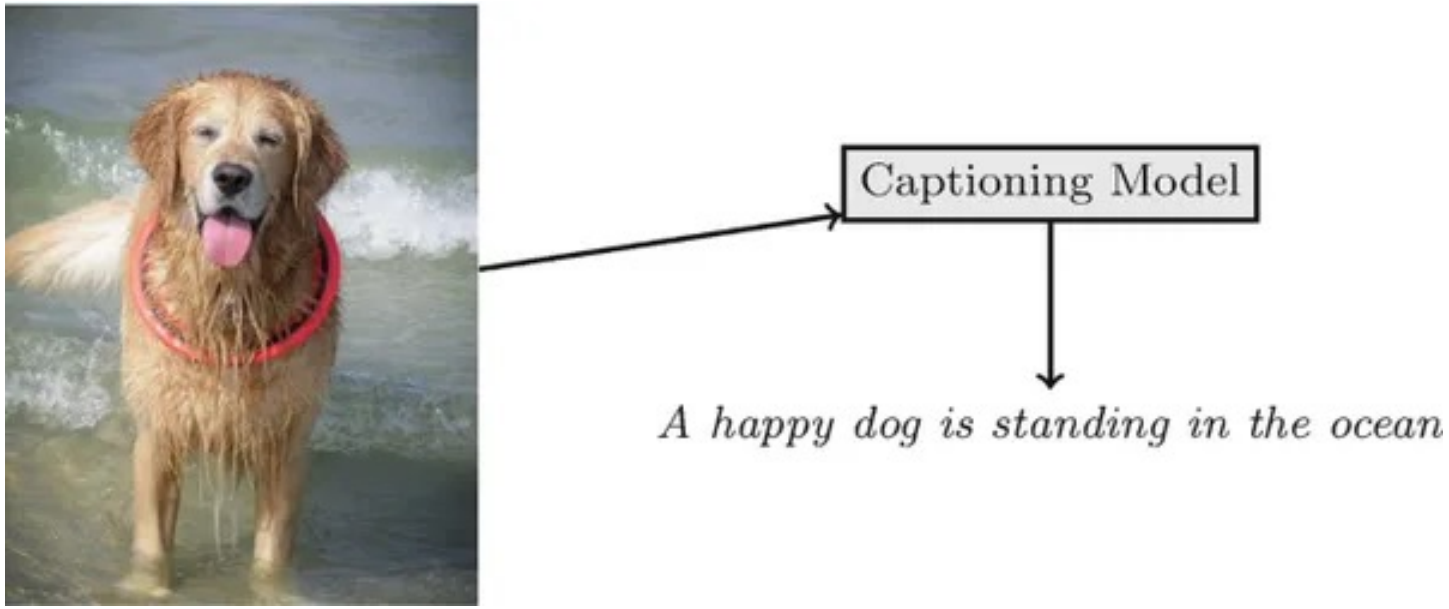
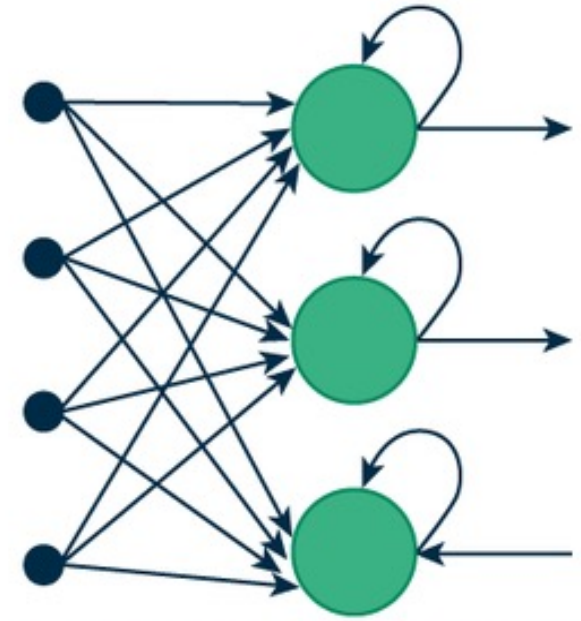
Recurrent Neural Networks

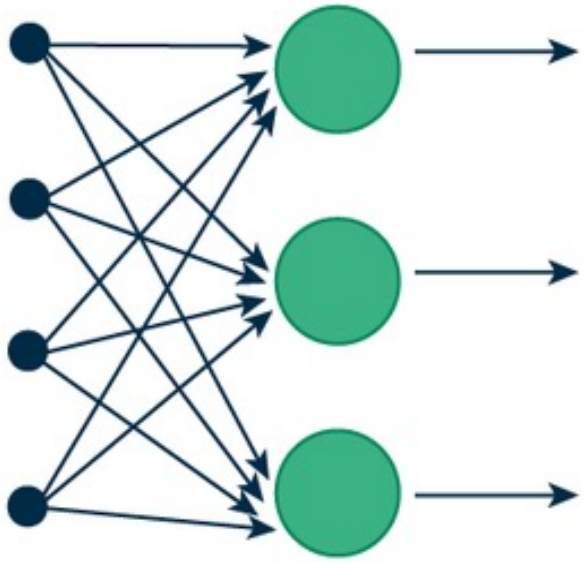
Deep Learning & Applications



Recurrent neural networks use sequential data to solve common temporal problems seen in language translation and speech recognition



(a) Recurrent Neural Network



(b) Feed-Forward Neural Network

A Classic Approach for Text Classification: Bag-of-Words Model

"Raw" training dataset

$\mathbf{x}^{[1]} =$ "The sun is shining"

$\mathbf{x}^{[2]} =$ "The weather is sweet"

$\mathbf{x}^{[3]} =$ "The sun is shining, the weather is sweet, and one and one is two"

$$\mathbf{y} = [0, 1, 0]$$

class labels

```
vocabulary = {
    'and': 0,
    'is': 1
    'one': 2,
    'shining': 3,
    'sun': 4,
    'sweet': 5,
    'the': 6,
    'two': 7,
    'weather': 8,
}
```

Training set as design matrix

$$\mathbf{X} = \begin{bmatrix} 0 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 1 \\ 2 & 3 & 2 & 1 & 1 & 1 & 2 & 1 & 1 \end{bmatrix}$$

$$\mathbf{y} = [0, 1, 0]$$

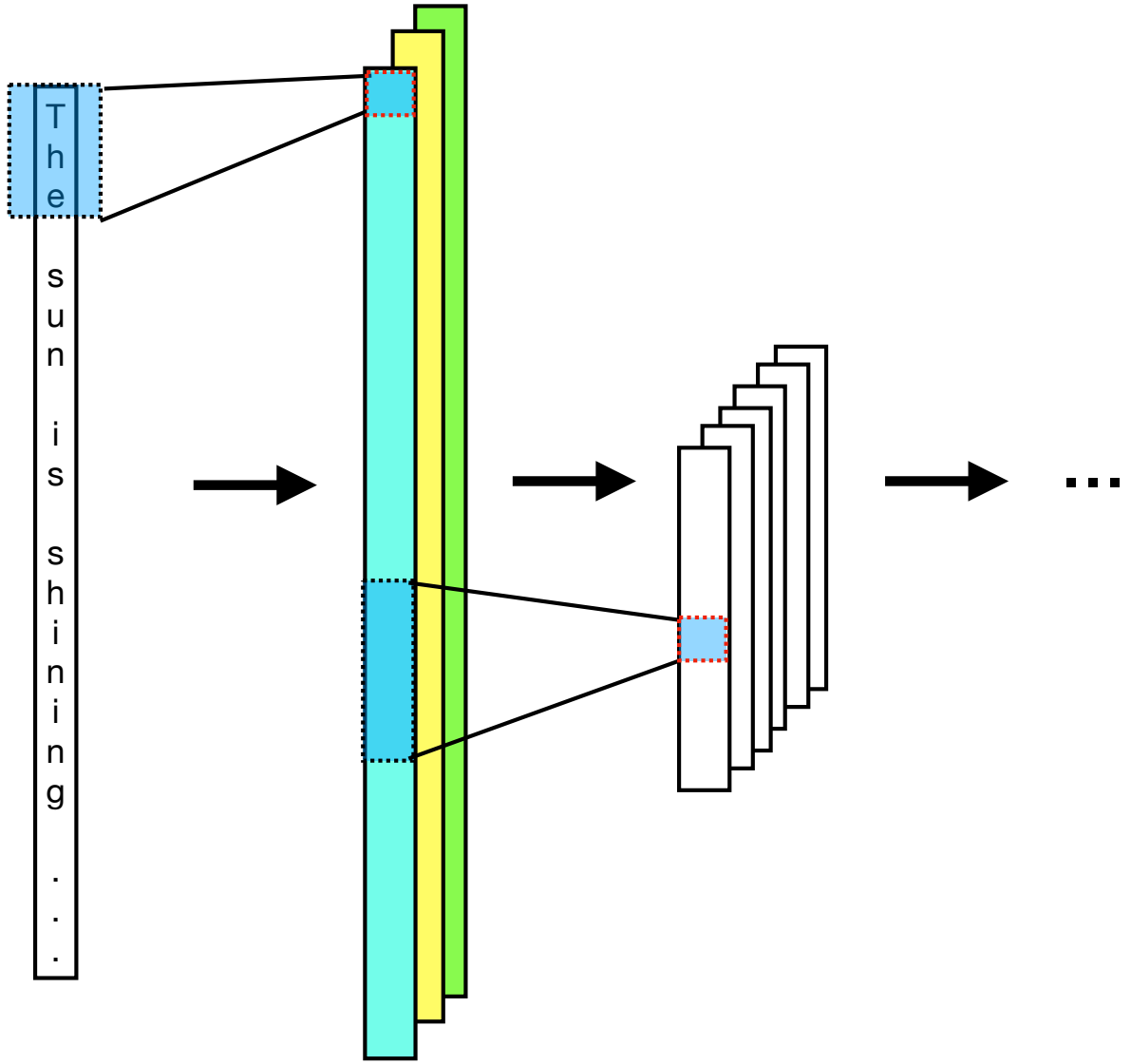
class labels

training

Classifier

(e.g., logistic regression, MLP, ...)

1D CNNs for text (and other sequence data)



Sequence data: order matters

The movie my friend has **not** seen is good
The movie my friend has seen is **not** good

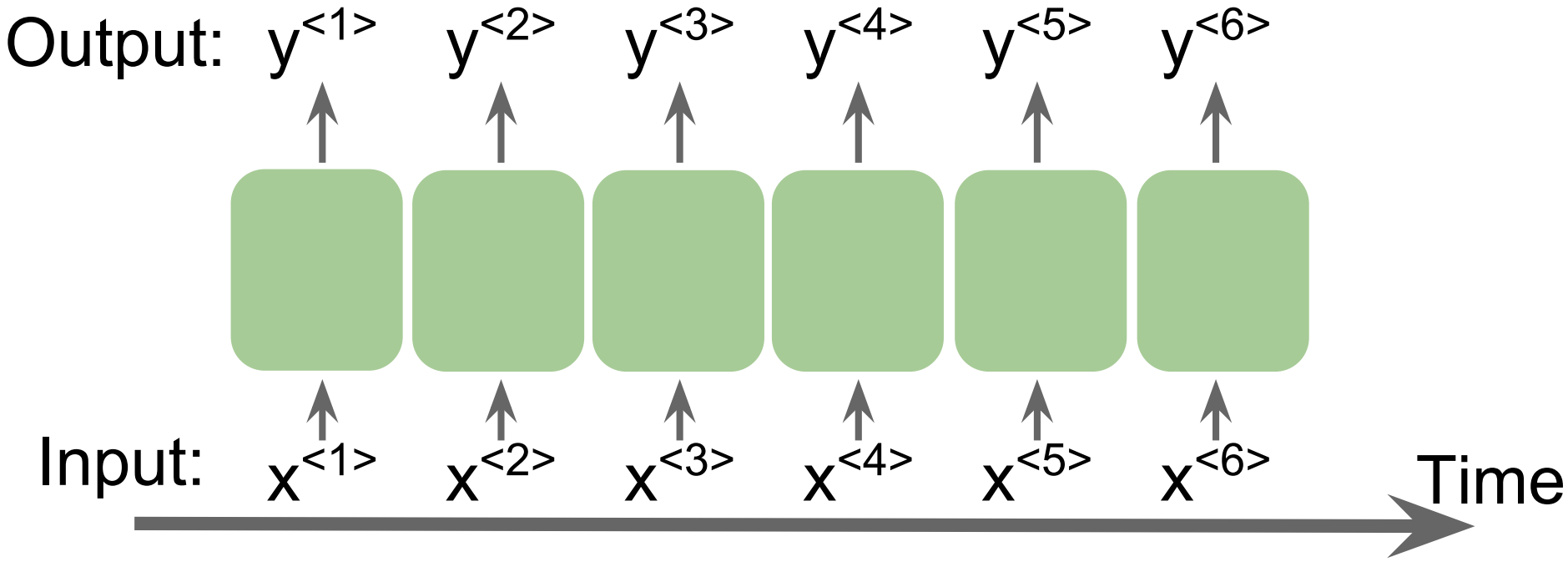
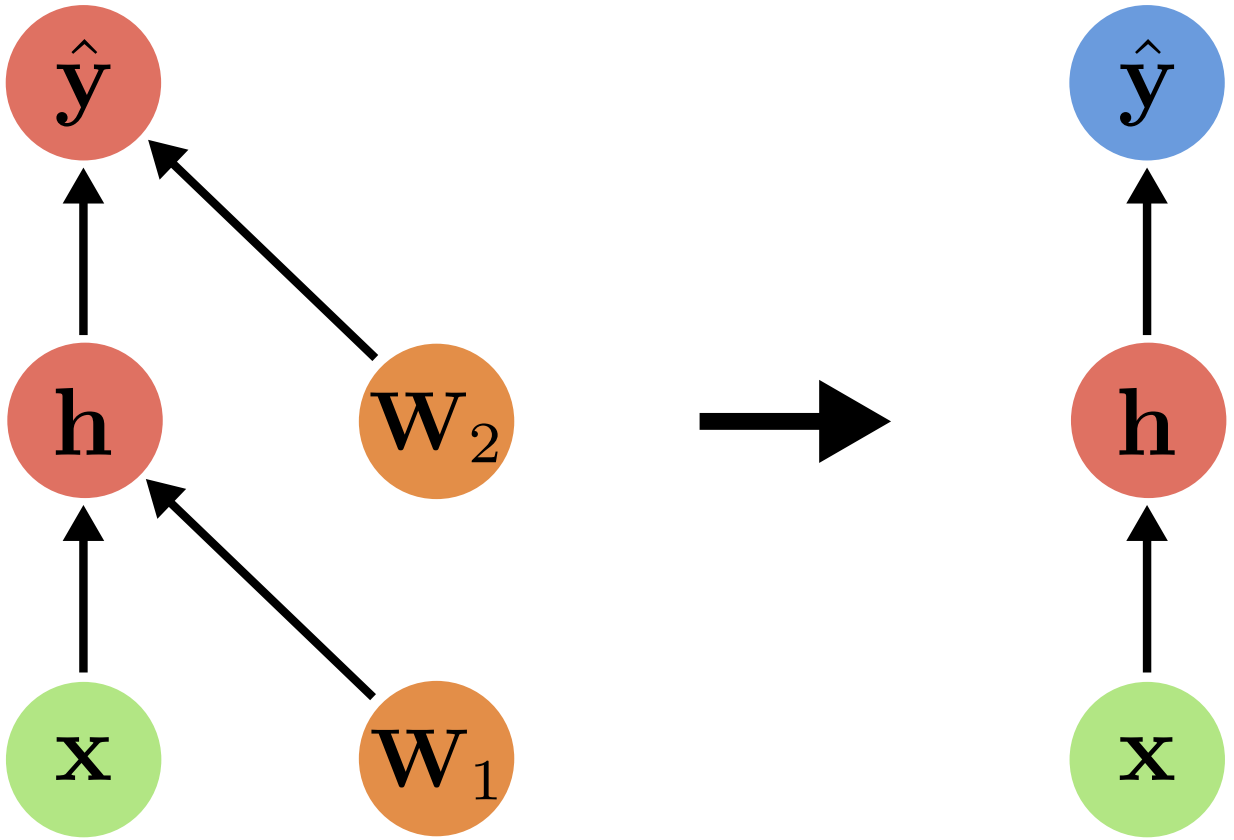


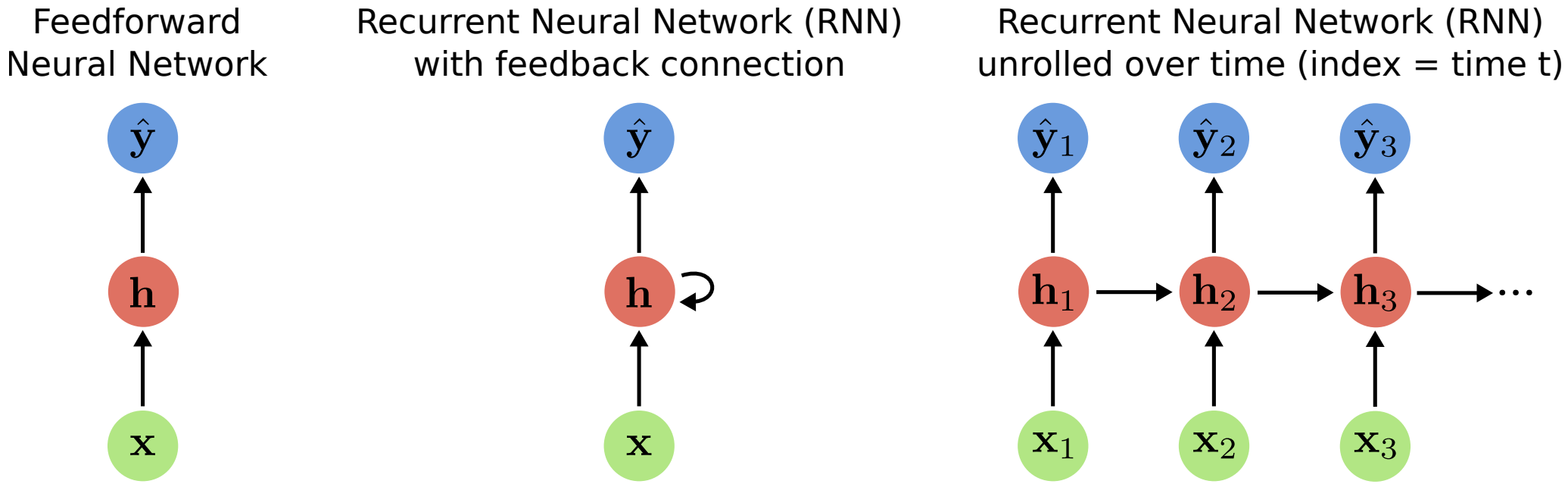
Image source: Sebastian Raschka, Vahid Mirjalili. *Python Machine Learning. 3rd Edition.* Packt, 2019

