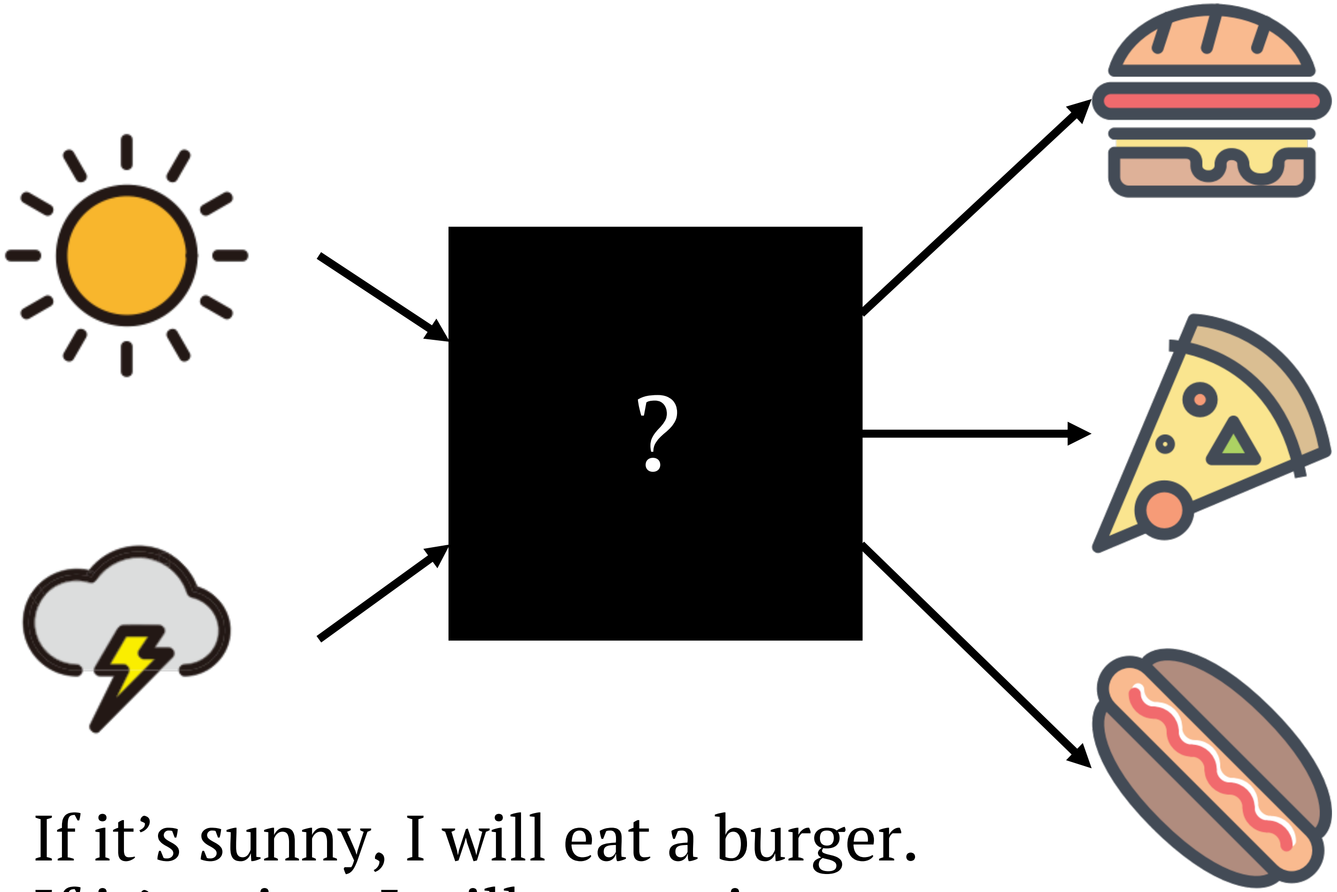
Recurrent Neural Networks

Dr. Unchalisa Taetragool

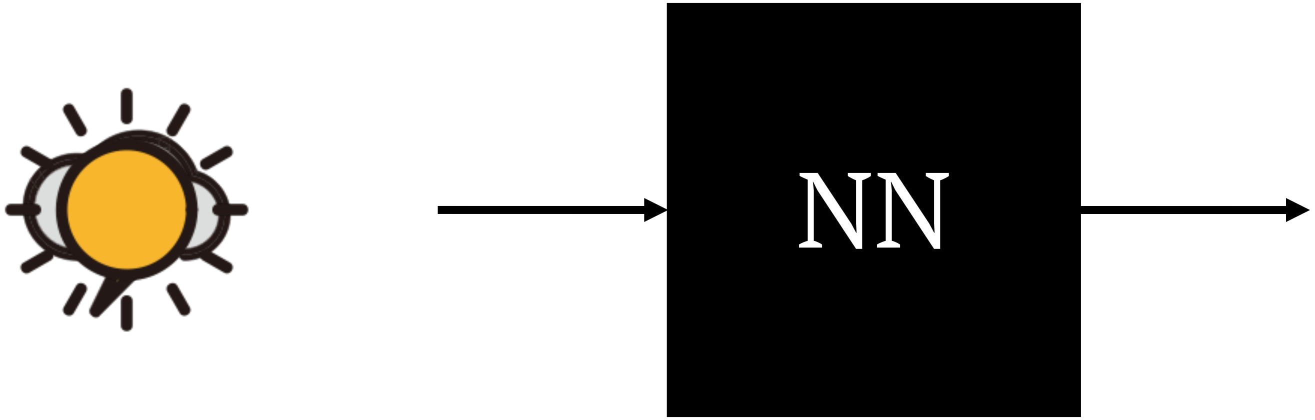
Department of Computer Engineering, Faculty of Engineering
King Mongkut's University of Technology Thonburi

Neural Network: Quiz!

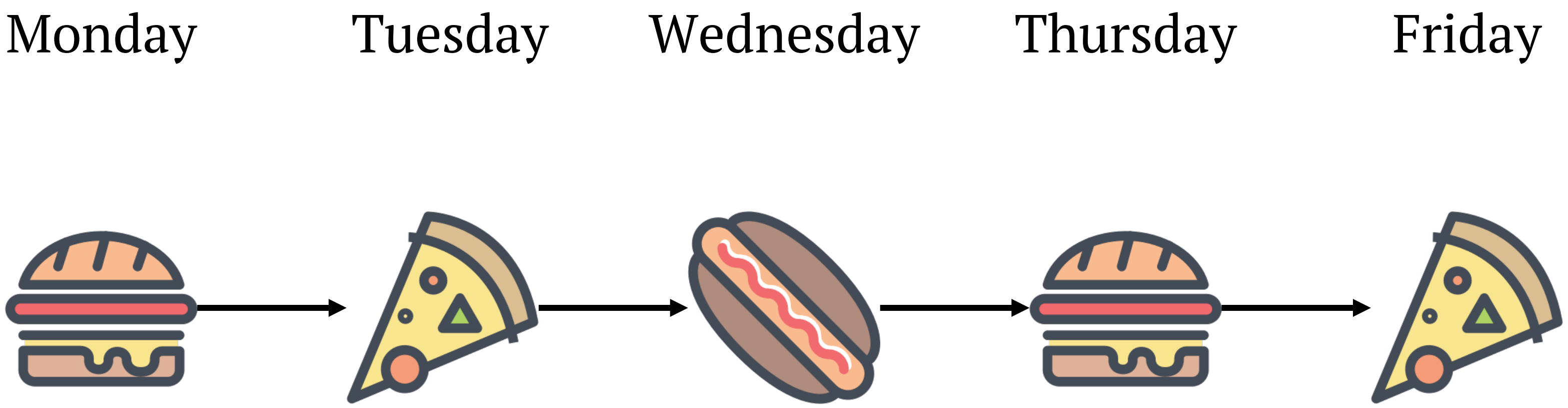


If it’s sunny, I will eat a burger.
If it’s rainy, I will eat a pizza.

Neural Network: Quiz!



Dinner Schedule



Monday

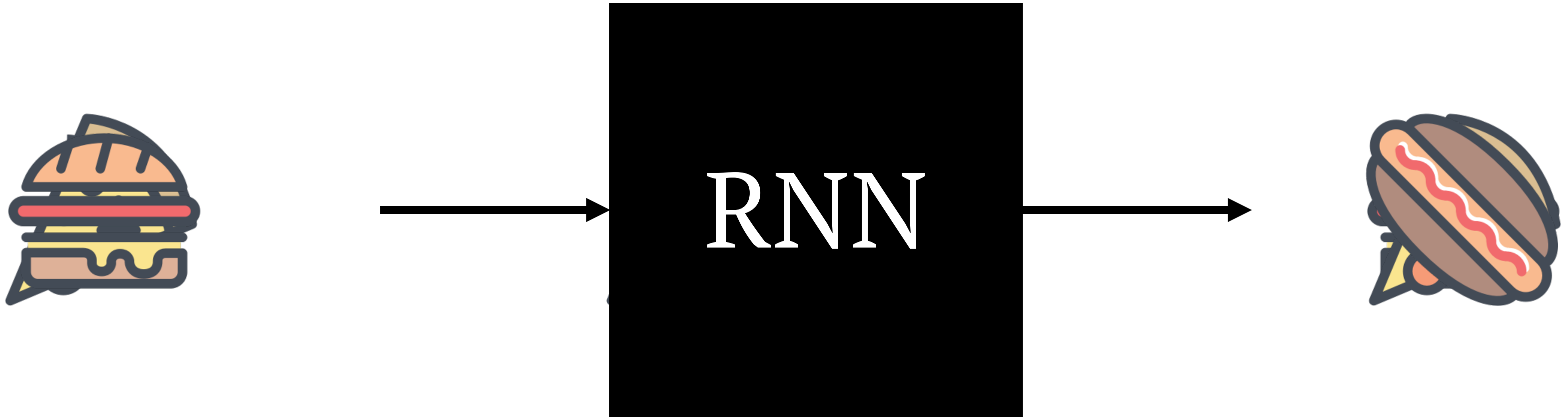
Tuesday

Wednesday

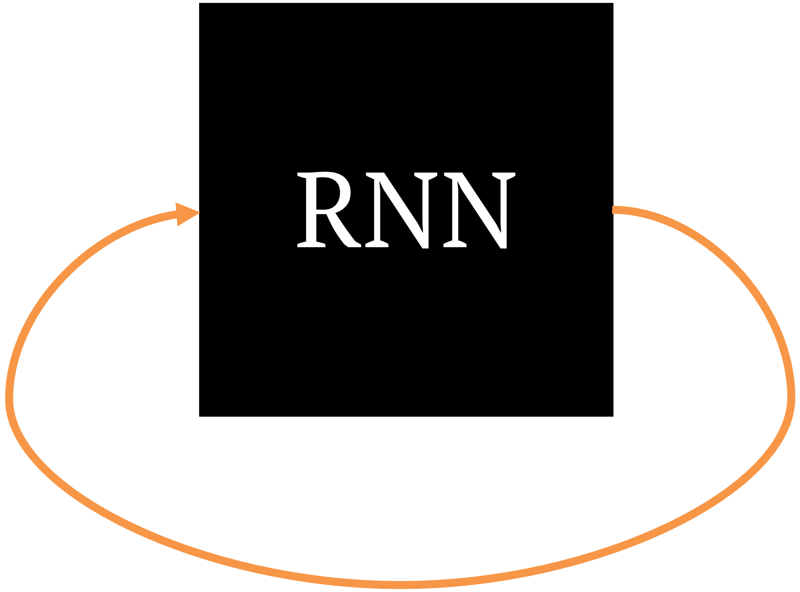
Thursday

Friday

Simple Recurrent Neural Network



Simple Recurrent Neural Network



Simple Recurrent Neural Network

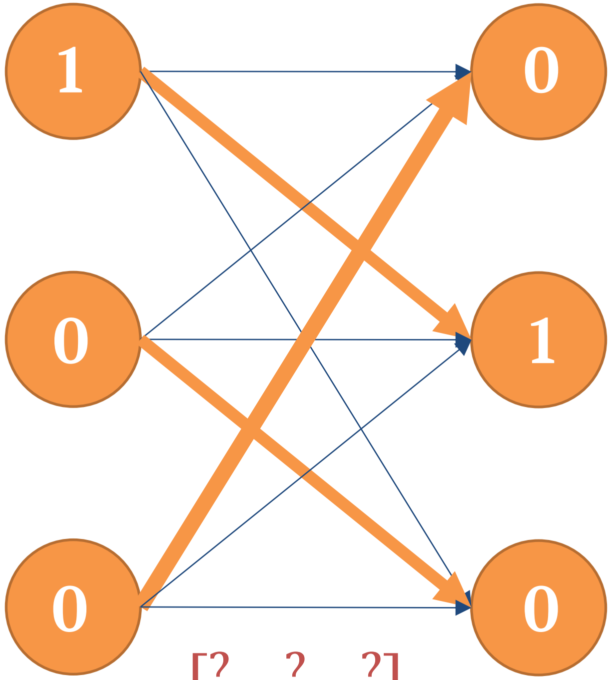
$$\begin{bmatrix}1\\0\\0\end{bmatrix} \longrightarrow \begin{bmatrix}0\\1\\0\end{bmatrix}$$

$$\begin{bmatrix}0\\1\\0\end{bmatrix} \longrightarrow \begin{bmatrix}0\\0\\1\end{bmatrix}$$

$$\begin{bmatrix}0\\0\\1\end{bmatrix} \longrightarrow \begin{bmatrix}1\\0\\0\end{bmatrix}$$

