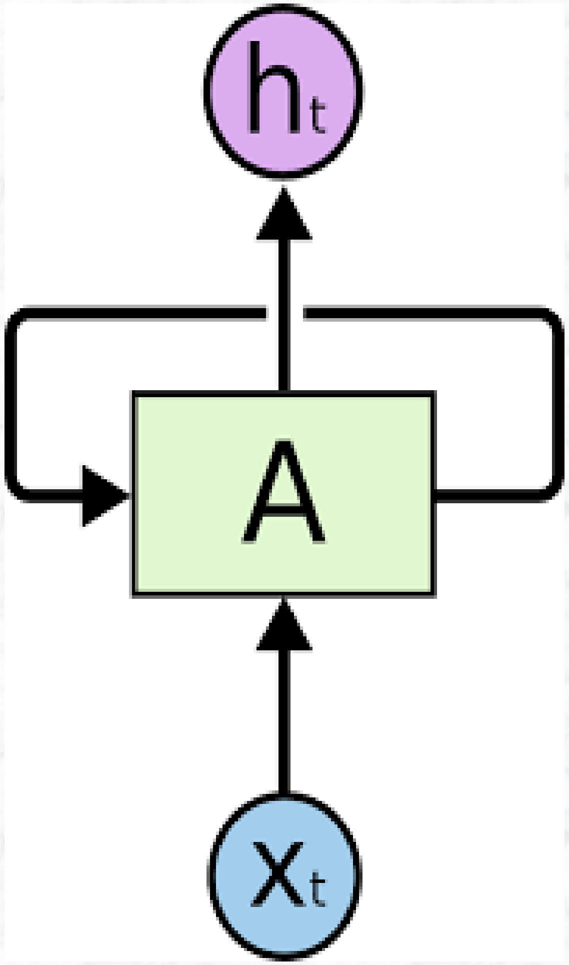
Recurrent Neutal network

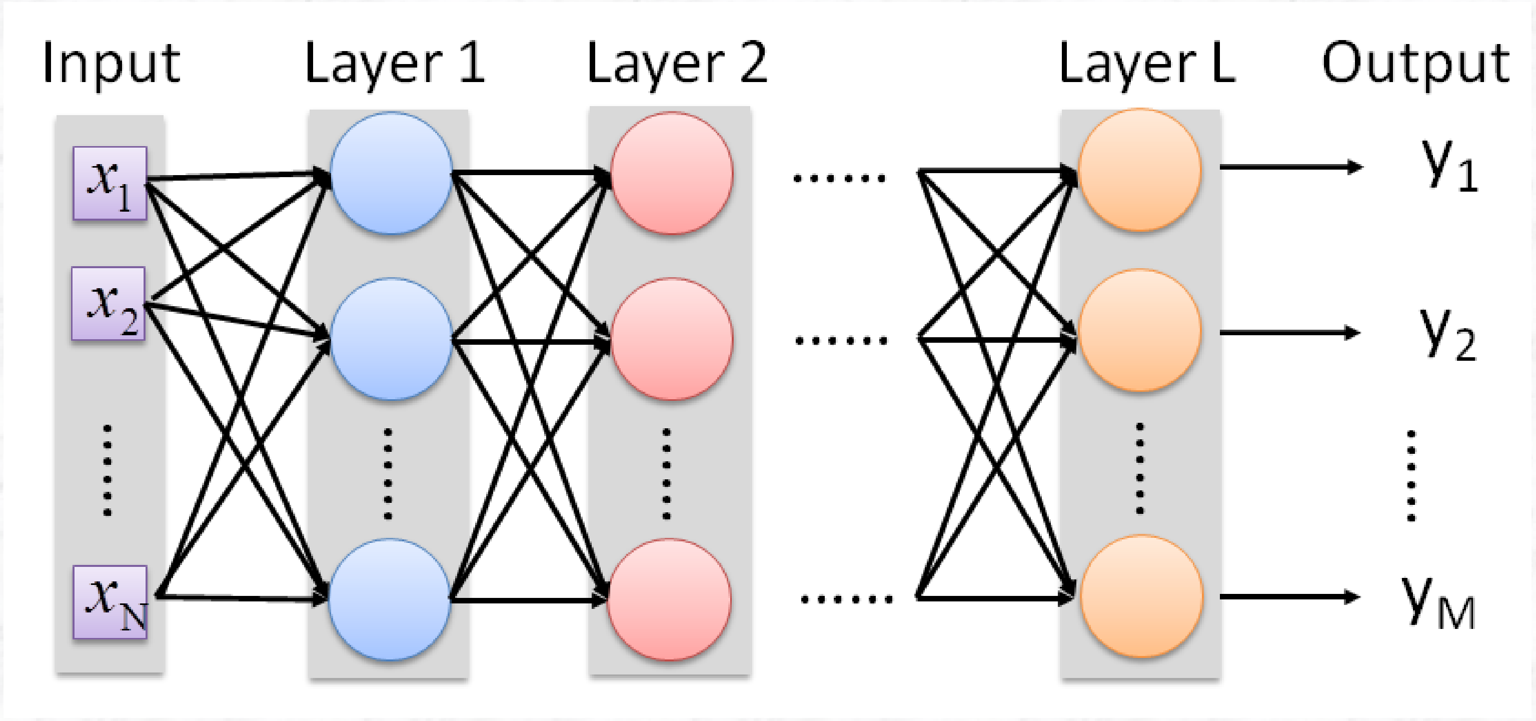


RNN

Instructor : Dr. Hanaa Bayomi Ali
Mail : h.mobarz @ fci-cu.edu.eg

RNN VS Vanilla

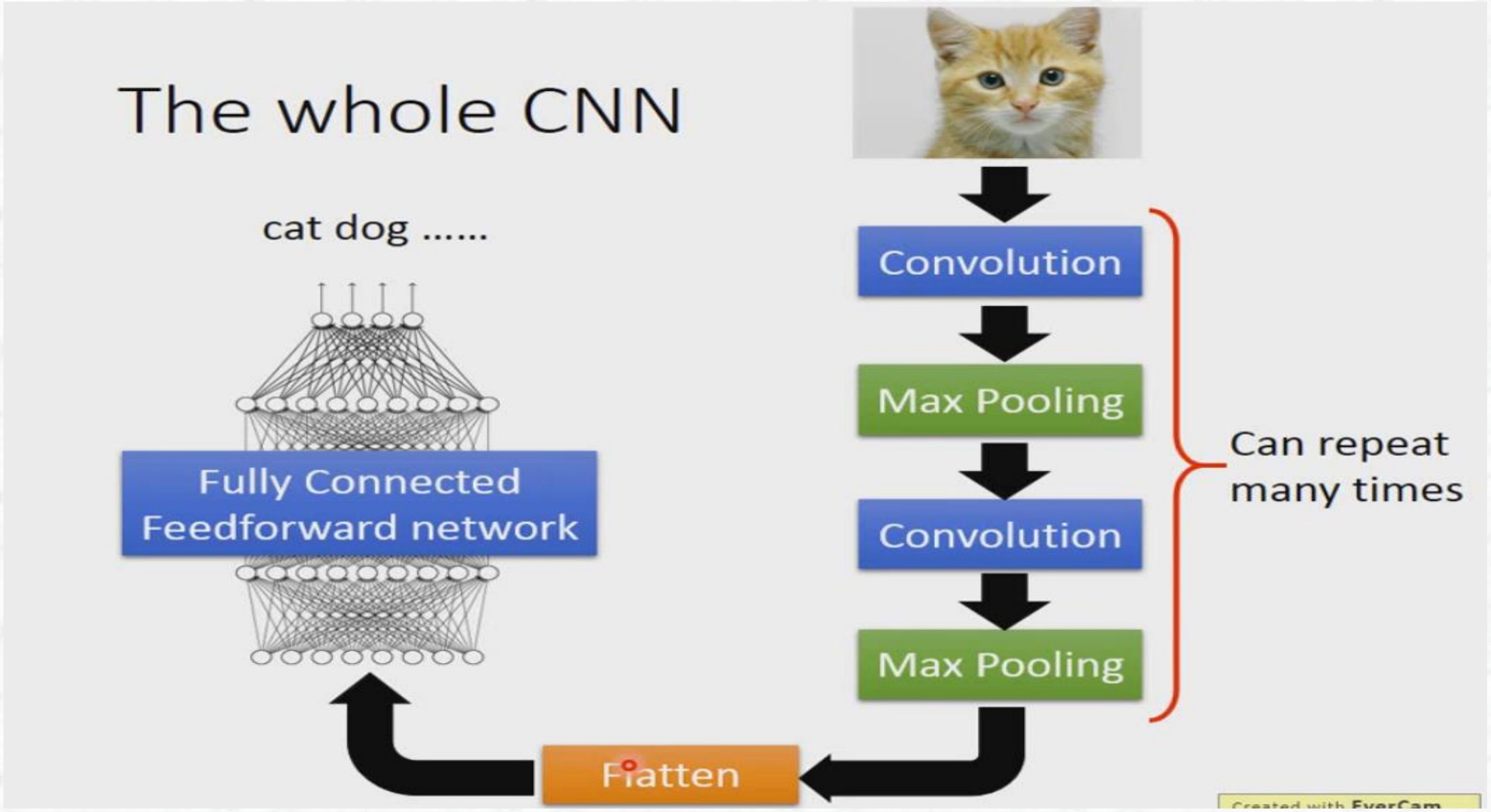
Vanilla



- pass all input in the same time
- inputs are independent in each other
- fixed input and fixed output
- using different parameters with different layers in the network

RNN VS CNN

Convolution NN

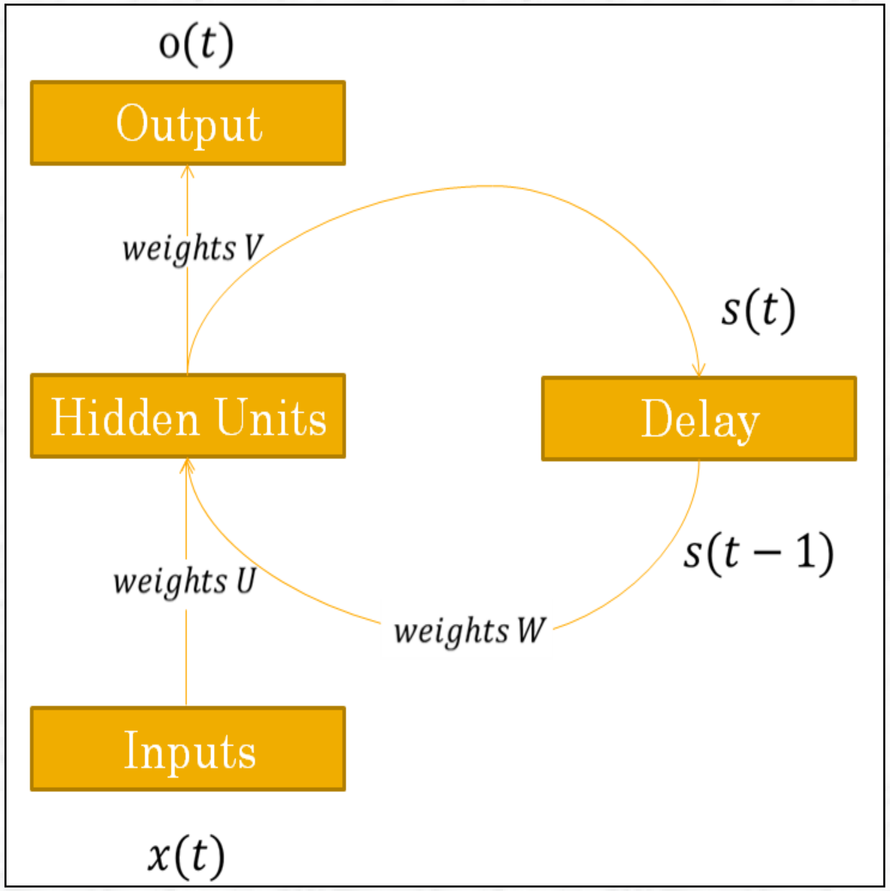


•features to be captured regardless of where they are in the input sample

RNN architecture

- RNNs are called *recurrent* because they perform the same task for every element of a sequence, with the output being depended on the previous computations (memory).

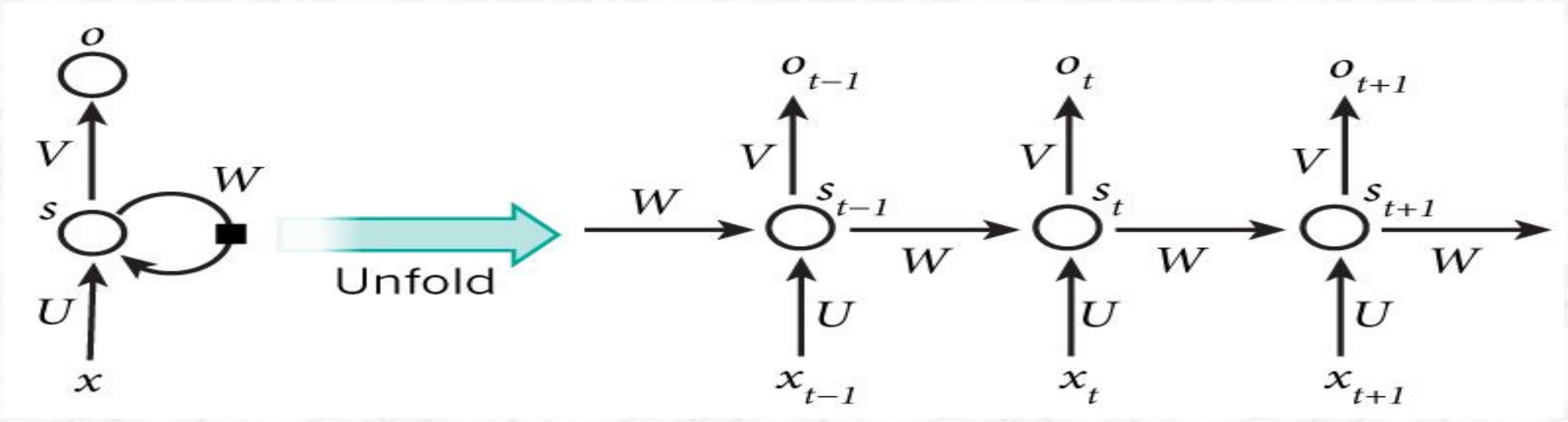
- Inputs ***x(t)*** outputs ***y(t)*** hidden state ***s(t)*** the memory of the network
A delay unit is introduced to hold activation until they are processed at the next step.



