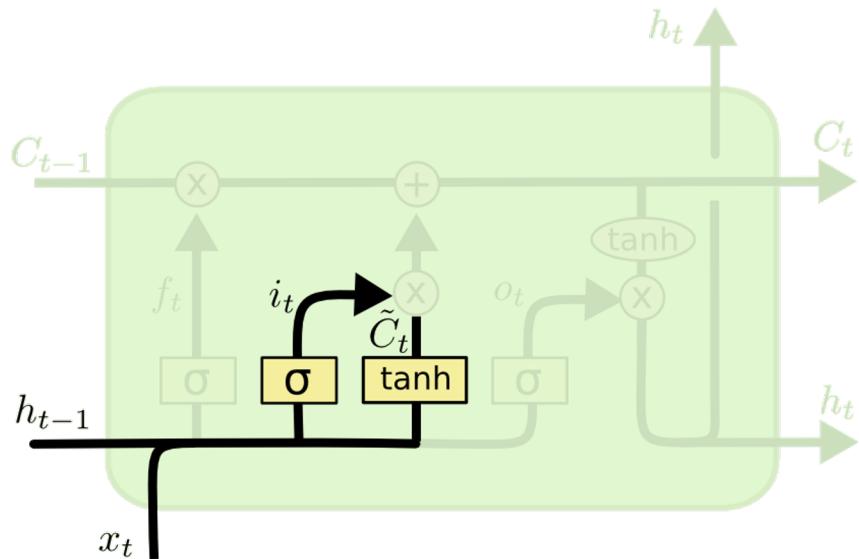


$$f_t = \sigma (W_f \cdot [h_{t-1}, x_t] + b_f)$$

Information from previous cell state is filtered by current input and previous hidden layer output (cell output) using forget gate  $f_t$ .

Example: “Maria likes summer. She enjoys the beach. Jim likes tennis.”  
Meaning: Let’s change our topic from Maria.

# LSTM Cell Candidate State Update [by Chris Olah]



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

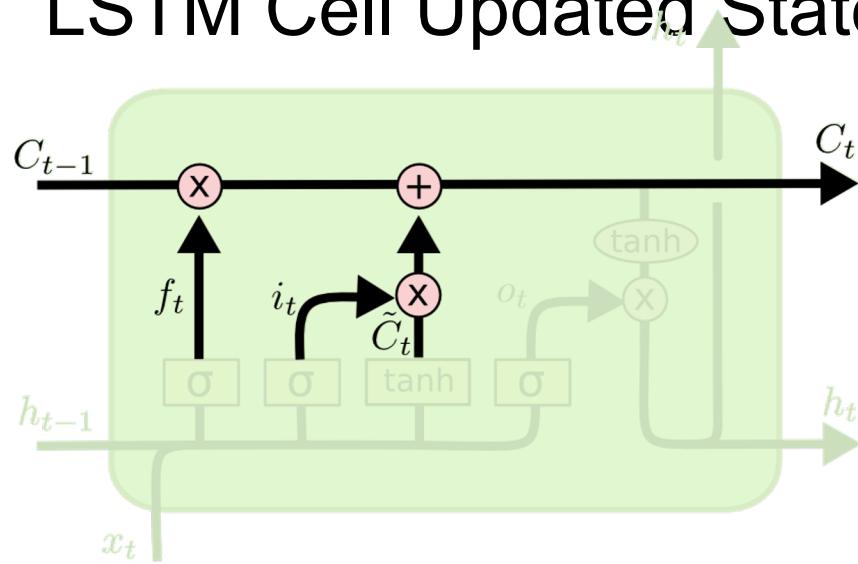
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C)$$

$\tilde{C}_t$  is the proposed update to the previous state.  $i_t$  is another forget gate that filters the proposed update. Both are controlled by present input and previous cell output

Example: “Maria likes summer. She enjoys the beach. Jim likes tennis.”

Meaning: Let’s talk about Jim now.

