Recurrent Neural Networks (RNN)

Rowel Atienza
rowel@eee.upd.edu.ph
University of the Philippines
2024

Recurrent Neural Network (RNN)

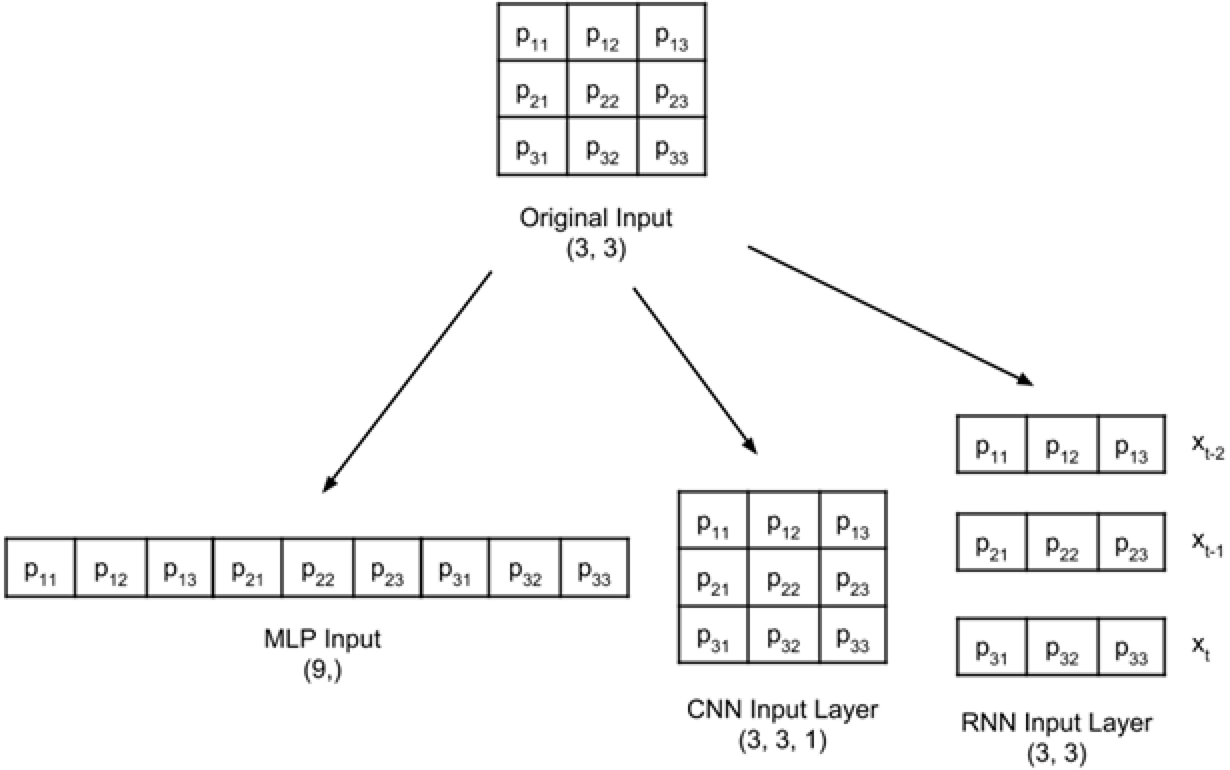
Specializes in processing sequential data: $\boldsymbol{x}_0, \boldsymbol{x}_1, .. \boldsymbol{x}_t$

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

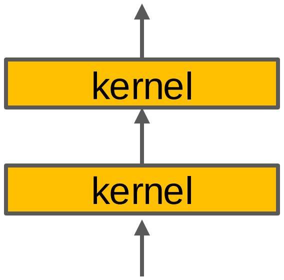


Recurrent Neural Network

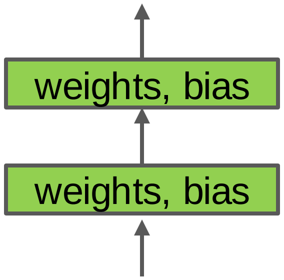
One difference between RNN and CNN and MLP is parameters to be learned are shared among inputs

$\boldsymbol{x}_0, \boldsymbol{x}_1, .. \boldsymbol{x}_t$ share the same set of parameters over the sequence

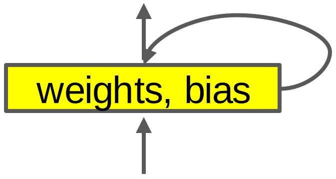
CNN and MLP share the same set of parameter over the entire input



CNN



MLP



RNN

Recurrent Neural Network

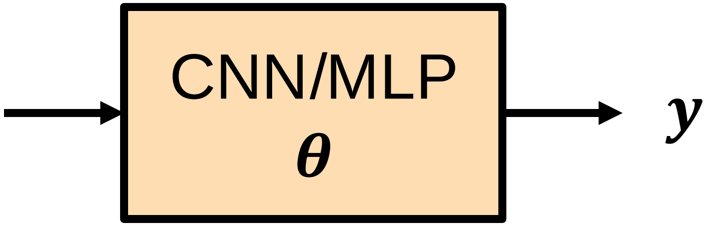
Given a sequential data: $\boldsymbol{x} = \{\boldsymbol{x}_0, \boldsymbol{x}_1, .. \boldsymbol{x}_t\}$

In MLP and CNN, $\boldsymbol{x} = \{\boldsymbol{x}_0, \boldsymbol{x}_1, .. \boldsymbol{x}_t\}$ in treated as an independent input whose features needed to be learned

In RNN, sequential inputs are dependent on one another, $\boldsymbol{x}_0, \boldsymbol{x}_1, .. \boldsymbol{x}_t$

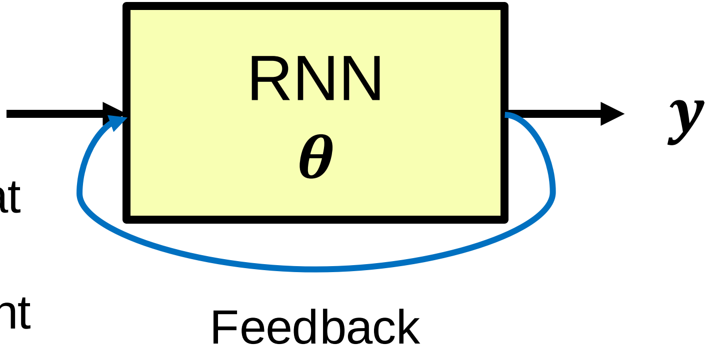
For example:
$\boldsymbol{x}$ is an image made of rows of pixels $\boldsymbol{x}_i$