- The decision a recurrent net reached at time step ***t-1*** affects the decision it will reach one moment later at time step ***t***. So recurrent networks *have two sources of input*, **the present and the recent past**, which combine to determine how they respond to new data

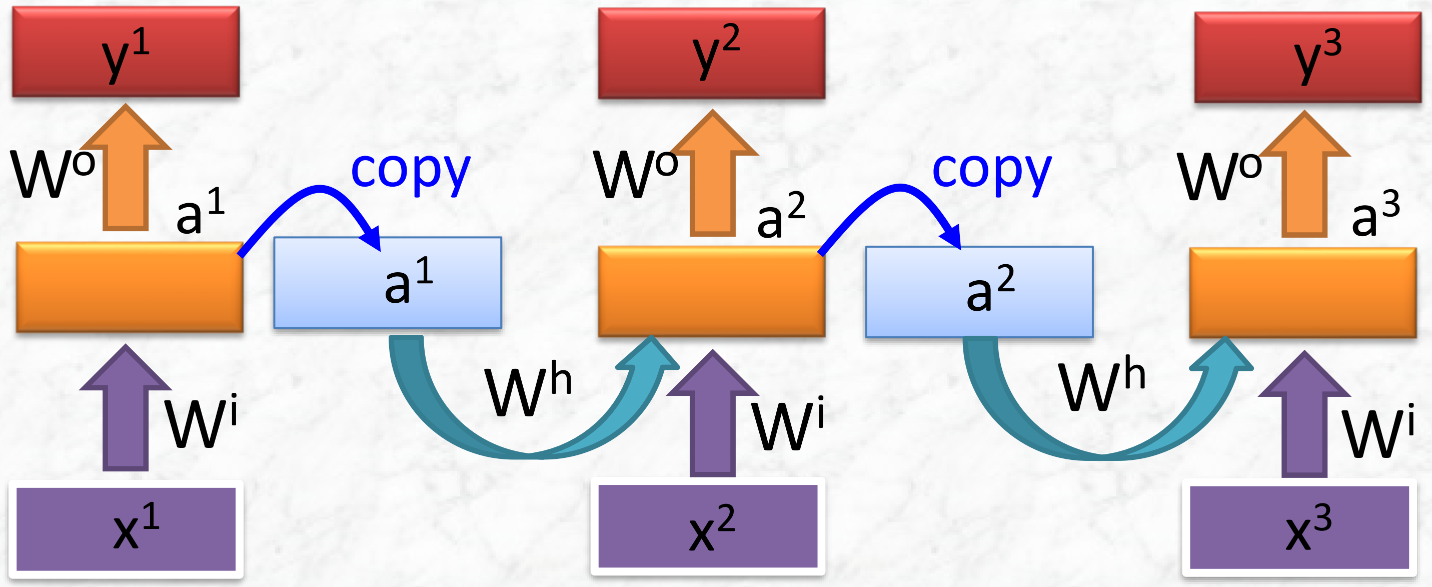
What are RNNs?

- The recurrent network can be converted into a feed-forward network by ***unfolding over time***



An unfolded recurrent network. Each node represents a layer of network units at a single time step. The weighted connections from the input layer to hidden layer are labelled ‘w1’, those from the hidden layer to itself (i.e. the recurrent weights) are labelled ‘w2’ and the hidden to output weights are labelled‘w3’. Note that the same weights are reused at every time step. Bias weights are omitted for clarity.

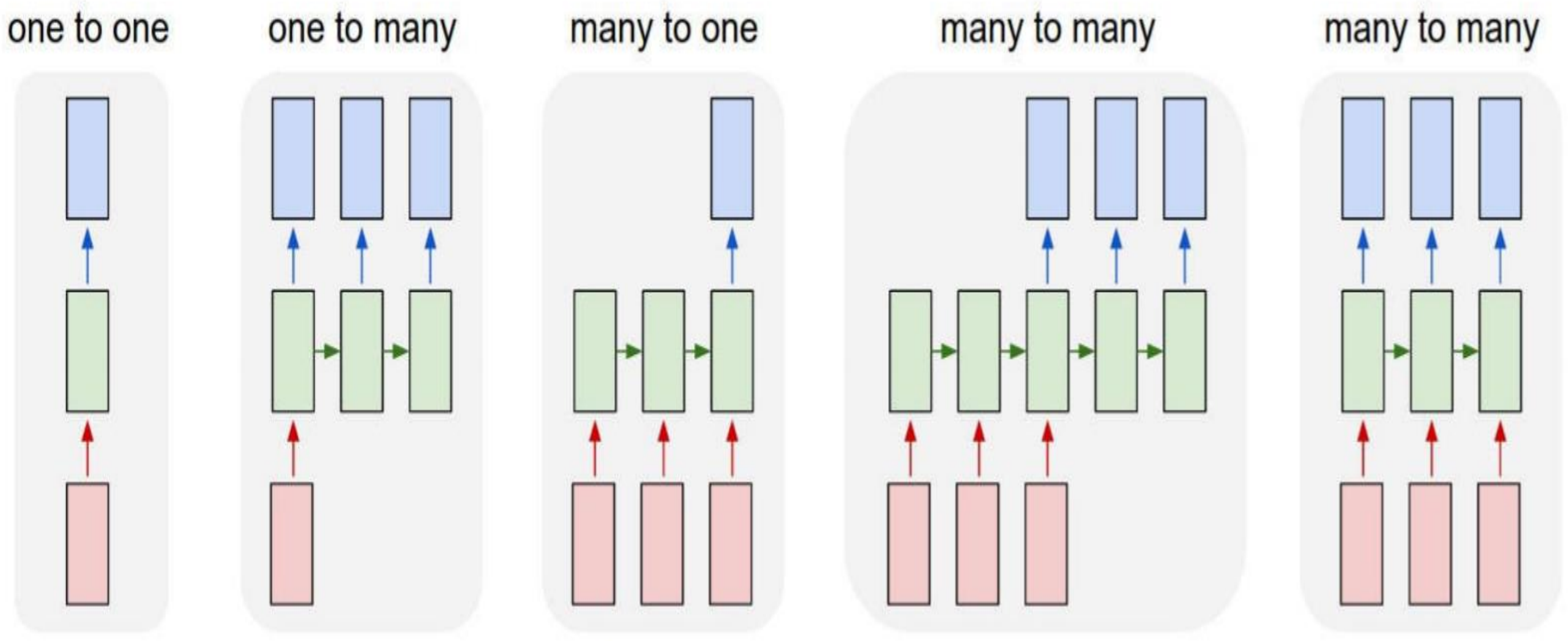
RNN



The same network is used again and again.

Output y^i depends on x^1, x^2, …… x^i

Recurrent Networks offer a lot of flexibility:



(1) fixed-sized input to fixed-sized output (e.g. image classification)

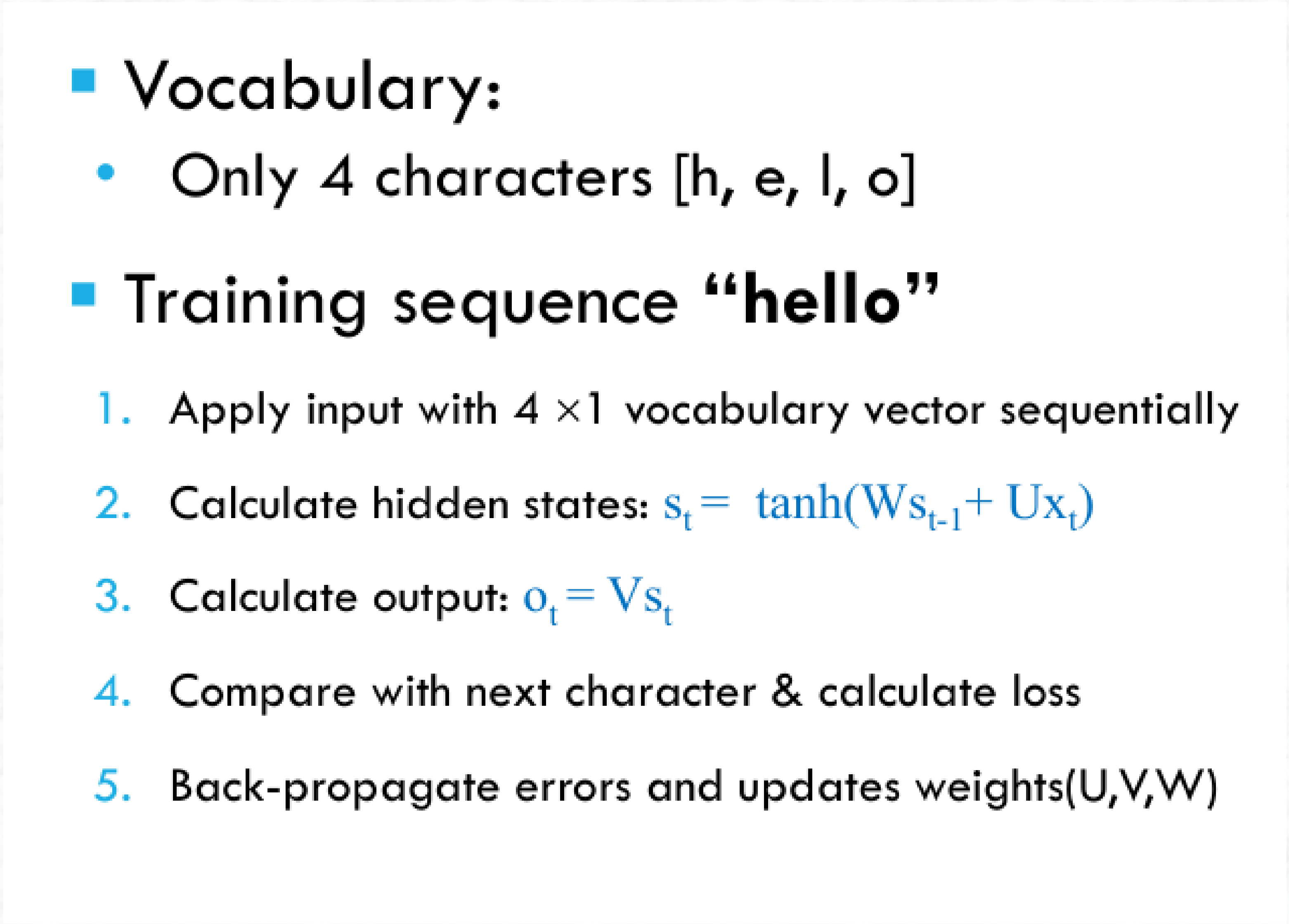
(2) Sequence output (e.g. image captioning takes an image and outputs a sentence of words).

(3) Sequence input (e.g. sentiment analysis where a given sentence is classified as expressing positive or negative sentiment).

(4) Sequence input and sequence output (e.g. Machine Translation: an RNN reads a sentence in English and then outputs a sentence in French).

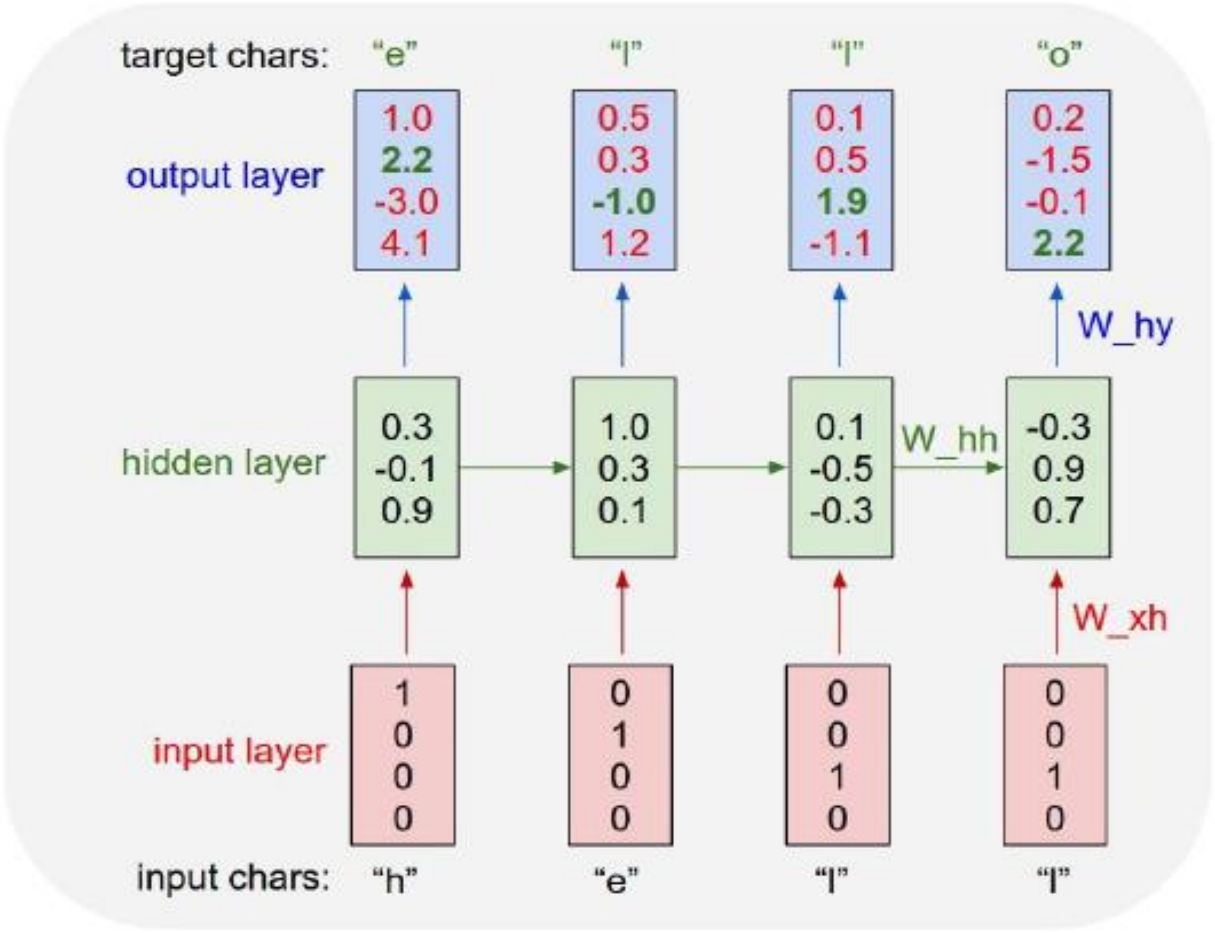
(5) Synced sequence input and output (e.g. video classification where we wish to label each frame of the video).