Simple Recurrent Neural Network

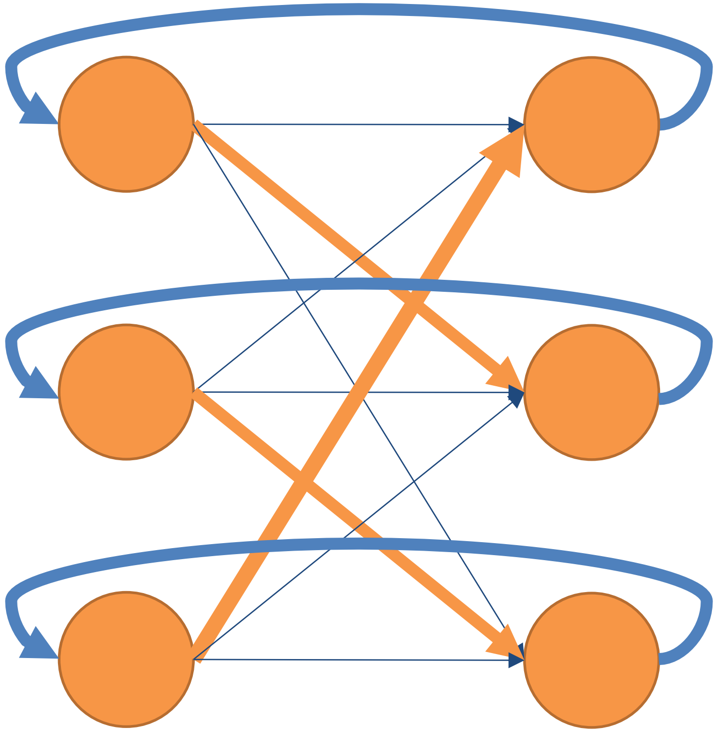
$$\begin{bmatrix}1\\0\\0\end{bmatrix}$$



$$\begin{bmatrix}?&?&?\\?&?&?\\?&?&?\end{bmatrix}$$

$$\begin{bmatrix}0\\1\\0\end{bmatrix}$$

Simple Recurrent Neural Network

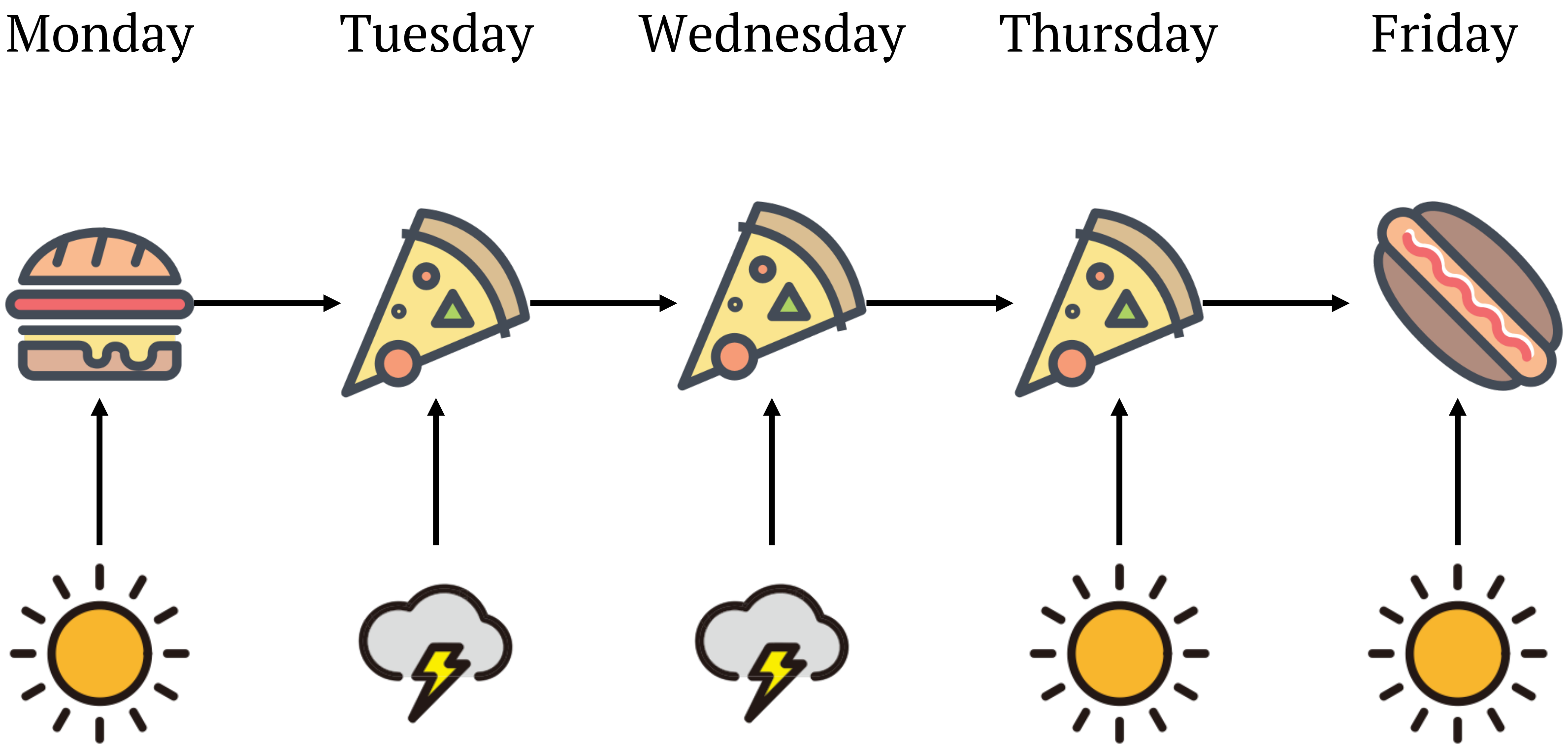


Weather Effect

Sunny
New food

Rain
Leftover

Weather Effect



Monday

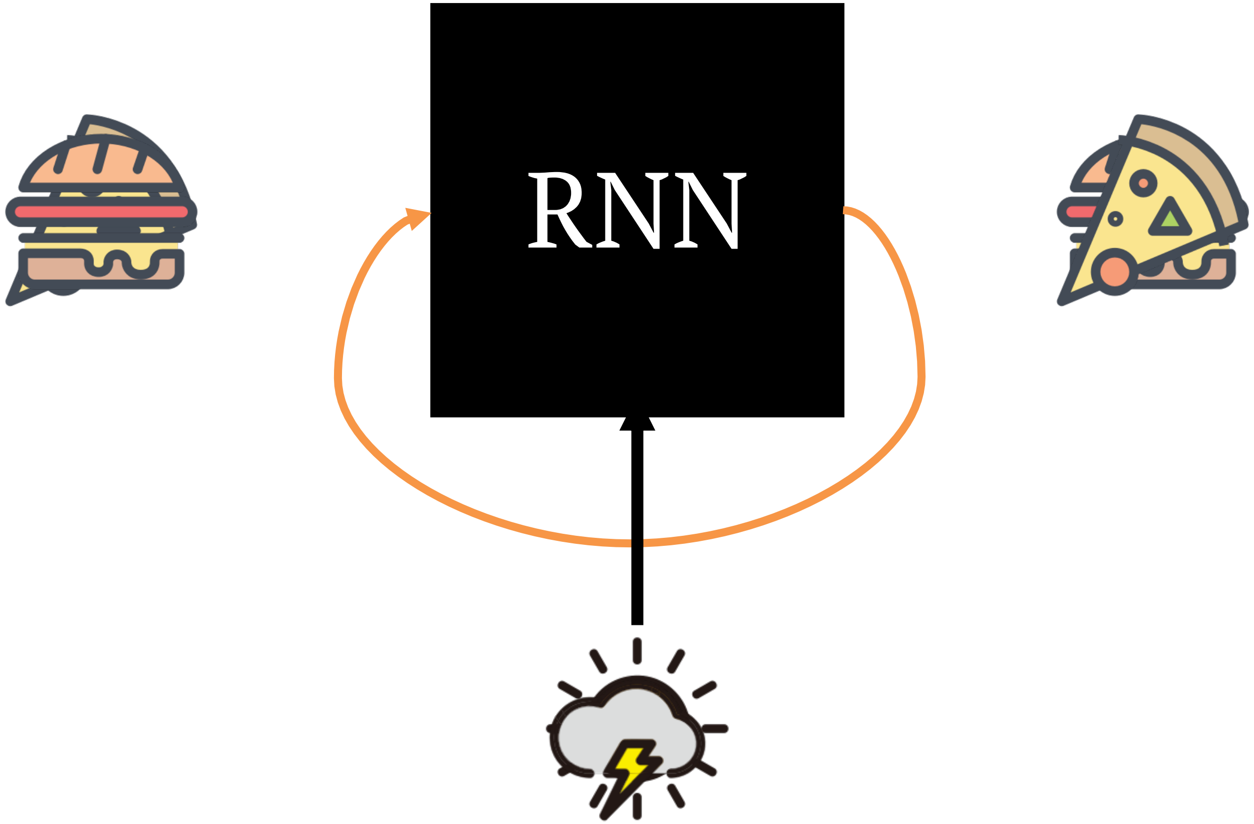
Tuesday

Wednesday

Thursday

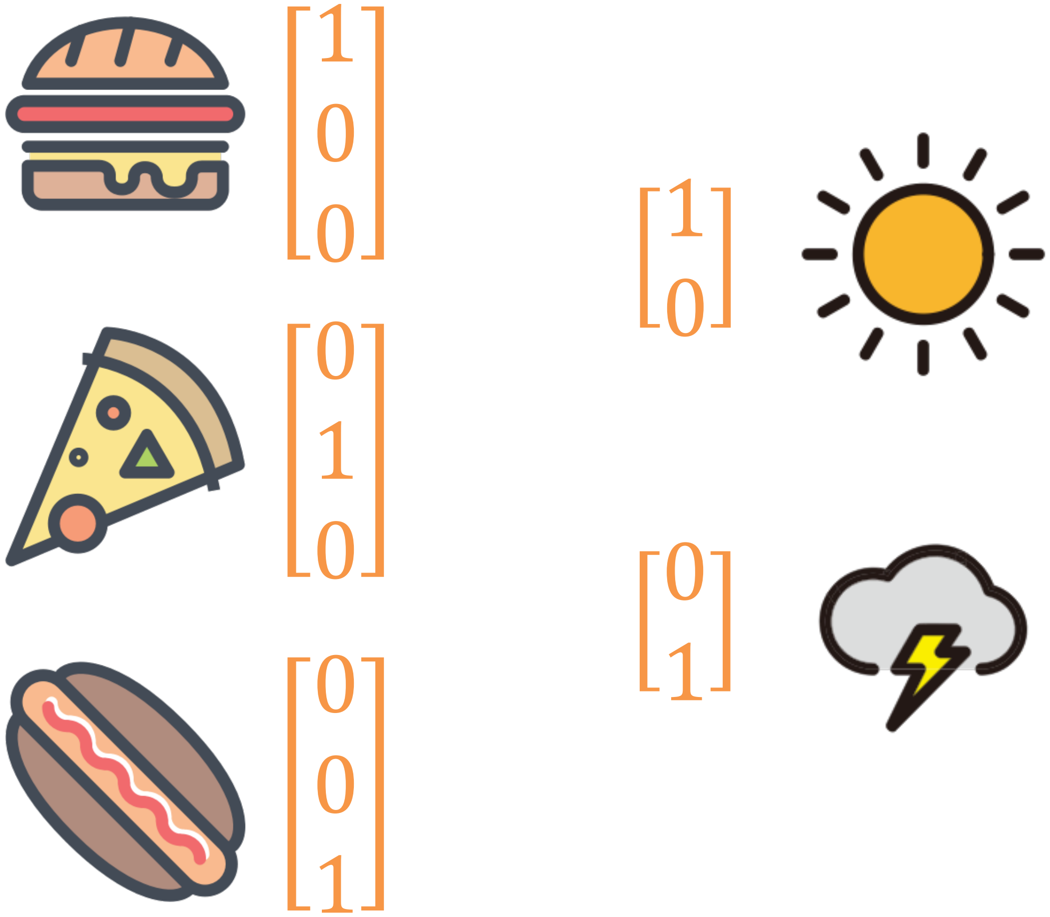
Friday

More Complicated RNN



RNN

Vectors



$$\begin{bmatrix}1\\0\\0\end{bmatrix} \quad \begin{bmatrix}0\\1\\0\end{bmatrix} \quad \begin{bmatrix}0\\0\\1\end{bmatrix}$$

$$\begin{bmatrix}1\\0\end{bmatrix} \quad \begin{bmatrix}0\\1\end{bmatrix}$$

More Complicated RNN

$$\left[\begin{array}{ccc} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ \hdashline 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{array}\right] + \left[\begin{array}{cc} 0 & 1 \\ 0 & 1 \\ 0 & 1 \\ \hdashline 1 & 0 \\ 1 & 0 \\ 1 & 0 \end{array}\right] \rightarrow$$

Food Add Weather Merge

Food

$$\begin{bmatrix}1\\0\\0\end{bmatrix}$$

$$\begin{bmatrix}0\\1\\0\end{bmatrix}$$

$$\begin{bmatrix}0\\0\\1\end{bmatrix}$$

$$\left[\begin{array}{ccc}1 & 0 & 0\\ 0 & 1 & 0\\ 0 & 0 & 1\\ \hdashline 0 & 0 & 1\\ 1 & 0 & 0\\ 0 & 1 & 0\end{array}\right]\begin{bmatrix}1\\0\\0\end{bmatrix}$$

$$= \left[\begin{array}{c}1\\0\\0\\ \hdashline 0\\1\\0\end{array}\right]$$

Food

Same

Next day

Weather

$\begin{bmatrix} 1 \\ 0 \end{bmatrix}$ ☀

$\begin{bmatrix} 0 \\ 1 \end{bmatrix}$ ⛈

$$\begin{bmatrix} 0 & 1 \\ 0 & 1 \\ 0 & 1 \\ \hline 1 & 0 \\ 1 & 0 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} ☀ = \begin{array}{cl} \begin{bmatrix} 0 \\ 0 \\ 0 \\ \hline 1 \\ 1 \\ 1 \end{bmatrix} & \begin{array}{l} \\ \text{Leftover} \\ \\ \\ \text{New food} \\ \\ \end{array} \end{array}$$

Weather

More Complicated RNN

☀ $\begin{bmatrix}1\\0\end{bmatrix}$ ⛈ $\begin{bmatrix}0\\1\end{bmatrix}$

$\begin{bmatrix}1\\0\\0\end{bmatrix}$ 🍔 $\begin{bmatrix}0\\1\\0\end{bmatrix}$ 🍕 $\begin{bmatrix}0\\0\\1\end{bmatrix}$ 🌭

$$\left[\begin{array}{ccc}1 & 0 & 0\\0 & 1 & 0\\0 & 0 & 1\\\hline 0 & 0 & 1\\1 & 0 & 0\\0 & 1 & 0\end{array}\right] + \left[\begin{array}{cc}0 & 1\\0 & 1\\0 & 1\\\hline 1 & 0\\1 & 0\\1 & 0\end{array}\right]$$

Food Add Weather Merge

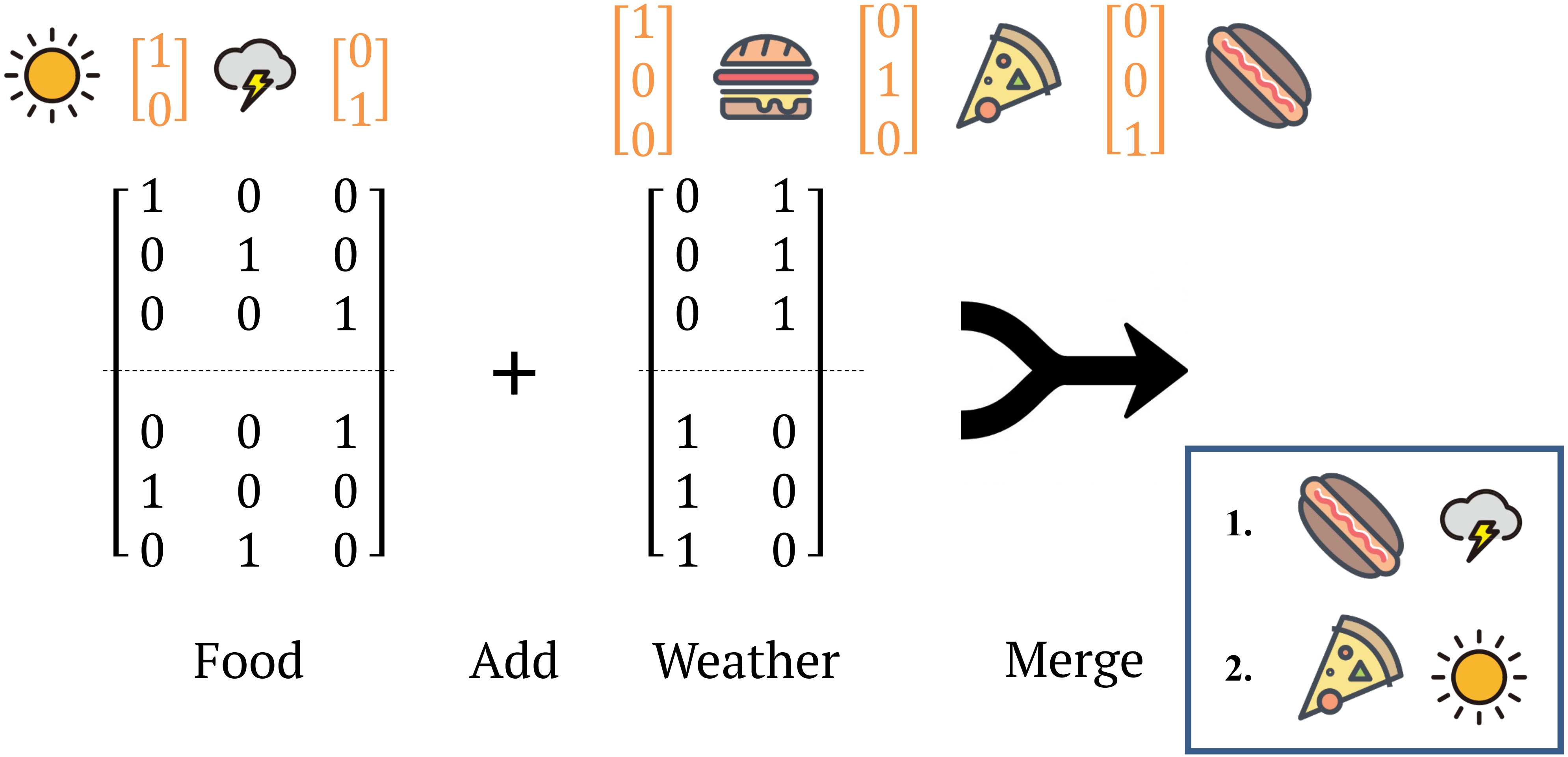
Add 🍔 ☀

$$\left[\begin{array}{c}1\\0\\0\\\hline 0\\1\\0\end{array}\right]\begin{array}{l}\text{Same}\\ \text{🍔}\\ \\ \\ \text{🍕}\\ \text{Next day}\end{array} + \left[\begin{array}{c}0\\0\\0\\\hline 1\\1\\1\end{array}\right]\begin{array}{l}\\ \text{Leftover}\\ \\ \\ \text{New food}\\ \\ \end{array} = \left[\begin{array}{c}1\\0\\0\\\hline 1\\\enclose{circle}{2}\\1\end{array}\right]$$