$$\boldsymbol{x} = \{\boldsymbol{x}_0, \boldsymbol{x}_1, ..\boldsymbol{x}_t\}$$



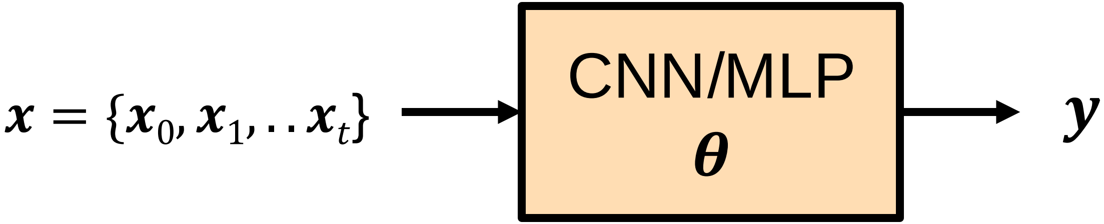
Inputs are processed as one
Inputs can be shuffled; Order not important

$$\boldsymbol{x}_0, \boldsymbol{x}_1, ..\boldsymbol{x}_t$$

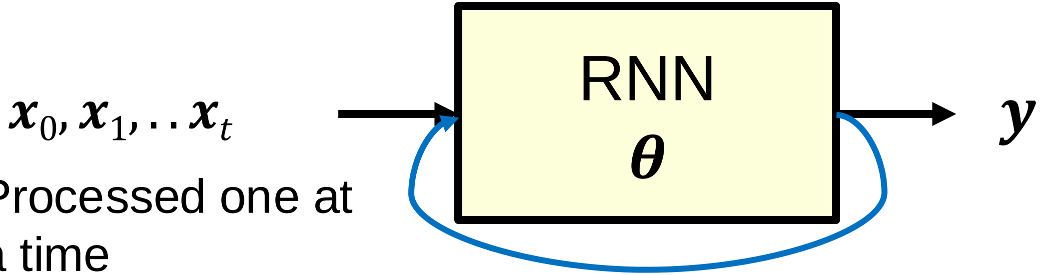


Processed one at a time
Order is important

$$\boldsymbol{y} = f(\boldsymbol{x}) \approx f_n(\cdot; \boldsymbol{\theta}_n) \circ f_{n-1}(\cdot; \boldsymbol{\theta}_{n-1}) \circ \cdots \circ f_1(\boldsymbol{x}; \boldsymbol{\theta}_1)$$



$$\boldsymbol{y}_i = f(\boldsymbol{s}_i, \boldsymbol{x}_i; \boldsymbol{\theta}) \quad for\ i = 1, \ldots, t$$



Processed one at a time
Order is important

Recurrent Neural Network - Unfolding

Consider a dynamical system with states: $\boldsymbol{s}_0, \boldsymbol{s}_1, \ldots, \boldsymbol{s}_t$

$$\boldsymbol{s}_t = f(\boldsymbol{s}_{t-1}; \boldsymbol{\theta})$$

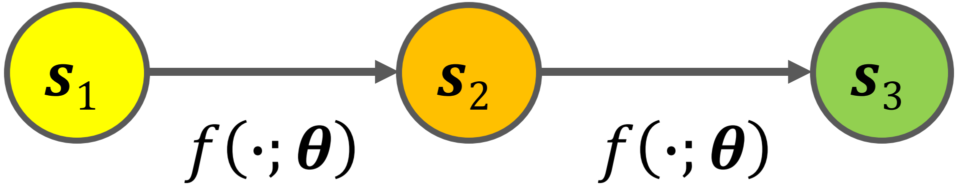
$\boldsymbol{\theta}$ are the parameters

For example:

$$\boldsymbol{s}_3 = f(\boldsymbol{s}_2; \boldsymbol{\theta}) = f(f(\boldsymbol{s}_1; \boldsymbol{\theta}); \boldsymbol{\theta})$$

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: $\boldsymbol{x}_t$

$$\boldsymbol{s}_t = f(\boldsymbol{s}_{t-1}, \boldsymbol{x}_t; \boldsymbol{\theta})$$

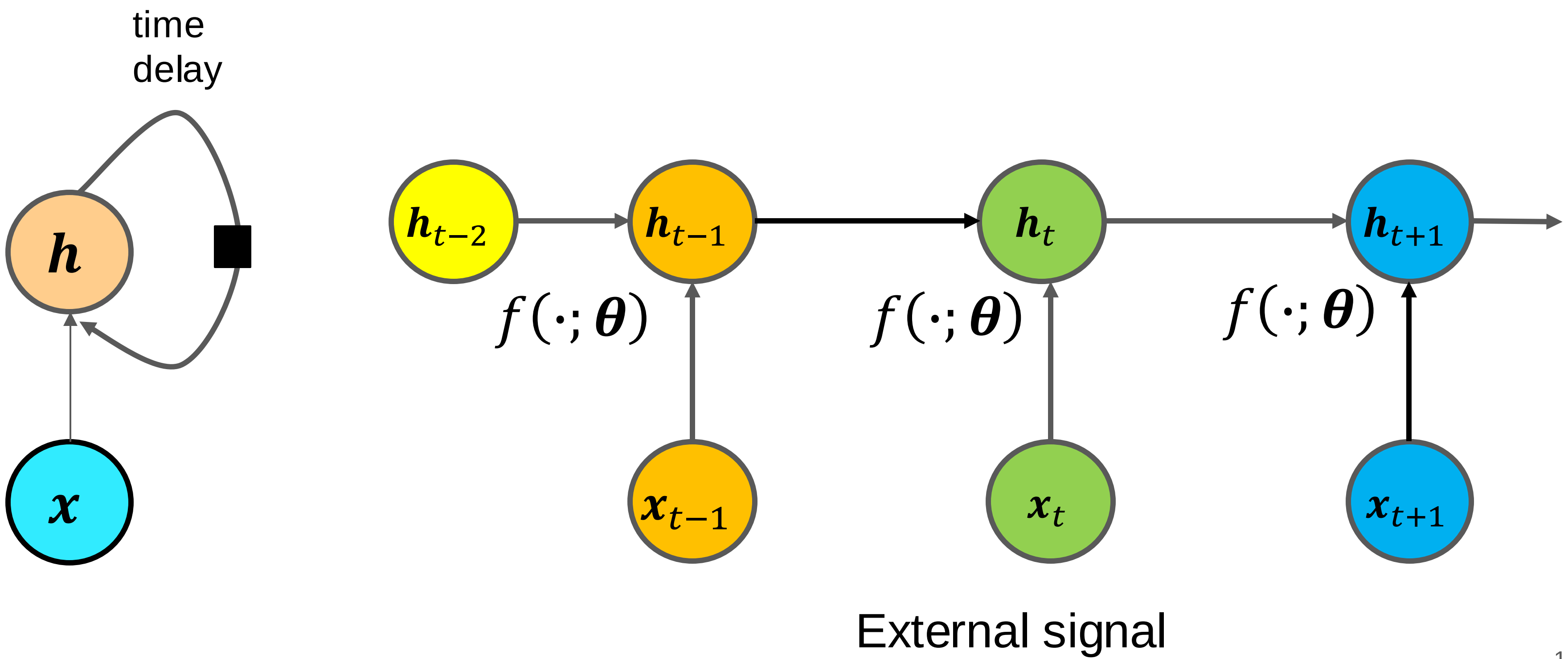
If we use a neural network to store the state, a hidden layer $\boldsymbol{h}_t$ can be modeled as:

$$\boldsymbol{h}_t = f(\boldsymbol{h}_{t-1}, \boldsymbol{x}_t; \boldsymbol{\theta})$$