Recap: Computation Graphs



- Example of a **feedforward neural network** (here: MLP with 1 hidden layer)

Feedforward vs. Recurrent Neural Networks

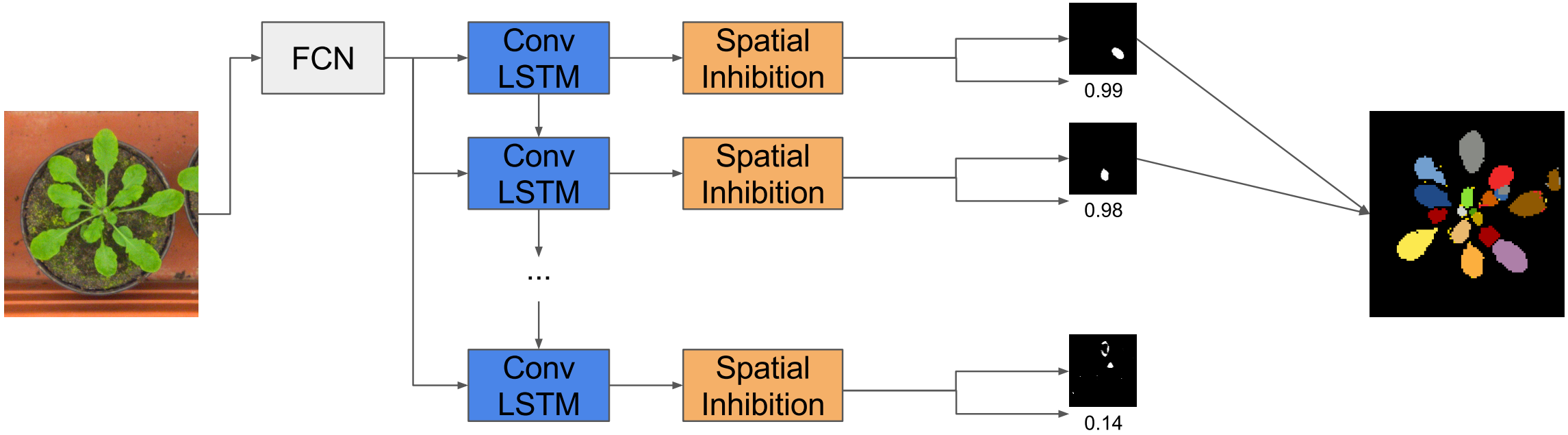


Recurrent Neural Networks (RNNs):

- Core idea: update **hidden state** $\mathbf{h}$ based on input and previous hidden state using same update rule (same/shared parameters) at each **time step**
- Allows for processing **sequences of variable length**, not only fixed-sized vectors
- **Infinite memory:** $\mathbf{h}$ is function of all previous inputs (long-term dependencies)

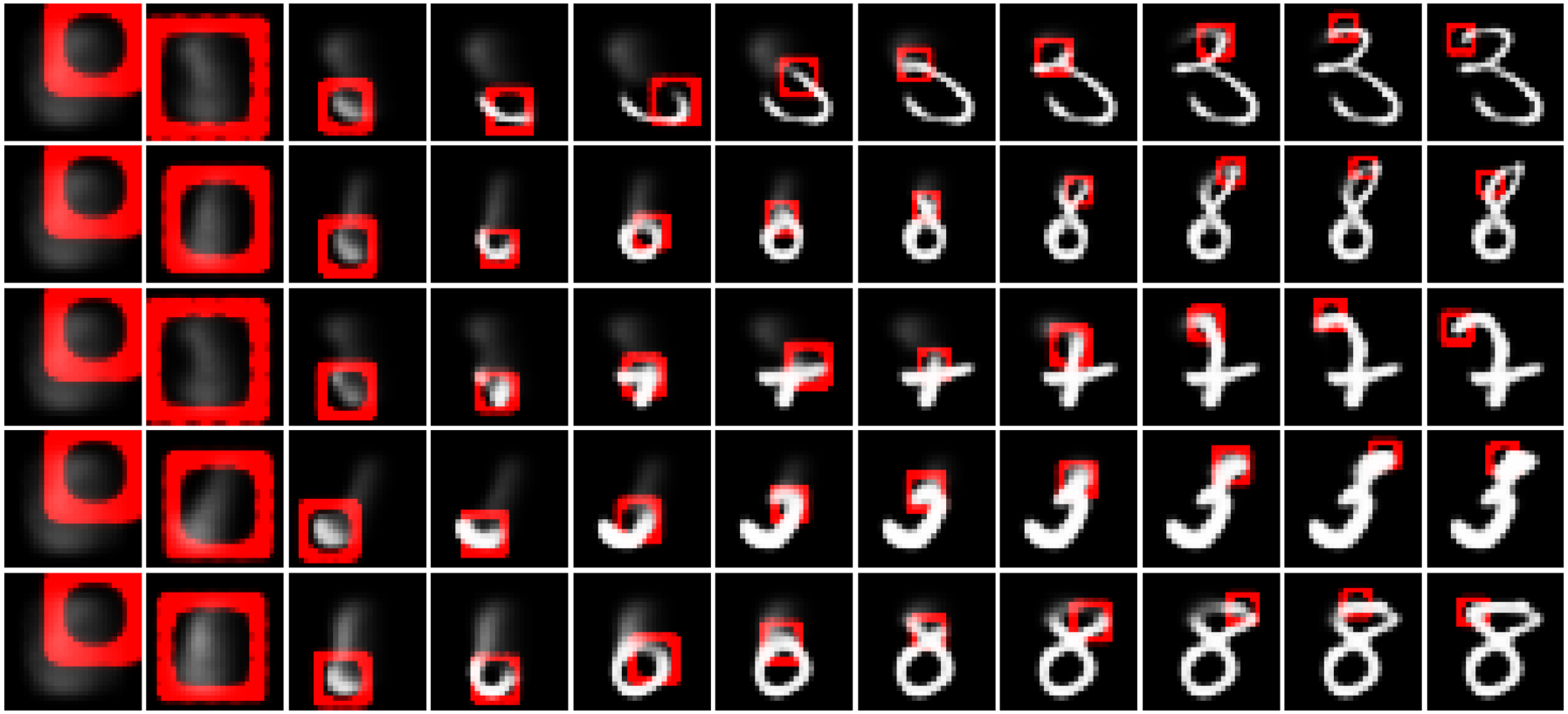
Recurrent Network Applications

Recurrent Instance Segmentation



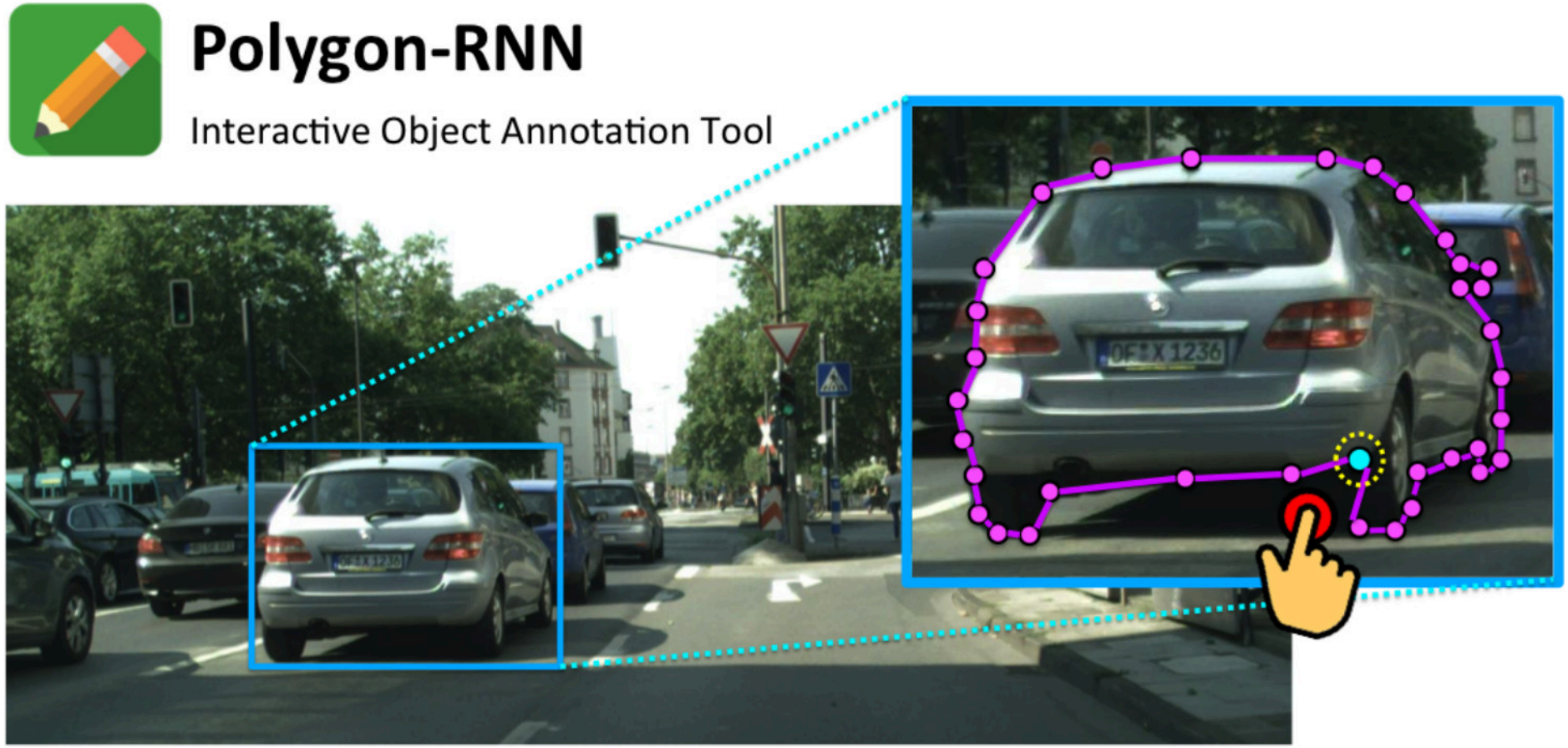
- At each time step, **segment** the next (not yet segmented) **part** of an object

Image Generation



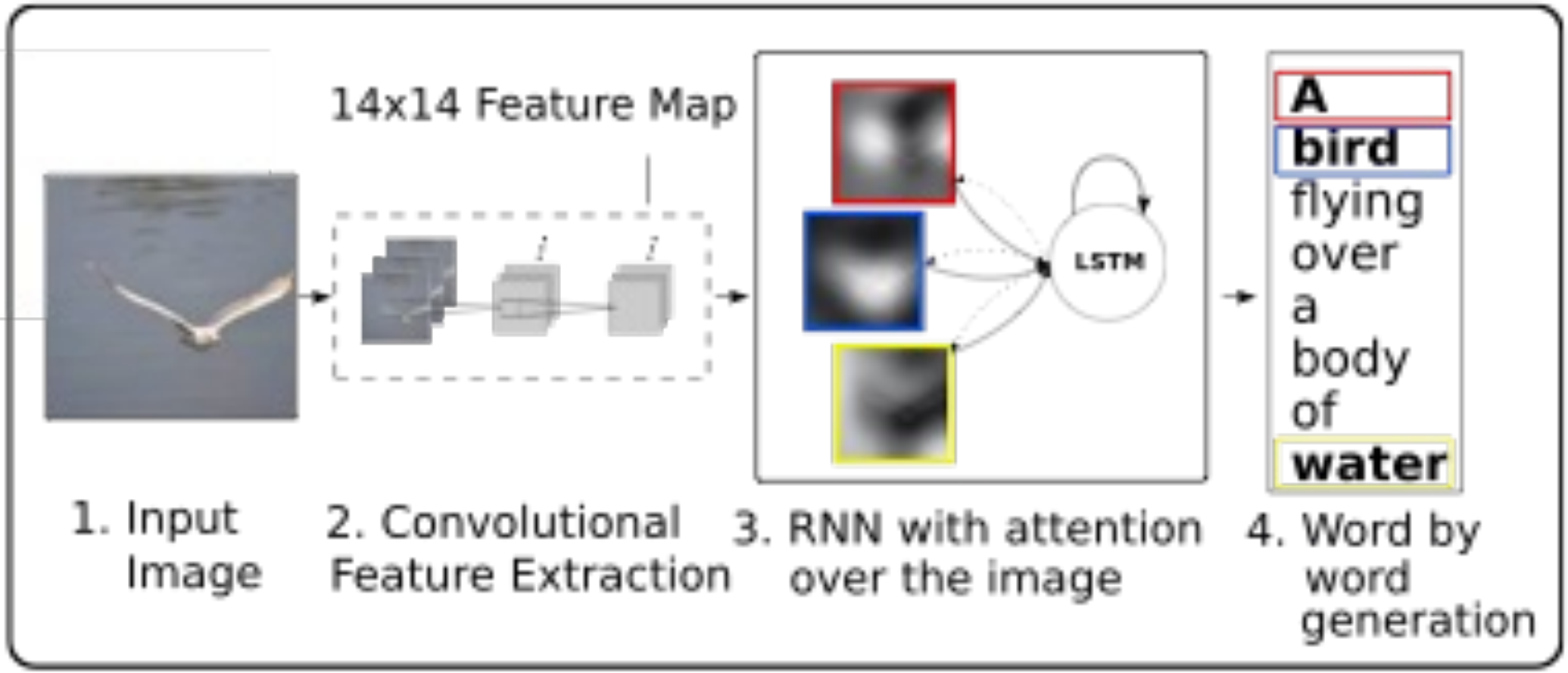
- Model for **sequential image generation** (red rectangle = attended region)
- Demonstrates that RNNs can also process/generate non-sequential data

Image Annotation



- Iteratively **annotate** the outline of 2D object instances in an image with **polygons**

Image Captioning



- Generate **image description** by sequentially looking at an image

Image Captioning

A woman is throwing a frisbee in a park.

A dog is standing on a hardwood floor.

A stop sign is on a road with a mountain in the background.

A little girl sitting on a bed with a teddy bear.

A group of people sitting on a boat in the water.

A giraffe standing in a forest with trees in the background.

- Successful caption generations

Xu et al.: Show, Attend and Tell: Neural Image Caption Generation with Visual Attention. ICML, 2015.

Image Captioning

A large white bird standing in a forest.

A woman holding a clock in her hand.

A man wearing a hat and a hat on a skateboard.

A person is standing on a beach with a surfboard.

A woman is sitting at a table with a large pizza.

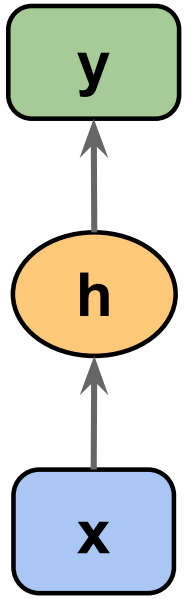
A man is talking on his cell phone while another man watches.

- Wrongly generated captions. Attention can reveal insights into what went wrong.

Xu et al.: Show, Attend and Tell: Neural Image Caption Generation with Visual Attention. ICML, 2015.