CHARACTER-LEVEL LANGUAGE MODEL



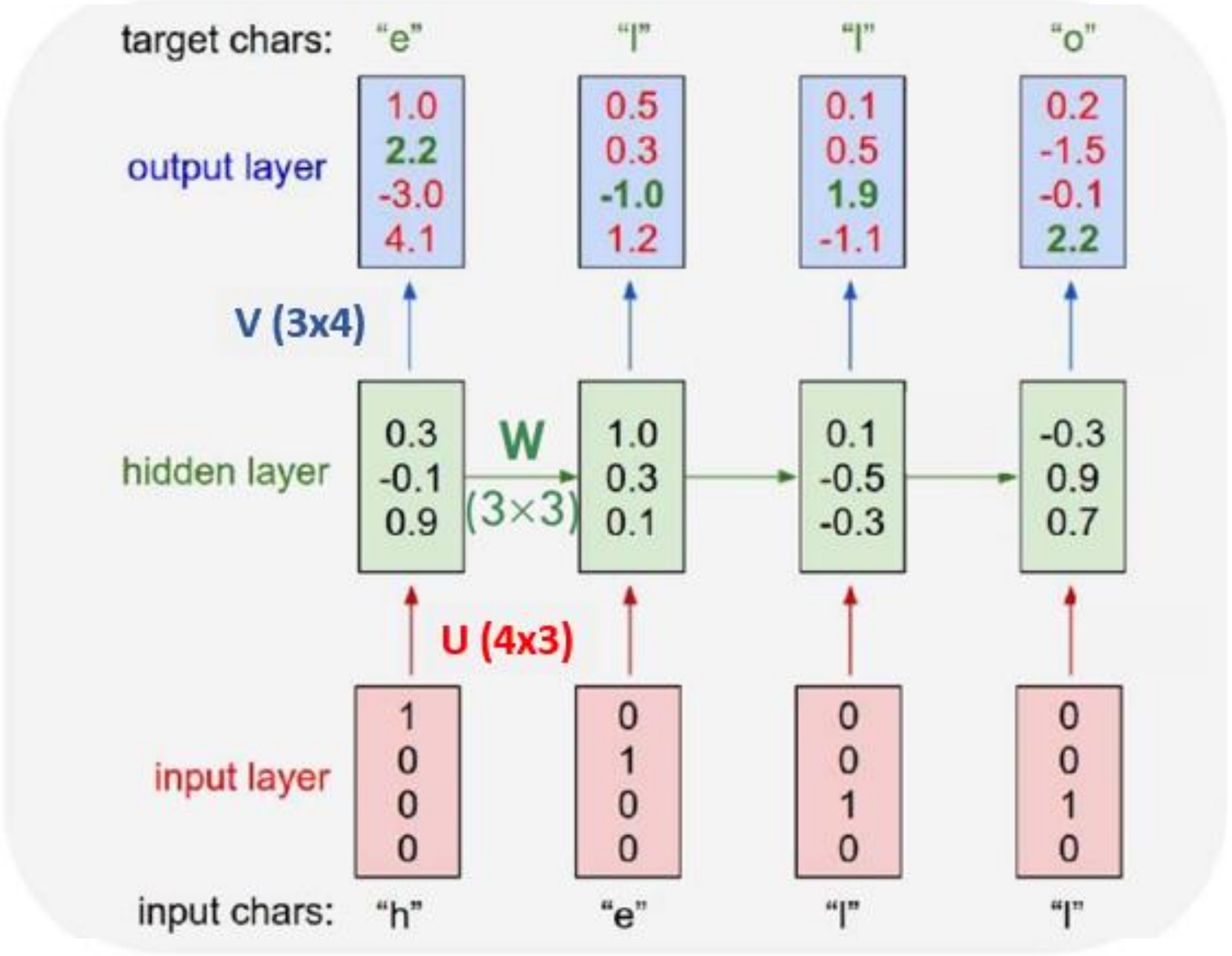
- Vocabulary:
  - Only 4 characters [h, e, l, o]
- Training sequence "**hello**"
  1. Apply input with 4 ×1 vocabulary vector sequentially
  2. Calculate hidden states: $s_t = \tanh(Ws_{t-1} + Ux_t)$
  3. Calculate output: $o_t = Vs_t$
  4. Compare with next character & calculate loss
  5. Back-propagate errors and updates weights(U,V,W)

RNN for Character-Level Language Models



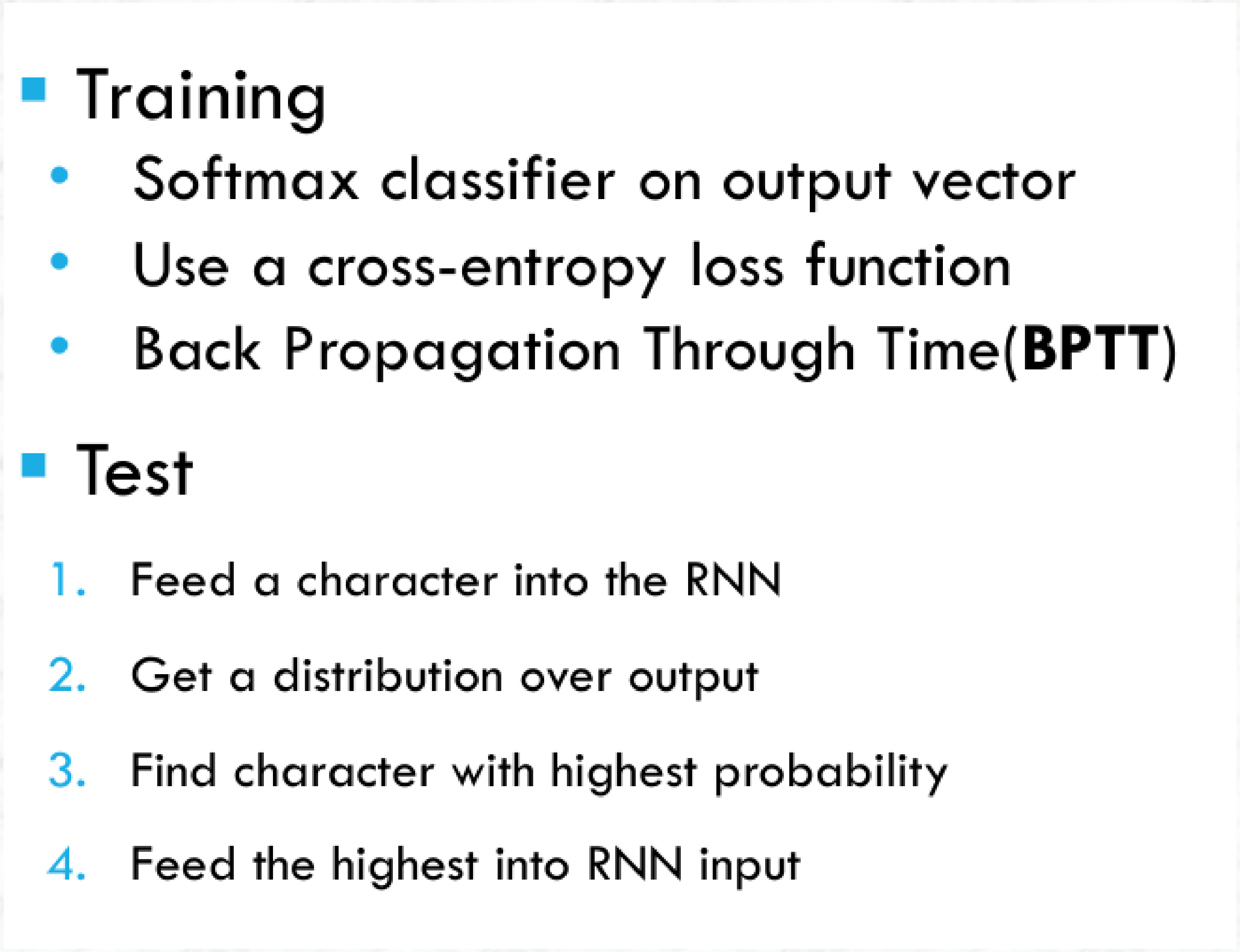
An example RNN with 4-dimensional input and output layers, and a hidden layer of 3 units (neurons). This diagram shows the activations in the forward pass when the RNN is fed the characters "hell" as input. The output layer contains confidences the RNN assigns for the next character (vocabulary is "h,e,l,o"); We want the green numbers to be high and red numbers to be low. …… Andrej Karpathy 2015

RNN for Character-Level Language Models



It uses the standard Softmax classifier (also commonly referred to as the cross-entropy loss) on every output vector simultaneously. The RNN is trained with mini-batch Stochastic Gradient Descent and I like to use RMSProp or Adam (per-parameter adaptive learning rate methods) to stabilize the updates.

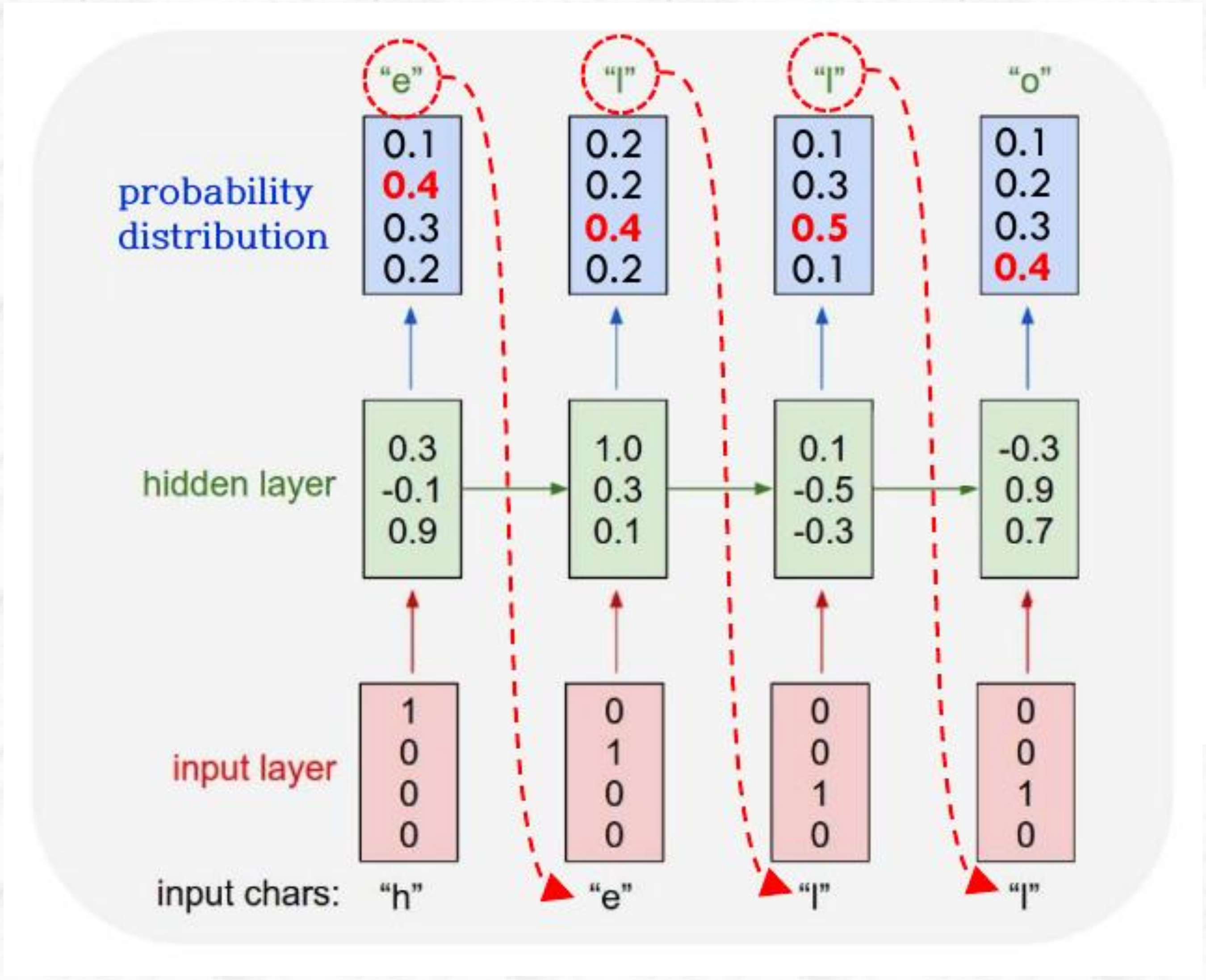
CHARACTER-LEVEL LANGUAGE MODEL

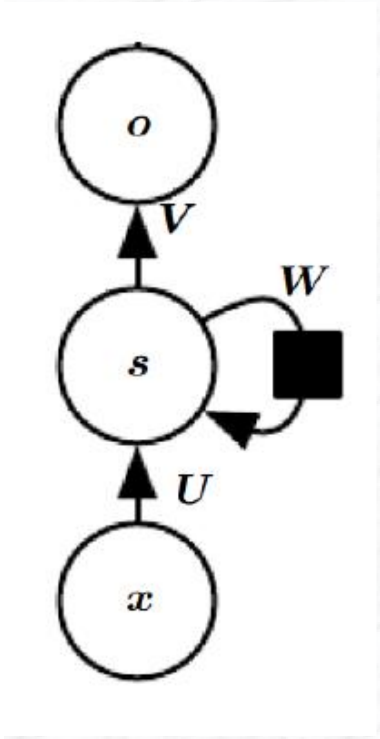


- Training
  - Softmax classifier on output vector
  - Use a cross-entropy loss function
  - Back Propagation Through Time(**BPTT**)
- Test