# LSTM Cell Updated State [by Chris Olah]



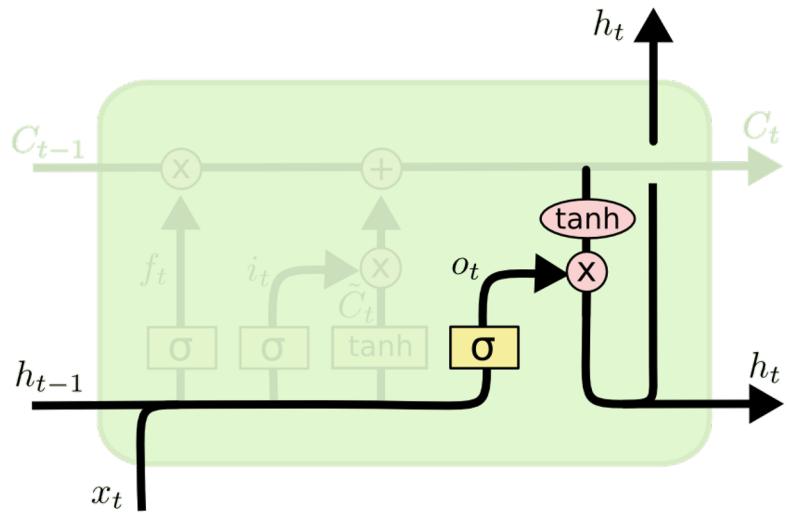
$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

The new state,  $C_t$ , is the old state with some old information forgotten plus the new proposed state with filtered information.

Example: “Maria likes summer. She enjoys the beach. Jim ...”

Meaning: Context: A person’s opinion on sports.. History is still intact. We have not forgotten what Maria likes.

# LSTM Cell Output [by Chris Olah]



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

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

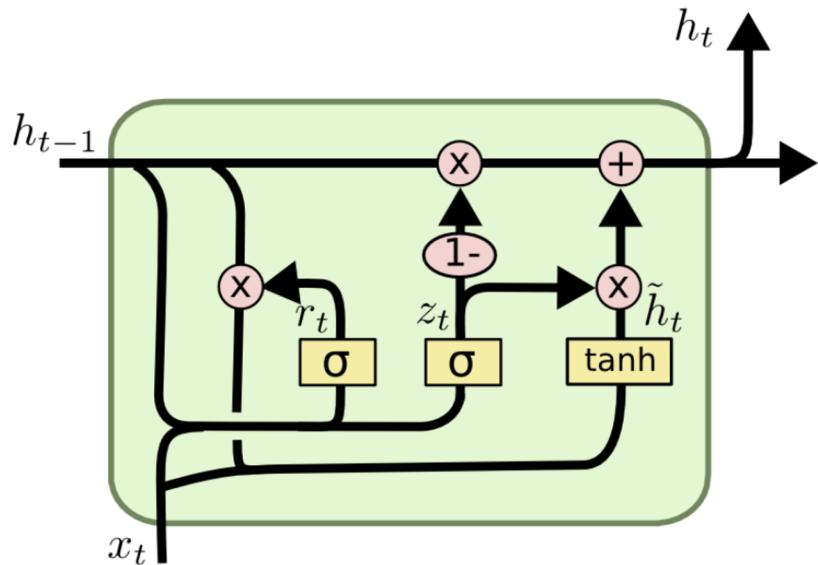
The final cell output is a squashed version of the new state (by applying non-linearity) filtered by another forget gate controlled by past cell output and new input

Example: “Maria likes summer. She enjoys the beach. Jim likes tennis.”

Meaning: If the network generates an image - A girl on a beach, it will start drawing an image of a man playing tennis near the beach.