Overview

Networks we used previously: also called feedforward neural networks



Recurrent Neural Network (RNN)

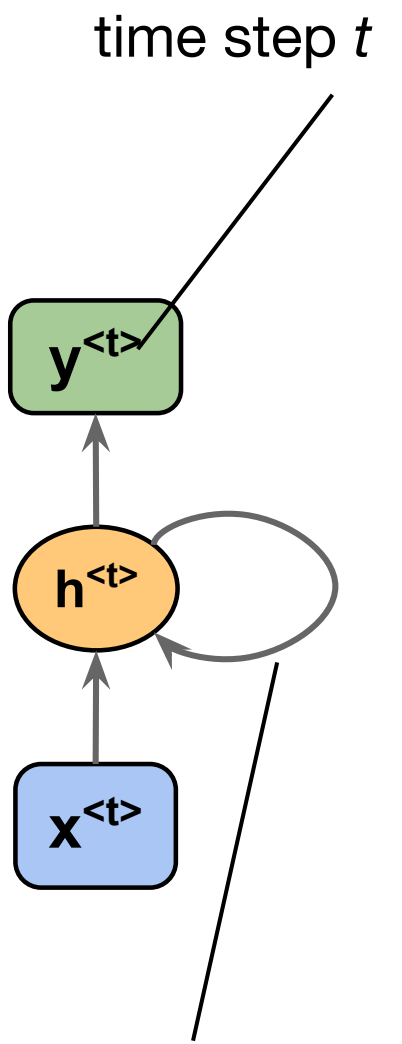
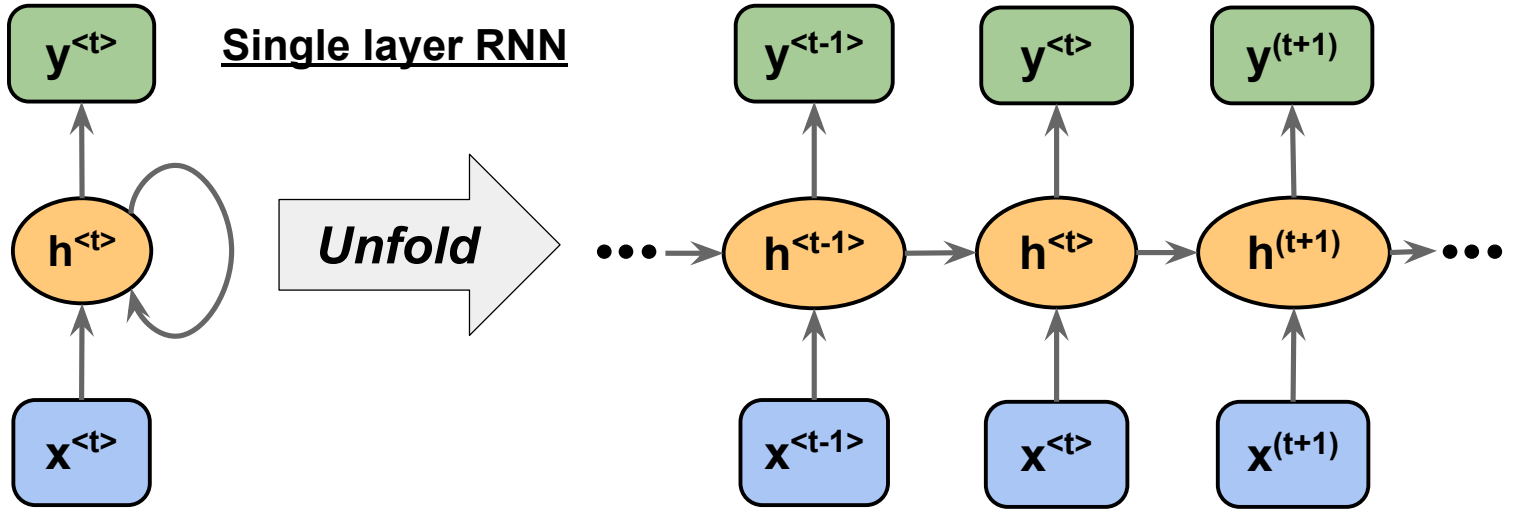


Image source: Sebastian Raschka, Vahid Mirjalili. *Python Machine Learning. 3rd Edition.* Packt, 2019

Overview



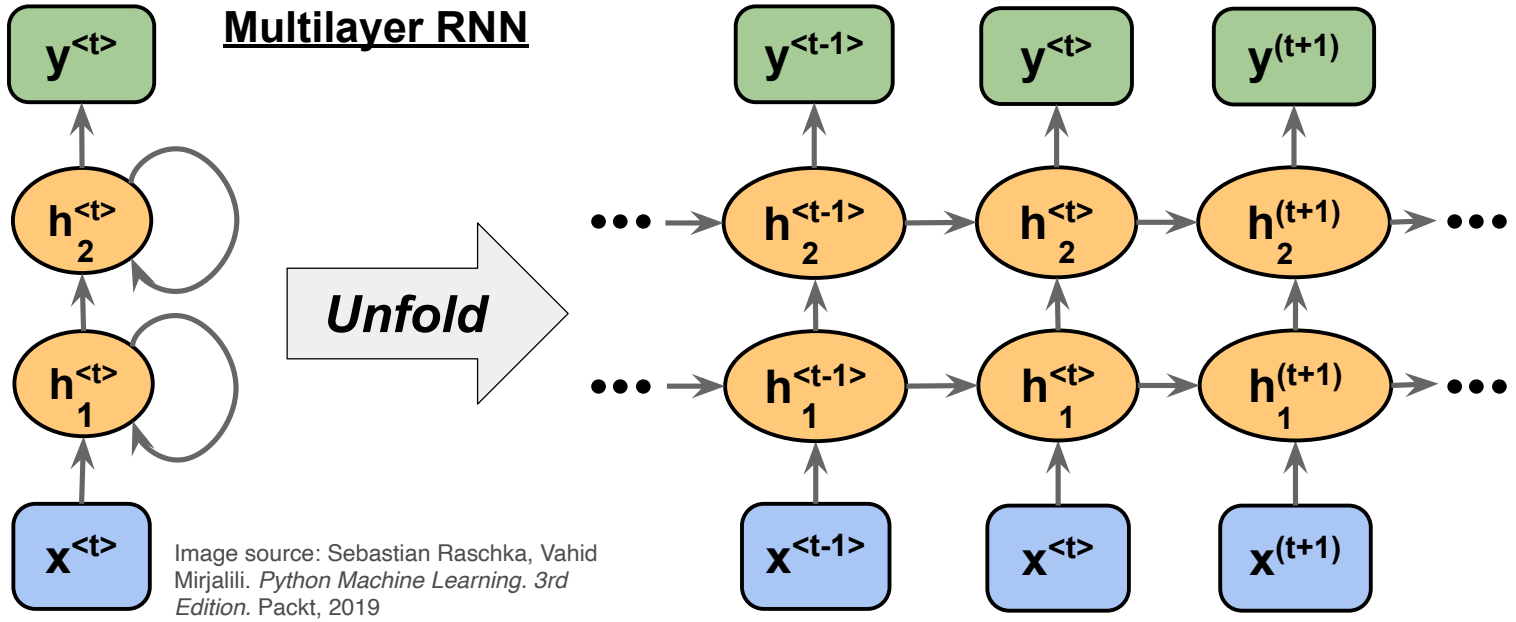


Image source: Sebastian Raschka, Vahid Mirjalili. *Python Machine Learning. 3rd Edition.* Packt, 2019

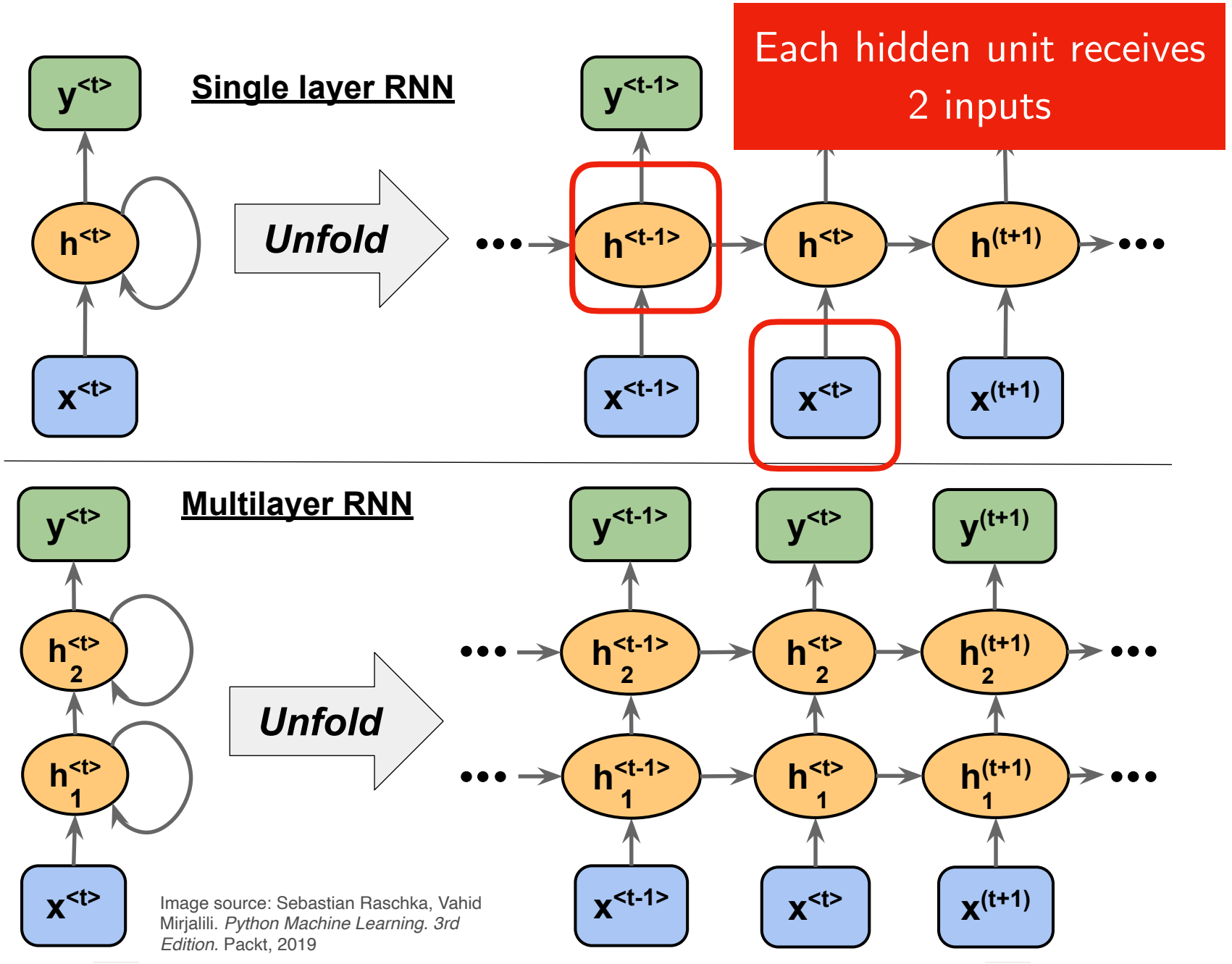
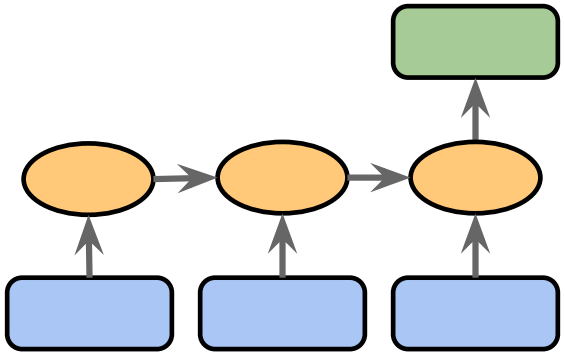
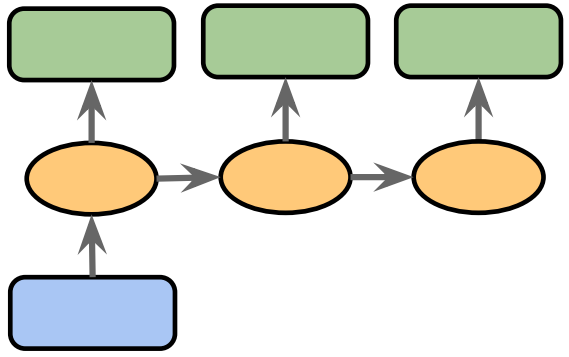


Image source: Sebastian Raschka, Vahid Mirjalili. *Python Machine Learning. 3rd Edition.* Packt, 2019

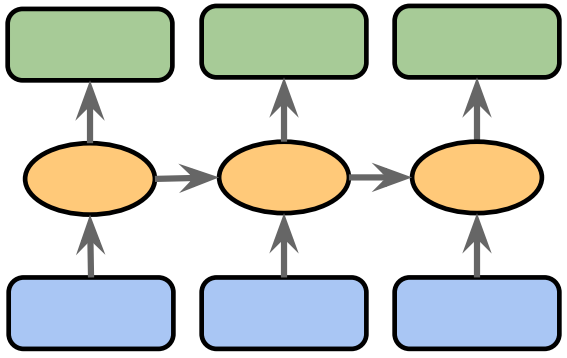
Different Types of Sequence Modeling Tasks



many-to-one



one-to-many



many-to-many

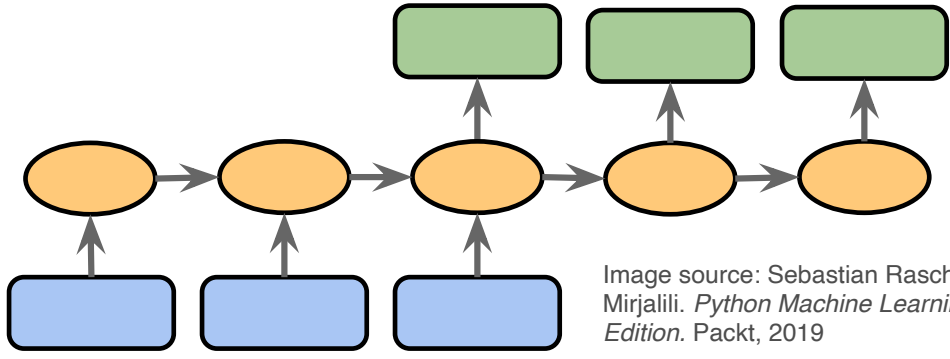
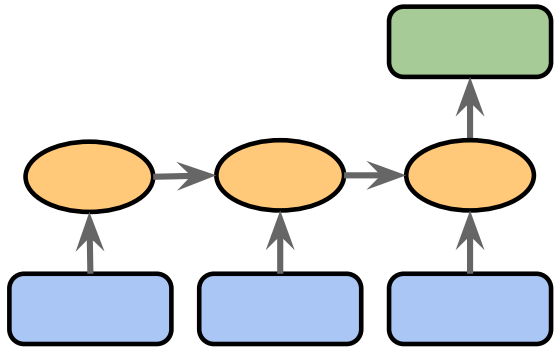


Image source: Sebastian Raschka, Vahid Mirjalili. *Python Machine Learning. 3rd Edition.* Packt, 2019

many-to-many

Figure based on:
The Unreasonable Effectiveness of Recurrent Neural Networks by Andrej Karpathy (http://karpathy.github.io/2015/05/21/rnn-effectiveness/)

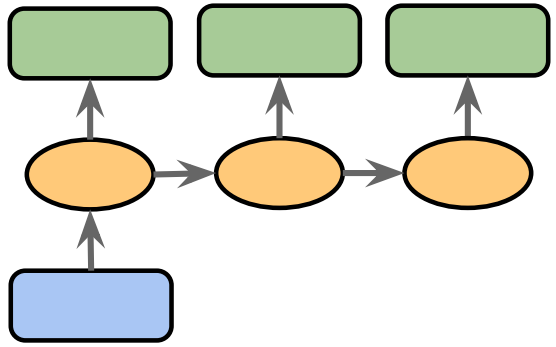
Different Types of Sequence Modeling Tasks



Many-to-one: The input data is a sequence, but the output is a fixed-size vector, not a sequence.

Ex.: sentiment analysis, the input is some text, and the output is a class label.

Different Types of Sequence Modeling Tasks



one-to-many

One-to-many: Input data is in a standard format (not a sequence), the output is a sequence.

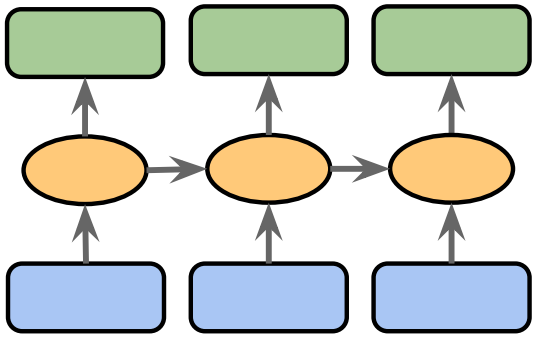
Ex.: Image captioning, where the input is an image, the output is a text description of that image

Different Types of Sequence Modeling Tasks

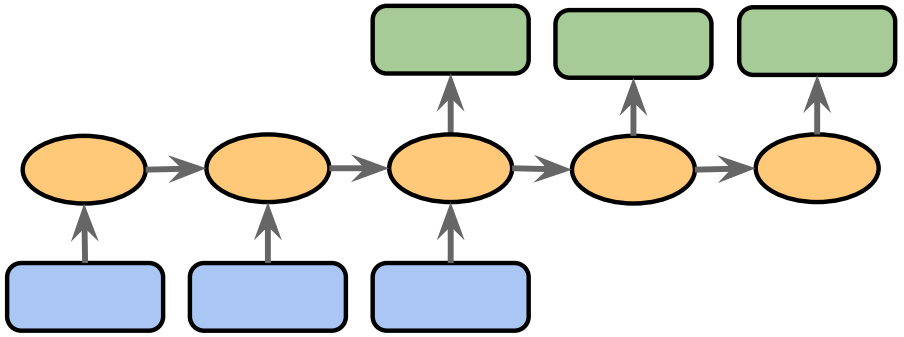
Many-to-many: Both inputs and outputs are sequences. Can be direct or delayed.

Ex.: Video-captioning, i.e., describing a sequence of images via text (direct).