1. Feed a character into the RNN
2. Get a distribution over output
3. Find character with highest probability
4. Feed the highest into RNN input

CHARACTER-LEVEL LANGUAGE MODEL (Generative Model)

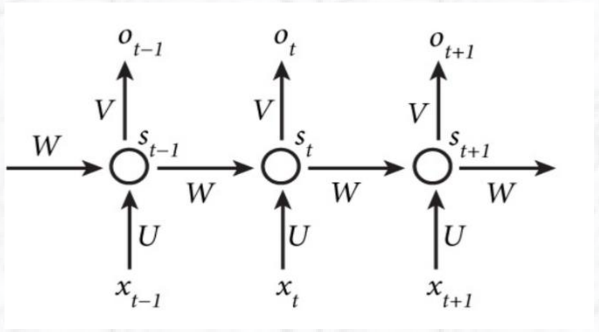




What are RNNs?

Training RNNs (determine the parameters) •

Back Propagation Through Time (BPTT) is often used to learn the RNN
BPTT is an extension of the back-propagation (BP)



The output of this RNN is $\hat{y}_t$ ●

$$s_t = \tanh(Ux_t + Ws_{t-1})$$
$$\hat{y}_t = softmax(Vs_t)$$

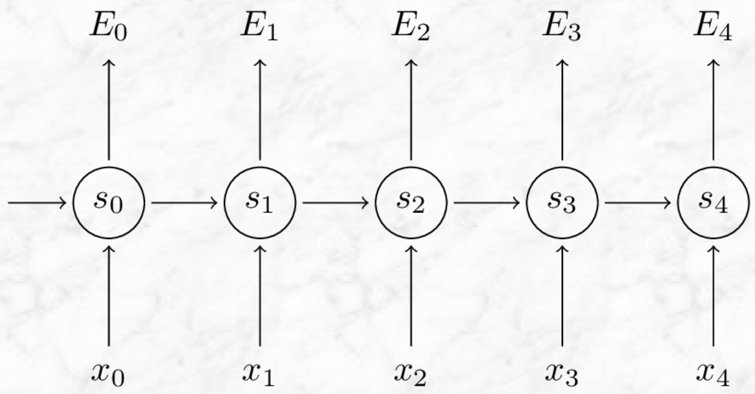
The loss/error function of this network is ●

$$E_t(y_t, \hat{y}_t) = -y_t \log \hat{y}_t \longrightarrow$$

The error at each time step
"the cross entropy loss"

$$E(y, \hat{y}) = \sum_t E_t(y_t, \hat{y}_t) \longrightarrow$$

the total loss is the sum of the errors at each time step



What are RNNs?

- Training RNNs (determine the parameters)

The gradients of the error with respect to our parameters ✓

Just like we sum up the errors, we also sum up the gradients at each time step for one training example. For parameter W, the gradient is

$$\frac{\partial E}{\partial W} = \sum_t \frac{\partial E_t}{\partial W}$$

✓ The gradient at each time step

we use time 3 as an example

$$\frac{\partial E_3}{\partial W} = \frac{\partial E_3}{\partial \hat{y}_3} \frac{\partial \hat{y}_3}{\partial s_3} \frac{\partial s_3}{\partial W} \longrightarrow \boxed{\text{Chain Rule}}$$

$$s_3 = \tanh(Ux_1 + Ws_2) \longrightarrow \boxed{s_3 \text{ depends on } W \text{ and } s_1\text{, we cannot simply treat } s_2 \text{ a constant}}$$

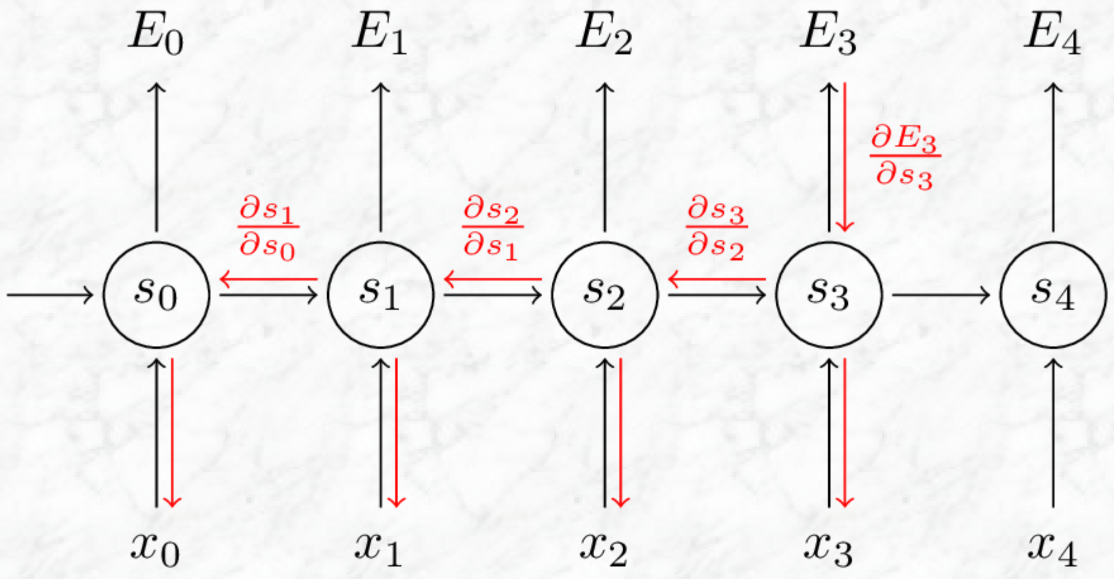
$$\frac{\partial E_3}{\partial W} = \sum_{k=0}^{3} \frac{\partial E_3}{\partial \hat{y}_3} \frac{\partial \hat{y}_3}{\partial s_3} \frac{\partial s_3}{\partial s_k} \frac{\partial s_k}{\partial W} \longrightarrow \boxed{\text{Apply Chain Rule again on } s_k}$$

What are RNNs?

- Training RNNs (determine the parameters)

$$\frac{\partial E_3}{\partial W} = \sum_{k=0}^{3} \frac{\partial E_3}{\partial \hat{y}_3} \frac{\partial \hat{y}_3}{\partial s_3} \frac{\partial s_3}{\partial s_k} \frac{\partial s_k}{\partial W}$$

Becaise W is used in every step up to the output we care about, we need to back-propagate gradients from $t = 3$ through the network all the way to $t = 0$

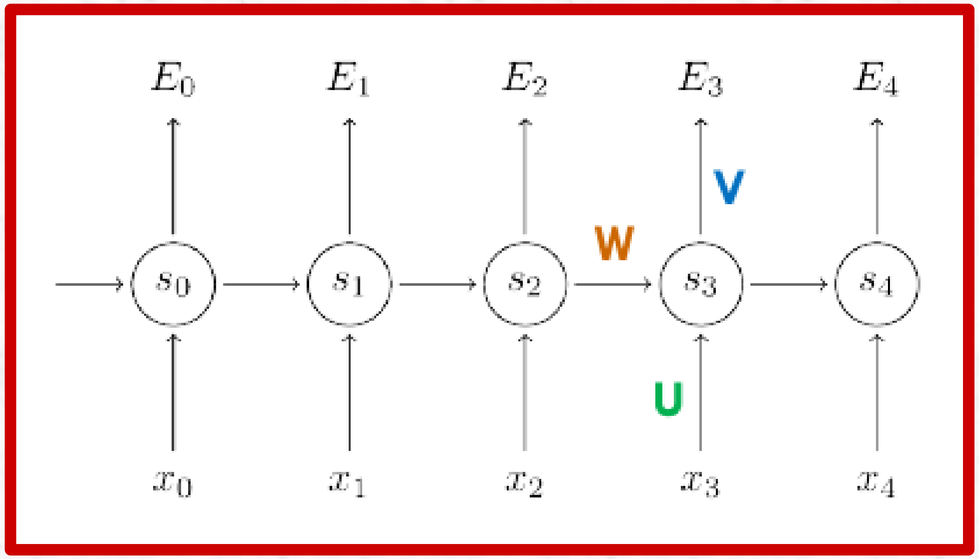


E_0
E_1
E_2
E_3
E_4
$\frac{\partial E_3}{\partial s_3}$
$\frac{\partial s_1}{\partial s_0}$
$\frac{\partial s_2}{\partial s_1}$
$\frac{\partial s_3}{\partial s_2}$
s_0
s_1
s_2
s_3
s_4
x_0
x_1
x_2
x_3
x_4

BACKPROPAGATION THROUGH TIME (BPTT)

Calculate gradients of error with respect to: **U, V, W.**

$$s_3 = \tanh(\mathrm{W}s_2 + \mathrm{U}x_3)$$
$$s_2 = \tanh(\mathrm{W}s_1 + \mathrm{U}x_2)$$
$$s_1 = \tanh(\mathrm{W}s_0 + \mathrm{U}x_1)$$



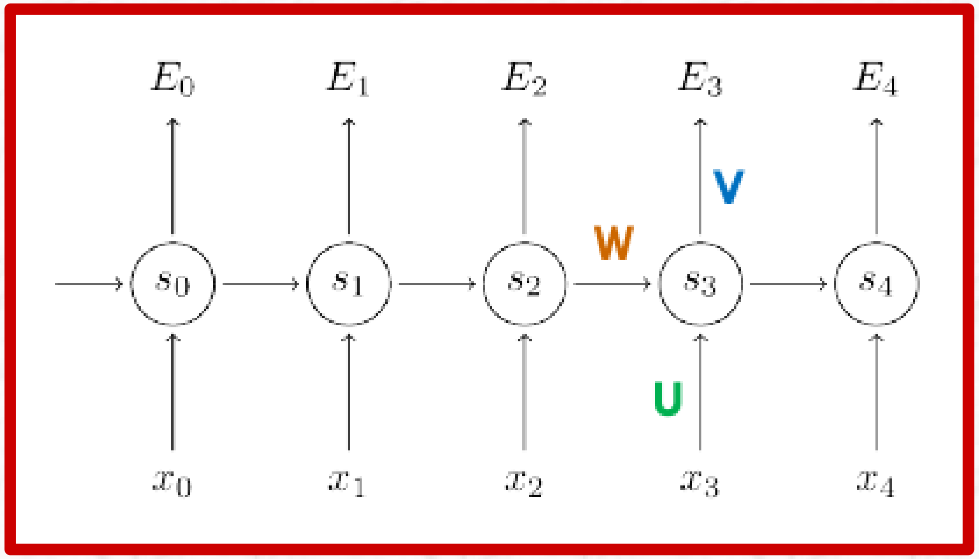
Sum up the gradients at each time step

$$\frac{\partial E}{\partial U} = \sum_t \frac{\partial E_t}{\partial U}, \quad \frac{\partial E}{\partial V} = \sum_t \frac{\partial E_t}{\partial V}, \quad \frac{\partial E}{\partial W} = \sum_t \frac{\partial E_t}{\partial W}$$

BACKPROPAGATION THROUGH TIME (BPTT)

Calculate Gradients by chain Rule

$$\frac{\partial E_3}{\partial V} = \frac{\partial E_3}{\partial \hat{y}_3} \cdot \frac{\partial \hat{y}_3}{\partial o_3} \cdot \frac{\partial o_3}{\partial V}$$



Remember:

$$E_t(y_t, \hat{y}_t) = -\,y_t \log \hat{y}_t$$

$$\hat{y}_t = \text{softmax}(o_t) = \text{softmax}(Vs_t)$$

$$o_t = Vs_t$$

$$s_t = \tanh(Ws_{t-1} + Ux_t)$$

$$s_3 = \tanh(Ws_2 + Ux_3)$$
$$s_2 = \tanh(Ws_1 + Ux_2)$$
$$s_1 = \tanh(Ws_0 + Ux_1)$$

Softmax f^n

$$\hat{y}_k = \frac{e^{a_k}}{\sum_{k'=1}^{K} e^{a_{k'}}} \quad \text{for } k = 1 \text{ to } k = K.$$