Merge 🍔 ☀️

$$\begin{bmatrix}1\\0\\0\\ \hdashline 1\\ \textcircled{2}\\1\end{bmatrix} \xrightarrow{\text{(non-linear)}} \begin{bmatrix}0\\0\\0\\ \hdashline 0\\1\\0\end{bmatrix} \rightarrowtail \begin{bmatrix}0+0\\0+1\\0+0\end{bmatrix} = \begin{bmatrix}0\\1\\0\end{bmatrix}$$ 🍕

More Complicated RNN

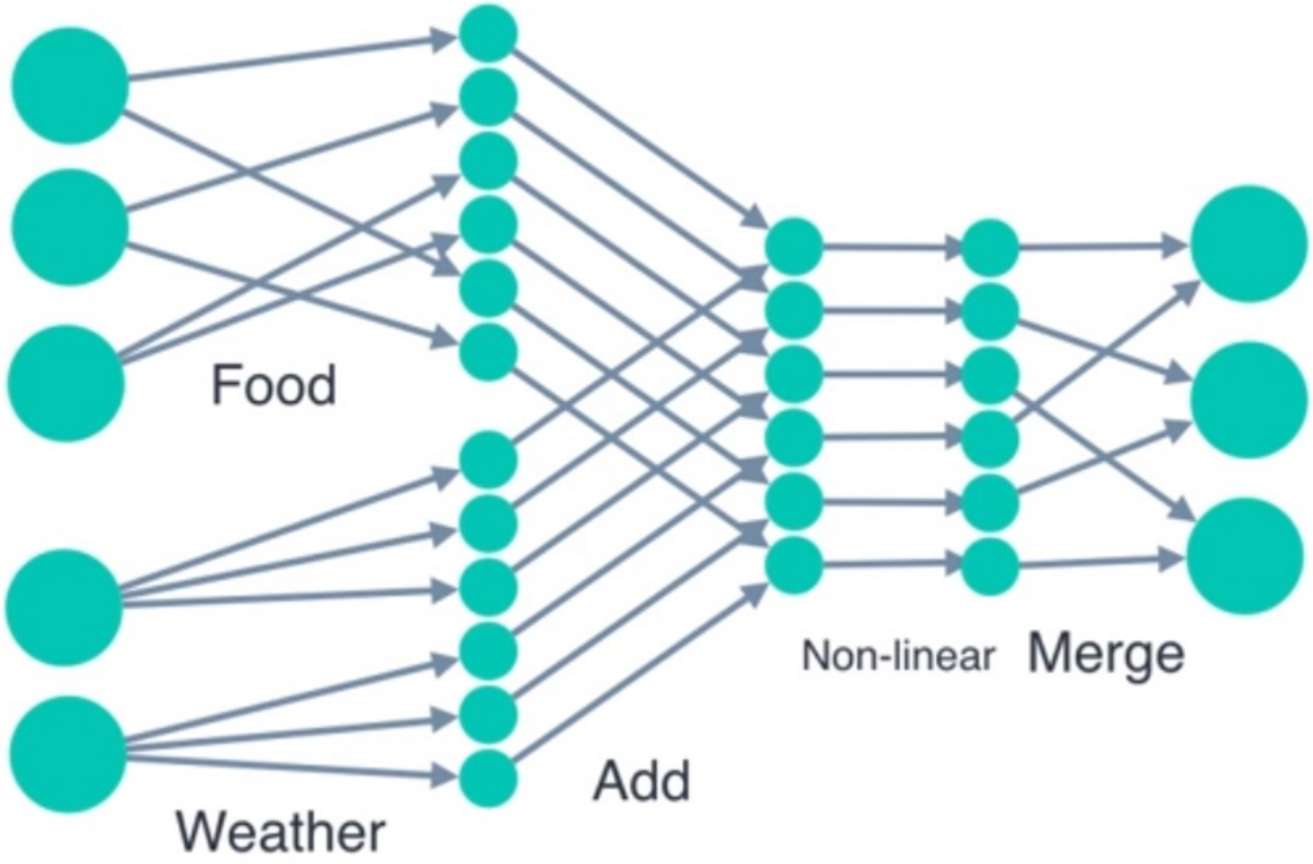


$$\begin{bmatrix}1\\0\end{bmatrix} \quad \begin{bmatrix}0\\1\end{bmatrix} \qquad \begin{bmatrix}1\\0\\0\end{bmatrix} \quad \begin{bmatrix}0\\1\\0\end{bmatrix} \quad \begin{bmatrix}0\\0\\1\end{bmatrix}$$

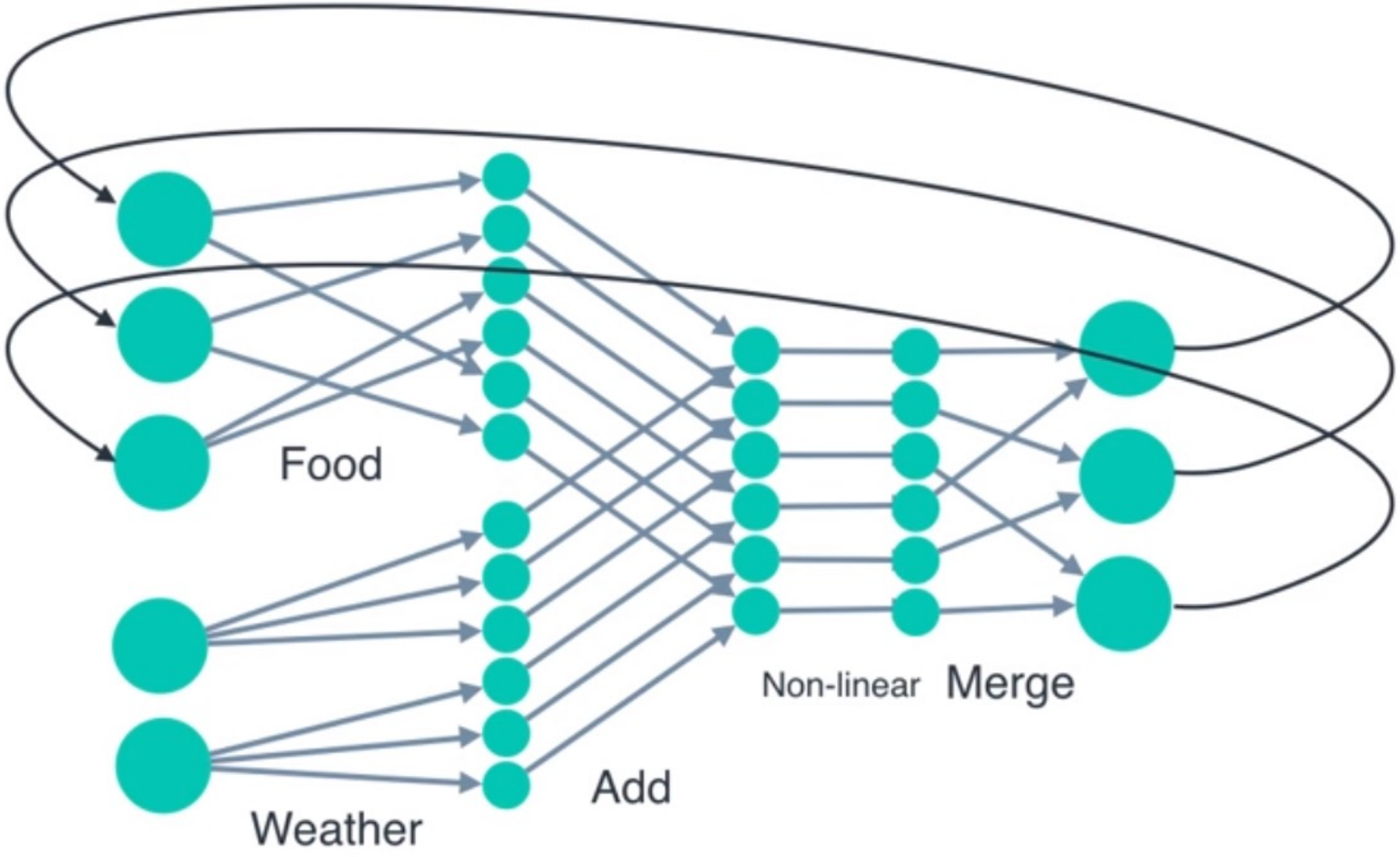
$$\begin{bmatrix}1 & 0 & 0\\0 & 1 & 0\\0 & 0 & 1\\ \hdashline 0 & 0 & 1\\1 & 0 & 0\\0 & 1 & 0\end{bmatrix} + \begin{bmatrix}0 & 1\\0 & 1\\0 & 1\\ \hdashline 1 & 0\\1 & 0\\1 & 0\end{bmatrix}$$