Translating one language into another (delayed)



many-to-many



many-to-many

Weight matrices in a single-hidden layer RNN

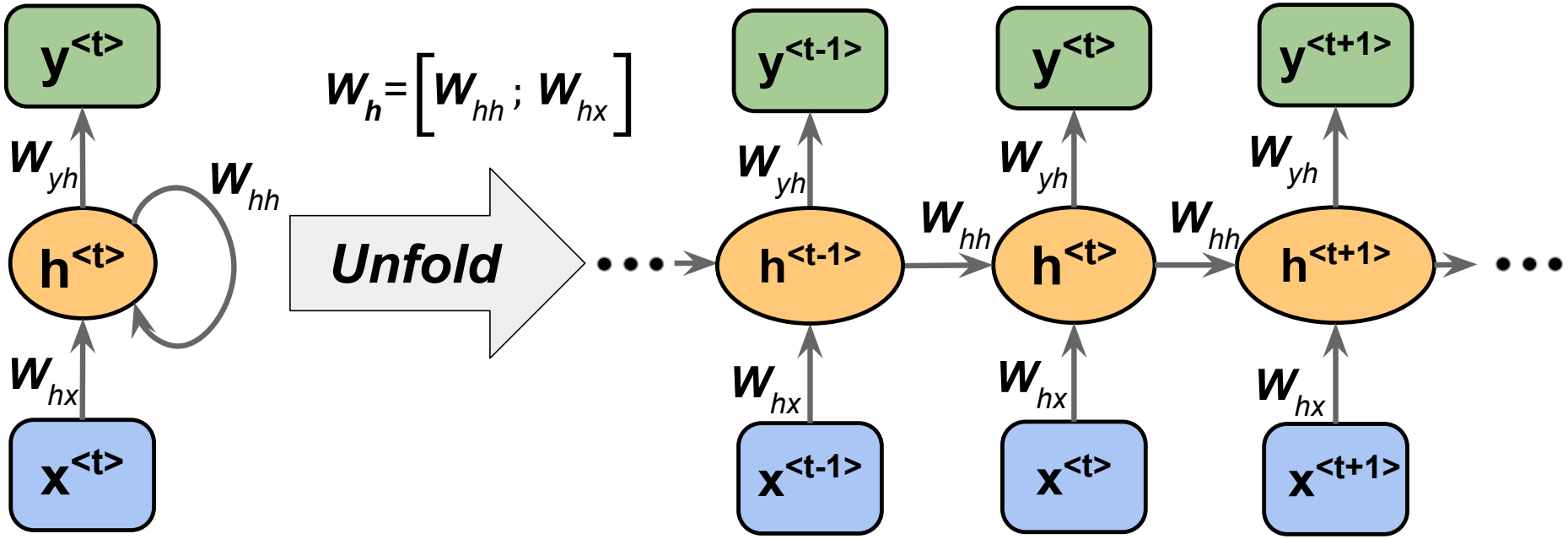


Image source: Sebastian Raschka, Vahid Mirjalili. *Python Machine Learning. 3rd Edition.* Packt, 2019

Weight matrices in a single-hidden layer RNN

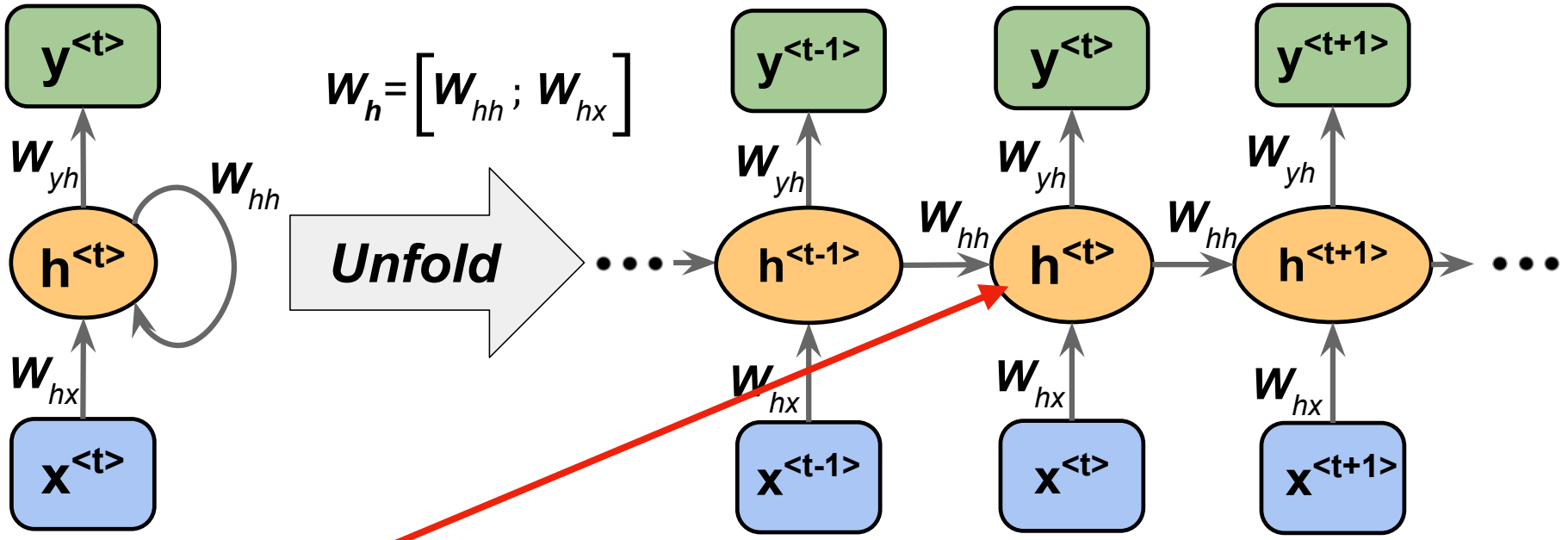


Image source: Sebastian Raschka, Vahid Mirjalili. *Python Machine Learning. 3rd Edition.* Packt, 2019

Net input:

$$\mathbf{z}_h^{\langle t \rangle} = \mathbf{W}_{hx}\mathbf{x}^{\langle t \rangle} + \mathbf{W}_{hh}\mathbf{h}^{\langle t-1 \rangle} + \mathbf{b}_h$$

Activation:

$$\mathbf{h}^{\langle t \rangle} = \sigma_h\left(\mathbf{z}_h^{\langle t \rangle}\right)$$

Weight matrices in a single-hidden layer RNN

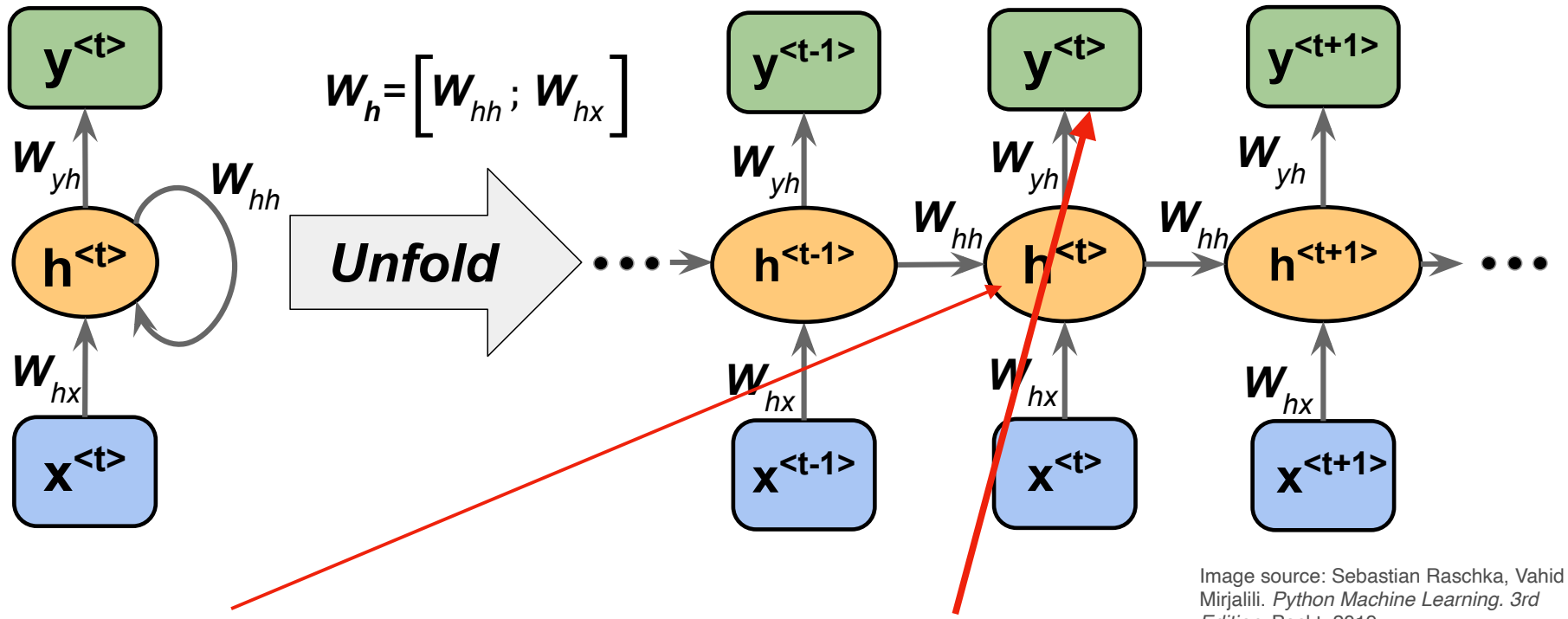


Image source: Sebastian Raschka, Vahid Mirjalili. *Python Machine Learning. 3rd Edition.* Packt, 2019

Net input:

$$\mathbf{z}_h^{\langle t \rangle} = \mathbf{W}_{hx}\mathbf{x}^{\langle t \rangle} + \mathbf{W}_{hh}\mathbf{h}^{\langle t-1 \rangle} + \mathbf{b}_h$$

Activation:

$$\mathbf{h}^{\langle t \rangle} = \sigma_h\left(\mathbf{z}_h^{\langle t \rangle}\right)$$

Net input:

$$\mathbf{z}_y^{\langle t \rangle} = \mathbf{W}_{yh}\mathbf{h}^{\langle t \rangle} + \mathbf{b}_y$$

Output:

$$\mathbf{y}^{\langle t \rangle} = \sigma_y\left(\mathbf{z}_y^{\langle t \rangle}\right)$$

Backpropagation through time

The overall loss can be computed as the sum over all time steps

$$L = \sum_{t=1}^{T} L^{\langle t \rangle}$$

Werbos, Paul J. "Backpropagation through time: what it does and how to do it." *Proceedings of the IEEE* 78, no. 10 (1990): 1550-1560.

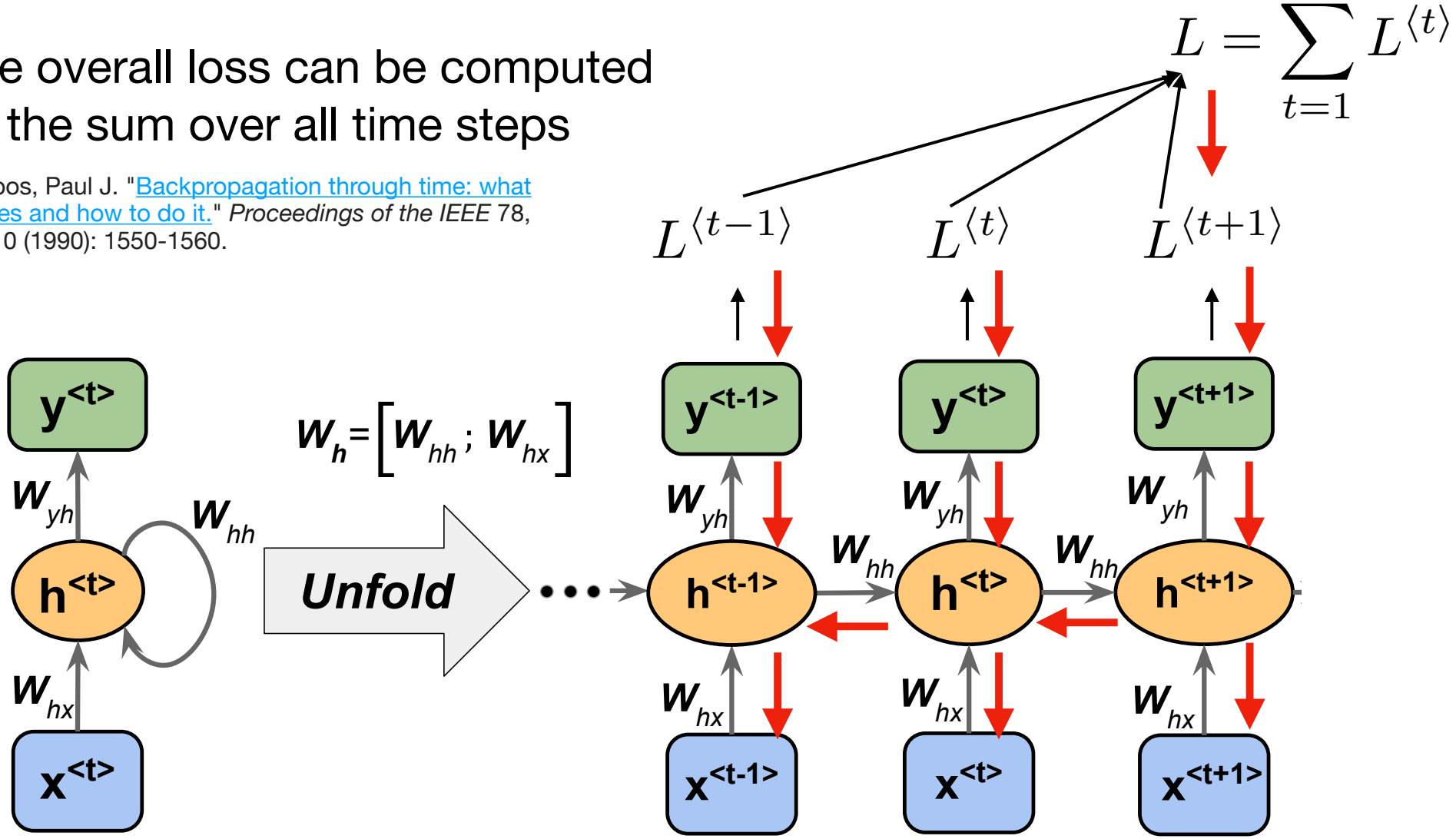
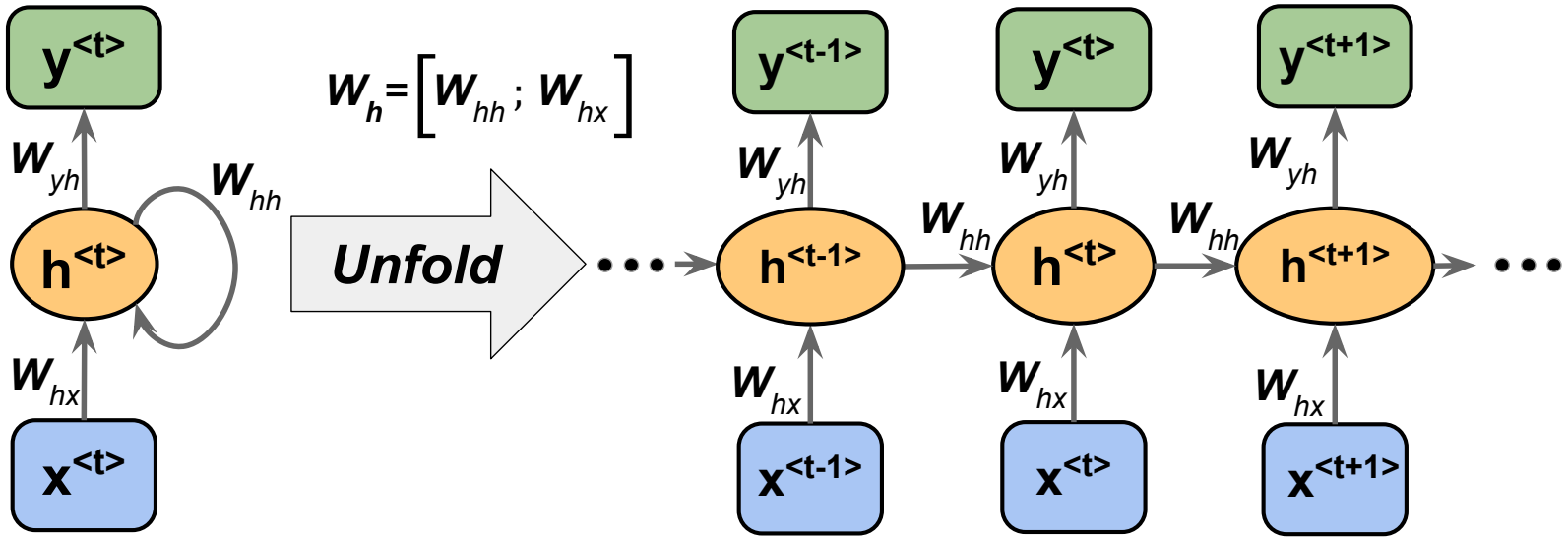


Image source: Sebastian Raschka, Vahid Mirjalili. *Python Machine Learning. 3rd Edition.* Packt, 2019

Backpropagation through time

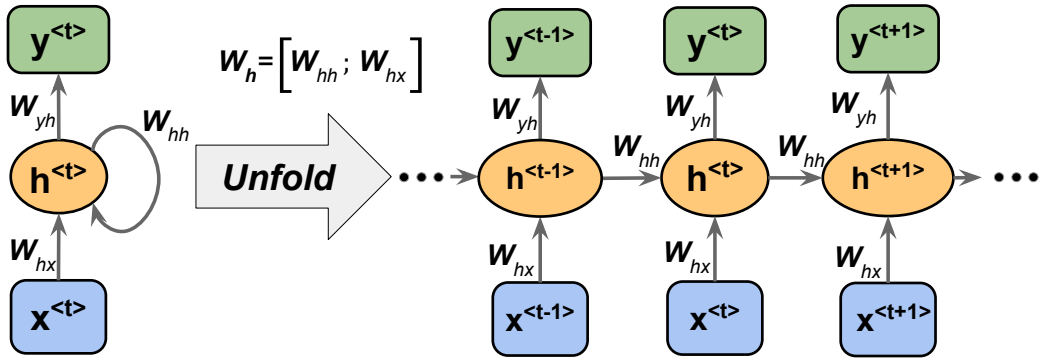


Werbos, Paul J. "Backpropagation through time: what it does and how to do it." *Proceedings of the IEEE* 78, no. 10 (1990): 1550-1560.

$$L = \sum_{t=1}^{T} L^{(t)}$$

$$\frac{\partial L^{(t)}}{\partial \mathbf{W}_{hh}} = \frac{\partial L^{(t)}}{\partial y^{(t)}} \cdot \frac{\partial y^{(t)}}{\partial \mathbf{h}^{(t)}} \cdot \left(\sum_{k=1}^{t} \frac{\partial \mathbf{h}^{(t)}}{\partial \mathbf{h}^{(k)}} \cdot \frac{\partial \mathbf{h}^{(k)}}{\partial \mathbf{W}_{hh}} \right)$$