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

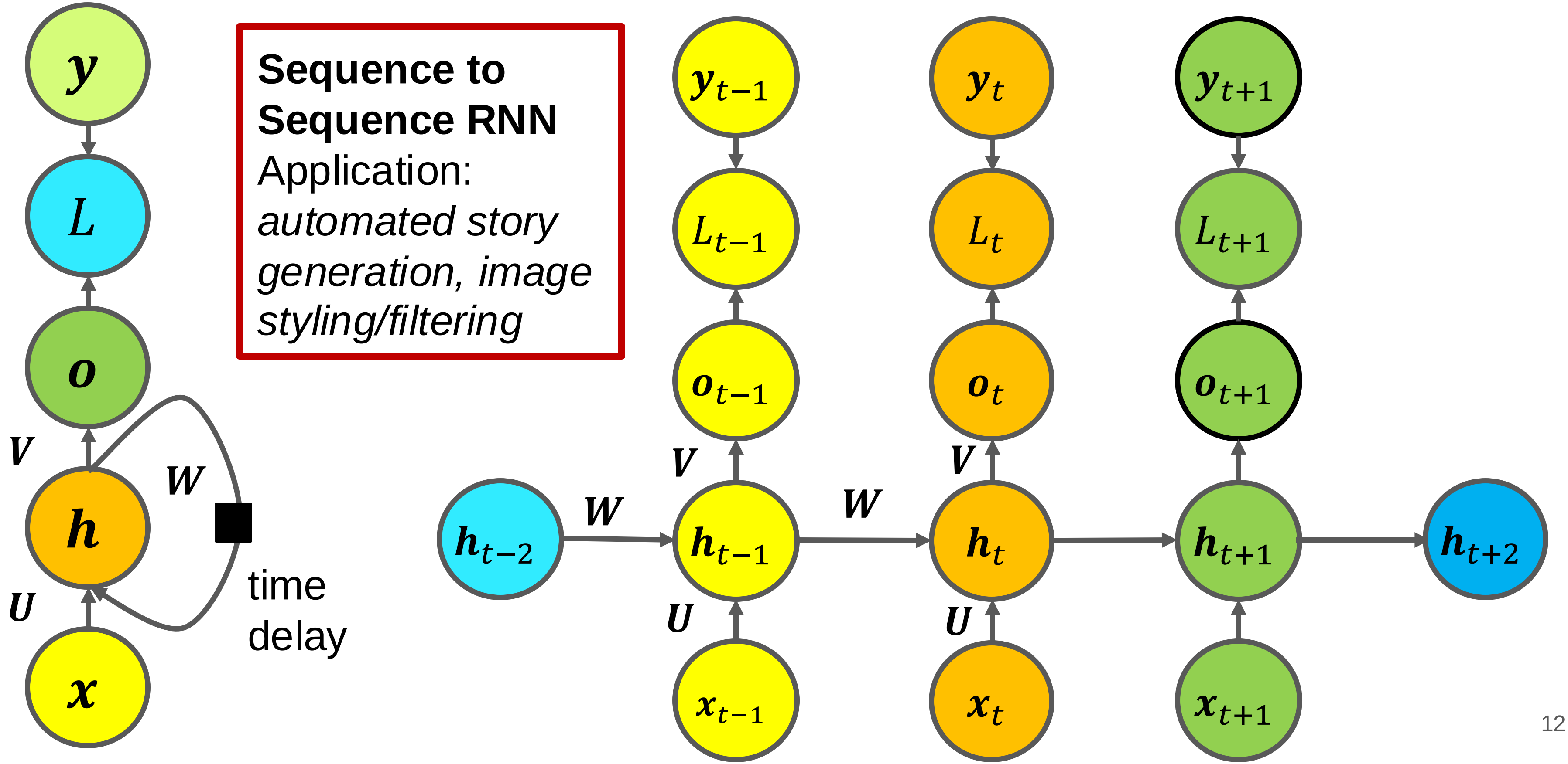
$\boldsymbol{h}_t$ encodes (lossy) all previous inputs $\boldsymbol{x}_0, \boldsymbol{x}_1, \ldots, \boldsymbol{x}_t$

Unfolding of RNN (Graphical Model)



seq2seq RNN - Recurrence between hidden layers

Sequence to Sequence RNN
Application: *automated story generation, image styling/filtering*



seq2seq RNN - Recurrence between hidden layers

$$\boldsymbol{a}_t = \boldsymbol{b} + \boldsymbol{W}\boldsymbol{h}_{t-1} + \boldsymbol{U}\boldsymbol{x}_t$$
$$\boldsymbol{h}_t = \tanh(\boldsymbol{a}_t)$$
$$\boldsymbol{o}_t = \boldsymbol{c} + \boldsymbol{V}\boldsymbol{h}_t$$
$$L_t = -\log p(\boldsymbol{y}_t | \{\boldsymbol{x}_1, \boldsymbol{x}_2, \dots, \boldsymbol{x}_t\})$$
$$\hat{\boldsymbol{y}}_t = softmax(\boldsymbol{o}_t)$$

$\boldsymbol{\theta} =$ $\boldsymbol{W}$, $\boldsymbol{U}$ and $\boldsymbol{V}$ are parameter matrices + $\boldsymbol{b}$ and $\boldsymbol{c}$ are biases

In Pytorch, RNN implementation is slightly different. See:

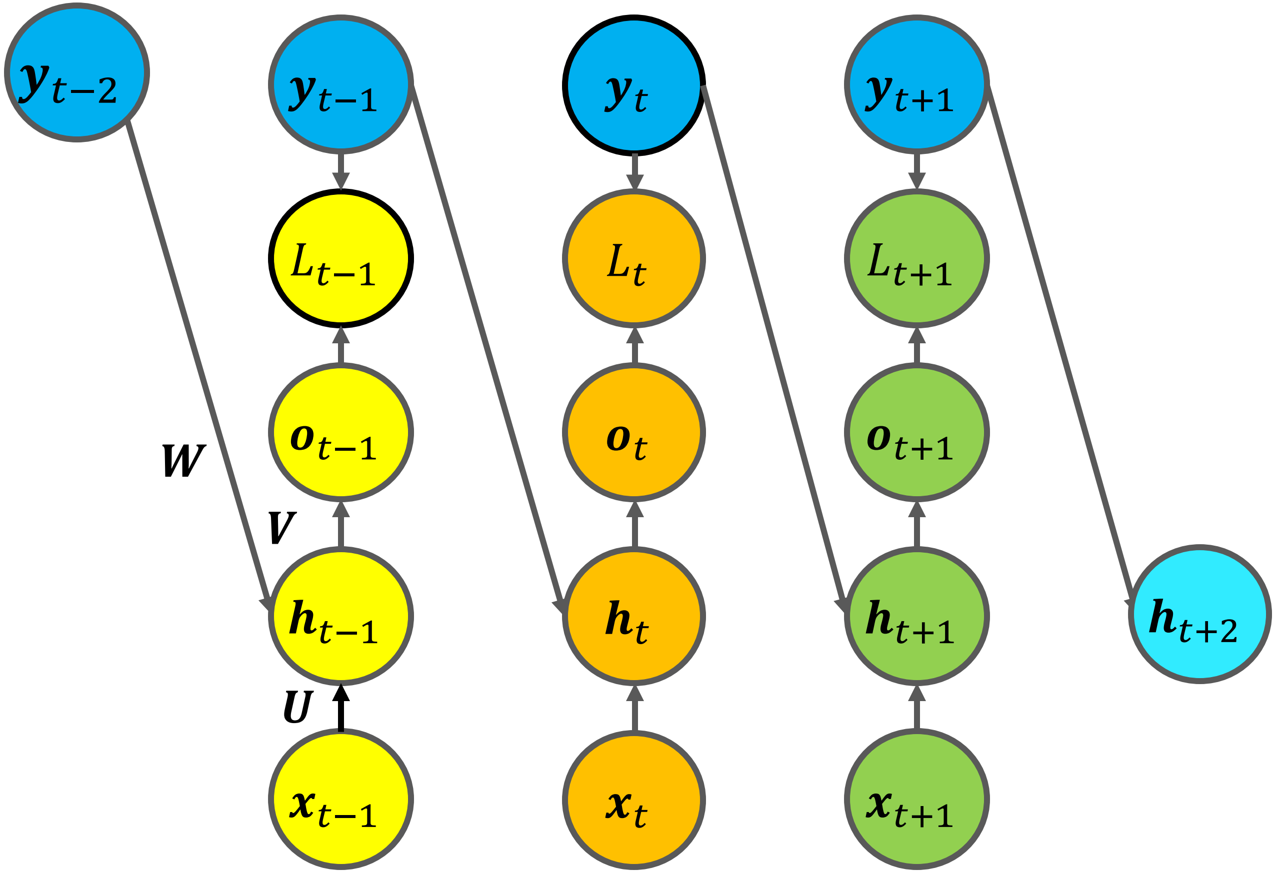
https://pytorch.org/docs/stable/generated/torch.nn.RNN.html

RNN maximizes the conditional probability to minimize L

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

By nature, RNN is not parallelizable

seq2seq RNN - Teacher forcing (Parallelizable RNN)



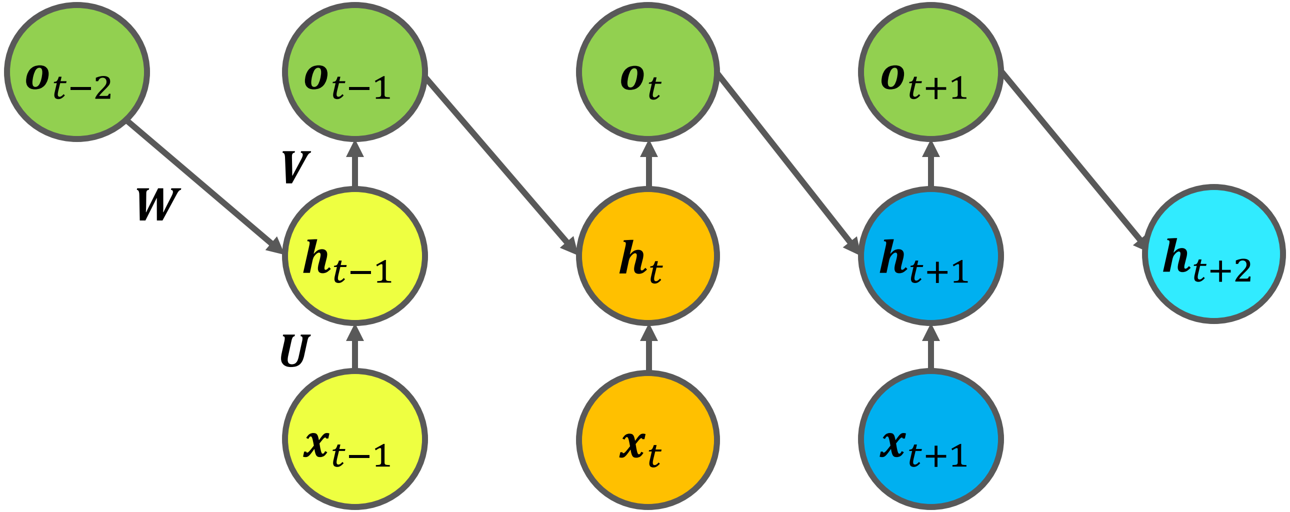
Teacher forcing maximizes the conditional probability to minimize L

$$L_t = -\log p(\boldsymbol{y}_t, \boldsymbol{y}_{t-1} | \{\boldsymbol{x}_1, \boldsymbol{x}_2, \dots, \boldsymbol{x}_t\})$$

By Bayes Theorem

$$L_t = -\log p(\boldsymbol{y}_t | \{\boldsymbol{x}_1, \boldsymbol{x}_2, \dots, \boldsymbol{x}_t, \boldsymbol{y}_{t-1}\}) - \log p(\boldsymbol{y}_{t-1} | \{\boldsymbol{x}_1, \boldsymbol{x}_2, \dots, \boldsymbol{x}_t\})$$