Food Add Weather Merge

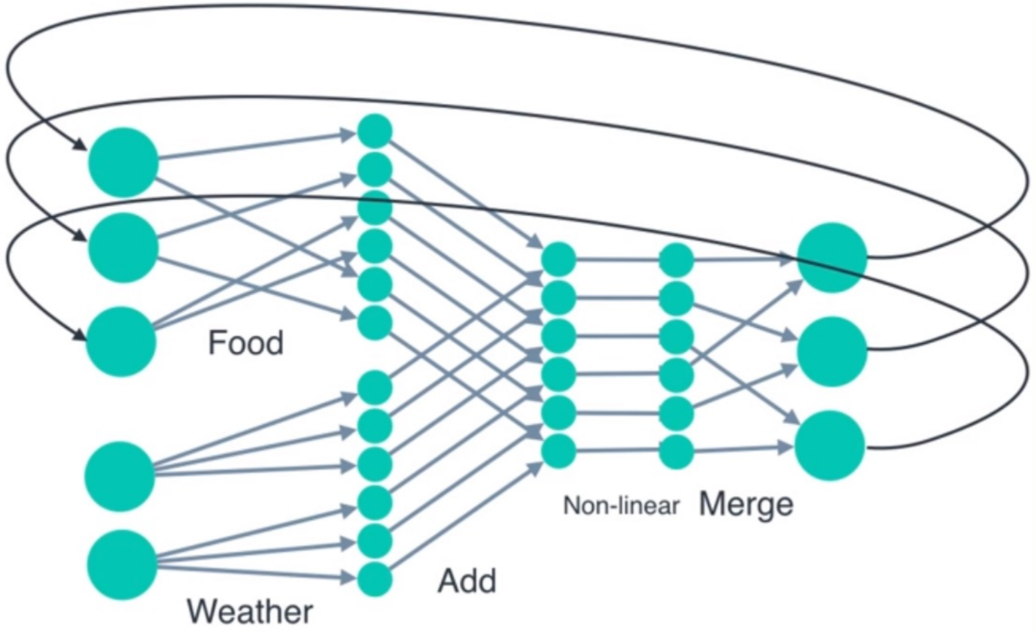
Recurrent Neural Network

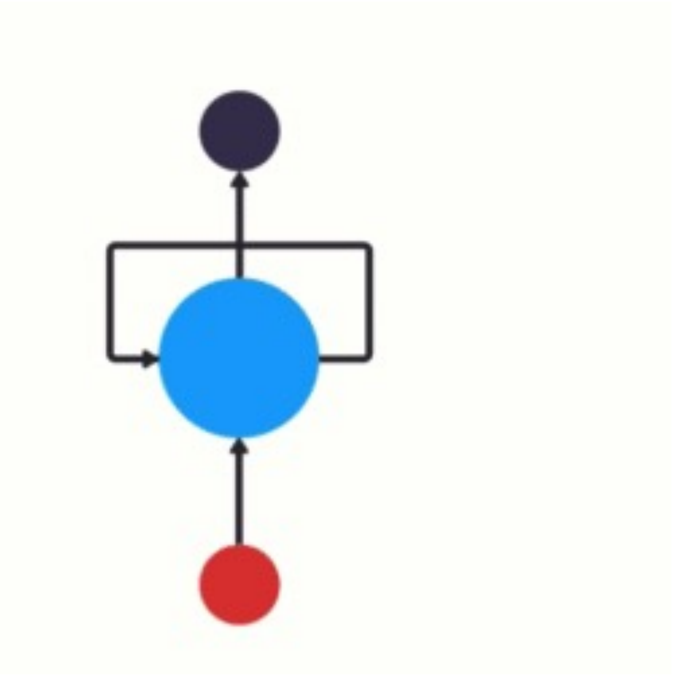


Recurrent Neural Network



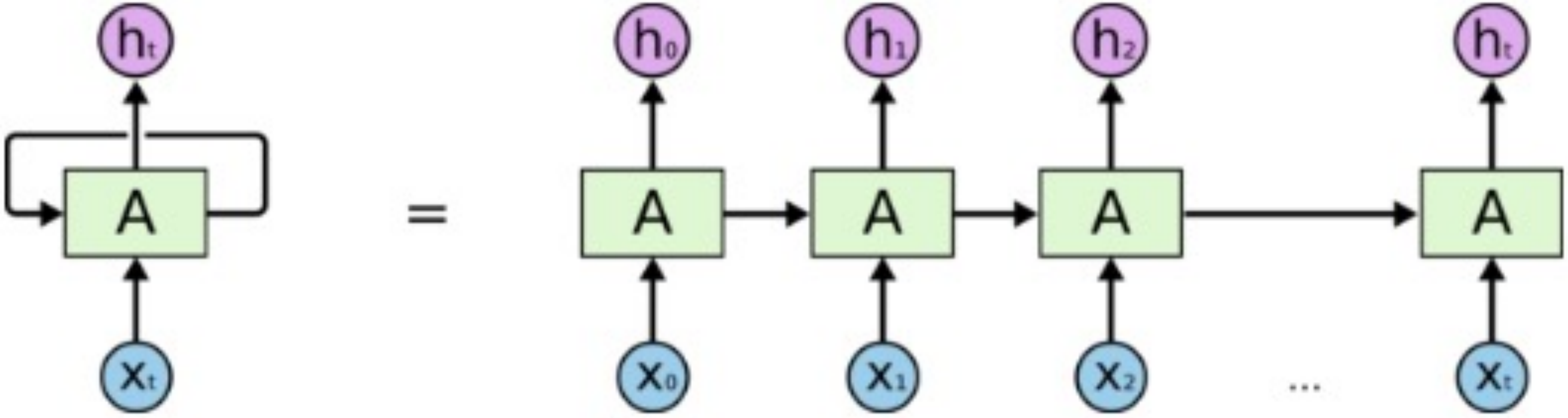
Recurrent Neural Network





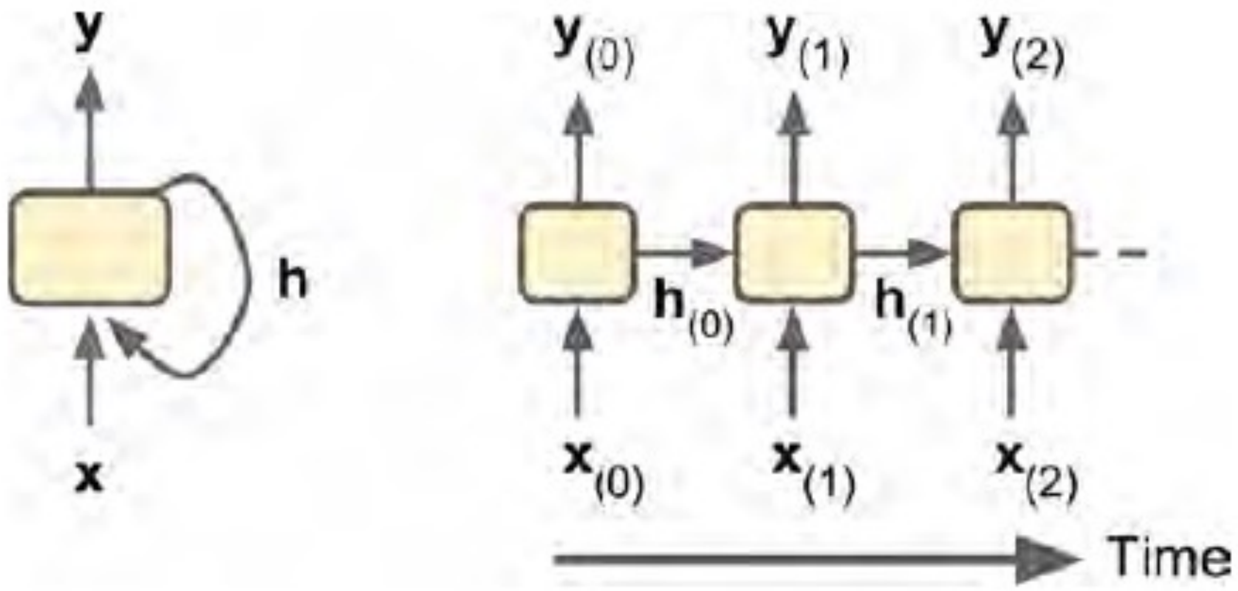
Credit: https://towardsdatascience.com

Recurrent Neural Network

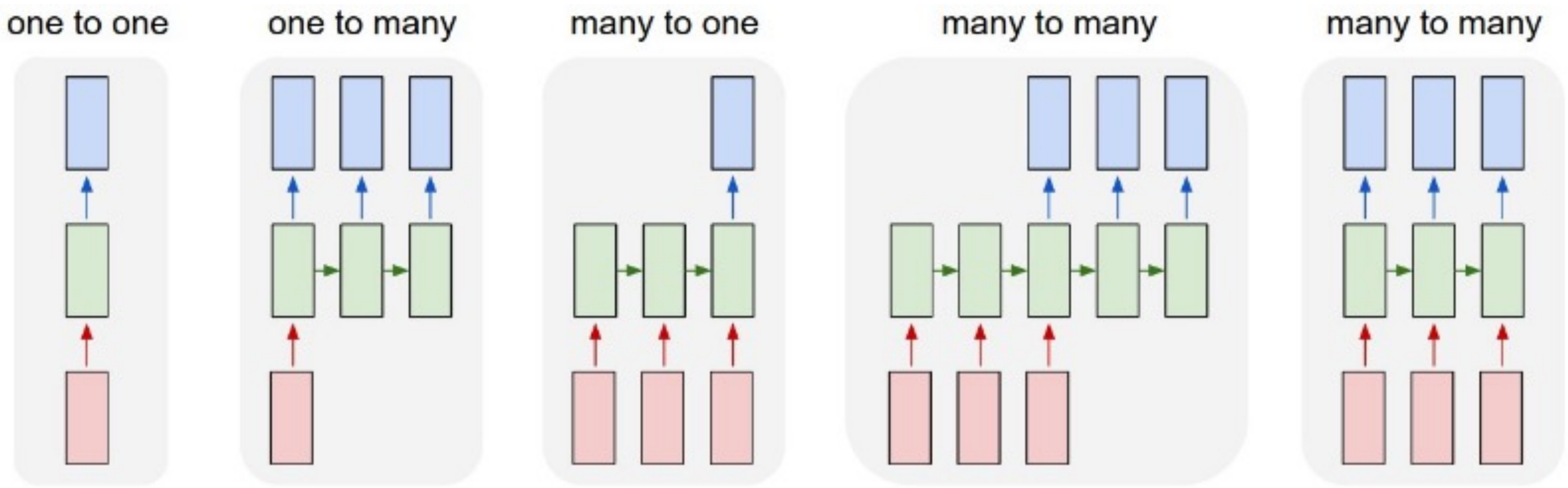


An unrolled recurrent neural network.

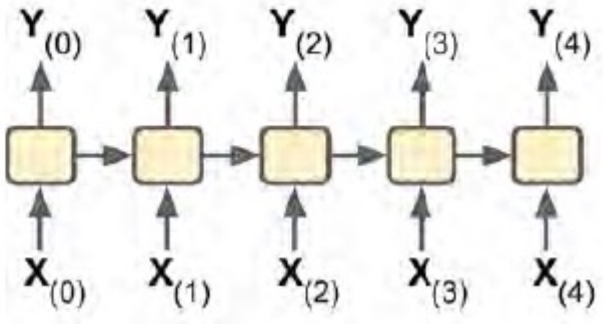
Memory Cells



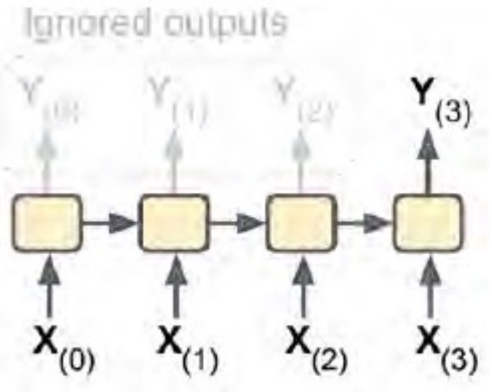
Recurrent Neural Network



Input and Output Sequences

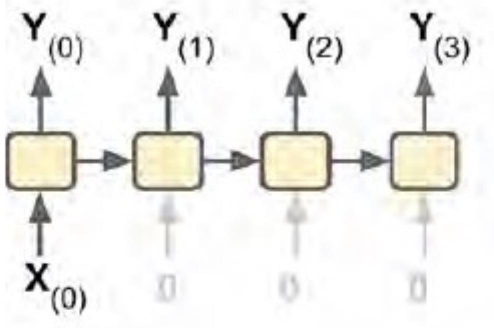


Sequence to Sequence
I/P – the prices over the last N days
O/P – the prices shifted by one day into the future

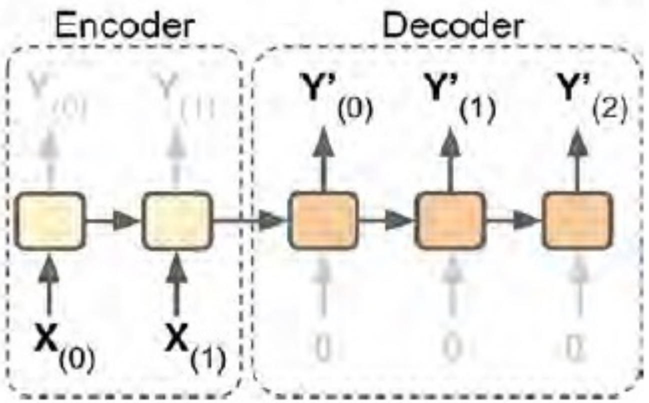


Sequence to Vector
I/P – the sequence of words corresponding to a review
O/P – a sentiment score

Input and Output Sequences



Vector to Sequence
I/P – an image
O/P – a caption for that image



Delayed Sequence to Sequence
e.g. translating a sentence from one language to another
Two-step model called an “Encoder-Decoder”

- Encoder – a sequence-to-vector network
- Decoder – a vector-to-sequence network

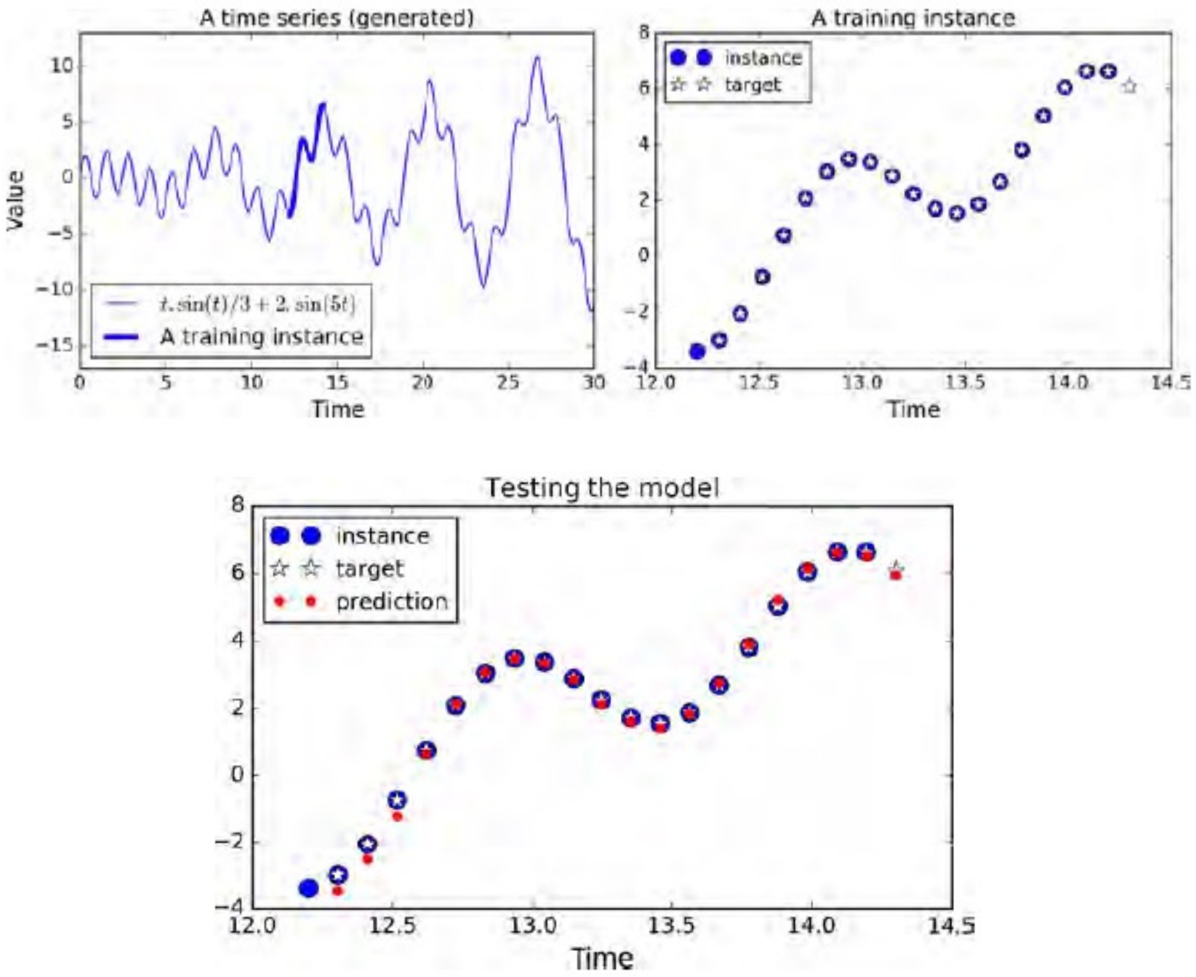
works much better than trying to translate on the fly with a single sequence-to-sequence RNN

RNN Applications

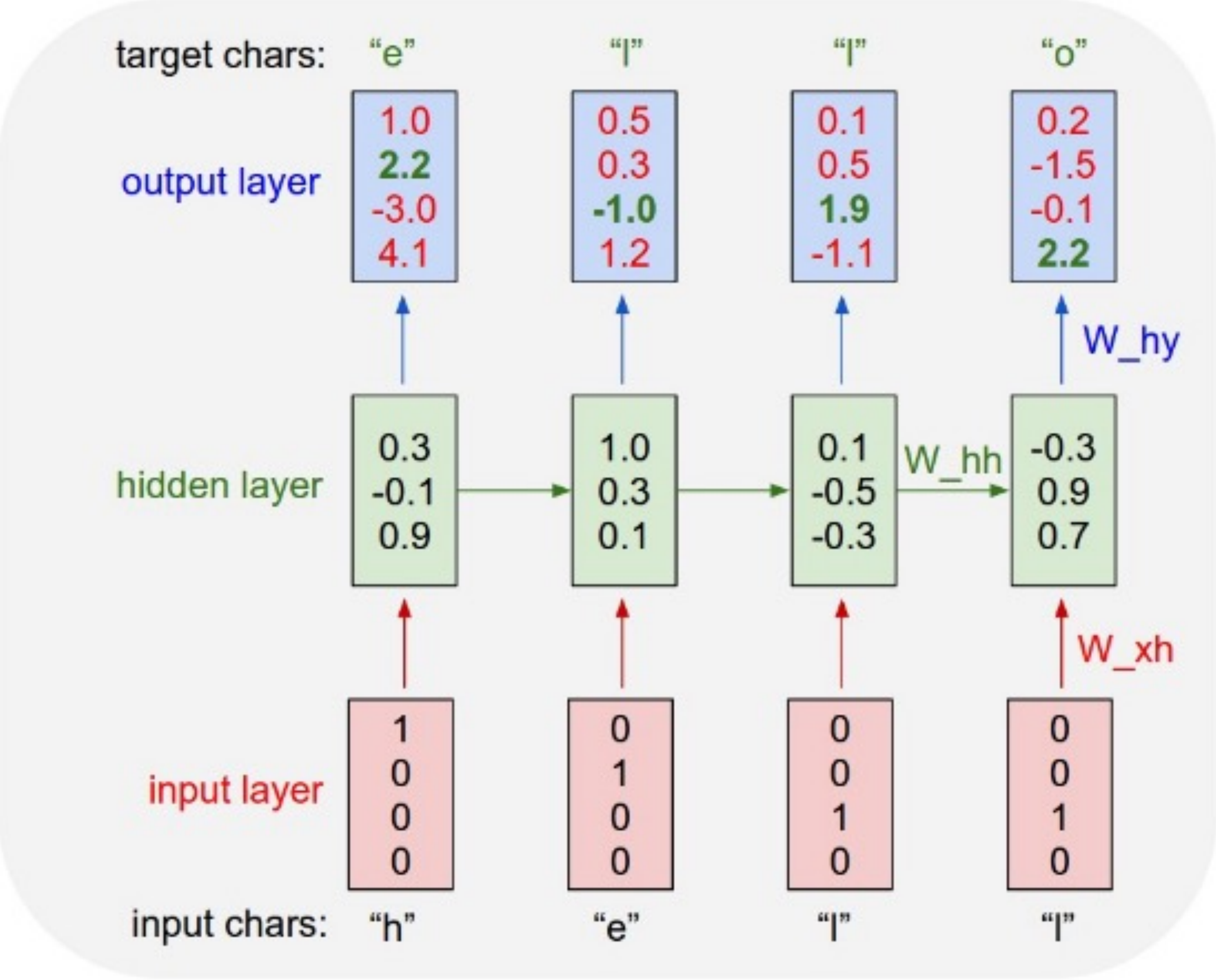
	x - input		y - output
Speech recognition		⟹	"Fuzzy Wuzzy was a bear. Fuzzy Wuzzy had no hair."
Music generation	Ø	⟹	
Sentiment classification	"Decent effort. The plot could have been better."	⟹	★★★☆☆
DNA sequence analysis	ACTGTACCCATGTGACTGCCC	⟹	ACTGTACCCATGTGACTGCCC
Machine translation	"El que no arriesga, no gana."	⟹	"If you don't take risks, you cannot win."
Video activity recognition		⟹	Running
Name entity recognition	"Ygritte says Jon Snow knows nothing."	⟹	"Ygritte says Jon Snow knows nothing."