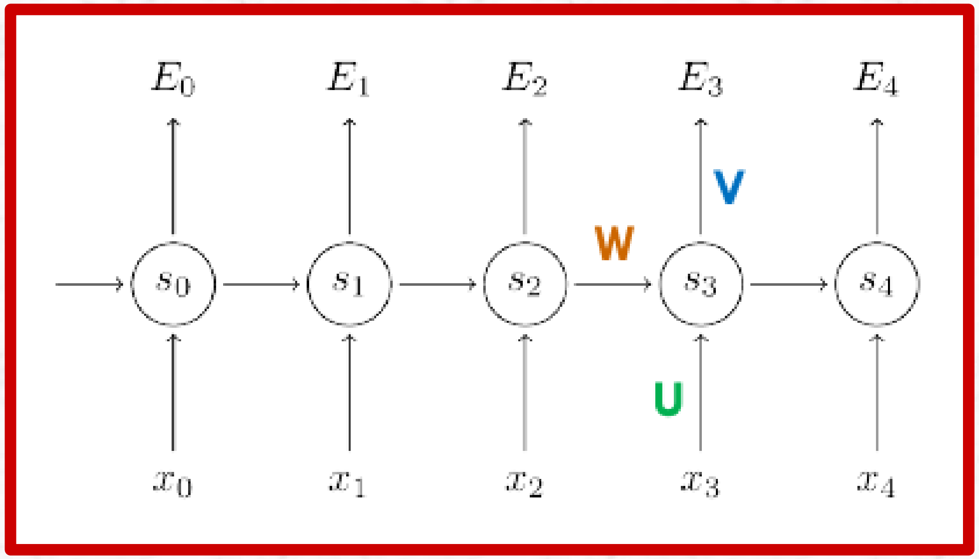
BACKPROPAGATION THROUGH TIME (BPTT)

$$\frac{\partial E_3}{\partial W} = \frac{\partial E_3}{\partial \hat{y}_3} \cdot \frac{\partial \hat{y}_3}{\partial s_3} \cdot \frac{\partial s_3}{\partial W}$$

$$\frac{\partial s_3}{\partial W} \ ?$$

S$_1$ and S$_2$ Depends on W and U too

$$\frac{\partial s_3}{\partial W} \rightarrow \sum_k \frac{\partial s_3}{\partial s_k} \cdot \frac{\partial s_k}{\partial W}$$



$$\frac{\partial E_3}{\partial W} = \sum_k \frac{\partial E_3}{\partial \hat{y}_3} \cdot \frac{\partial \hat{y}_3}{\partial s_3} \cdot \frac{\partial s_3}{\partial s_k} \cdot \frac{\partial s_k}{\partial W}$$

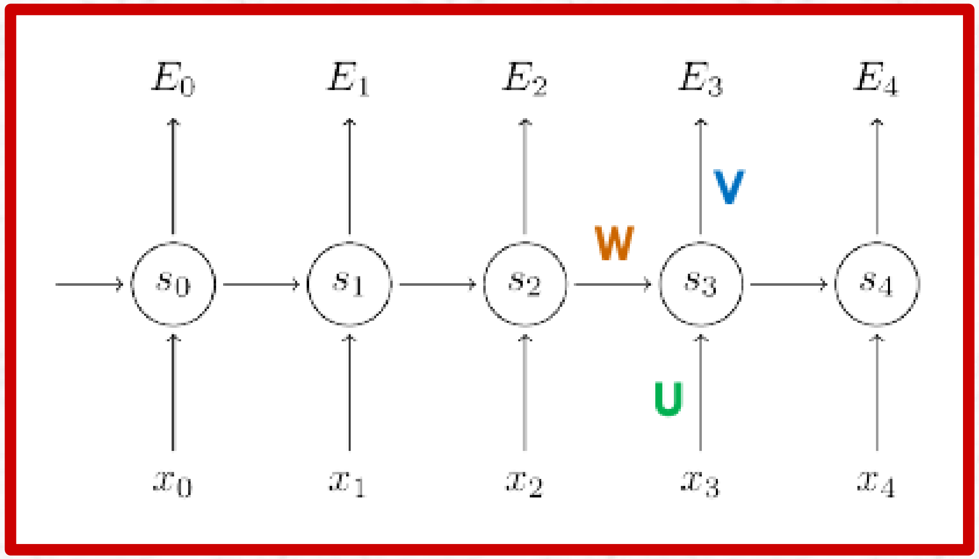
$$s_3 = \tanh(Ws_2 + Ux_3)$$
$$s_2 = \tanh(Ws_1 + Ux_2)$$
$$s_1 = \tanh(Ws_0 + Ux_1)$$

$$\frac{\partial E_3}{\partial W} = \frac{\partial E_3}{\partial \hat{y}_3} \cdot \frac{\partial \hat{y}_3}{\partial s_3} \cdot \frac{\partial s_3}{\partial W} + \frac{\partial E_3}{\partial \hat{y}_3} \cdot \frac{\partial \hat{y}_3}{\partial s_3} \cdot \frac{\partial s_3}{\partial s_2} \cdot \frac{\partial s_2}{\partial W} + \frac{\partial E_3}{\partial \hat{y}_3} \cdot \frac{\partial \hat{y}_3}{\partial s_3} \cdot \frac{\partial s_3}{\partial s_1} \cdot \frac{\partial s_1}{\partial W} + \frac{\partial E_3}{\partial \hat{y}_3} \cdot \frac{\partial \hat{y}_3}{\partial s_3} \cdot \frac{\partial s_3}{\partial s_0} \cdot \frac{\partial s_0}{\partial W}$$

BACKPROPAGATION THROUGH TIME (BPTT)

$$\frac{\partial E_3}{\partial U} = \frac{\partial E_3}{\partial \hat{y}_3} \cdot \frac{\partial \hat{y}_3}{\partial s_3} \cdot \frac{\partial s_3}{\partial U}$$

$$\frac{\partial s_3}{\partial U} \; ?$$



S_1 and S_2 Depends on W and U too

$$\frac{\partial s_3}{\partial U} \rightarrow \sum_k \frac{\partial s_3}{\partial s_k} \cdot \frac{\partial s_k}{\partial U}$$

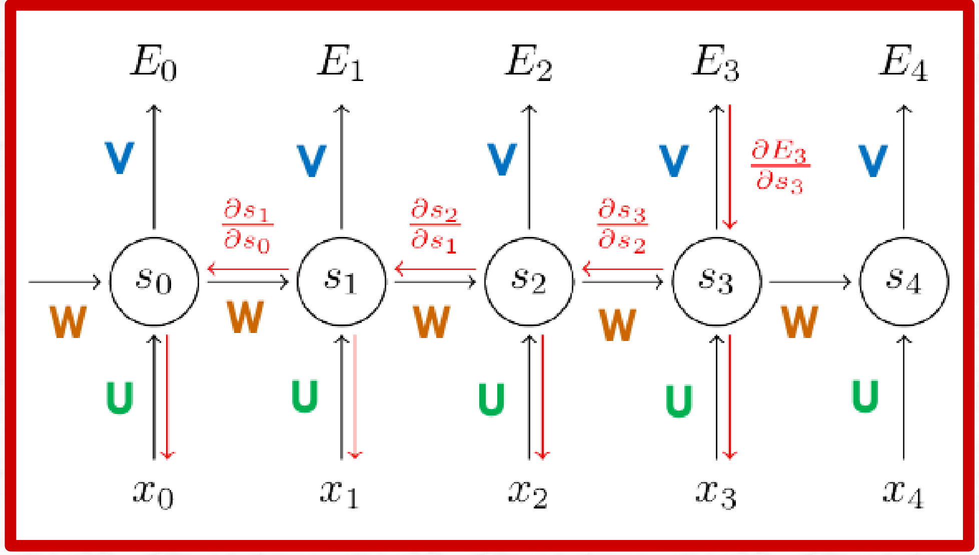
$$\frac{\partial E_3}{\partial U} = \sum_k \frac{\partial E_3}{\partial \hat{y}_3} \cdot \frac{\partial \hat{y}_3}{\partial s_3} \cdot \frac{\partial s_3}{\partial s_k} \cdot \frac{\partial s_k}{\partial U}$$

$$\begin{aligned} s_3 &= \tanh(W s_2 + U x_3) \\ s_2 &= \tanh(W s_1 + U x_2) \\ s_1 &= \tanh(W s_0 + U x_1) \end{aligned}$$

$$\frac{\partial E_3}{\partial U} = \frac{\partial E_3}{\partial \hat{y}_3} \cdot \frac{\partial \hat{y}_3}{\partial s_3} \cdot \frac{\partial s_3}{\partial U} + \frac{\partial E_3}{\partial \hat{y}_3} \cdot \frac{\partial \hat{y}_3}{\partial s_3} \cdot \frac{\partial s_3}{\partial s_2} \cdot \frac{\partial s_2}{\partial U} + \frac{\partial E_3}{\partial \hat{y}_3} \cdot \frac{\partial \hat{y}_3}{\partial s_3} \cdot \frac{\partial s_3}{\partial s_1} \cdot \frac{\partial s_1}{\partial U} + \frac{\partial E_3}{\partial \hat{y}_3} \cdot \frac{\partial \hat{y}_3}{\partial s_3} \cdot \frac{\partial s_3}{\partial s_0} \cdot \frac{\partial s_0}{\partial U}$$

BACKPROPAGATION THROUGH TIME (BPTT)

Sum up the gradients at each time step



E_0
E_1
E_2
E_3
E_4
V
U
W
s_0
s_1
s_2
s_3
s_4
x_0
x_1
x_2
x_3
x_4
$\frac{\partial s_1}{\partial s_0}$
$\frac{\partial s_2}{\partial s_1}$
$\frac{\partial s_3}{\partial s_2}$
$\frac{\partial E_3}{\partial s_3}$

$$\frac{\partial E}{\partial V} = \sum_t \frac{\partial E_t}{\partial V} = \sum_t \frac{\partial E_t}{\partial \hat{y}_t} \cdot \frac{\partial \hat{y}_t}{\partial o_t} \cdot \frac{\partial o_t}{\partial V}$$

$$\frac{\partial E}{\partial W} = \sum_t \frac{\partial E_t}{\partial W} = \sum_t \sum_k \frac{\partial E_t}{\partial \hat{y}_t} \cdot \frac{\partial \hat{y}_t}{\partial s_t} \cdot \frac{\partial s_t}{\partial s_k} \cdot \frac{\partial s_k}{\partial W}$$

$$\frac{\partial E}{\partial U} = \sum_t \frac{\partial E_t}{\partial U} = \sum_t \sum_k \frac{\partial E_t}{\partial \hat{y}_t} \cdot \frac{\partial \hat{y}_t}{\partial s_t} \cdot \frac{\partial s_t}{\partial s_k} \cdot \frac{\partial s_k}{\partial U}$$

Propagation through time (RNN)

=

Propagation through layers (FNN)

What are RNNs?

- The vanishing gradient problem

To understand why, let’s take a closer look at the gradient we calculated above:

$$\frac{\partial E_3}{\partial W} = \sum_{k=0}^{3} \frac{\partial E_3}{\partial \hat{y}_3} \frac{\partial \hat{y}_3}{\partial s_3} \frac{\partial s_3}{\partial s_k} \frac{\partial s_k}{\partial W} \longrightarrow \frac{\partial E_3}{\partial W} = \sum_{k=0}^{3} \frac{\partial E_3}{\partial \hat{y}_3} \frac{\partial \hat{y}_3}{\partial s_3} \underbrace{\prod_{j=k+1}^{3} \frac{\partial s_j}{\partial s_{j-1}}} \frac{\partial s_k}{\partial W}$$

Because the layers and time steps of deep neural networks relate to each other through multiplication, derivatives are susceptible to vanishing

Gradient contributions from “far away” steps become zero, and the state at those steps doesn’t contribute to what you are learning: You end up not learning long-range dependencies.

What are RNNs?

- How to solve the vanishing gradient problem?