# Variations of LSTM

# Simpler LSTM is GRU



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

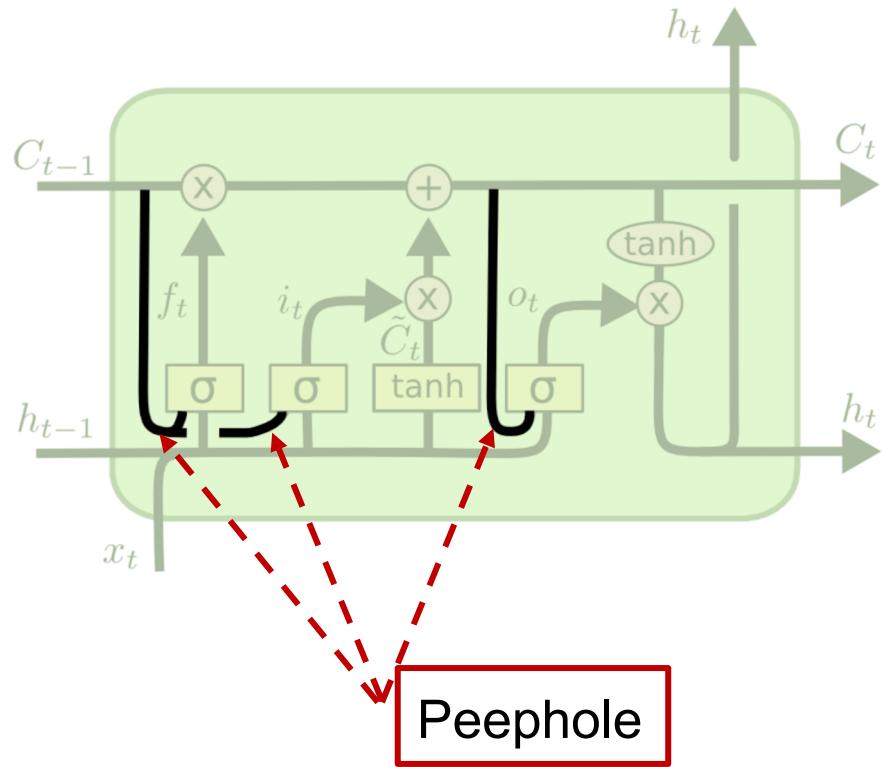
$$r_t = \sigma (W_r \cdot [h_{t-1}, x_t])$$

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

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

Cell and output states are merged. Forget and input gates perform a single update.

# LSTM with Peephole

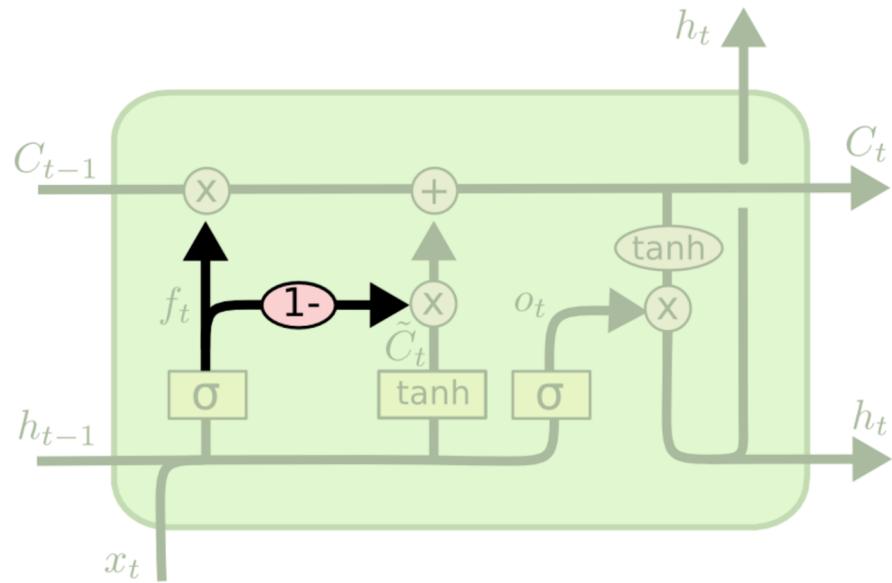


$$f_t = \sigma (W_f \cdot [C_{t-1}, h_{t-1}, x_t] + b_f)$$

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

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

# LSTM with combined forget and input gates



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

In short, the percentage of information to forget is inversely proportional to the new information to introduce