Courtesy of Strahinja Zivkovic

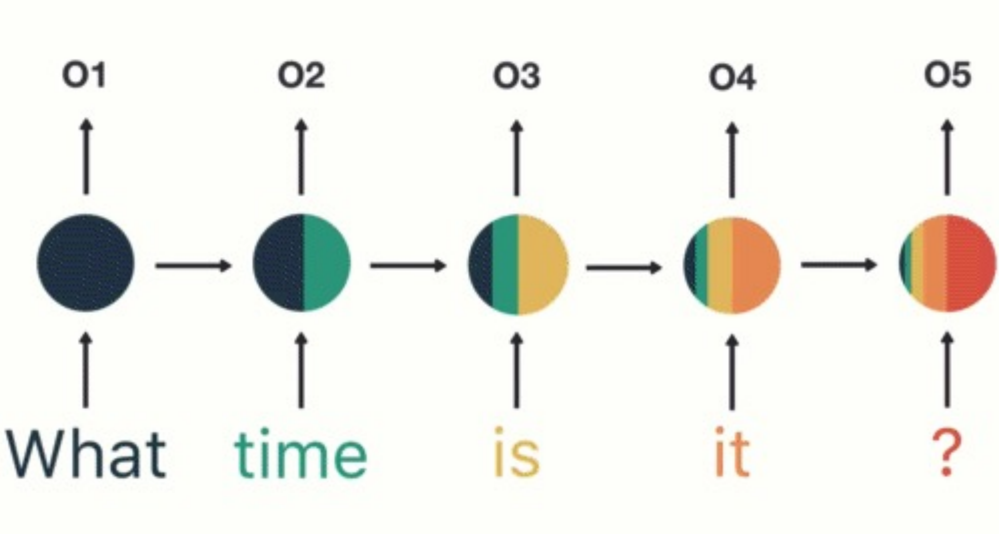
Example: Predict Time Series



Example:
Character-Level Language Models

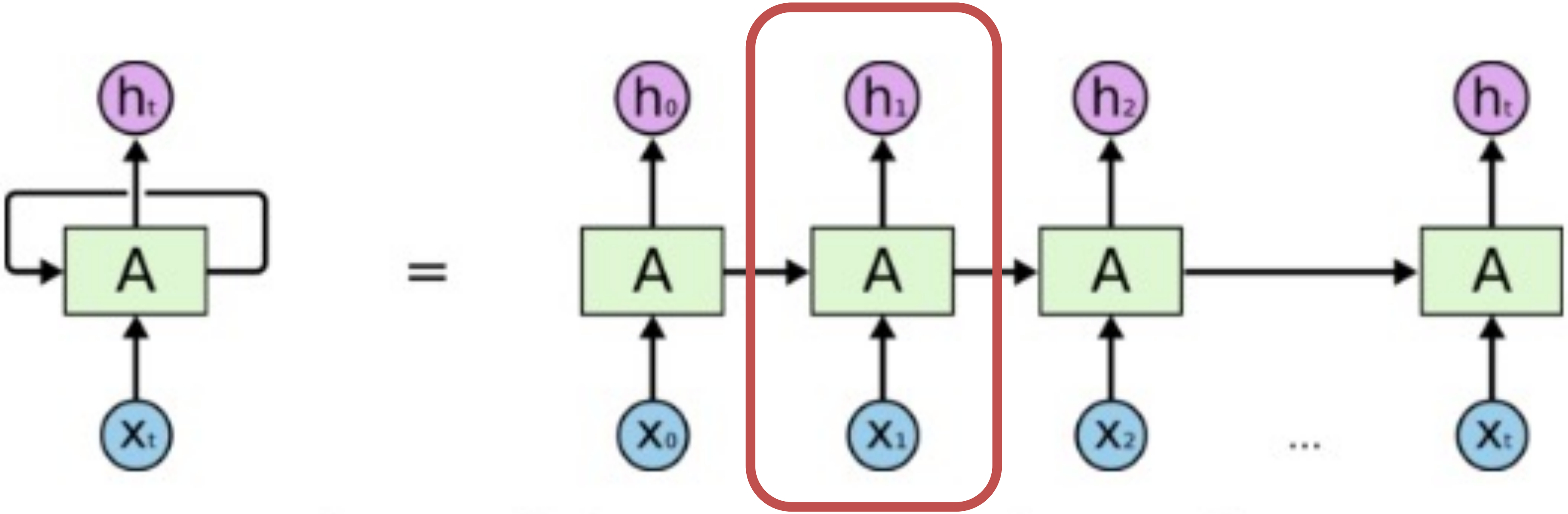


Example: Encoder



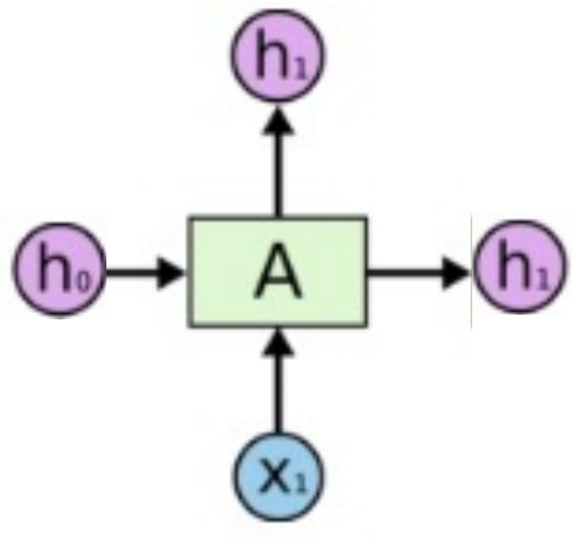
Credit: https://towardsdatascience.com

How to Train RNN



An unrolled recurrent neural network.

Training RNN



- Feed forward:
  - A simple one layer of input

$$\mathrm{h}_t = u_t \mathrm{X}_t + w_t \mathrm{h}_{t-1}$$

- Back propagation: **Gradient descent!**

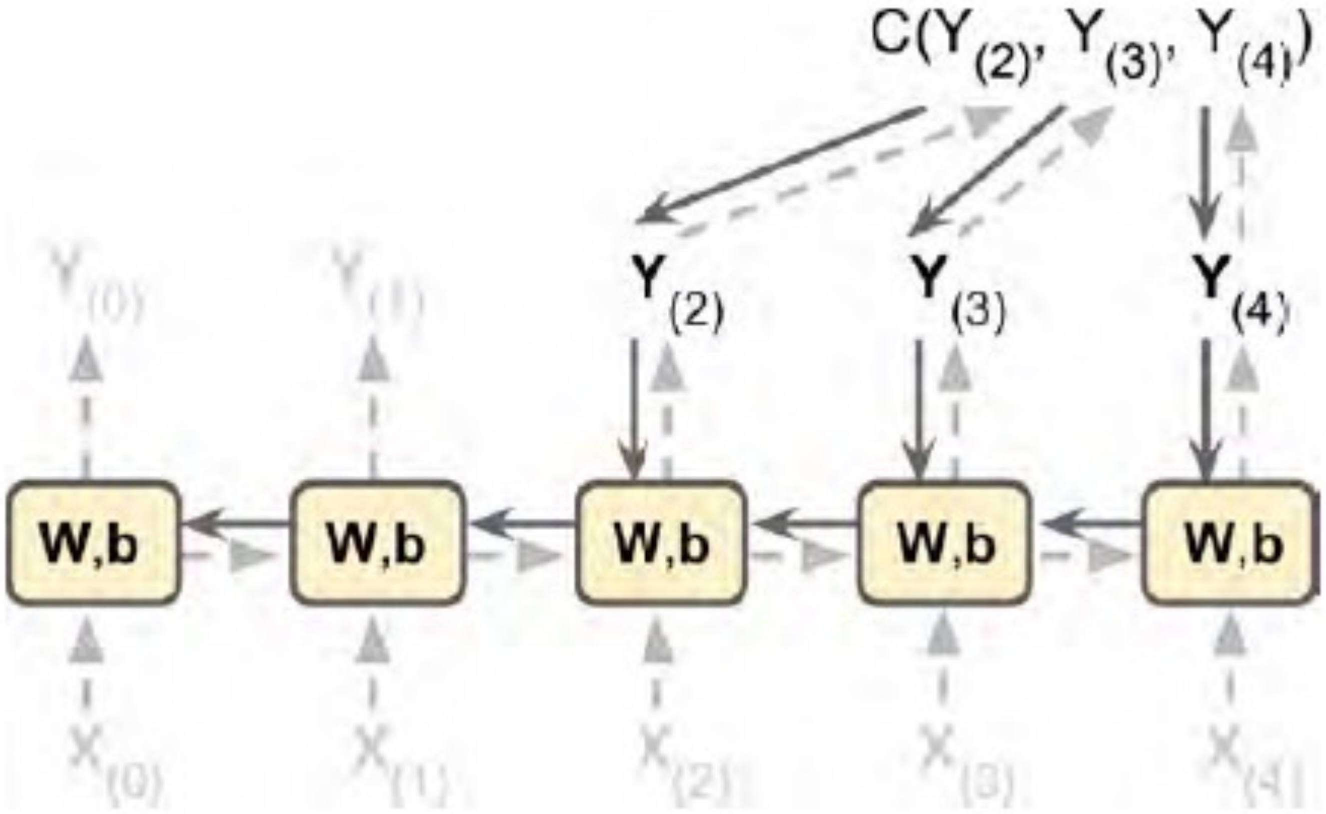
$$E = \frac{1}{N}\sum_{i=1}^{n}\left(y_t^i - \mathrm{h}_t^i\right)^2 \text{ then find } \frac{\partial E}{\partial u_t} \text{ and } \frac{\partial E}{\partial w_t}$$

$$u_t^{new} = u_t^{old} - \frac{\partial E}{\partial u_t} \text{ and } w_t^{new} = w_t^{old} - \frac{\partial E}{\partial w_t}$$

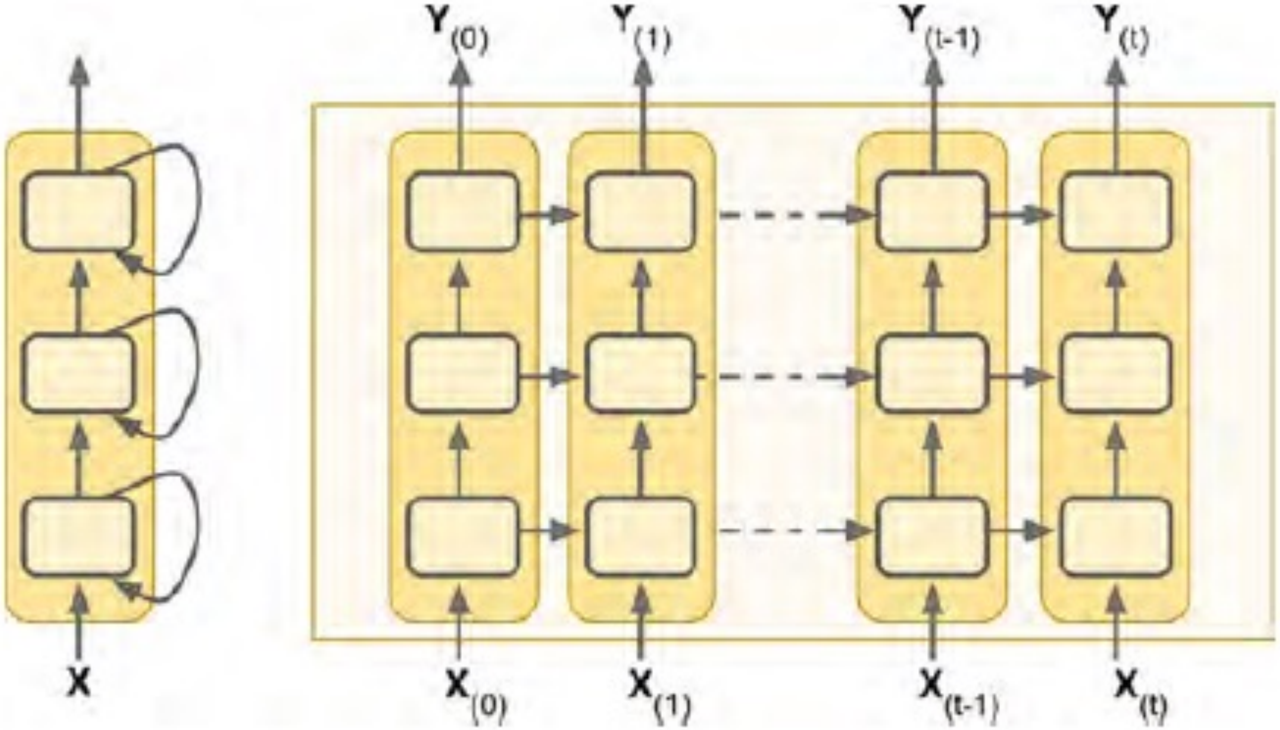
Training RNN: Backpropagation Through Time (BPTT)

- Just like in regular backpropagation,
  - there is a first forward pass through the network
  - then the output sequence is evaluated using a cost function
  - the gradients of that cost function are propagated backward through the network
  - finally, the model parameters are updated using the gradients computed during BPTT

Training RNN: Backpropagation Through Time (BPTT)



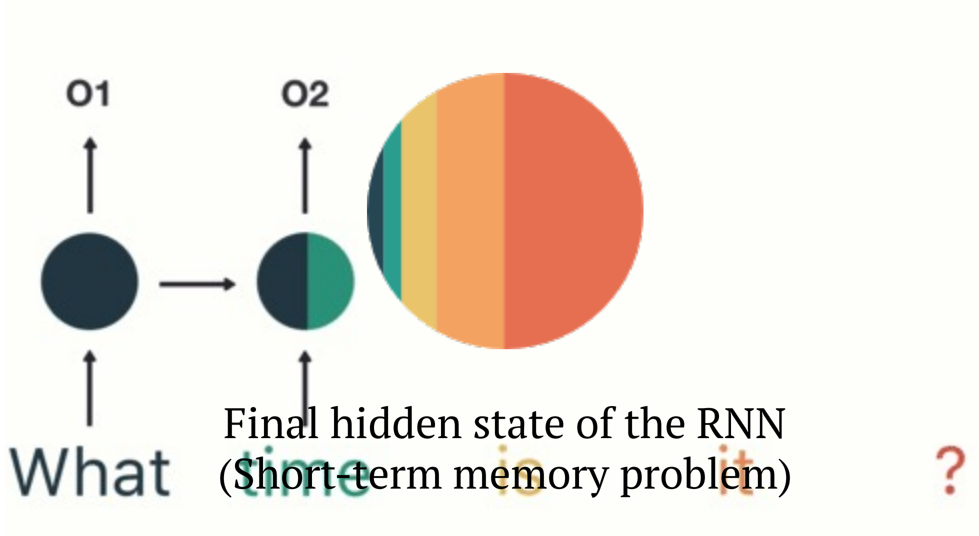
Deep RNN



Difficulties of Training RNN

- To train an RNN on long sequences, you will need to run it over many time steps, making the RNN a very deep network.
- Just like deep neural network, it may suffer from
  - taking forever to train
  - vanishing gradient problem
- To alleviate these problems, you can use
  - good parameter initialization
  - non-saturating activation functions (e.g., ReLU)
  - faster optimizers
  - etc.