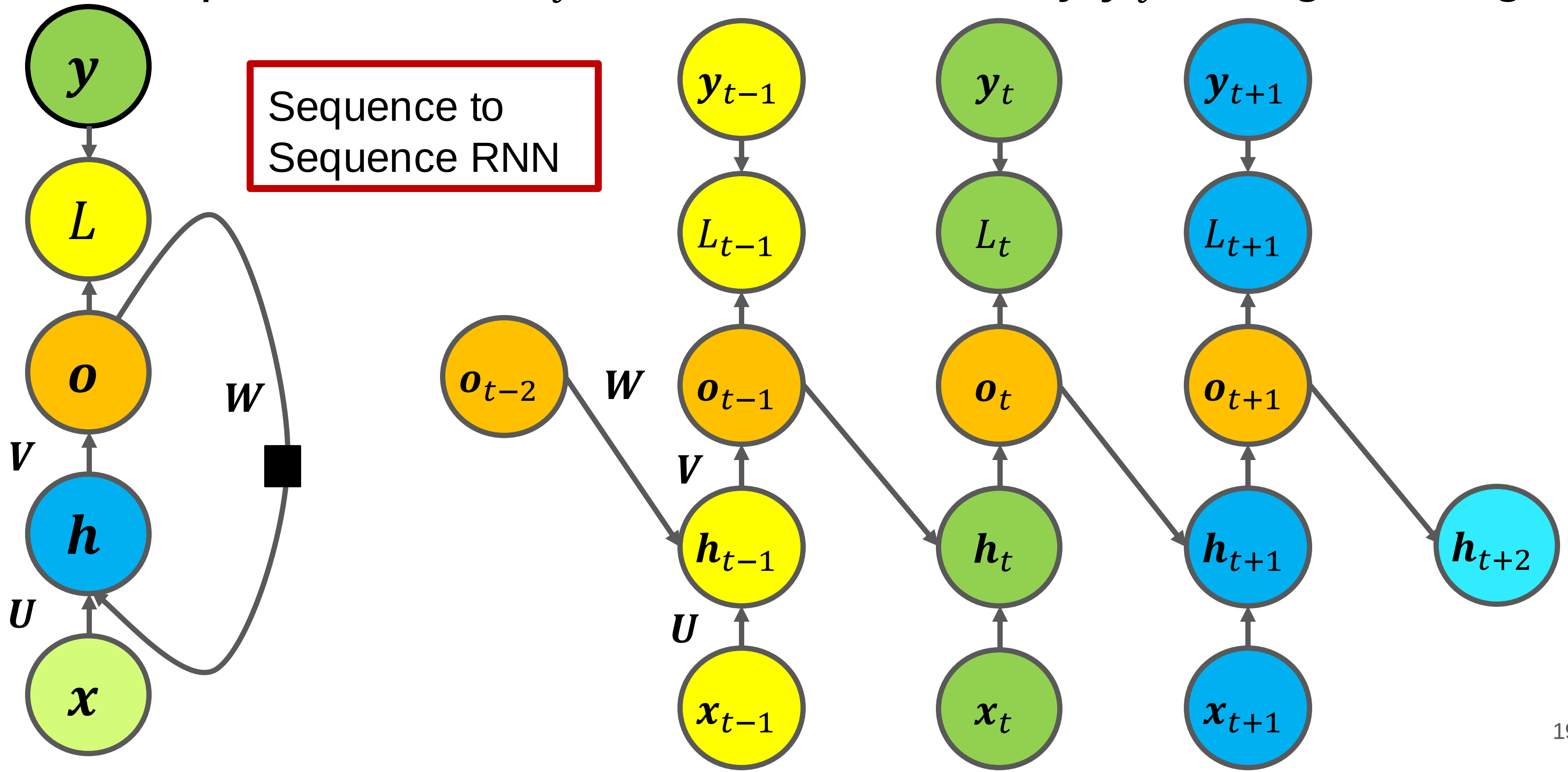
seq2seq RNN - Teacher forcing (testing)



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

Teacher forcing - Recurrence between output & hidden layers

Can be parallelized! $\boldsymbol{o}_t$ can be estimated by $\boldsymbol{y}_t$ during training



Teaching forcing RNN - Recurrence between output & hidden layers

$$\boldsymbol{a}_t = \boldsymbol{b} + \boldsymbol{W}\boldsymbol{h}_t + \boldsymbol{U}\boldsymbol{x}_t$$
$$\boldsymbol{h}_t = \tanh(\boldsymbol{a}_t)$$
$$\boldsymbol{o}_t = \boldsymbol{c} + \boldsymbol{V}\boldsymbol{h}_t$$
$$L = -\log p(\boldsymbol{y}_t | \{\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_t\})$$
$$\boldsymbol{y}_t = softmax(\boldsymbol{o}_t)$$

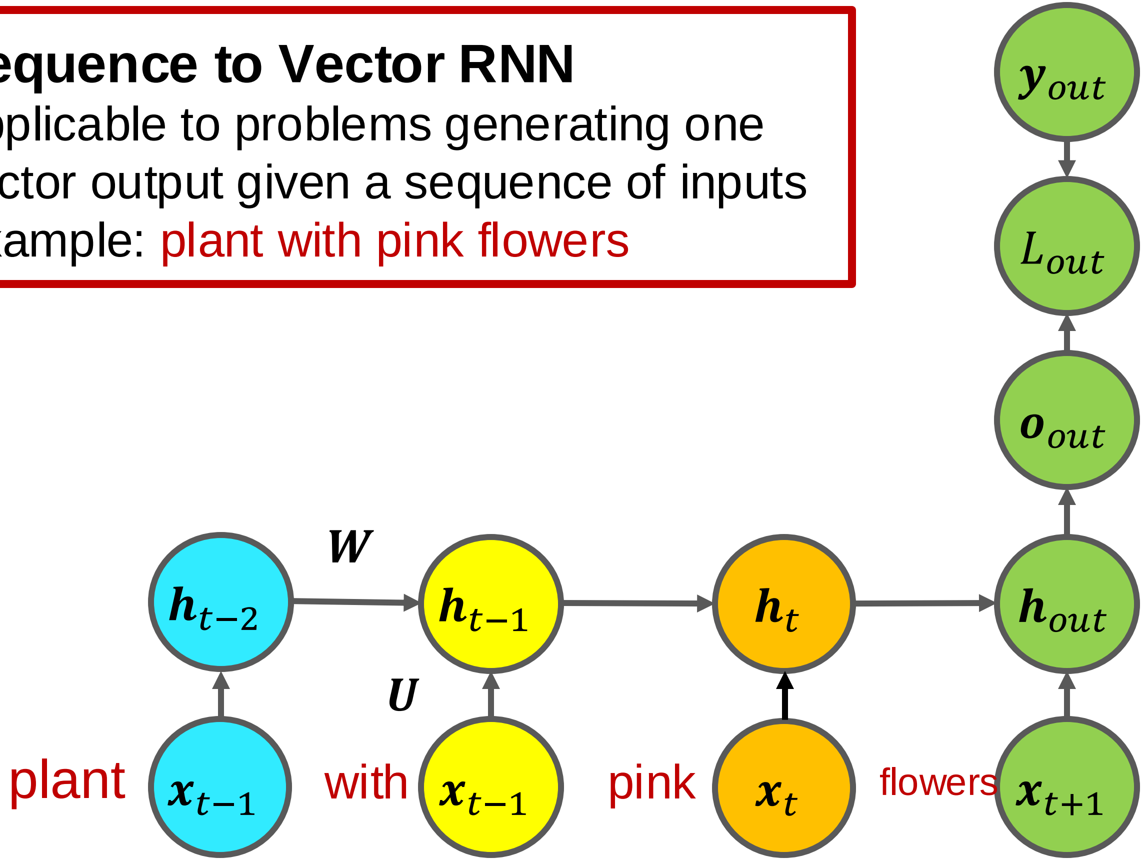
$\boldsymbol{W}$, $\boldsymbol{U}$ and $\boldsymbol{V}$ are parameter matrices