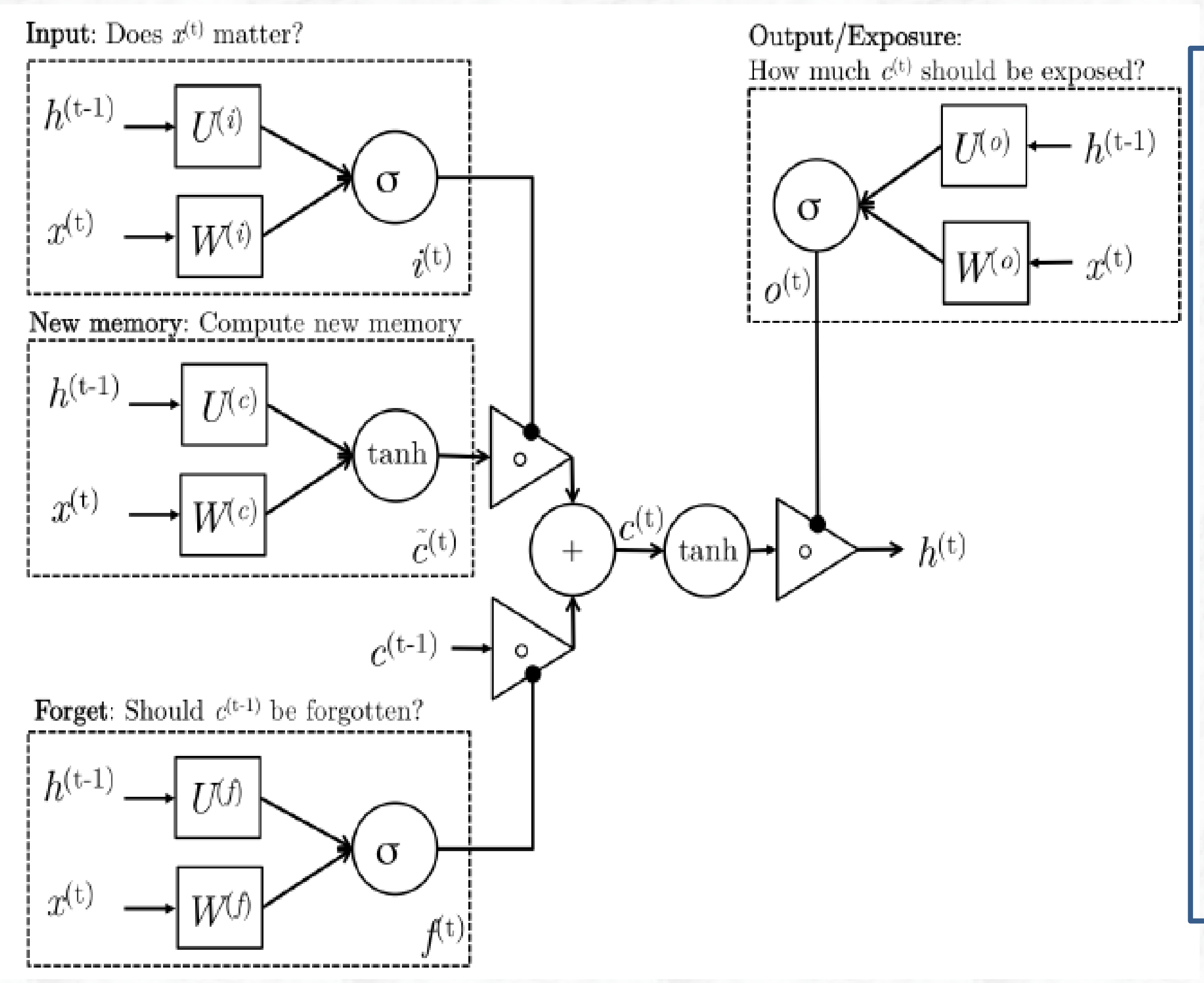
- ☐ Proper initialization of the W matrix can reduce the effect of vanishing gradients
- ☐ Use ReLU instead of tanh or sigmoid activation function

  ReLU derivate is a constant of either 0 or 1, so it isn't likely to suffer from vanishing gradients
- ☐ Use Long Short-Term Memory or Gated Recurrent unit architectures

  LSTM will be introduced later

RNN Extensions: Long Short-term Memory

The vanishing gradient problem prevents standard RNNs from learning long-term dependencies. LSTMs (Hochreiter and Schmidhuber, 1997) were designed to combat vanishing gradients through a *gating* mechanism.



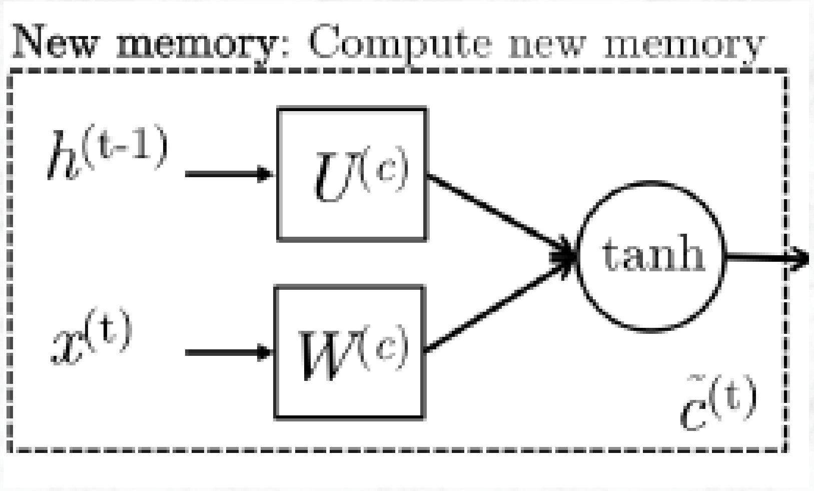
A gating mechanism of the LSTM , which generates the current hidden state by the paste hidden state and current input ..It contains five modules:

•<u>input gate</u>,
•<u>new memory cell</u>,
•<u>forget gate</u>,
•<u>final memory generation</u>,
•<u>output gate</u>.

RNN Extensions: Long Short-term Memory

A gating mechanism of the LSTM

New memory cell



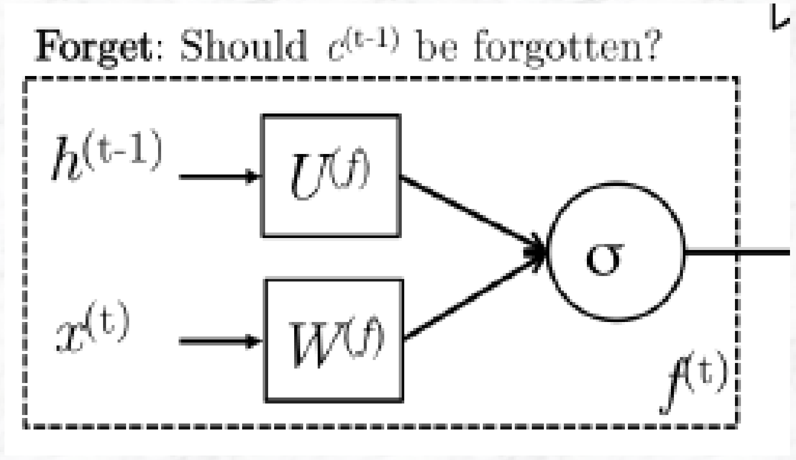
$$\underbrace{\tilde{c}_t}_{\text{New memory}} = \tanh(W^c x_t + U^c h_{t-1})$$

use the input word and the past hidden state to generate a new memory which includes aspects of the new input

RNN Extensions: Long Short-term Memory

A gating mechanism of the LSTM

Forget gate



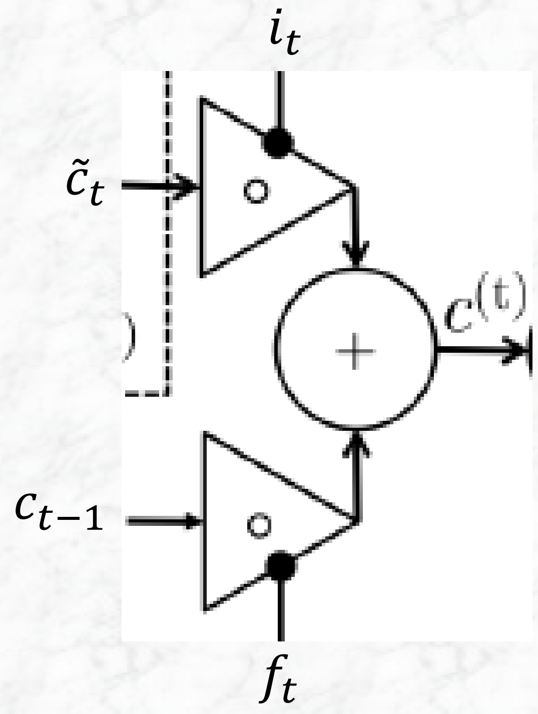
$$f_t = \sigma(W^f x_t + U^f h_{t-1})$$

The forget gate looks at the input word and the past hidden state and makes an assessment on whether the past memory cell is useful for the computation of the current memory cell

RNN Extensions: Long Short-term Memory

A gating mechanism of the LSTM

Final memory cell



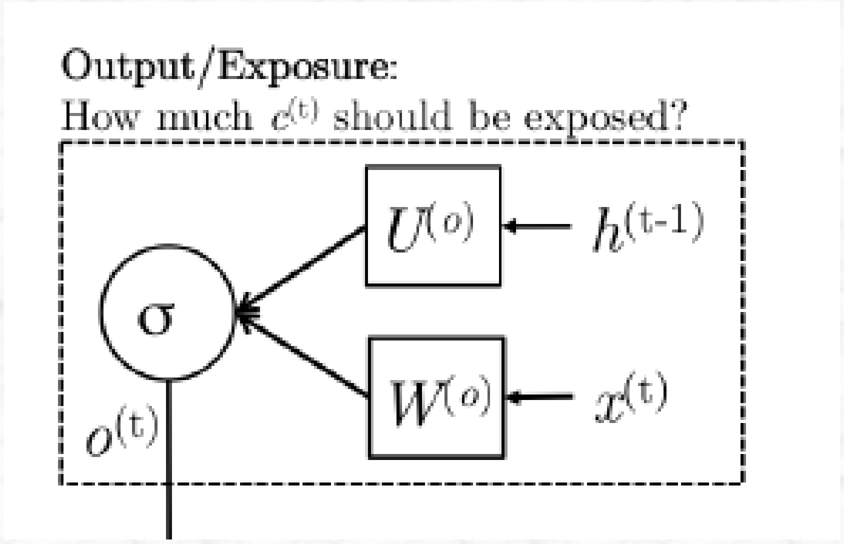
$$c_t = f_t \circ c_{t-1} + i_t \circ \tilde{c}_t$$

This stage first takes the advice of the forget gate f_t and accordingly forgets the past memory c_{t-1}. Similarly, it takes the advice of the input gate i_t and accordingly gates the new memory. It then sums these two results to produce the final memory

RNN Extensions: Long Short-term Memory

A gating mechanism of the LSTM

Output gate



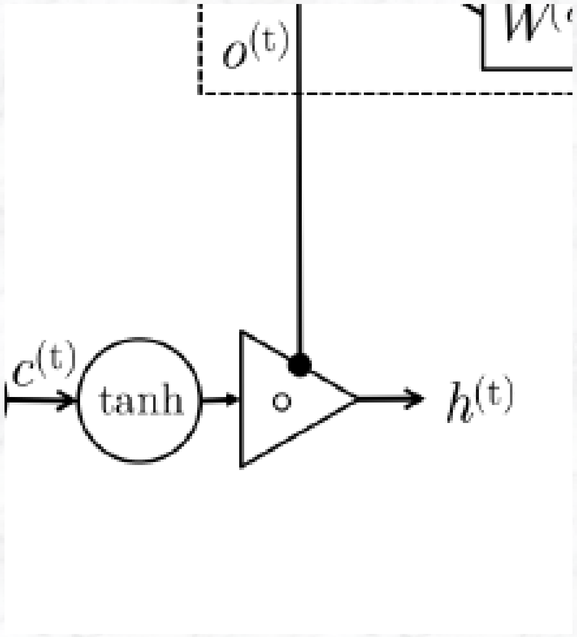
$$o_t = \sigma(W_o x_t + U_o h_{t-1})$$

This gate makes the assessment regarding what parts of the memory c_t needs to be exposed/present in the hidden state h_t.

RNN Extensions: Long Short-term Memory

A gating mechanism of the LSTM

The hidden state



$$h_t = o_t \circ \tanh(c_t)$$

Output

Final memory

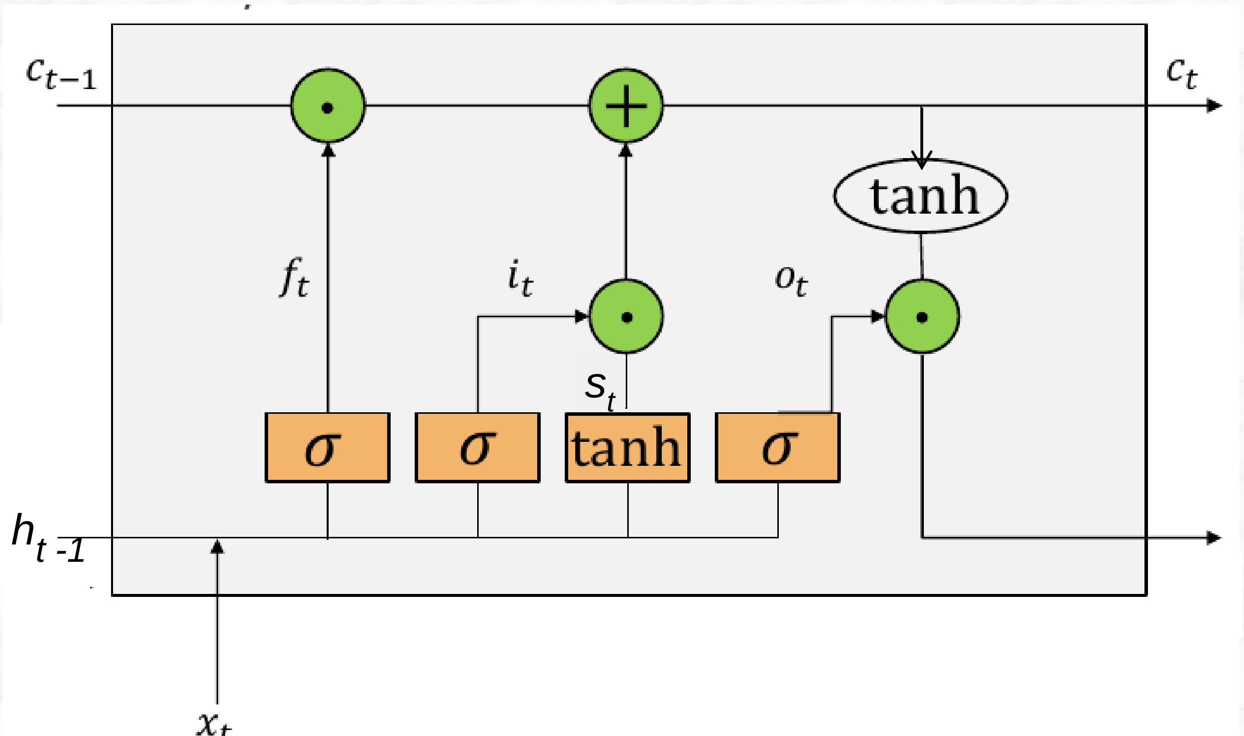
LSTM Cell
Activation Functions

σ ∈ (0, 1): control gate – something like a switch

tanh ∈ −1, 1 : recurrent nonlinearity

$$f_t = \sigma(W^f h_{t-1} + U^f x_t)$$
$$i_t = \sigma(W^i h_{t-1} + U^i x_t)$$
$$o_t = \sigma(W^o h_{t-1} + U^o x_t)$$

$$s_t = \tanh(W^s h_{t-1} + U^s x_t)$$
$$c_t = f_t \bullet c_{t-1} + i_t \bullet s_t$$
$$h_t = o_t \bullet \tanh(c_t)$$