Vanishing Gradient



Final hidden state of the RNN
(Short-term memory problem)

Credit: https://towardsdatascience.com

Vanishing Gradient

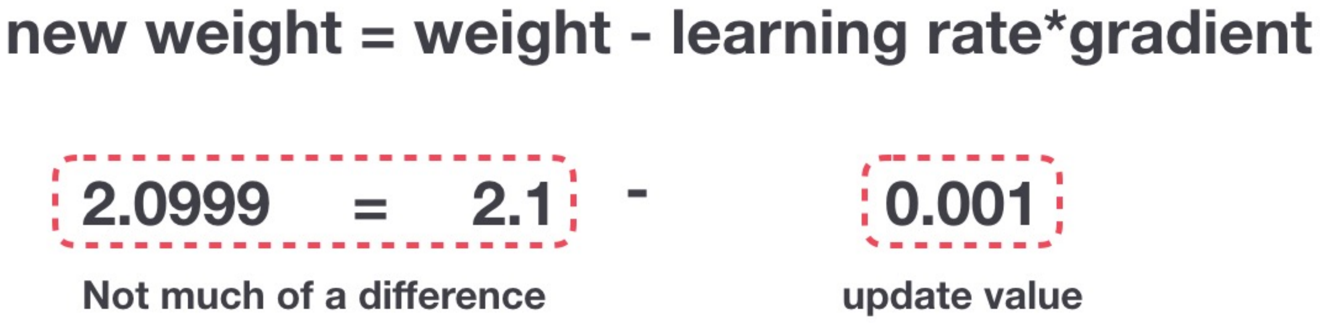
- The *vanishing gradient* is due to the nature of back-propagation
- Weight adjustment from the gradient descent algorithm

$$\underset{\substack{\uparrow\\ \text{new weight}}}{w_{ij}} := \underset{\substack{\uparrow\\ \text{old weight}}}{w_{ij}} - \underset{\substack{\uparrow\\ \text{learning}\\ \text{rate}}}{\alpha} * \underset{\substack{\uparrow\\ \text{gradient}\\ \text{of cost function}}}{\frac{\partial\ Cost}{\partial\ w_{ij}}}$$

- The bigger the gradient, the bigger the adjustments and vice versa.

Vanishing Gradient

- When doing back propagation, each node in a layer calculates it's gradient with respect to the effects of the gradients in the layer before it.
- So if the adjustments to the layers before it is small, then adjustments to the current layer will be even smaller.



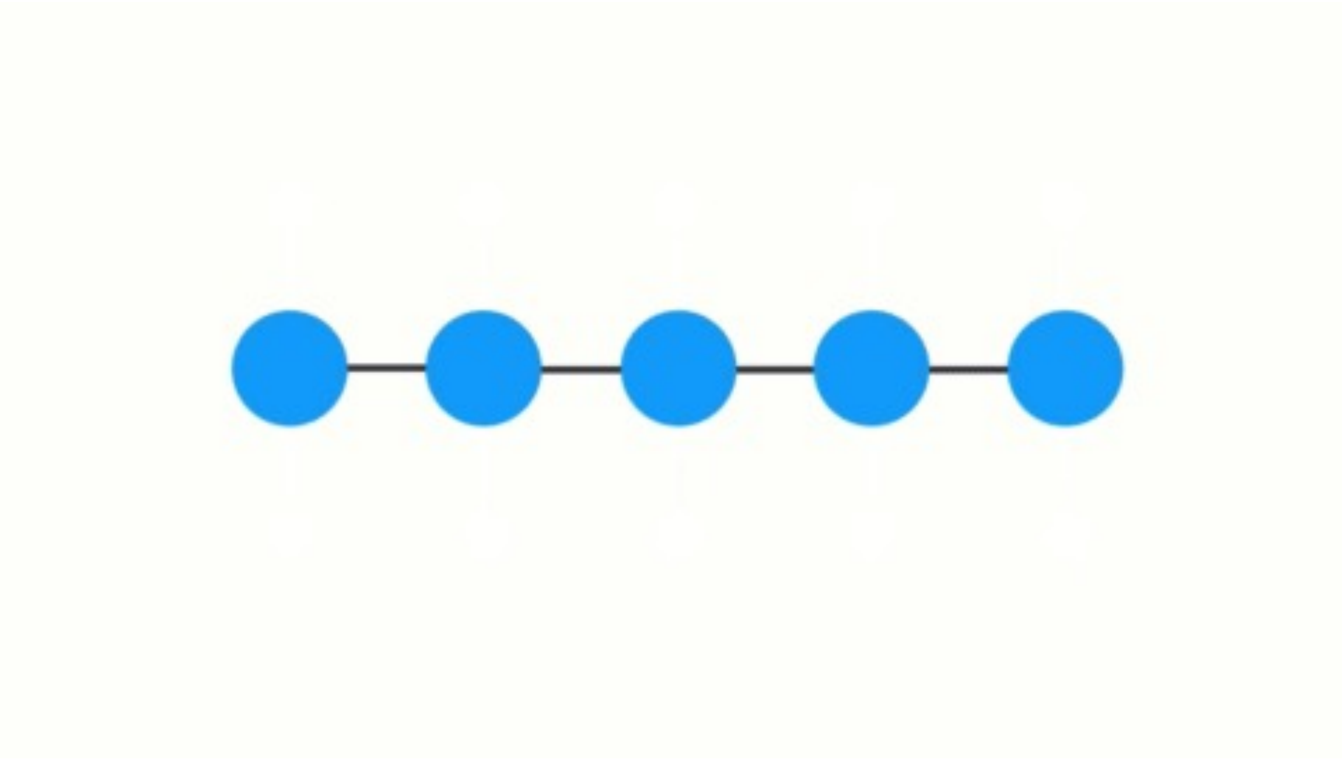
new weight = weight - learning rate*gradient

2.0999 = 2.1 - 0.001

Not much of a difference

update value

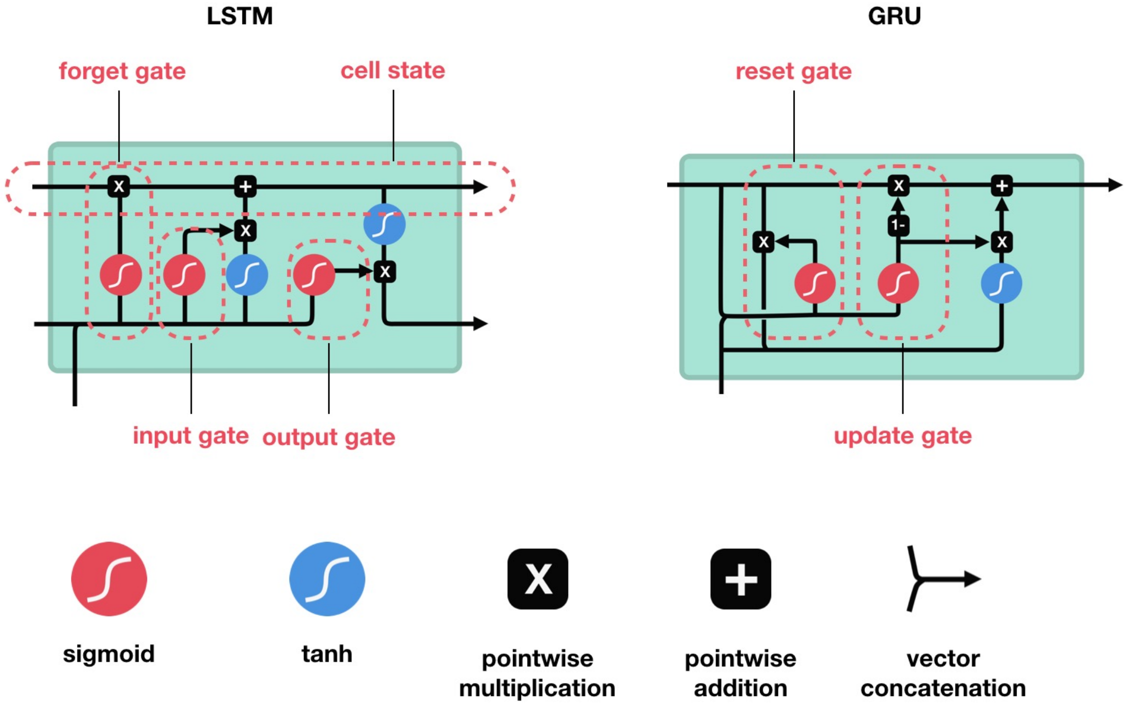
Vanishing Gradient



Gradients shrink as it back-propagates through time

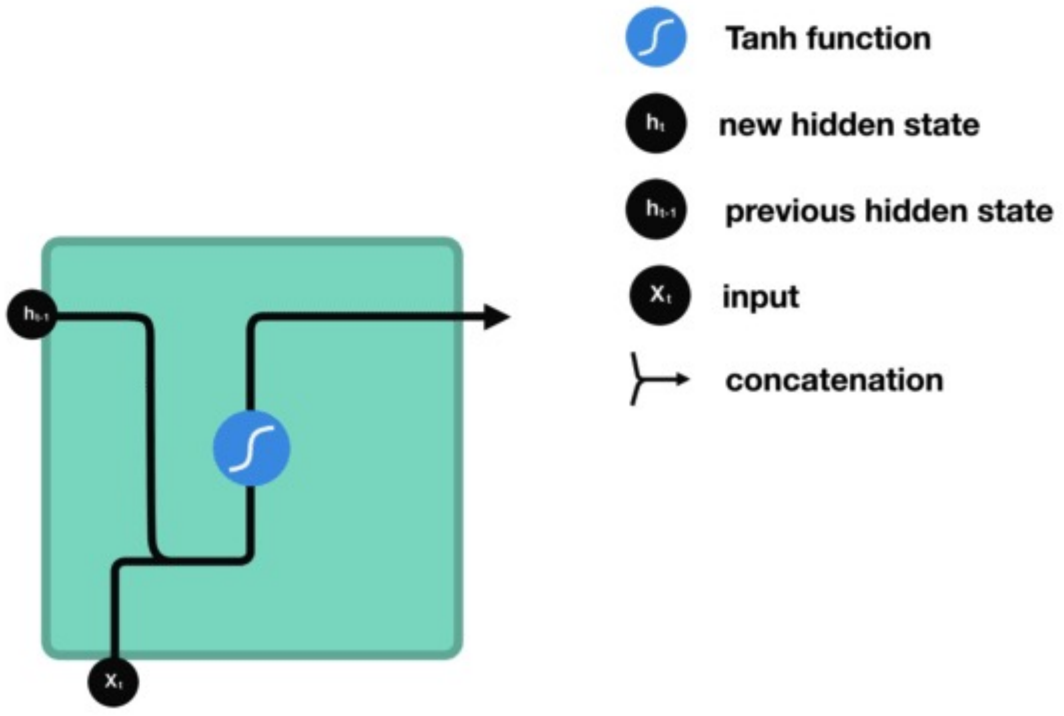
Solutions to short-term memory

- Long Short-Term Memory (LSTM)
- Gated Recurrent Units (GRU)



Credit: https://towardsdatascience.com

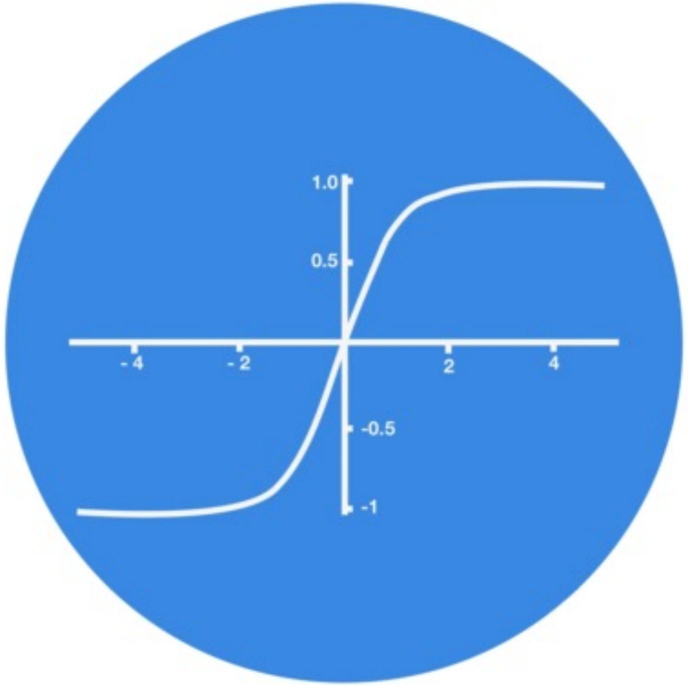
RNN (recap)