Backpropagation through time



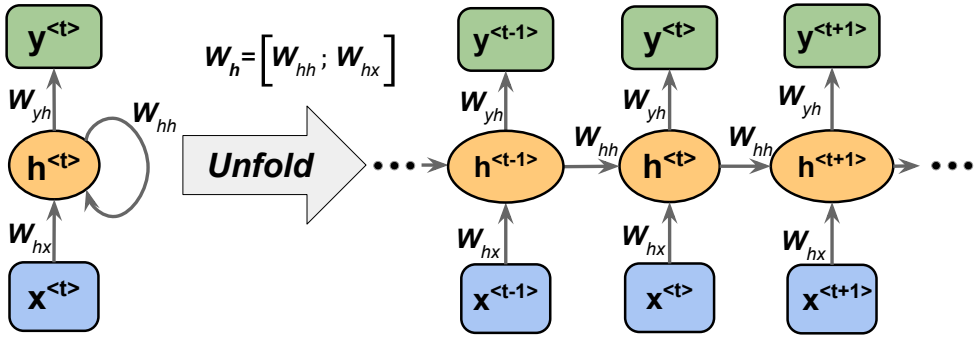
Werbos, Paul J. "Backpropagation through time: what it does and how to do it." *Proceedings of the IEEE* 78, no. 10 (1990): 1550-1560.

$$\frac{\partial L^{(t)}}{\partial \mathbf{W}_{hh}} = \frac{\partial L^{(t)}}{\partial y^{(t)}} \cdot \frac{\partial y^{(t)}}{\partial \mathbf{h}^{(t)}} \cdot \left(\sum_{k=1}^{t} \frac{\partial \mathbf{h}^{(t)}}{\partial \mathbf{h}^{(k)}} \cdot \frac{\partial \mathbf{h}^{(k)}}{\partial \mathbf{W}_{hh}} \right)$$

computed as a multiplication of adjacent time steps:

$$\frac{\partial \mathbf{h}^{(t)}}{\partial \mathbf{h}^{(k)}} = \prod_{i=k+1}^{t} \frac{\partial \mathbf{h}^{(i)}}{\partial \mathbf{h}^{(i-1)}}$$

Backpropagation through time



Werbos, Paul J. "Backpropagation through time: what it does and how to do it." *Proceedings of the IEEE* 78, no. 10 (1990): 1550-1560.

$$L = \sum_{t=1}^{T} L^{(t)} \qquad \frac{\partial L^{(t)}}{\partial \mathbf{W}_{hh}} = \frac{\partial L^{(t)}}{\partial y^{(t)}} \cdot \frac{\partial y^{(t)}}{\partial \mathbf{h}^{(t)}} \cdot \left(\sum_{k=1}^{t} \frac{\partial \mathbf{h}^{(t)}}{\partial \mathbf{h}^{(k)}} \cdot \frac{\partial \mathbf{h}^{(k)}}{\partial \mathbf{W}_{hh}} \right)$$

computed as a multiplication of adjacent time steps:

This is very problematic:
Vanishing/Exploding gradient problem!

$$\frac{\partial \mathbf{h}^{(t)}}{\partial \mathbf{h}^{(k)}} = \prod_{i=k+1}^{t} \frac{\partial \mathbf{h}^{(i)}}{\partial \mathbf{h}^{(i-1)}}$$

Solutions to the vanishing/exploding gradient problems

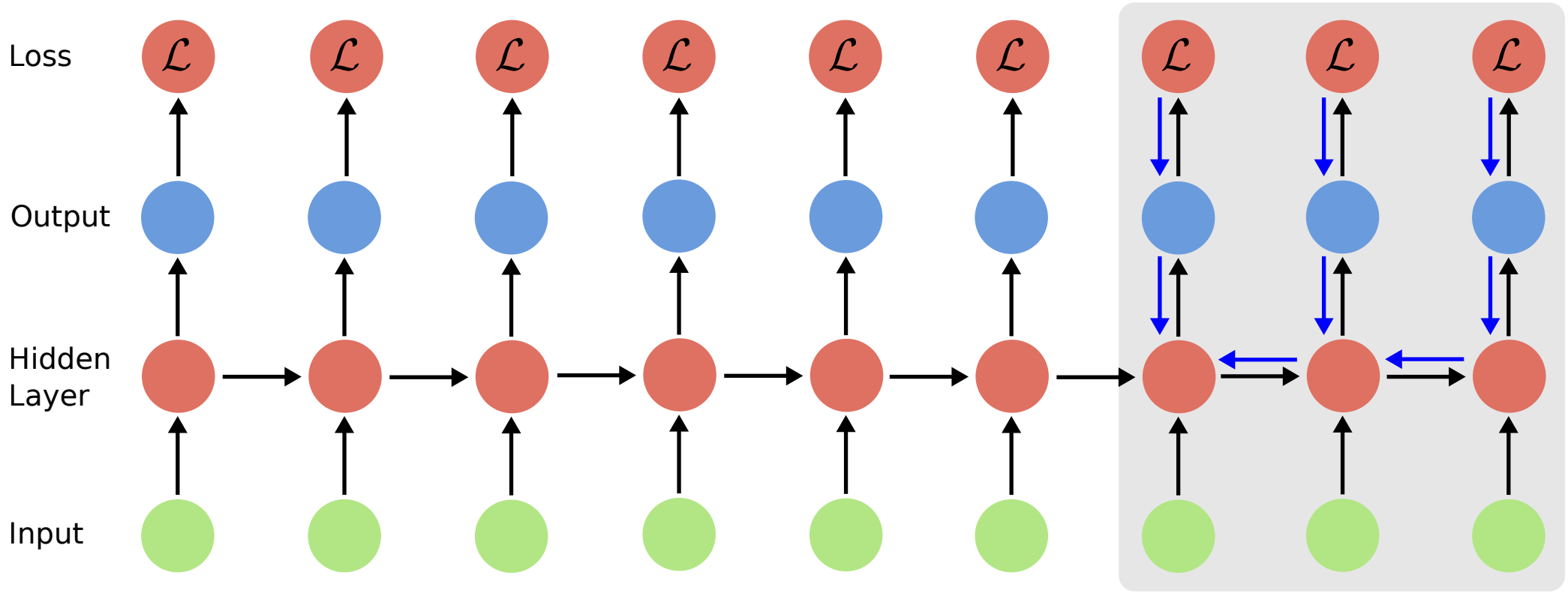
1) Gradient Clipping: set a max value for gradients if they grow to large (solves only exploding gradient problem)

2) Truncated backpropagation through time (TBPTT)
simply limits the number of time steps the signal can backpropagate after each forward pass. E.g., even if the
sequence has 100 elements/steps, we may only backpropagate through 20 or so

3) Long short-term memory (LSTM) -- uses a memory cell for modeling long-range dependencies and avoid vanishing gradient problems

Hochreiter, Sepp, and Jürgen Schmidhuber. "Long short-term memory." *Neural computation* 9, no. 8 (1997): 1735-1780.

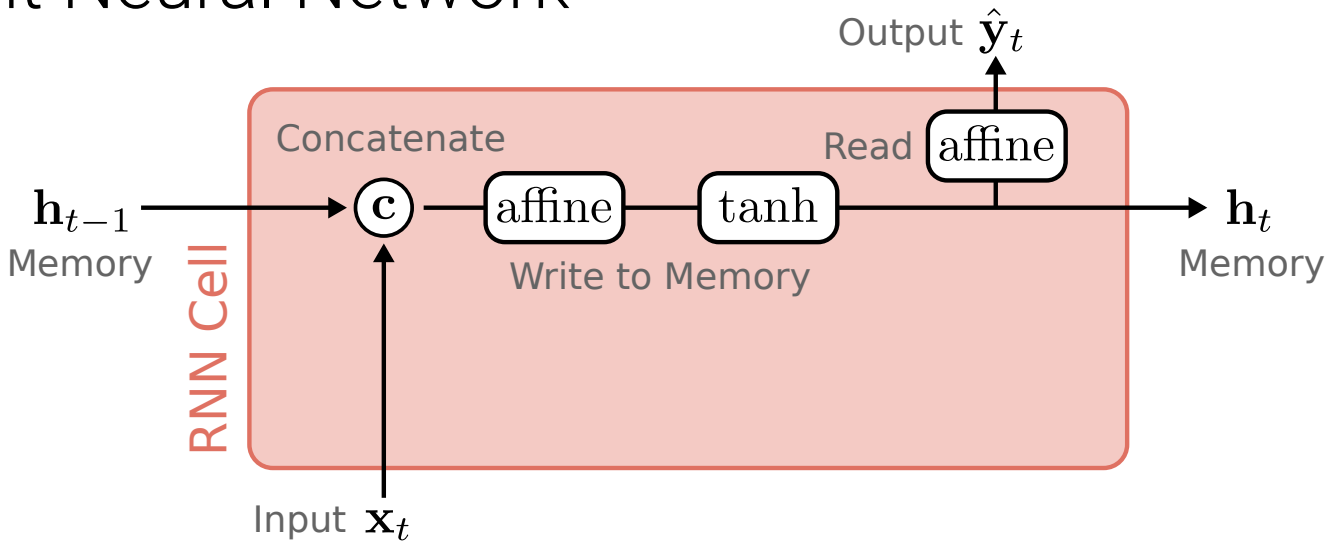
Truncated Backpropagation through Time



- Thus, one typically uses **truncated backpropagation through time** in practice
- Carry hidden states forward in time forever, but stop backpropagation earlier
- Total loss is sum of individual loss functions (= negative log likelihood)

Basic RNN Cell

Basic Recurrent Neural Network



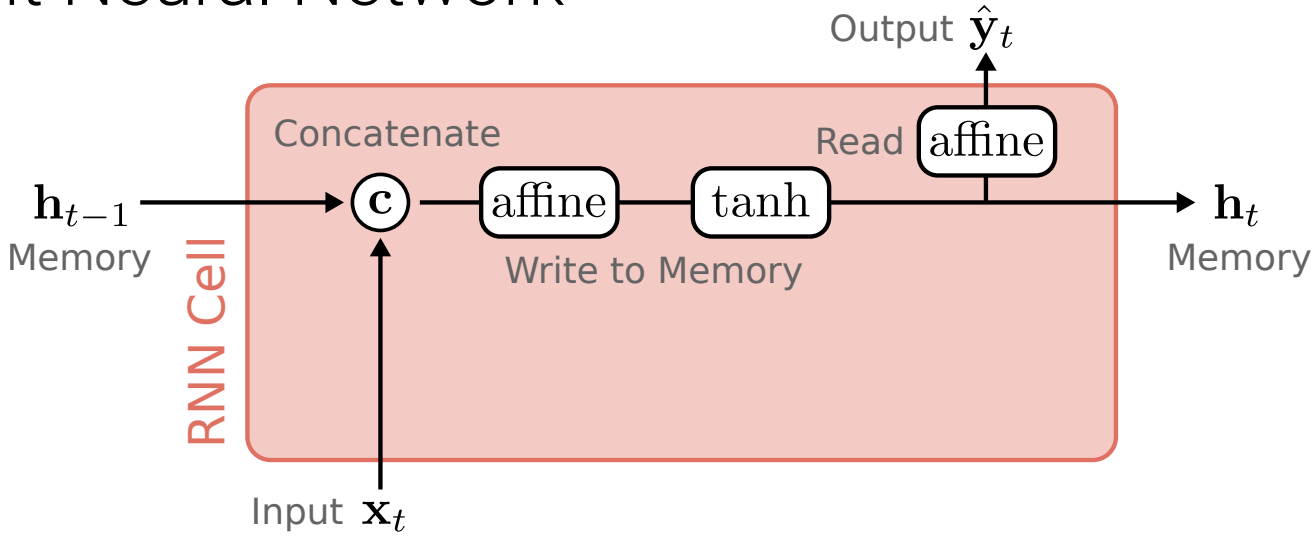
General Form:

$$\mathbf{h}_t = f_h(\mathbf{h}_{t-1}, \mathbf{x}_t)$$

$$\hat{\mathbf{y}}_t = f_y(\mathbf{h}_t)$$

- We use t as the **time index** (in feedforward networks we used i as layer index)
- General form does not specify the form of the hidden and output mappings
- Important: f_h and f_y **do not change over time**, unlike in layers of feedforward net

Basic Recurrent Neural Network



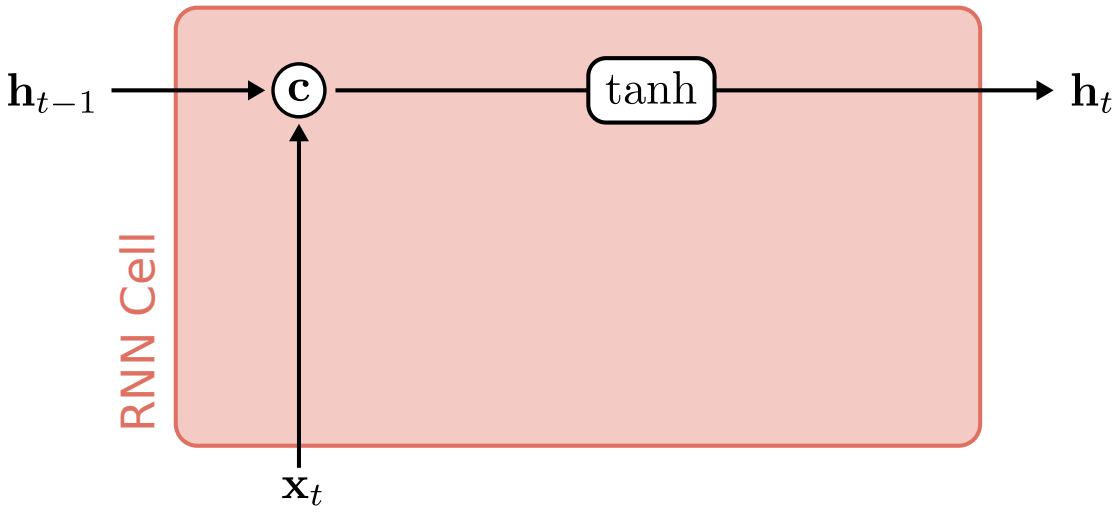
Single-Layer RNN:

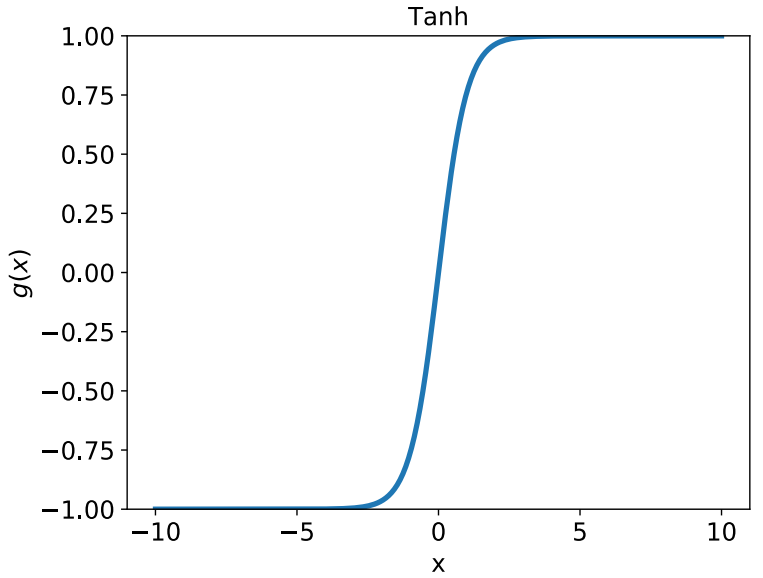
$$\mathbf{h}_t = \tanh(\mathbf{A}_h \mathbf{h}_{t-1} + \mathbf{A}_x \mathbf{x}_t + \mathbf{b}_h)$$

$$\hat{\mathbf{y}}_t = \mathbf{A}_y \mathbf{h}_t + \mathbf{b}_y$$

- Hidden state $\mathbf{h}_t$ = linear combination of input $\mathbf{x}_t$ and previous hidden state $\mathbf{h}_{t-1}$
- Output $\hat{\mathbf{y}}_t$ = linear prediction based on current hidden state $\mathbf{h}_t$
- $\tanh(\cdot)$ is commonly used as activation function (data is in the range $[-1, 1]$)
- Parameters $\mathbf{A}_h$, $\mathbf{A}_x$, $\mathbf{A}_y$, $\mathbf{b}_h$, $\mathbf{b}_y$ are constant over time

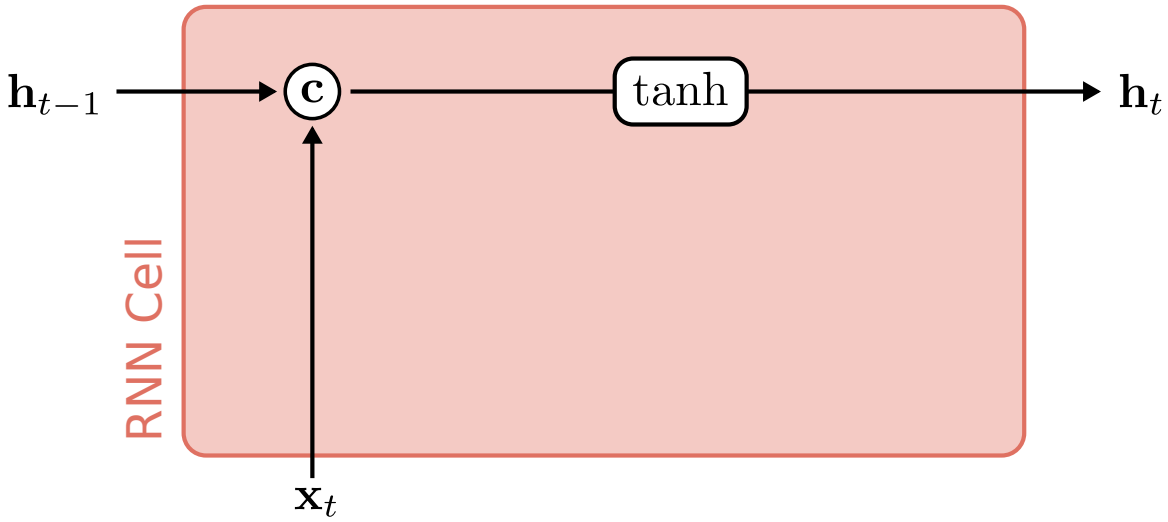
Recurrent Neural Network





- The state update H_t is modeled using a zero-centered $\tanh(\cdot)$
- $\tanh(\cdot)$ assumes that the processed data is in the range $[-1, 1]$
- Remark: we omit the affine transformations and the output layer for clarity

Vanishing / Exploding Gradients



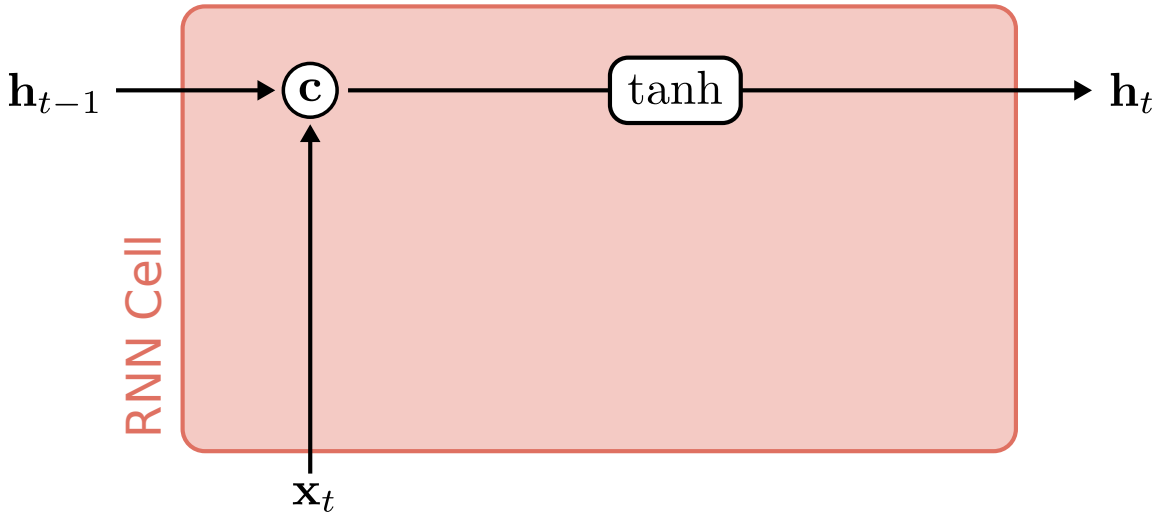
$$h_t = \tanh\left(a_h\, h_{t-1} \;+\; a_x\, x_t \;+\; b\right)$$

$$\frac{\partial h_t}{\partial h_{t-1}} = \tanh' a_h \quad \text{with} \quad \tanh' = \frac{\partial \tanh(x)}{\partial x}$$

What is the problem with vanilla RNNs?