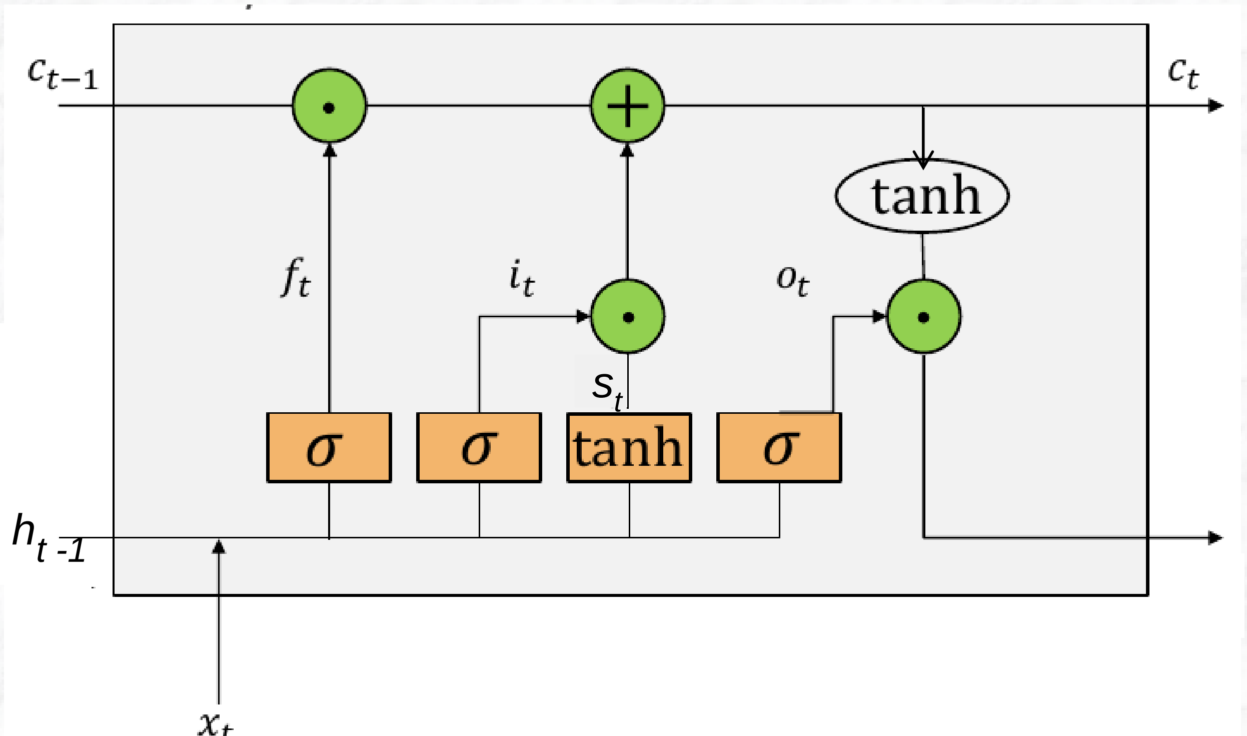
LSTM Cell
Activation Functions

σ ∈ (0, 1): control gate – something like a switch

tanh ∈ −1, 1 : recurrent nonlinearity

$$f_t = \sigma(W^f h_{t-1} + U^f x_t)$$
$$i_t = \sigma(W^i h_{t-1} + U^i x_t)$$
$$o_t = \sigma(W^o h_{t-1} + U^o x_t)$$

$$s_t = \tanh(W^s h_{t-1} + U^s x_t)$$
$$c_t = f_t \bullet c_{t-1} + i_t \bullet s_t$$
$$h_t = o_t \bullet \tanh(c_t)$$



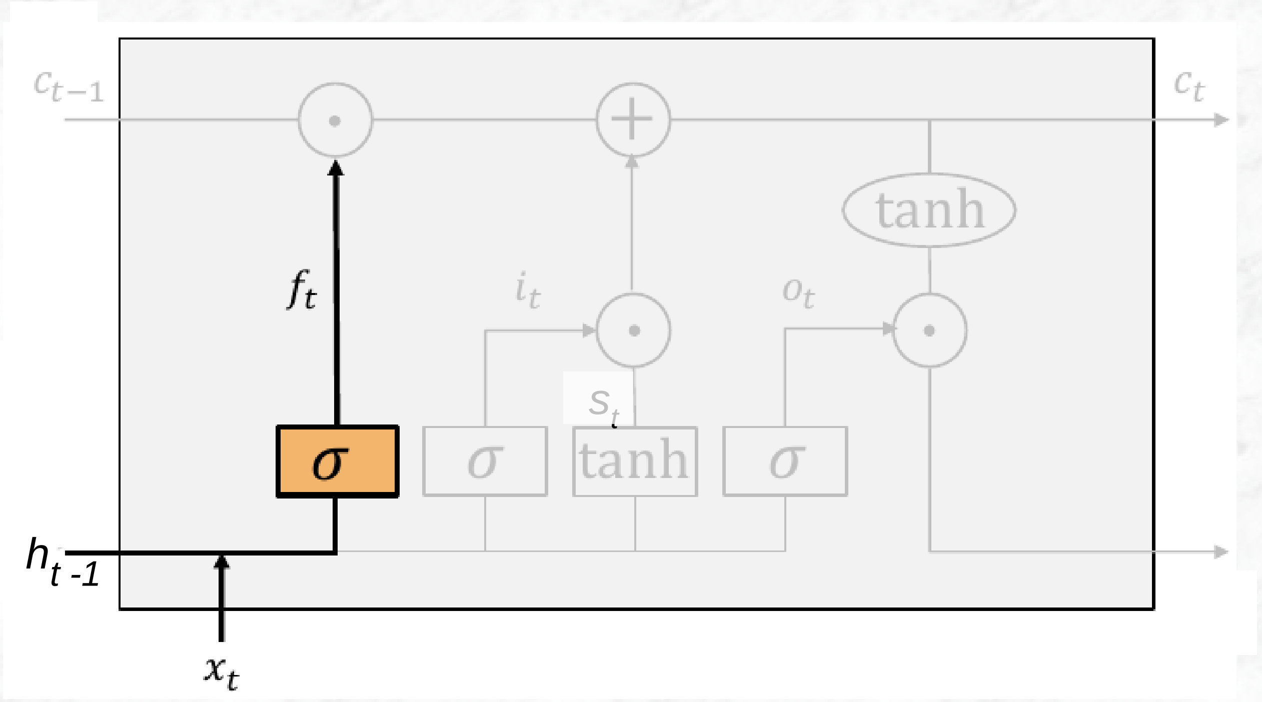
LSTM Cell
forget Gate

Decide what to forget and what to remember for the new memory

Sigmoid 1 ==> Remember everything
Sigmoid 0 ==> Forget everything

$$f_t = \sigma(W^f h_{t-1} + U^f x_t)$$
$$i_t = \sigma(W^i h_{t-1} + U^i x_t)$$
$$o_t = \sigma(W^o h_{t-1} + U^o x_t)$$

$$s_t = \tanh(W^s h_{t-1} + U^s x_t)$$
$$c_t = f_t \bullet c_{t-1} + i_t \bullet s_t$$
$$h_t = o_t \bullet \tanh(c_t)$$



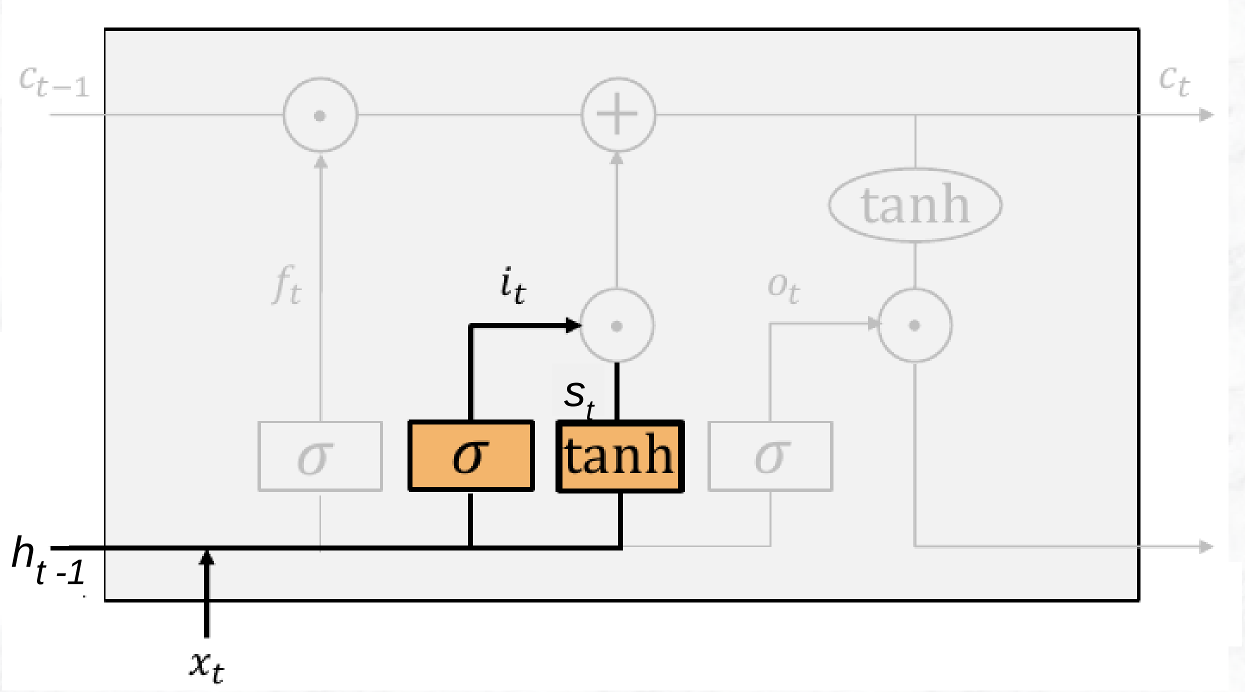
LSTM Cell
Input Gate

Decide what new information should you add to the new memory

Modulate the input it
Generate candidate memories C_t

$$f_t = \sigma(W^f h_{t-1} + U^f x_t)$$
$$i_t = \sigma(W^i h_{t-1} + U^i x_t)$$
$$o_t = \sigma(W^o h_{t-1} + U^o x_t)$$

$$s_t = \tanh(W^s h_{t-1} + U^s x_t)$$
$$c_t = f_t \bullet c_{t-1} + i_t \bullet s_t$$
$$h_t = o_t \bullet \tanh(c_t)$$



LSTM Cell
Update State

Compute and update the current cell state C_t

Depends on the previous cell state

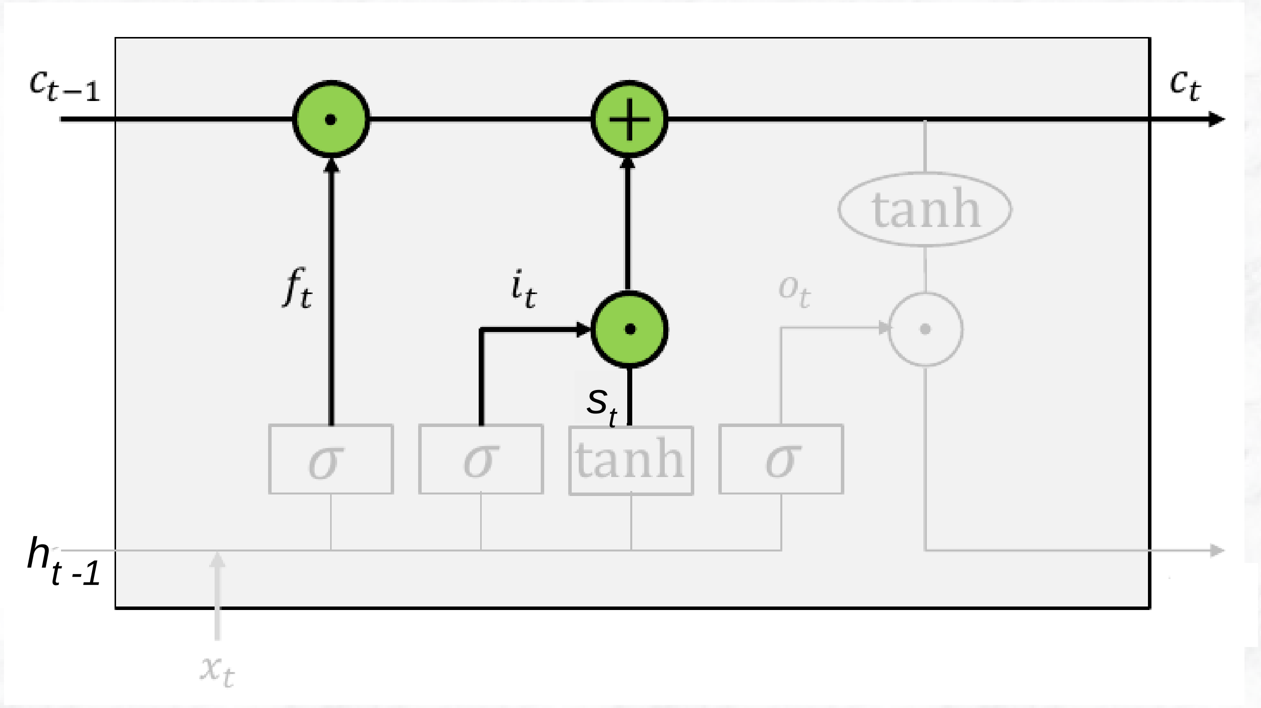
What we decide to forget

What inputs we allow

The candidate memories

$$f_t = \sigma(W^f h_{t-1} + U^f x_t)$$
$$i_t = \sigma(W^i h_{t-1} + U^i x_t)$$
$$o_t = \sigma(W^o h_{t-1} + U^o x_t)$$

$$s_t = \tanh(W^s h_{t-1} + U^s x_t)$$
$$c_t = f_t \bullet c_{t-1} + i_t \bullet s_t$$
$$h_t = o_t \bullet \tanh(c_t)$$