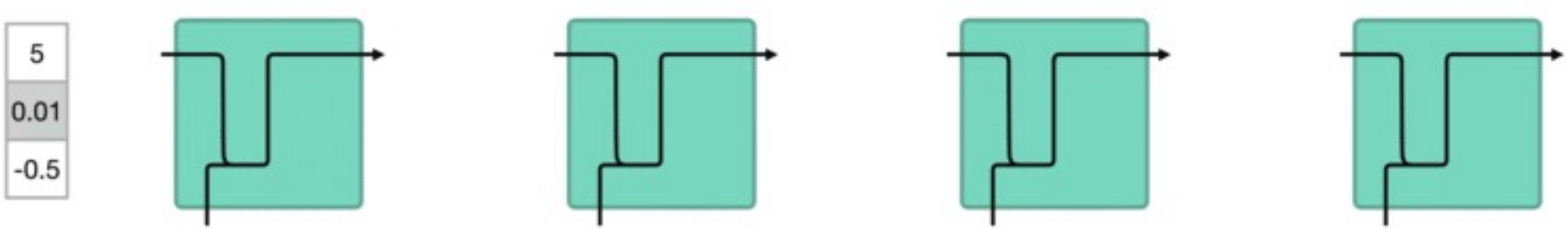
Credit: https://towardsdatascience.com

Tanh activation

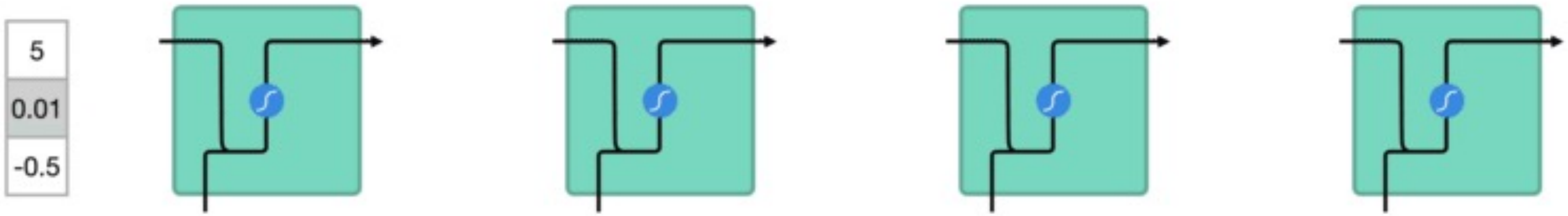
5
0.1
-0.5



Tanh activation



vector transformations without tanh



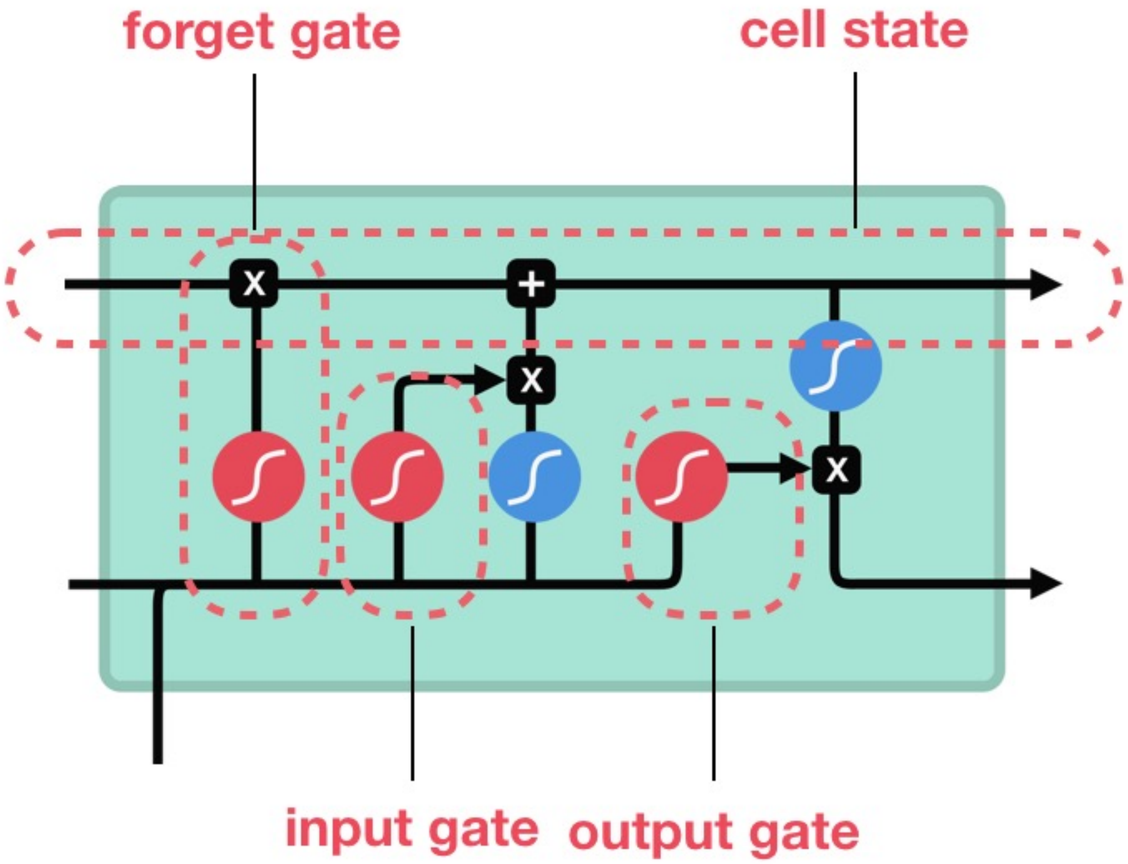
vector transformations with tanh

Credit: https://towardsdatascience.com

Long Short-Term Memory (LSTM)

- The core concept of LSTM’s are the **cell state**, and various **gates**.
- The cell state, in theory, can carry relevant information throughout the processing of the sequence.
- As the cell state goes on its journey, information get’s added or removed to the cell state via gates.
- The gates are different neural networks that decide which information is allowed on the cell state.
- The gates can learn what information is relevant to keep or forget during training.

Long Short-Term Memory (LSTM)

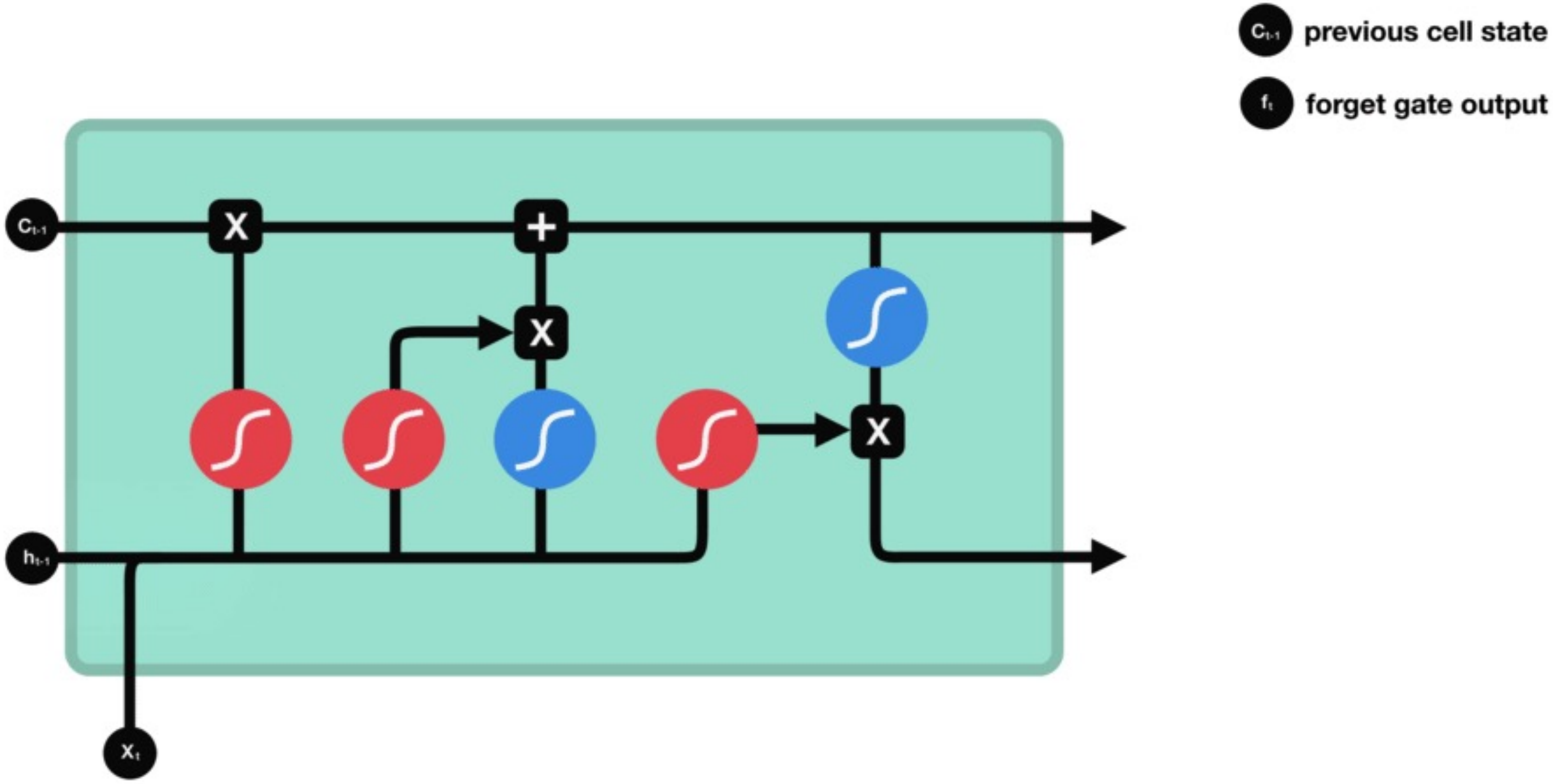


Credit: https://towardsdatascience.com

Long Short-Term Memory (LSTM)

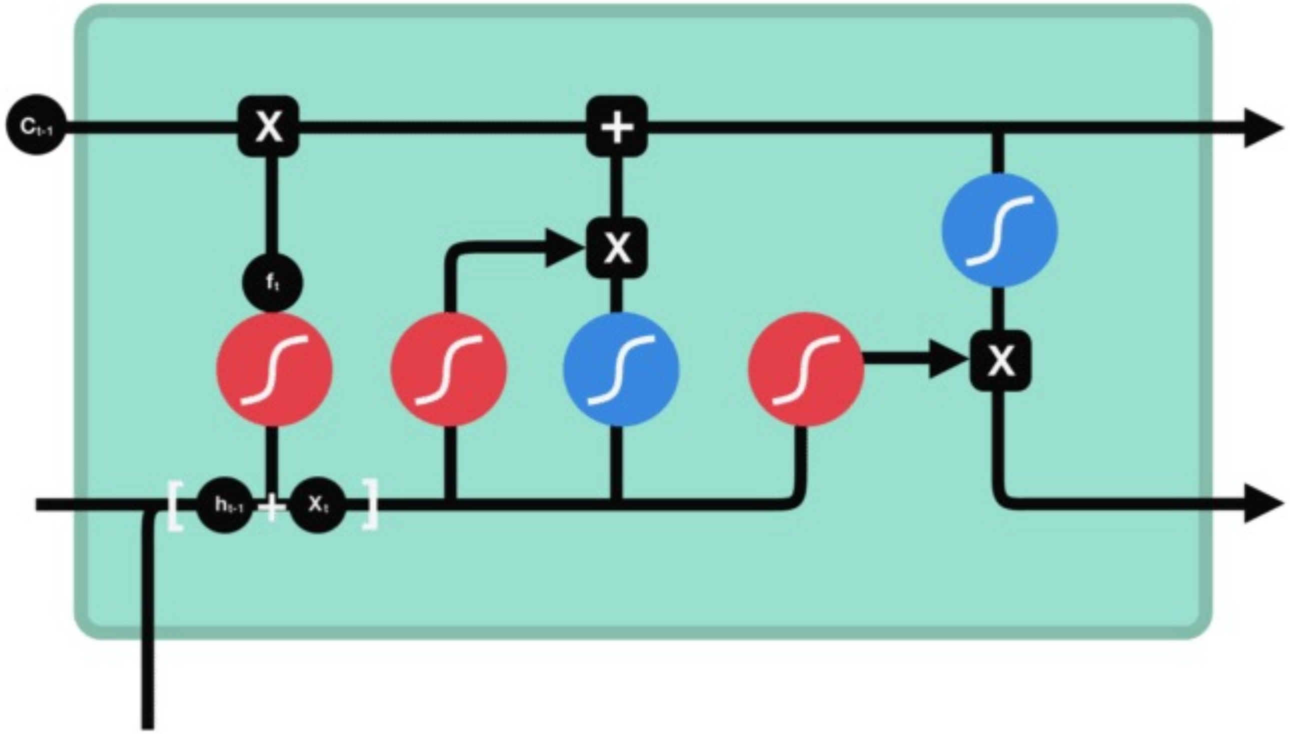
- 3 main gates:
  - Forget gate
    - decides what is relevant to keep from prior steps
  - Input gate
    - decides what information is relevant to add from the current step
  - Output gate
    - decides what the next hidden state should be