- Let us consider an RNN with one dimensional hidden state $h_t \in \mathbb{R}$
- We have: $$\frac{\partial h_t}{\partial h_{t-k}} = \frac{\partial h_t}{\partial h_{t-1}} \frac{\partial h_{t-1}}{\partial h_{t-2}} \cdots \frac{\partial h_{t-k+1}}{\partial h_{t-k}} = \left(\prod_{i=t-k+1}^{t} \tanh'_i \right) a_h^k$$
- Thus, the gradient vanishes if $\tanh(\cdot)$ saturates as in feedforward networks
- RNNs require careful initialization to avoid saturating activation functions

Vanishing / Exploding Gradients



$$h_t = \tanh\left(a_h\, h_{t-1} + a_x\, x_t + b\right)$$
$$\approx a_h\, h_{t-1} + a_x\, x_t + b$$
$$\frac{\partial h_t}{\partial h_{t-1}} \approx a_h$$

However, gradients might still misbehave:

- Let us now assume that the RNN has been initialized well such that the activation functions are not saturated $\Rightarrow a_h\, h_{t-1} + a_x\, x_t + b \in [-1, 1] \Rightarrow \tanh(x) \approx x$
- We now have:

$$\frac{\partial h_t}{\partial h_{t-k}} = \frac{\partial h_t}{\partial h_{t-1}} \frac{\partial h_{t-1}}{\partial h_{t-2}} \cdots \frac{\partial h_{t-k+1}}{\partial h_{t-k}} = a_h^k$$

Vanishing / Exploding Gradients

$$\frac{\partial h_t}{\partial h_{t-k}} = \frac{\partial h_t}{\partial h_{t-1}} \frac{\partial h_{t-1}}{\partial h_{t-2}} \cdots \frac{\partial h_{t-k+1}}{\partial h_{t-k}} = a_h^k$$

For $a_h > 1$ gradients will **explode** (become very large, cause divergence):

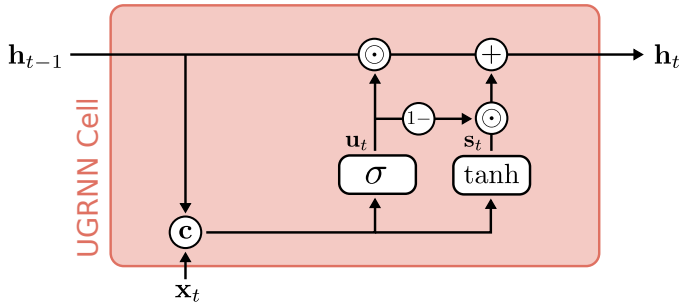
- Example: For $a_h = 1.1$ and $k = 100$ we have $\partial h_t / \partial h_{t-k} = a_h^k = 13781$
- This problem is often addressed in practice using **gradient clipping**
- Forward values do not explode due to bounded $\tanh(\cdot)$ activation function

For $a_h < 1$ gradients will **vanish** (no learning in earlier time steps):

- Example: For $a_h = 0.9$ and $k = 100$ we have $\partial h_t / \partial h_{t-k} = a_h^k = 0.0000266$
- Avoiding this problem requires an **architectural change**
- But residual connections do not work here as the parameters are shared across time and the input and desired output at each time step are different

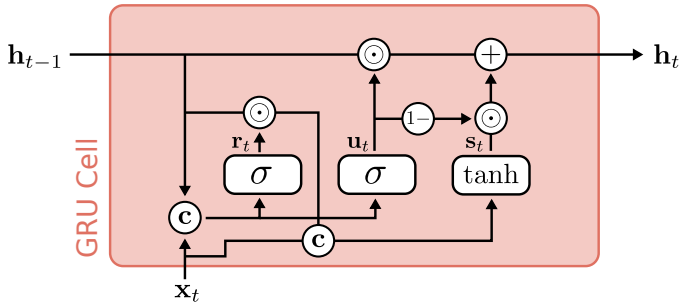
Gated Recurrent Networks

UGRNN



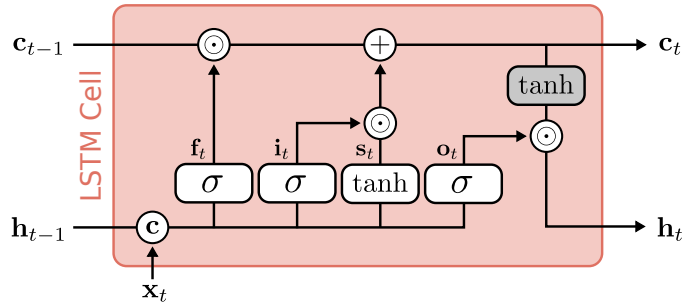
Collins, 2017

GRU



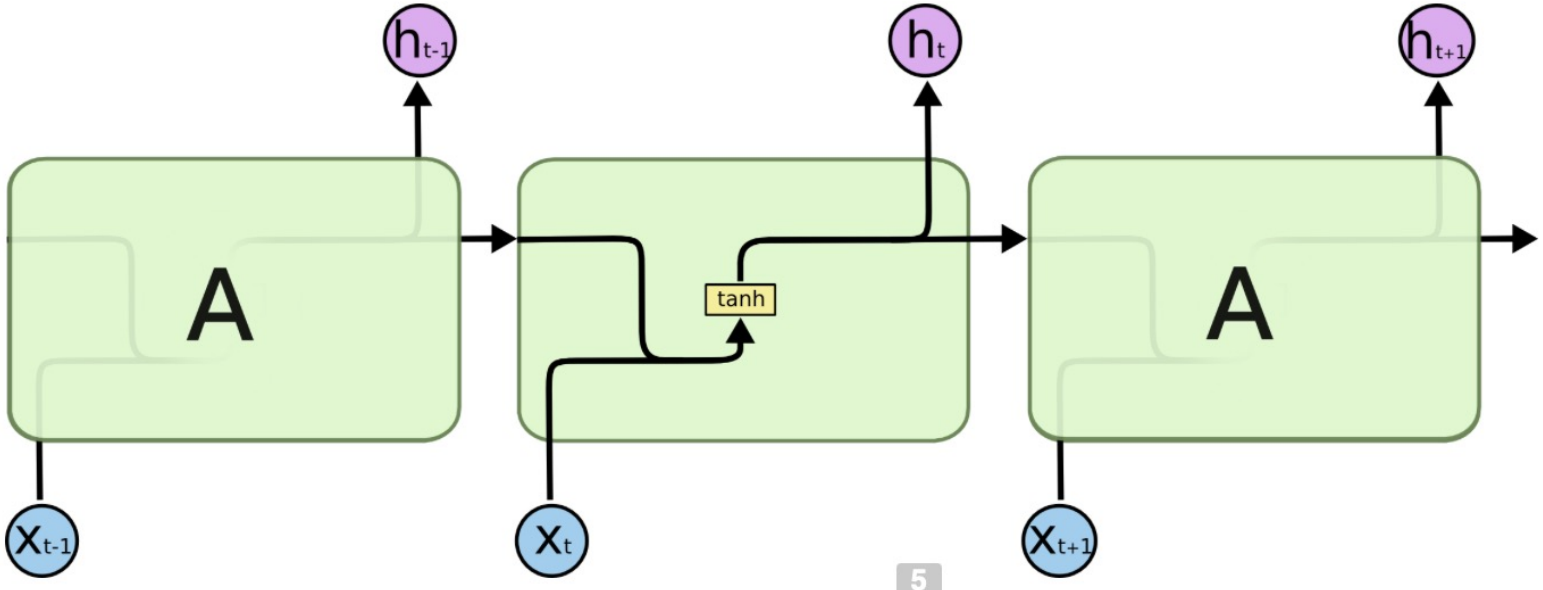
Cho, 2014

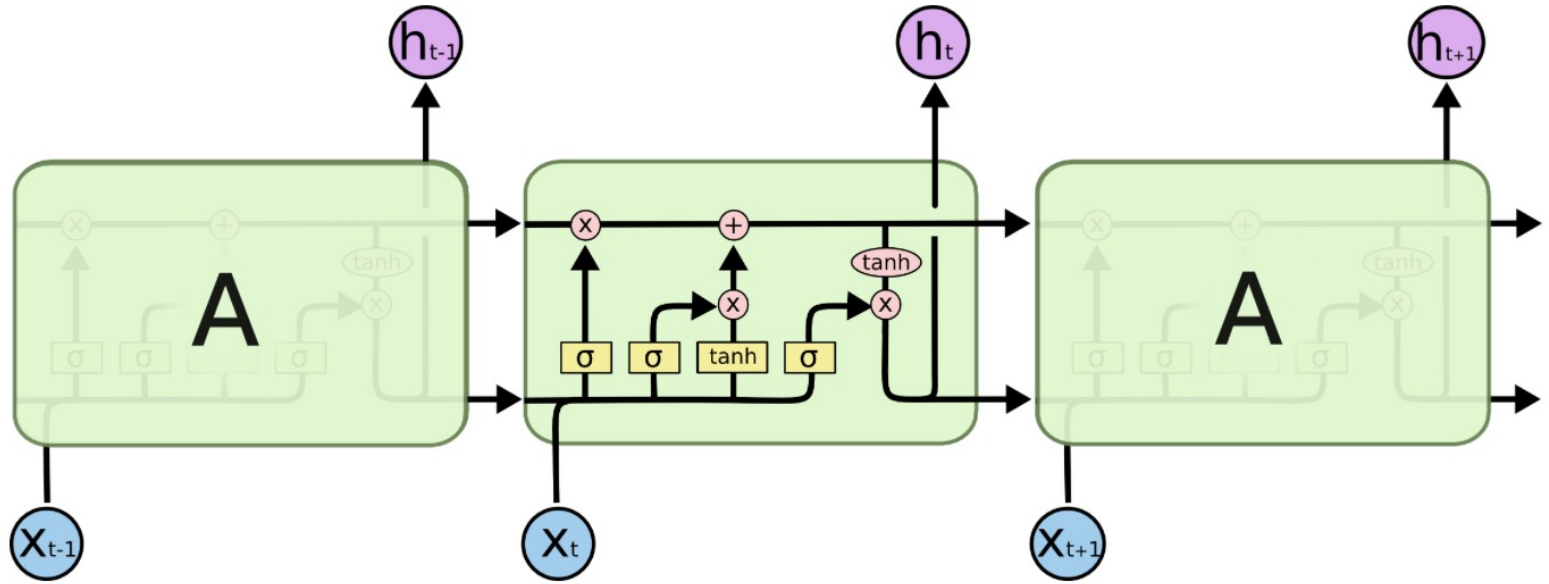
LSTM



Hochreiter, 1997

- **UGRNN:** Update Gate Recurrent Neural Network
- **GRU:** Gated Recurrent Unit
- **LSTM:** Long Short-Term Memory
- LSTM was the first and most transformative (revolutionized NLP in 2015, e.g. at Google), but also most complex model. UGRNN and GRU work similarly well.
- Common to all architectures: **gates** for filtering information


h_{t-1}
h_t
h_{t+1}
A
tanh
A
X_{t-1}
X_t
X_{t+1}


h_{t-1}
h_t
h_{t+1}
A
tanh
σ
σ
tanh
σ
A
X_{t-1}
X_t
X_{t+1}

Long-short term memory (LSTM)

LSTM cell:

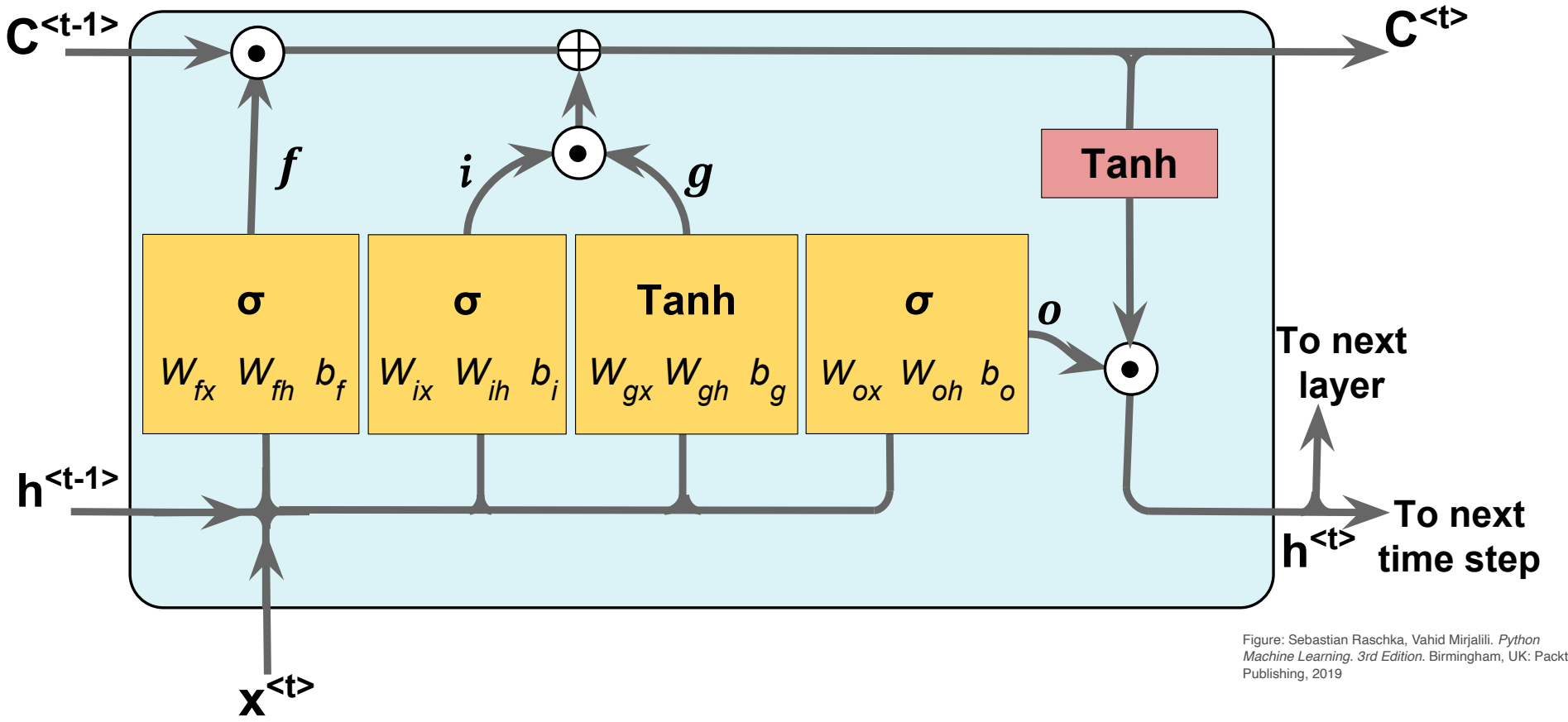
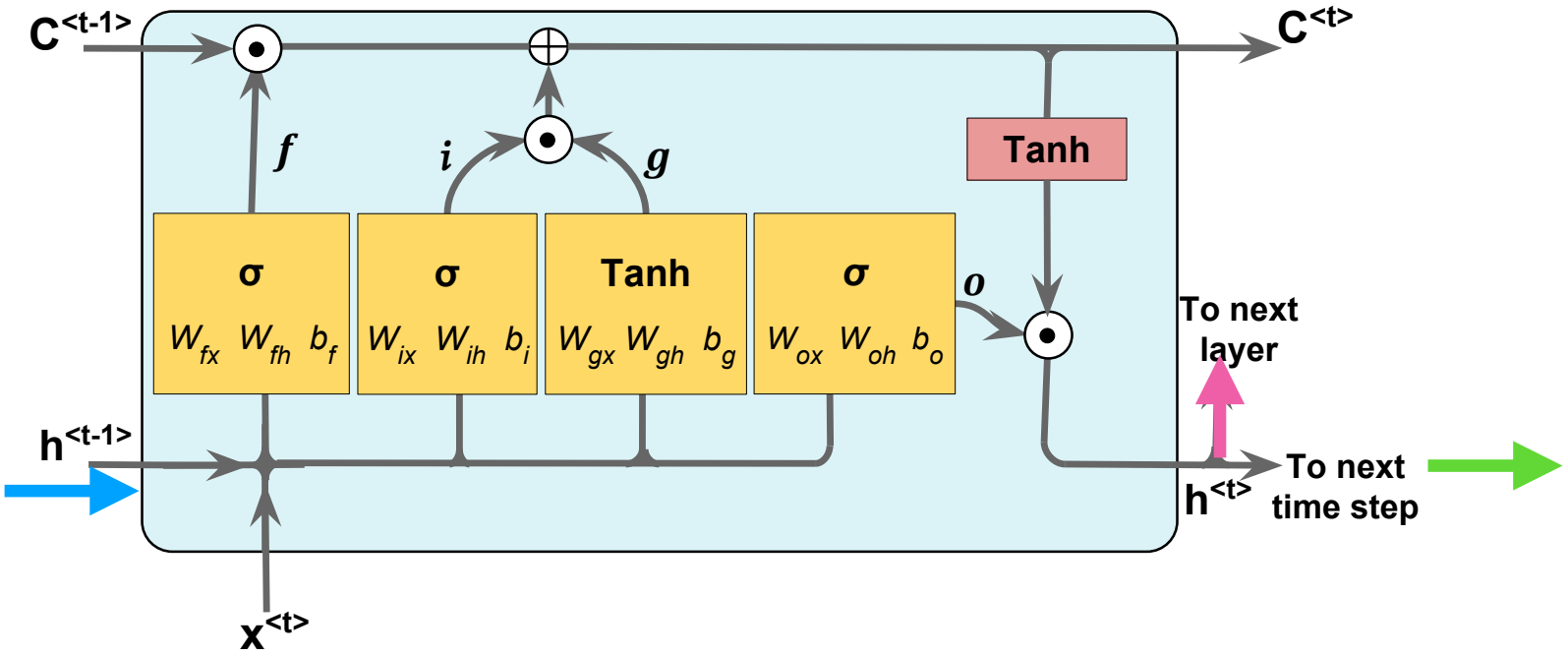
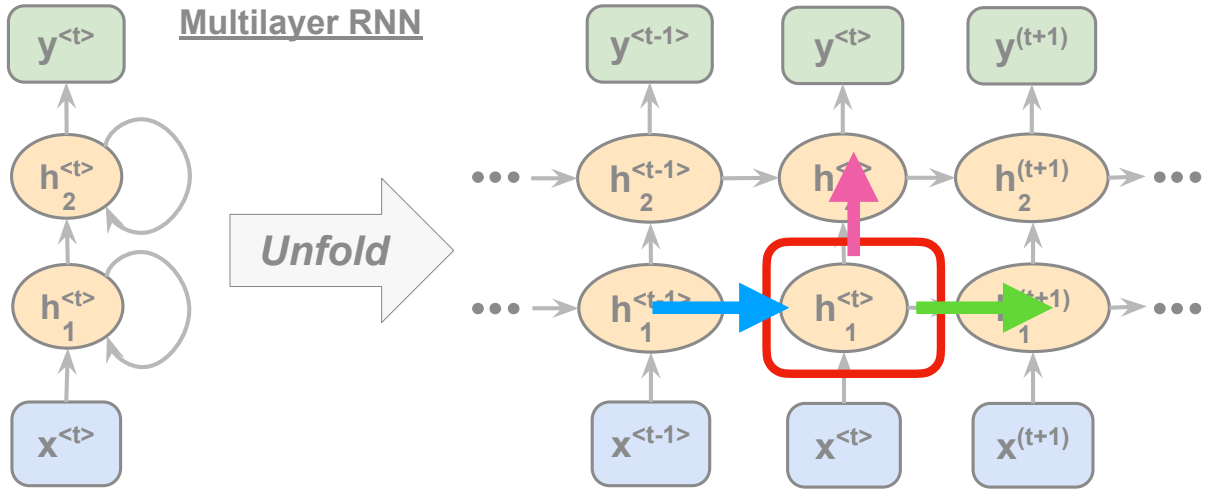


Figure: Sebastian Raschka, Vahid Mirjalili. *Python Machine Learning. 3rd Edition.* Birmingham, UK: Packt Publishing, 2019

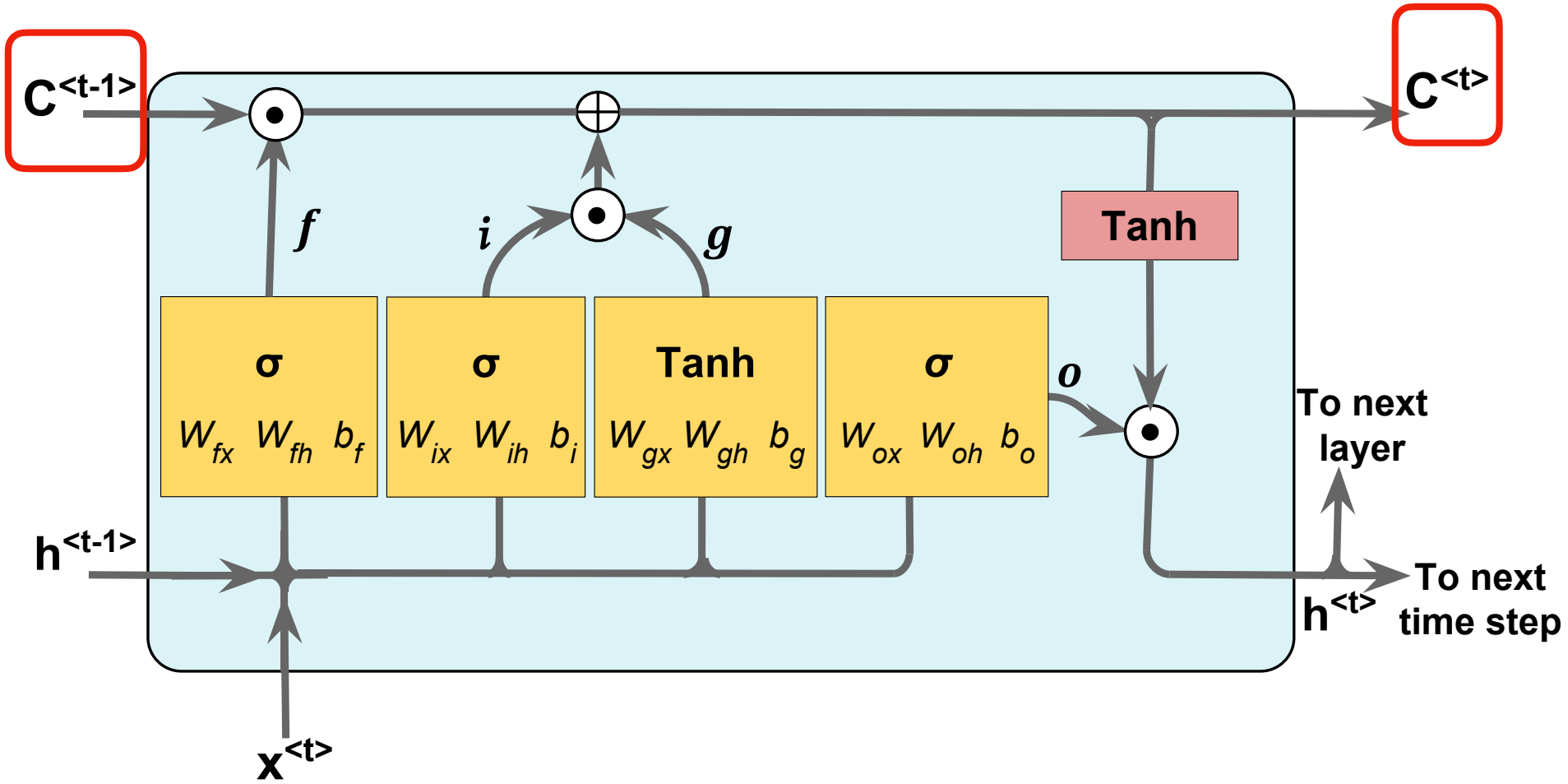

C<t-1>
C<t>
f
i
g
o
Tanh
σ
σ
Tanh
σ
W_fx W_fh b_f
W_ix W_ih b_i
W_gx W_gh b_g
W_ox W_oh b_o
h<t-1>
x<t>
To next layer
h<t>
To next time step


Multilayer RNN
y<t>
h2<t>
h1<t>
x<t>
Unfold
y<t-1>
y<t>
y(t+1)
h2<t-1>
h2(t+1)
h1<t-1>
h1<t>
x<t-1>
x<t>
x(t+1)

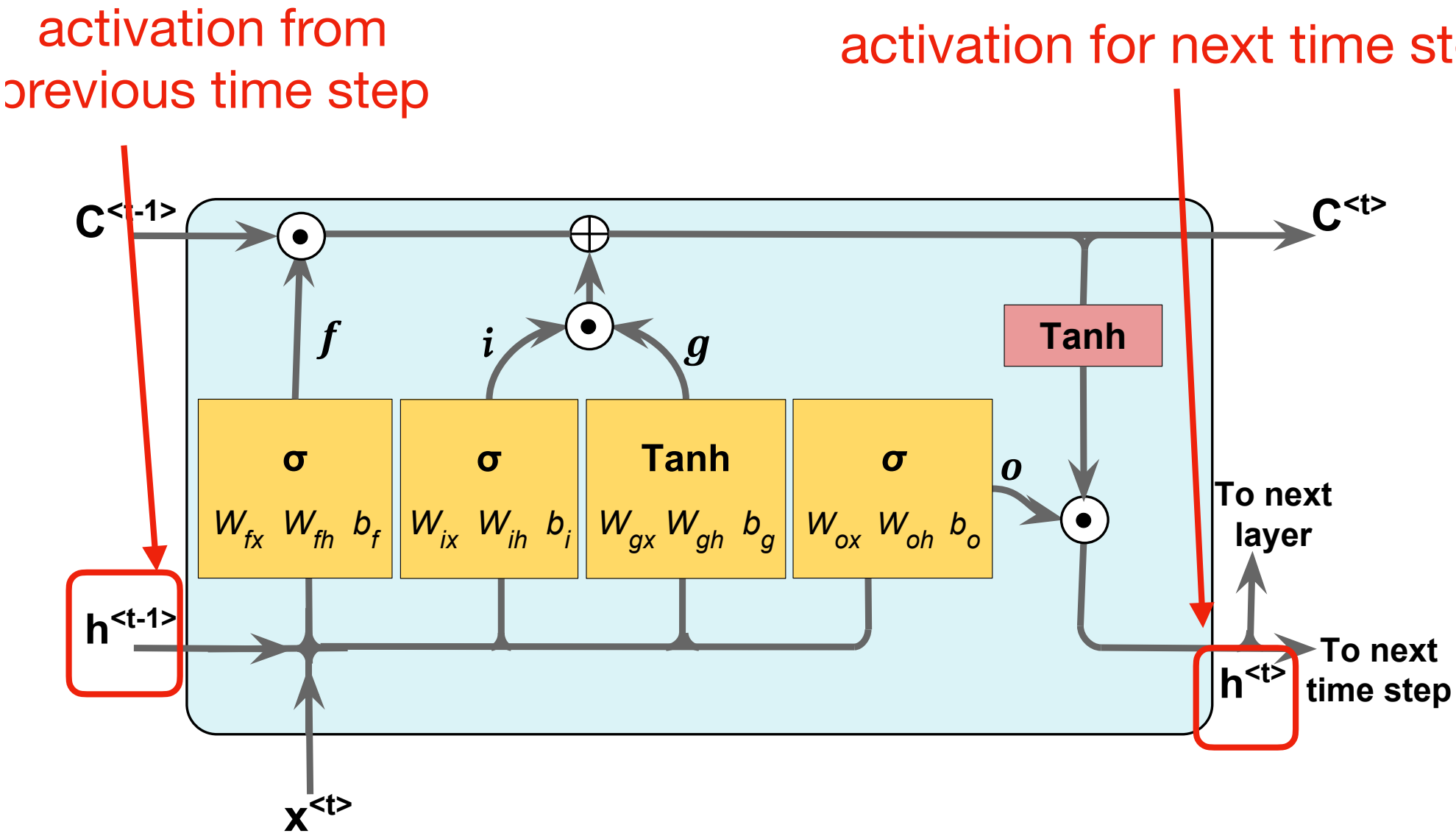
Long-short term memory (LSTM)

Cell state at previous time step

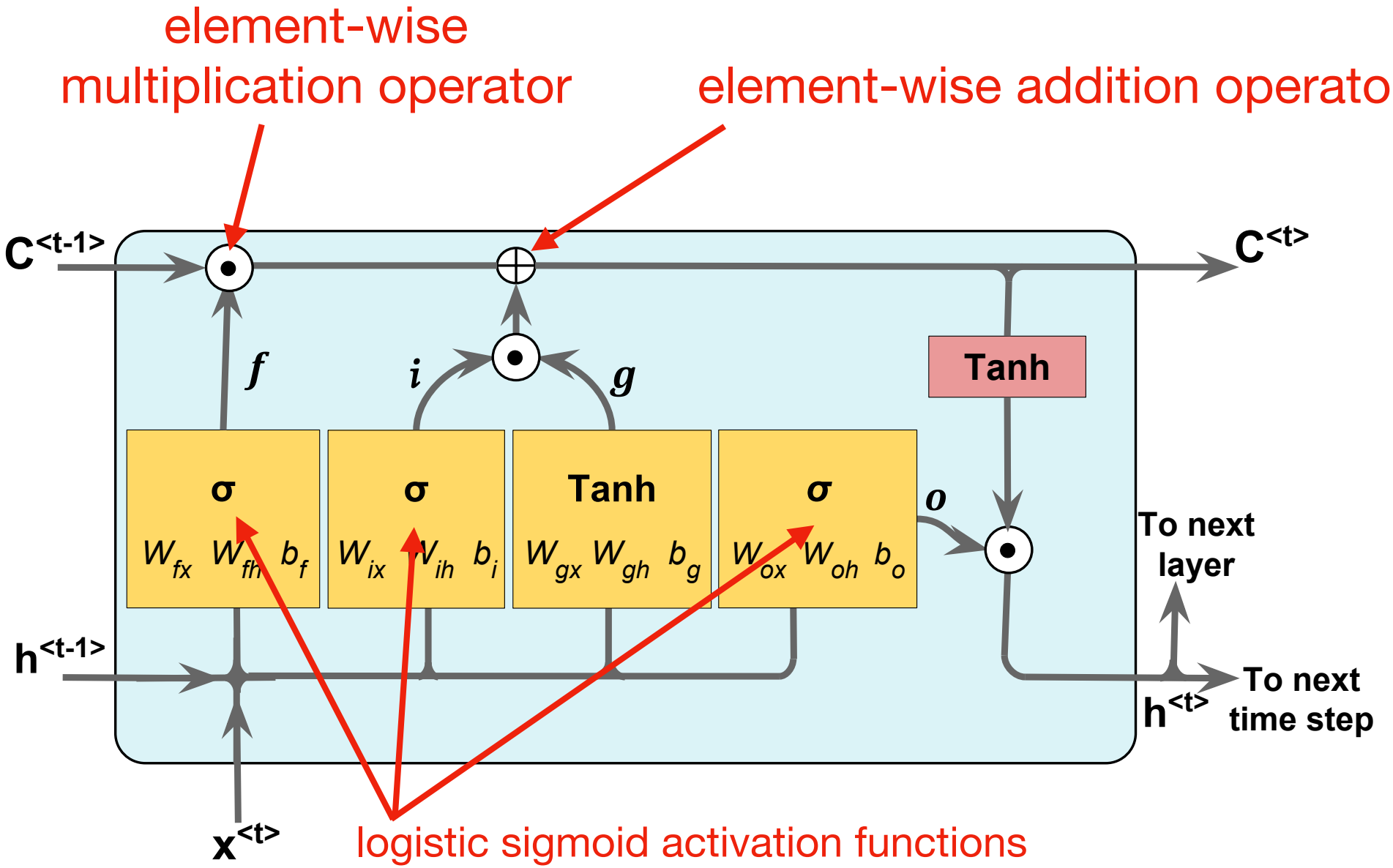
Cell state at current time step



Long-short term memory (LSTM)



Long-short term memory (LSTM)