# RNN in Keras

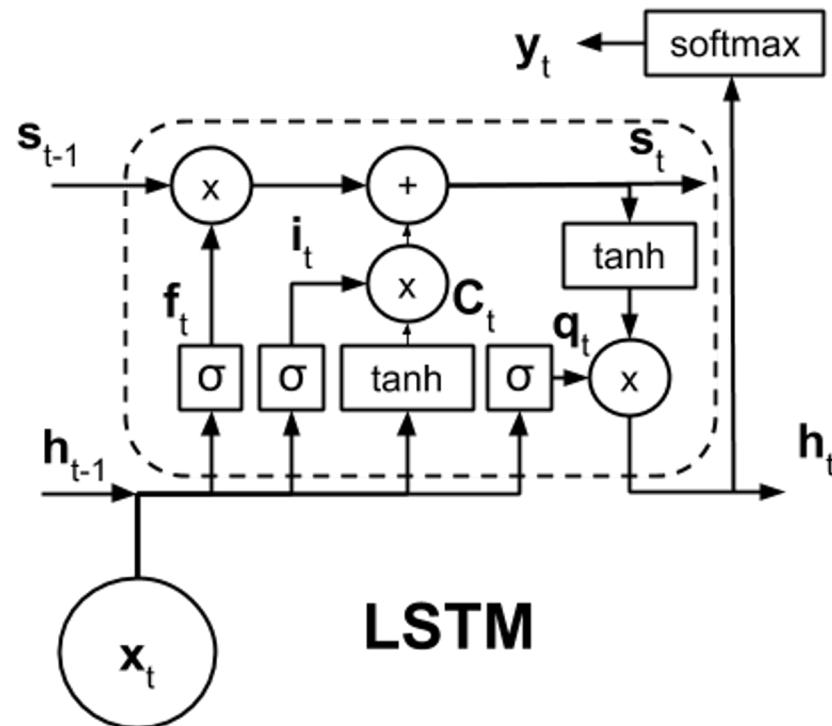
**SimpleRNN** - simplified implementation of RNN

**GRU** - GRU based on Cho's paper <https://arxiv.org/pdf/1406.1078v3.pdf>

**LSTM** - Long Short-Term Memory layer - Hochreiter 1997.

**ConvLSTM2D** - Convolutional LSTM. It is similar to an LSTM layer, but the input transformations and recurrent transformations are both convolutional.

# LSTM Cell



# Reference

Deep Learning, Ian Goodfellow and Yoshua Bengio and Aaron Courville, MIT Press, 2016, <http://www.deeplearningbook.org>

Kaparthy, A. The Unreasonable Effectiveness of RNN,  
<http://karpathy.github.io/2015/05/21/rnn-effectiveness/>

Olah, C. Understanding LSTM Networks, <http://colah.github.io/posts/2015-08-Understanding-LSTMs/>

# In Summary

RNN family excels in learning patterns in sequential data

RNN layers are inherently slow and tricks are needed to parallelize

Transformer network has emerged as a superior replacement of RNN

## seq2seq

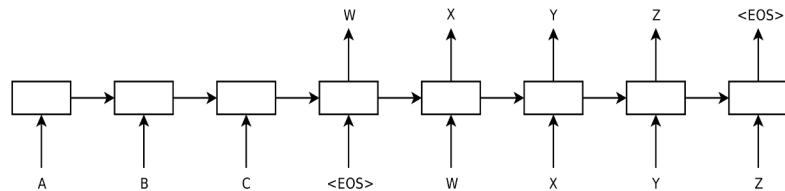


Figure 1: Our model reads an input sentence “ABC” and produces “WXYZ” as the output sentence. The model stops making predictions after outputting the end-of-sentence token. Note that the LSTM reads the input sentence in reverse, because doing so introduces many short term dependencies in the data that make the optimization problem much easier.

[2015 Sutskever et al Sequence to Sequence Learning with Neural Networks]