Forget Gate



Credit: https://towardsdatascience.com

Input Gate



C_{t-1} **previous cell state**

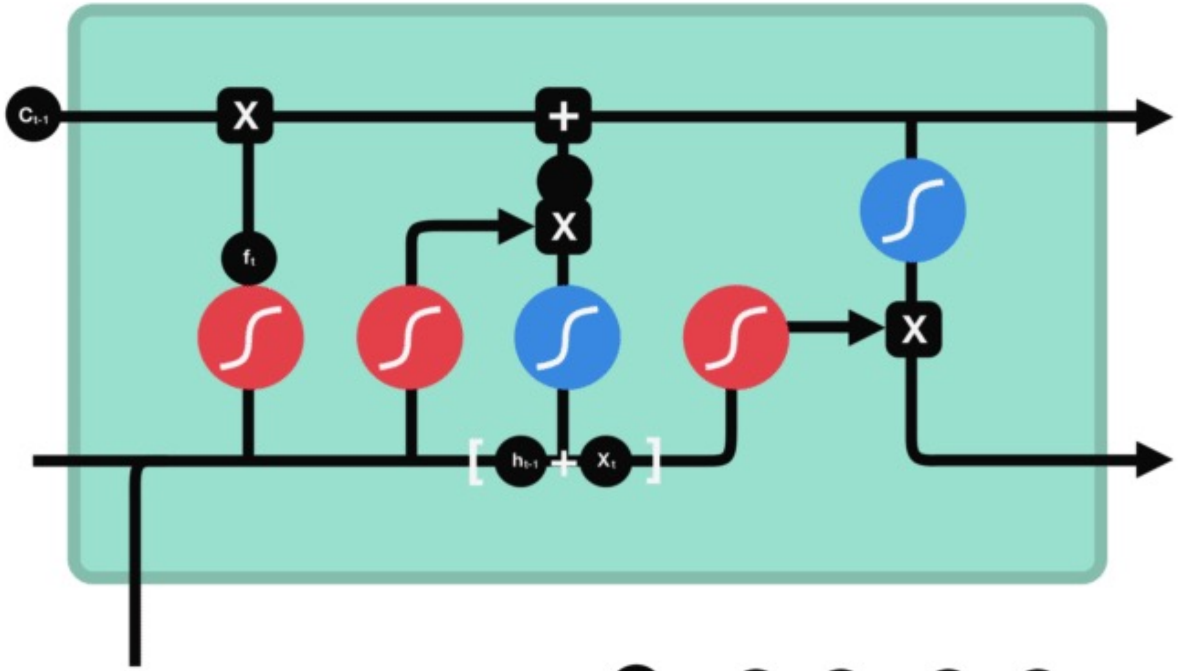
f_t **forget gate output**

i_t **input gate output**

$\tilde{C}_t$ **candidate**

Credit: https://towardsdatascience.com

Cell State

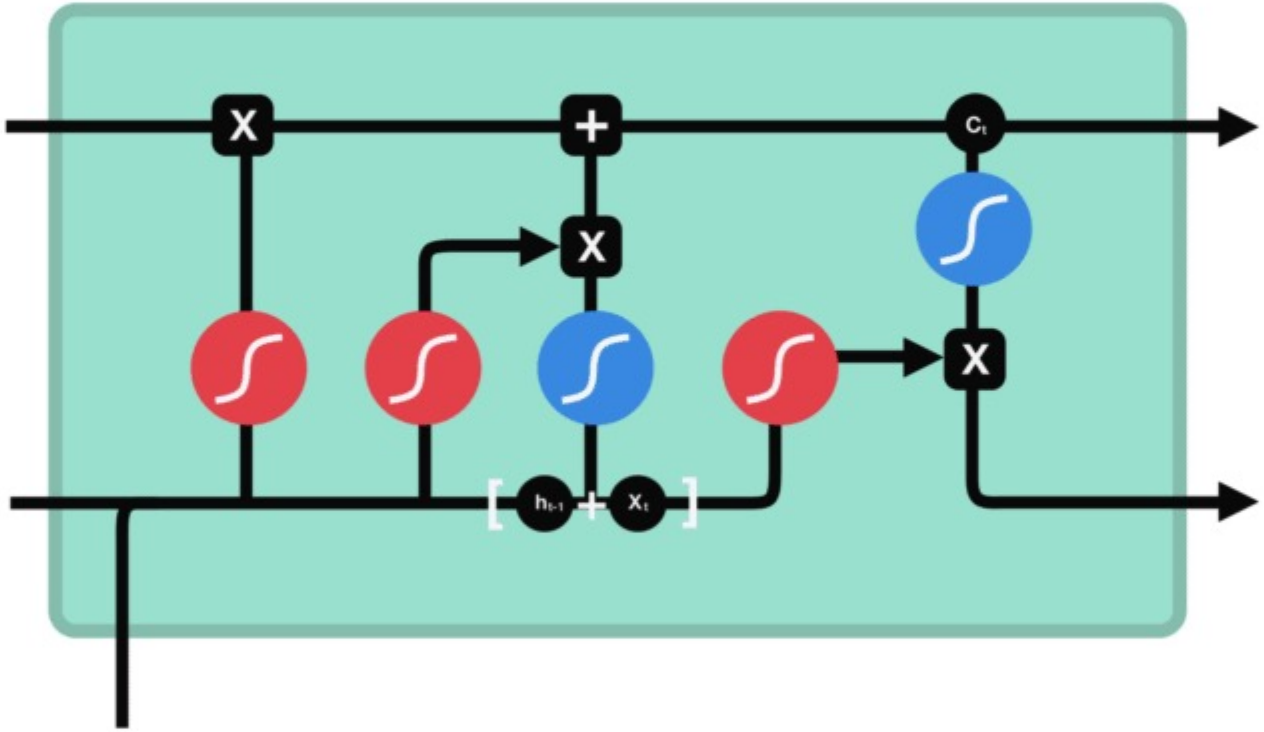


- C_{t-1} previous cell state
- f_t forget gate output
- i_t input gate output
- $\tilde{C}_t$ candidate
- C_t new cell state

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

Credit: https://towardsdatascience.com

Output Gate



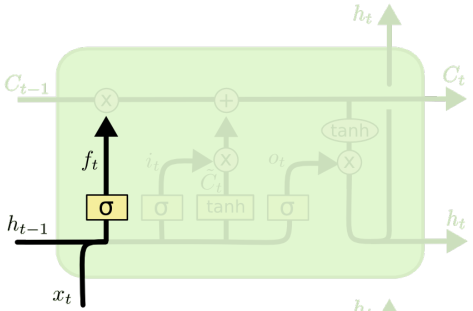
C_{t-1}	previous cell state
f_t	forget gate output
i_t	input gate output
$\tilde{C}_t$	candidate
C_t	new cell state
o_t	output gate output
h_t	hidden state

Credit: https://towardsdatascience.com

LSTM: input and new memory

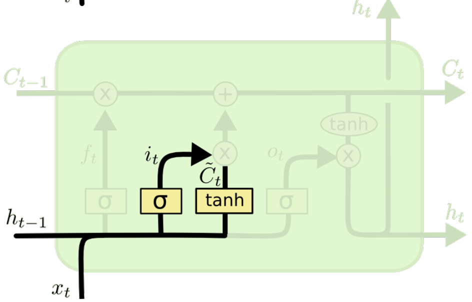
LSTM cells takes the following input vectors:

- the input x_t
- the past output h_{t-1}
- the past cell state c_{t-1}



Forget Gate

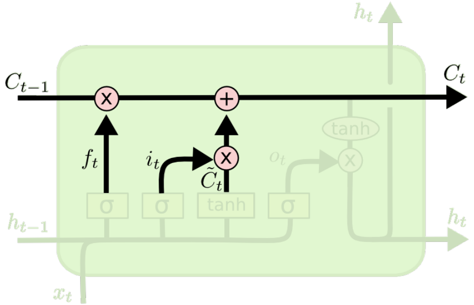
$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \ + \ b_f\right)$$



Input Gate

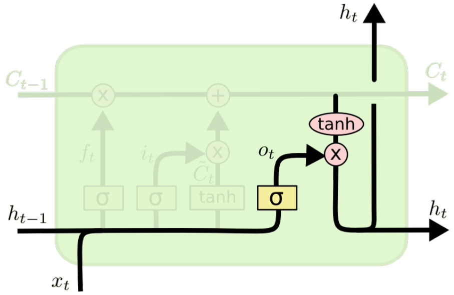
$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \ + \ b_i\right)$$

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



Cell State

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



Output Gate

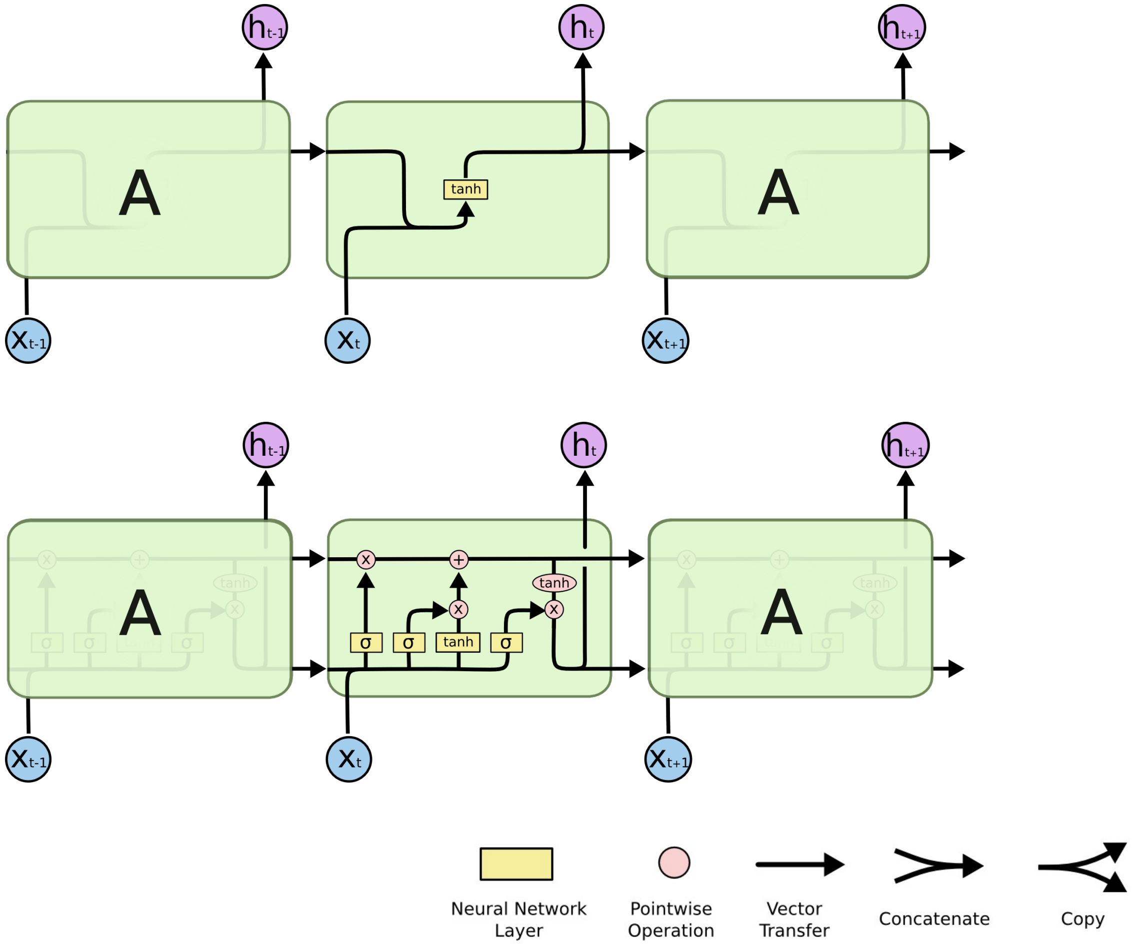
$$o_t = \sigma\left(W_o \ [h_{t-1}, x_t] \ + \ b_o\right)$$

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

Standard RNNs to LSTM

Standard

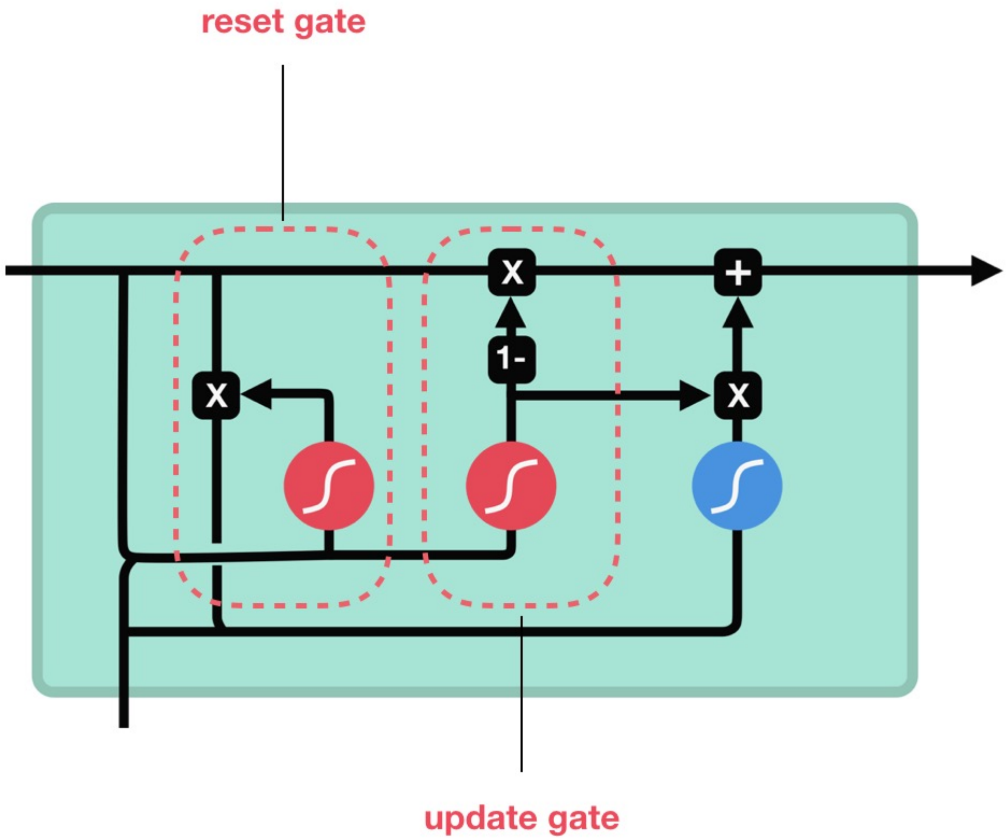
LSTM



Gated Recurrent Units (GRU)

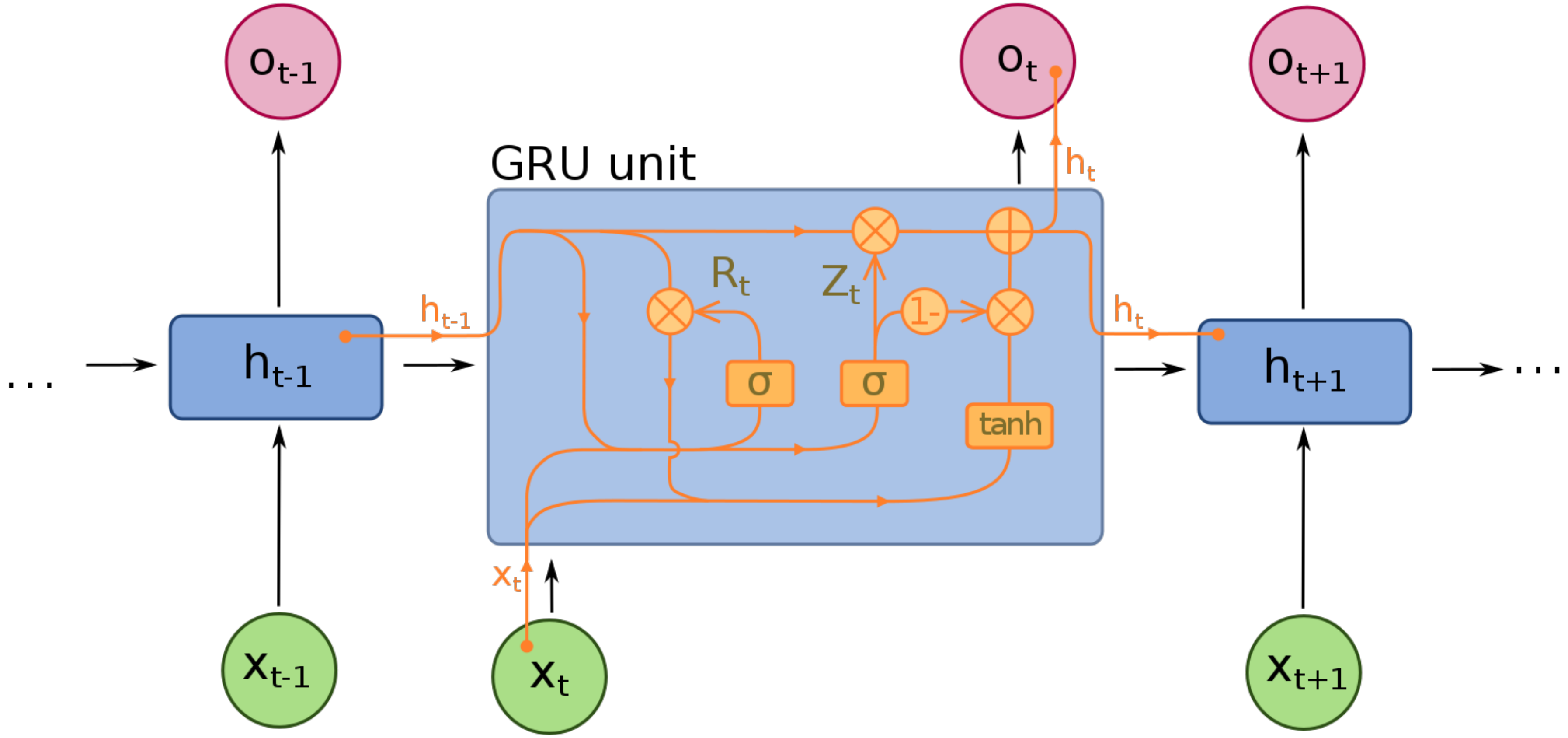
- GRU is the newer generation of Recurrent Neural networks and is pretty similar to an LSTM
- GRU’s got rid of the cell state and used the hidden state to transfer information
- It only has two gates,
    - an update gate
        - acts similar to the forget and input gate of an LSTM
        - decides what information to throw away and what new information to add
    - a reset gate
        - decide how much past information to forget

Gated Recurrent Units (GRU)



Credit: https://towardsdatascience.com

GRU Unit



Credit https://en.wikipedia.org/wiki/Recurrent_neural_network

A mostly complete chart of

Neural Networks