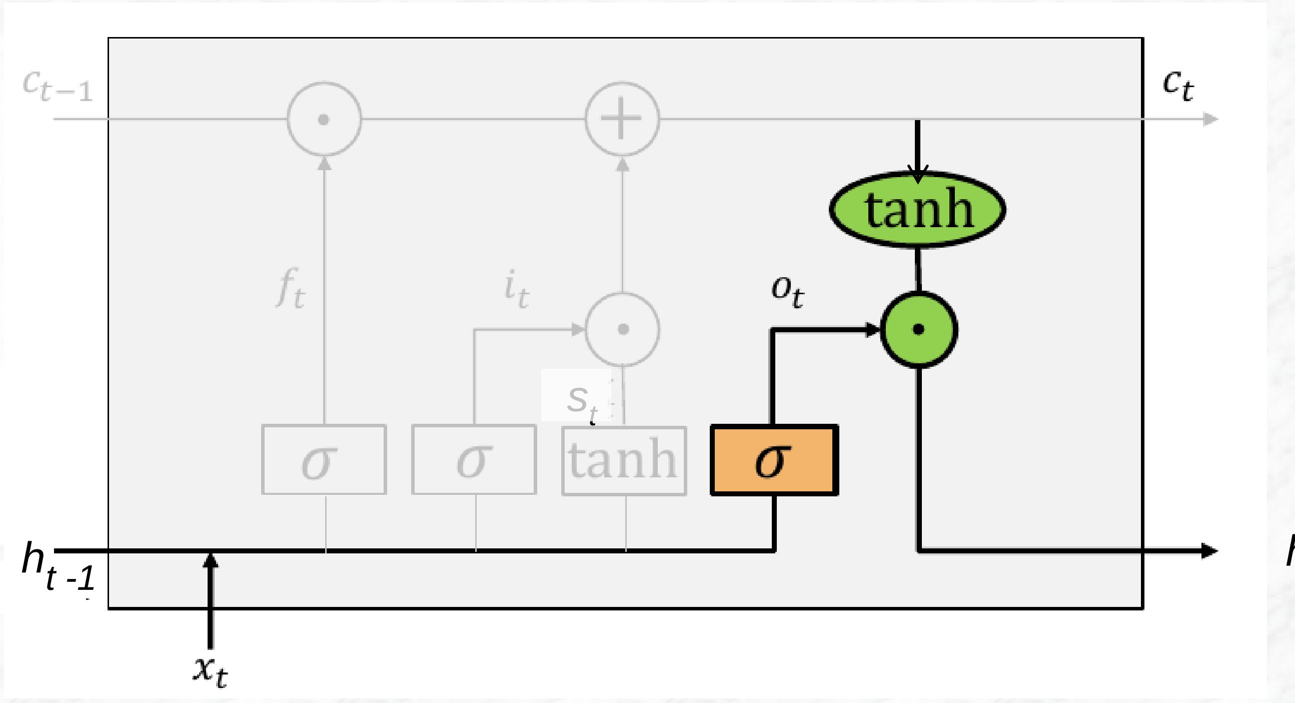
LSTM Cell
Cell Output

Modulate the output

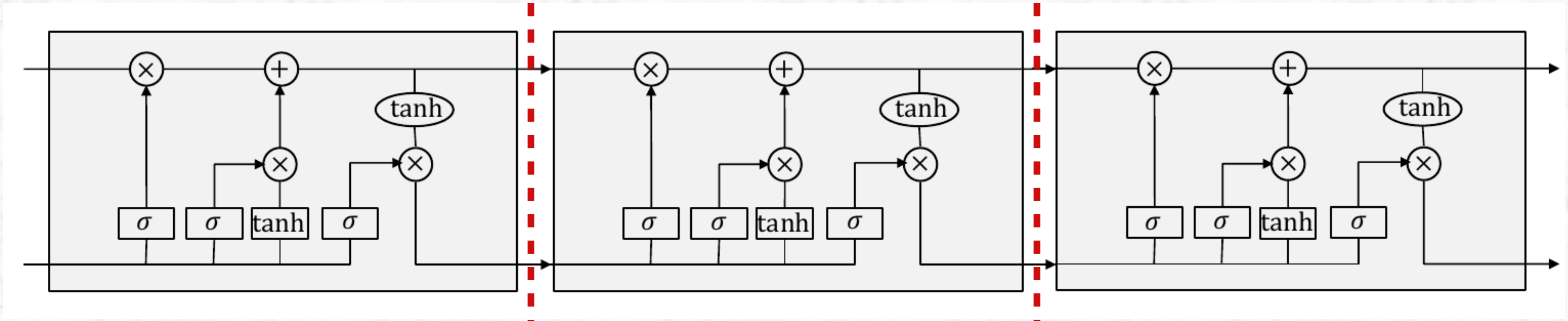
Does the cell state contain something relevant? --> Sigmoid 1

$$f_t = \sigma(W^f h_{t-1} + U^f x_t)$$
$$i_t = \sigma(W^i h_{t-1} + U^i x_t)$$
$$o_t = \sigma(W^o h_{t-1} + U^o x_t)$$

$$s_t = \tanh(W^s h_{t-1} + U^s x_t)$$
$$c_t = f_t \bullet c_{t-1} + i_t \bullet s_t$$
$$h_t = o_t \bullet \tanh(c_t)$$



Unrolled LSTM

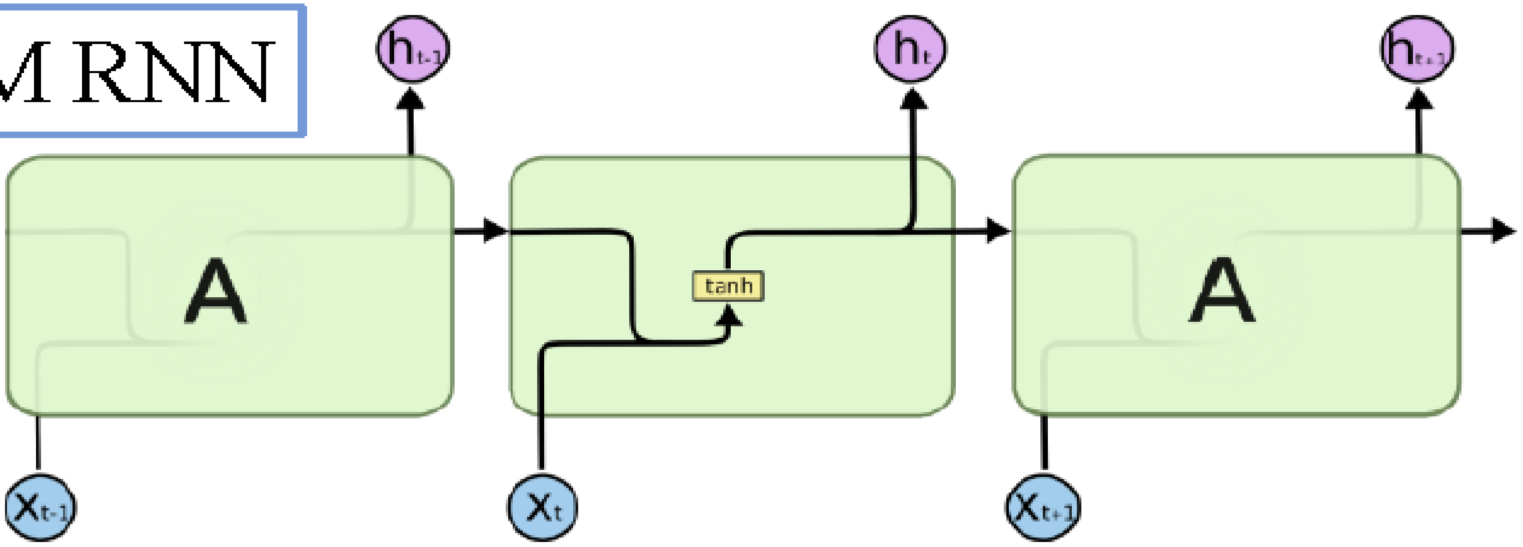


t-1

t

t+1

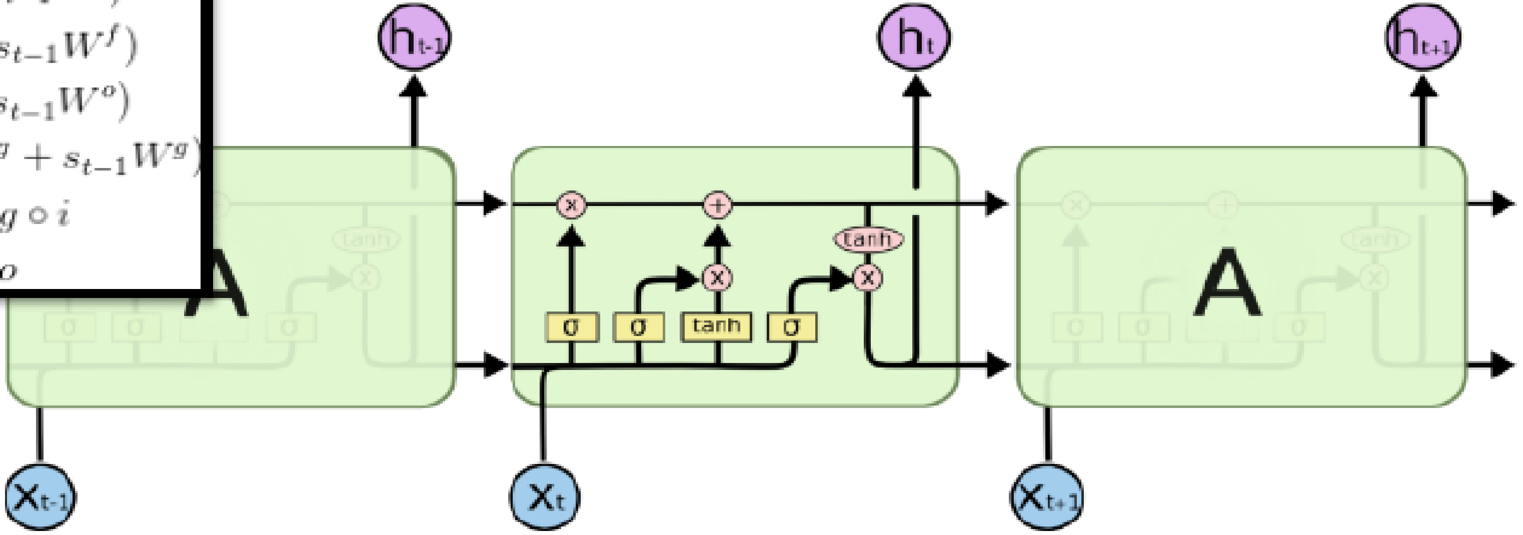
LSTM RNN



The repeating module in a standard RNN contains a single layer.

LSTMs also have this chain like structure, but the repeating module has a different structure. Instead of having a single neural network layer, there are four, interacting in a very special way.

$$
\begin{aligned}
i &= \sigma(x_t U^i + s_{t-1} W^i) \\
f &= \sigma(x_t U^f + s_{t-1} W^f) \\
o &= \sigma(x_t U^o + s_{t-1} W^o) \\
g &= \ tanh(x_t U^g + s_{t-1} W^g) \\
c_t &= c_{t-1} \circ f + g \circ i \\
s_t &= \tanh(c_t) \circ o
\end{aligned}
$$



The repeating module in an LSTM contains four interacting layers.

RNN extensions: Long Short-term memory

Conclusions on LSTM

LSTMs contain information outside the normal flow of the recurrent network in a gated cell. Information can be stored in, written to, or read from a cell, much like data in a computer’s memory. The cells learn when to allow data to enter, leave or be deleted through the iterative process of making guesses, back-propagating error, and adjusting weights via gradient descent.

RNN extensions: Long Short-term Memory

Why LSTM can combat the vanish gradient problem?

LSTMs help preserve the error that can be back-propagated through time and layers. By maintaining a more constant error, they allow recurrent nets to continue to learn over many time steps (over 1000), thereby opening a channel to link causes and effects remotely

Applications of RNN/LSTM include

- Robot control
- Image captioning
- Time series prediction
- Time series anomaly detection
- Sensor data analysis
- Speech recognition
- speech synthesis
- Rhythm learning (precise event timing of spikes)
- Music composition
- Grammar learning
- Semantic parsing
- Handwriting recognition
- Human action recognition
- Machine Translation
- Sign Language Translation
- Protein Homology Detection
- Predicting subcellular localization of proteins
- Prediction in medical care pathways
- Several prediction tasks in the area of business process managemen