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

Exercise: Modify the equation for the preceding figure

seq2vec RNN

Sequence to Vector RNN
Applicable to problems generating one vector output given a sequence of inputs
Example: plant with pink flowers



y_{out}

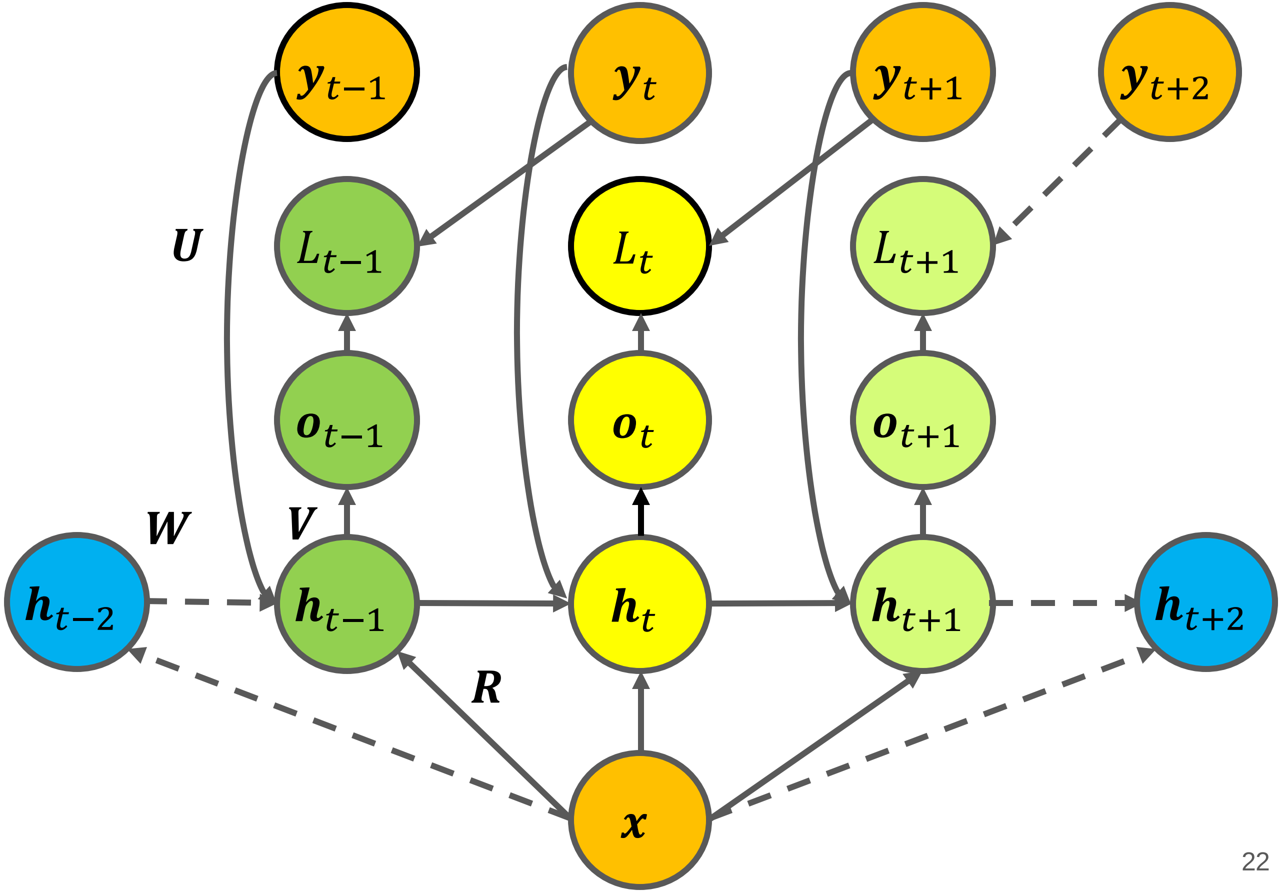
Exercise:
Write the equations for this network

vec2seq RNN

$\boldsymbol{y}_0$ $\boldsymbol{y}_1$ $\boldsymbol{y}_2$ $\boldsymbol{y}_1$

A ripe calamansi fruit

$\boldsymbol{x}$



vec2seq RNN

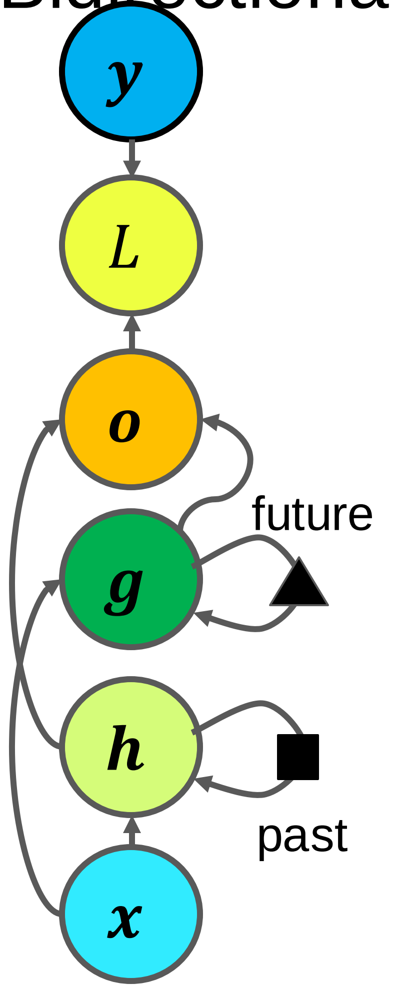
Applicable to problems converting fixed data to a variable length sequence

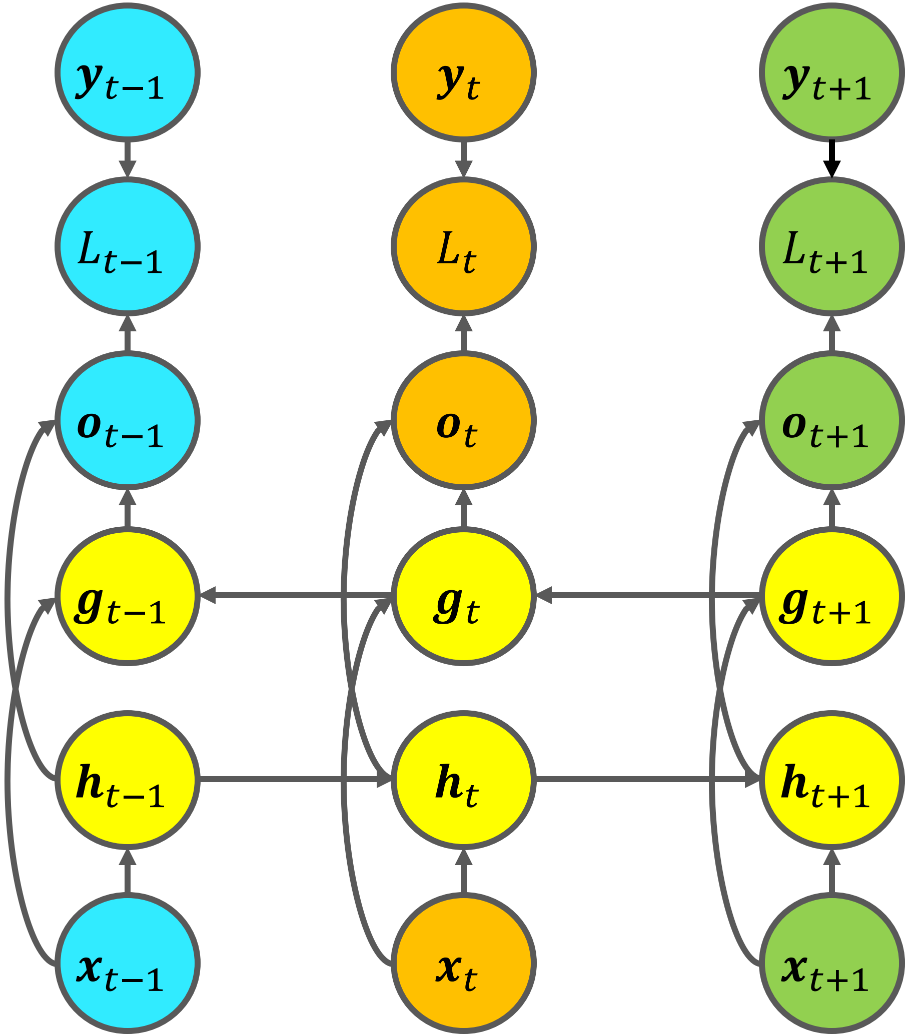
Image (fixed) → Caption (variable)

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

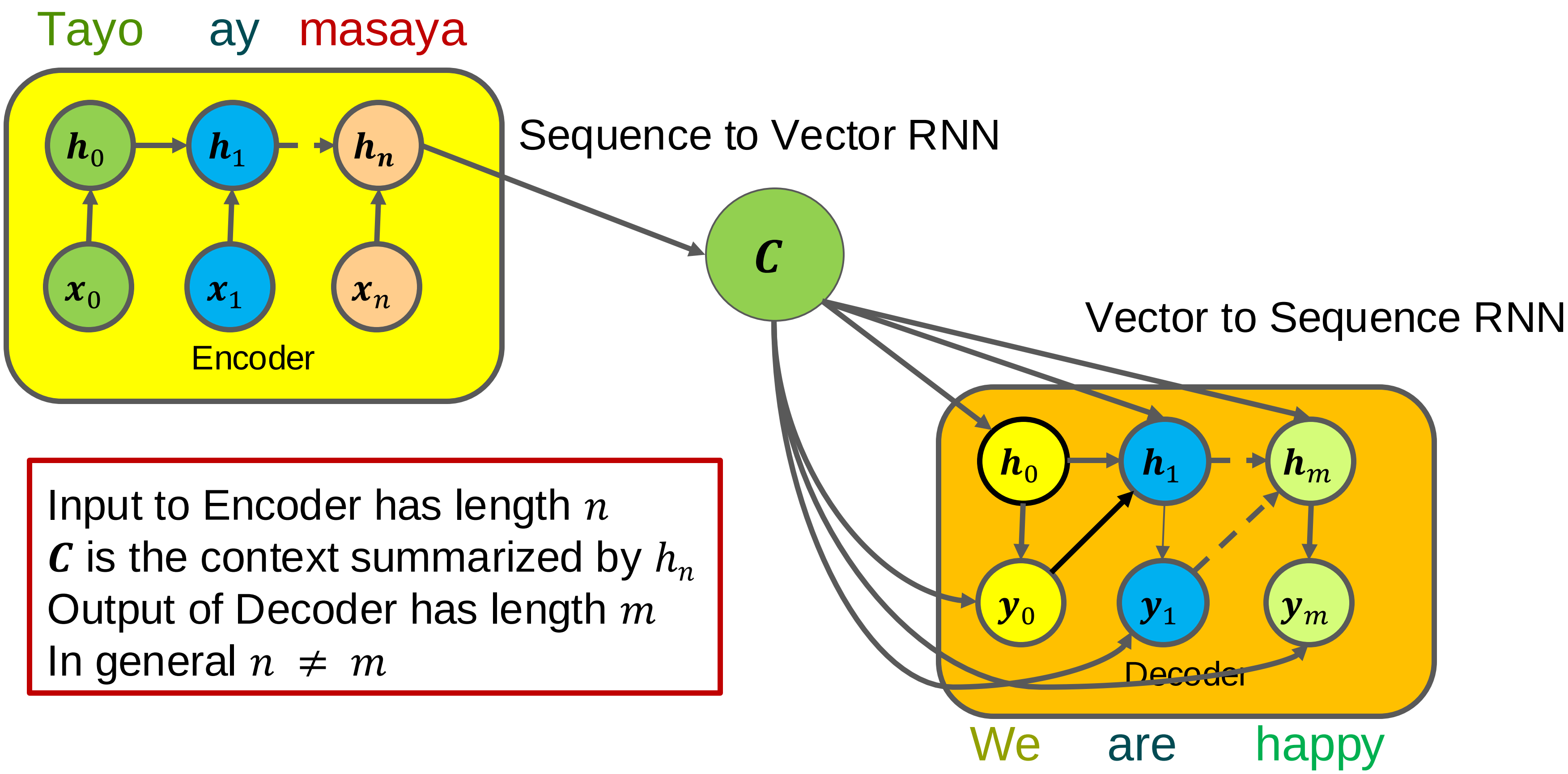
RNN prediction: *bring a golden* → ring

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



Input to Encoder has length n
$\boldsymbol{C}$ is the context summarized by h_n
Output of Decoder has length m
In general $n \neq m$

Encoder-Decoder Sequence-to-Sequence

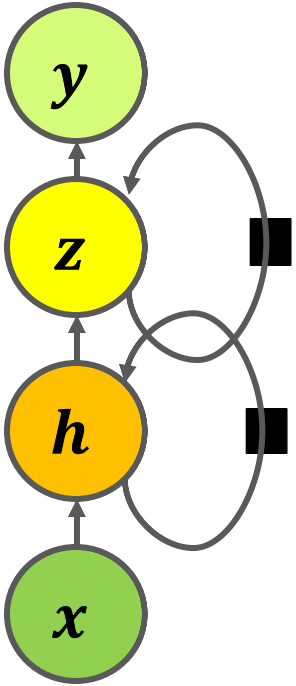
A vector, $\boldsymbol{x}$, is encoded into context, $\boldsymbol{C}$. The information in $\boldsymbol{C}$ is used to decode output vector, $\boldsymbol{y}$.

Applicable in problems generating a sequence from a given sequence:

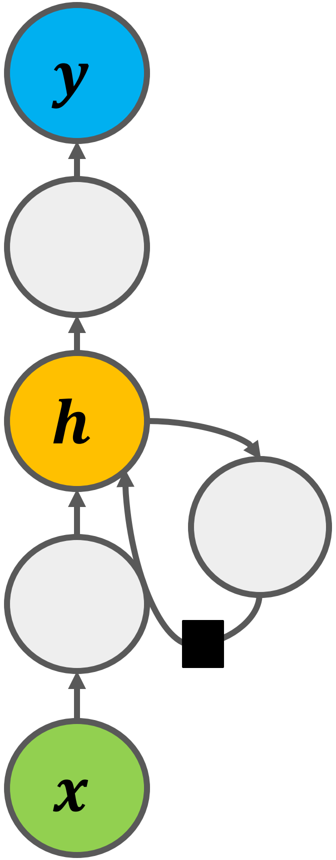
Language translation, speech to text, question-answering, etc

Deep RNN

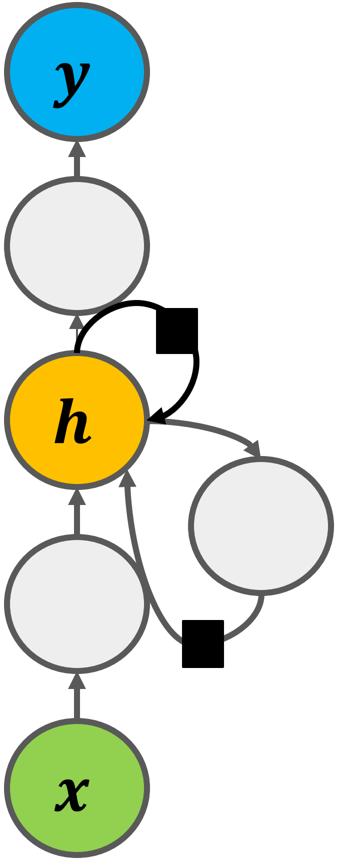
Deep Hidden Layer



MLP in input, hidden and output layers



Skip connection



Problem with Long-Term Dependency

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

As $t \to \infty$, 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. $t \rightarrow t/2$

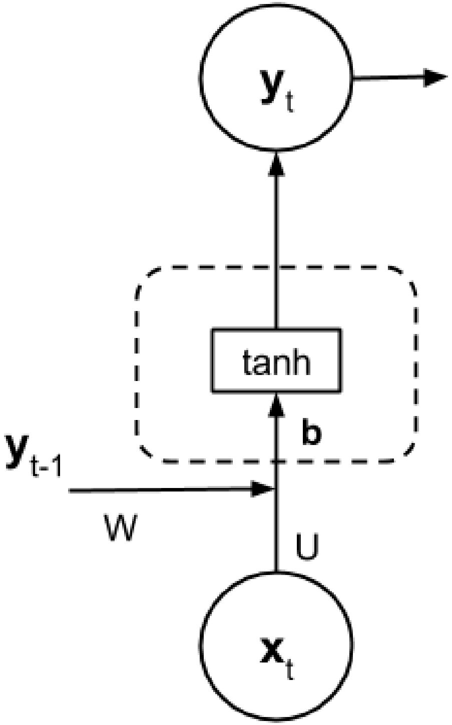
Skip connections/Remove connections

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

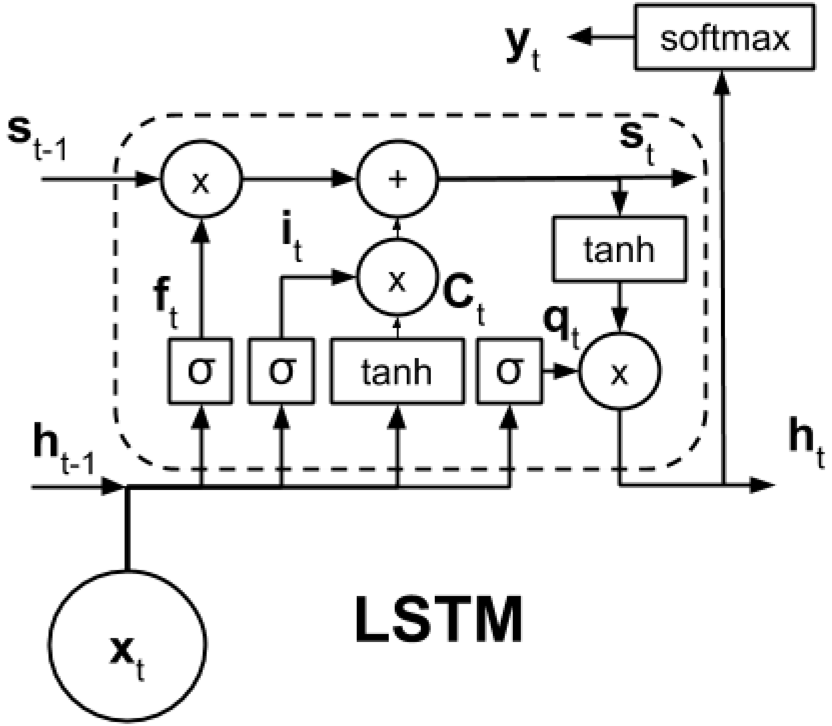
Gradient clipping

Use LSTM

RNN Cell



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

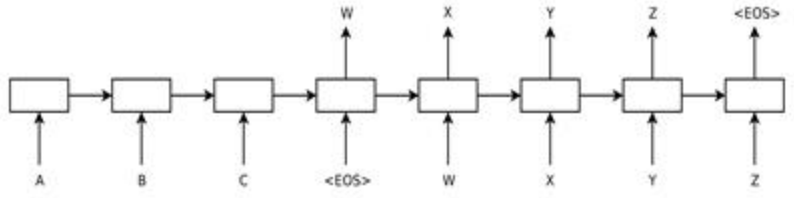


Figure 1: Our model reads an input sentence "ABC" and produces "WXYZ" as the output sentence. The model stops making predictions after outputting the end-of-sentence token. Note that the LSTM reads the input sentence in reverse, because doing so introduces many short term dependencies in the data that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

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