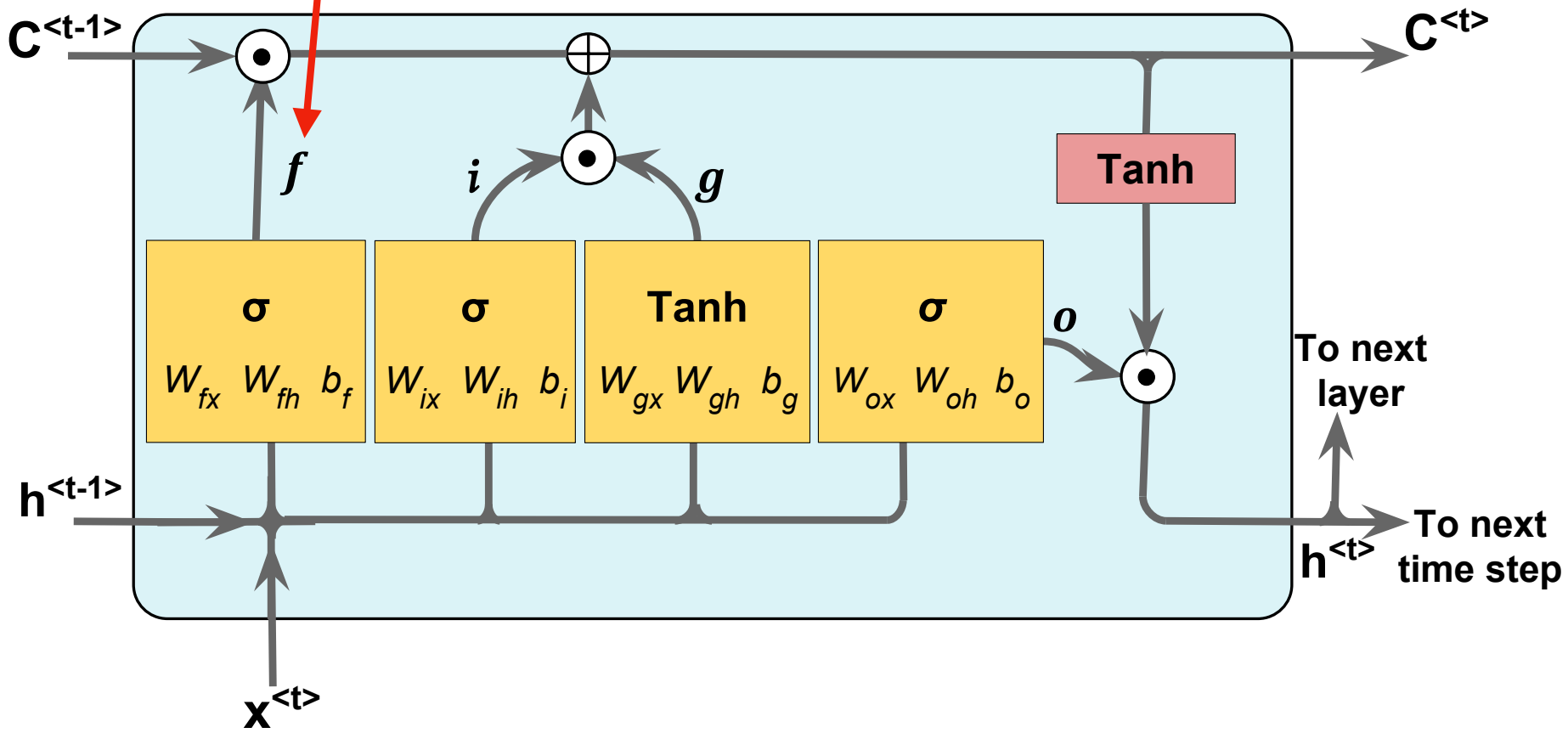


Long-short term memory (LSTM)

Gers, Felix A., Jürgen Schmidhuber, and Fred Cummins. "Learning to forget: Continual prediction with LSTM." (1999): 850-855.

"Forget Gate": controls which information is remembered, and which is forgotten; can reset the cell state

$$f_t = \sigma\left(\mathbf{W}_{fx}\mathbf{x}^{\langle t\rangle} + \mathbf{W}_{fh}\mathbf{h}^{\langle t-1\rangle} + \mathbf{b}_f\right)$$

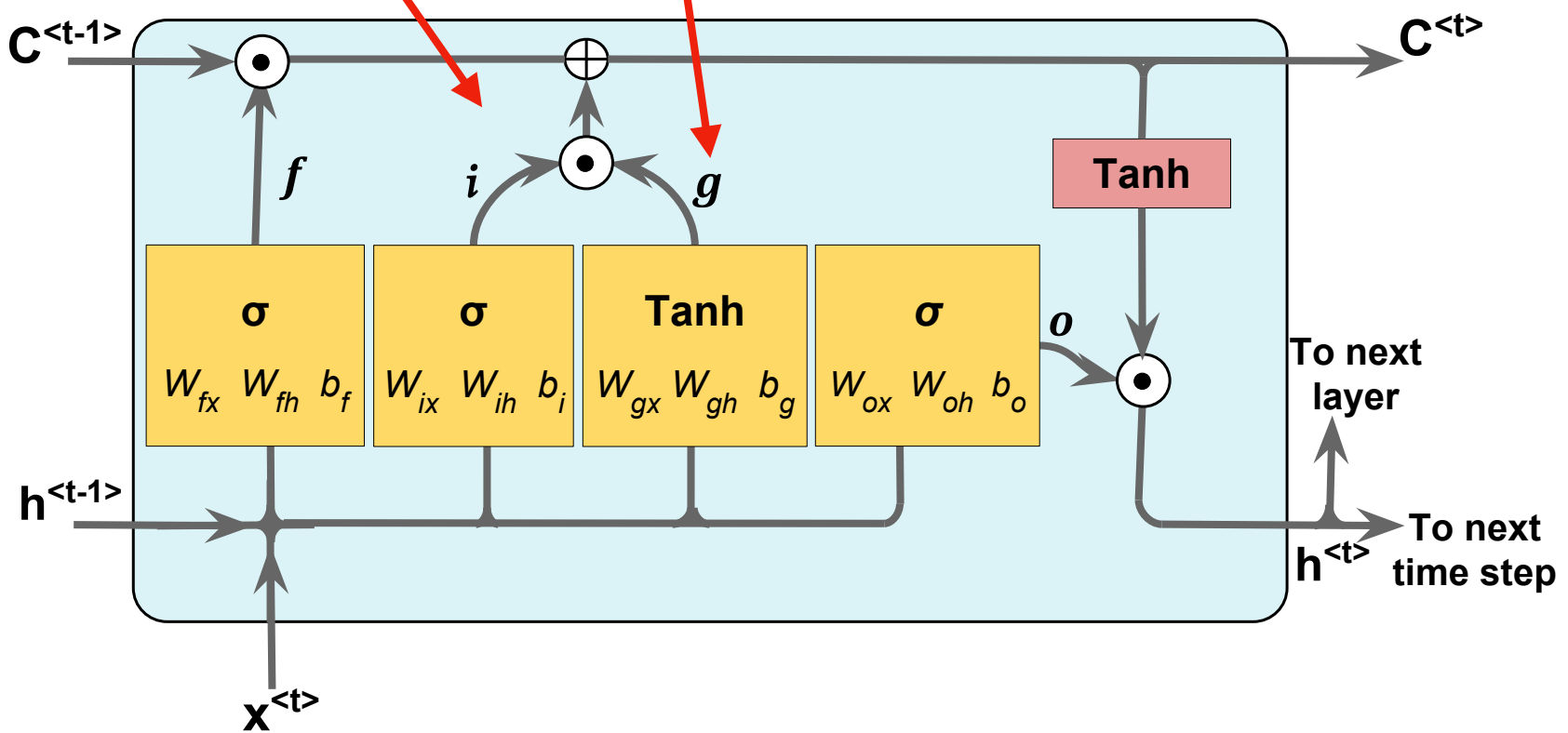


Long-short term memory (LSTM)

"Input Gate": $\mathbf{i}_t = \sigma\left(\mathbf{W}_{ix}\mathbf{x}^{\langle t\rangle} + \mathbf{W}_{ih}\mathbf{h}^{\langle t-1\rangle} + \mathbf{b}_i\right)$

"Input Node":

$$\mathbf{g}_t = \tanh\left(\mathbf{W}_{gx}\mathbf{x}^{\langle t\rangle} + \mathbf{W}_{gh}\mathbf{h}^{\langle t-1\rangle} + \mathbf{b}_g\right)$$



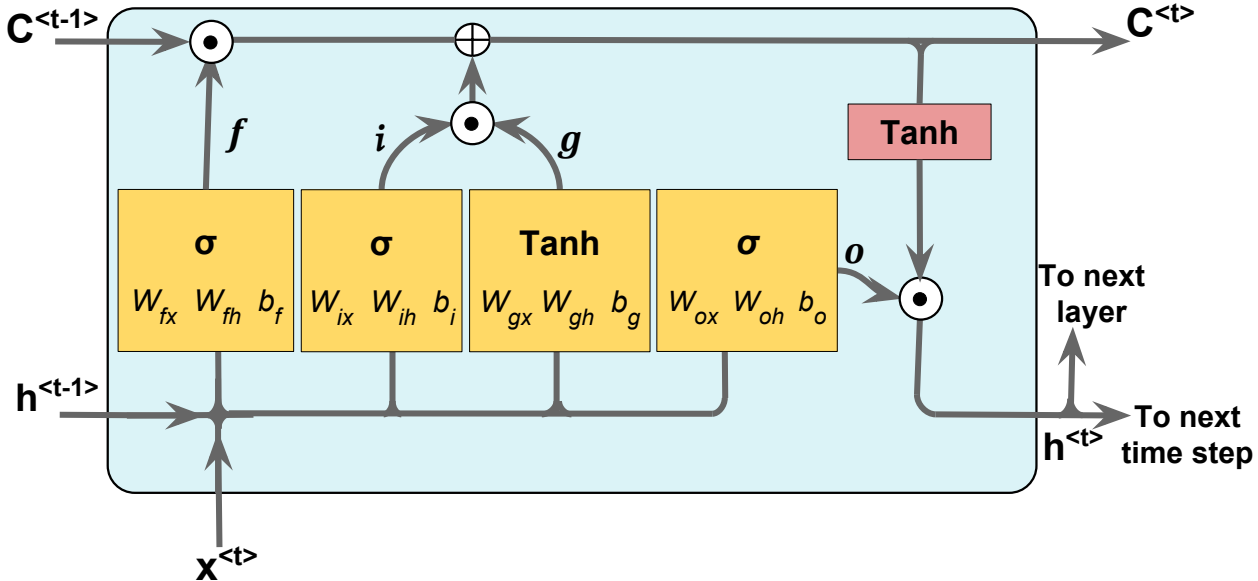
Long-short term memory (LSTM)

Brief summary of the gates so far ...

Forget Gate Input Node Input Gate

$$C^{\langle t \rangle} = \left(C^{\langle t-1 \rangle} \odot f_t \right) \oplus \left(i_t \odot g_t \right)$$

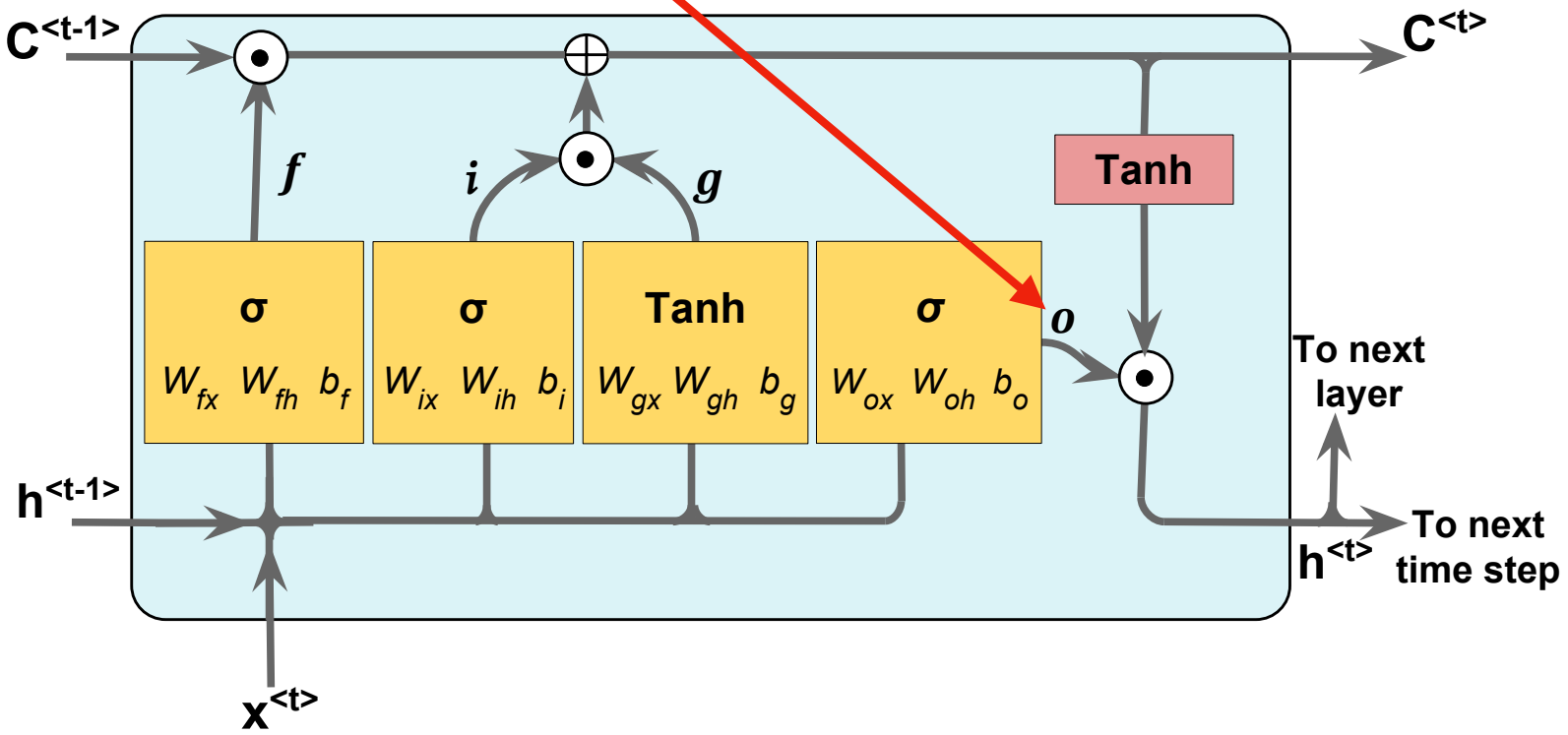
For updating the cell state



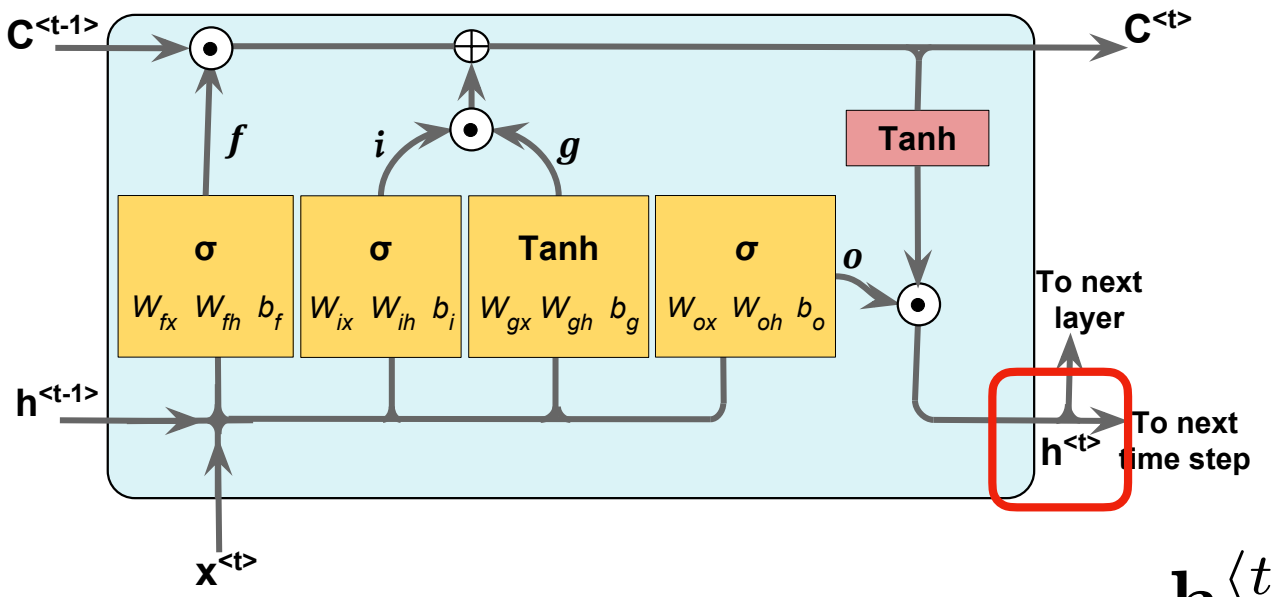
Long-short term memory (LSTM)

Output gate for updating the values of hidden units:

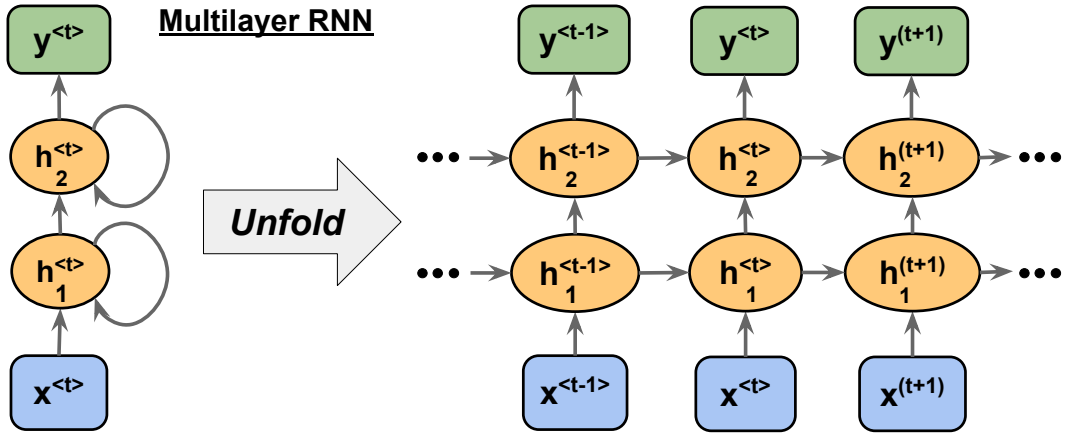
$$\mathbf{o}_t = \sigma\left(\mathbf{W}_{ox}\mathbf{x}^{\langle t\rangle} + \mathbf{W}_{oh}\mathbf{h}^{\langle t-1\rangle} + \mathbf{b}_o\right)$$



Long-short term memory (LSTM)

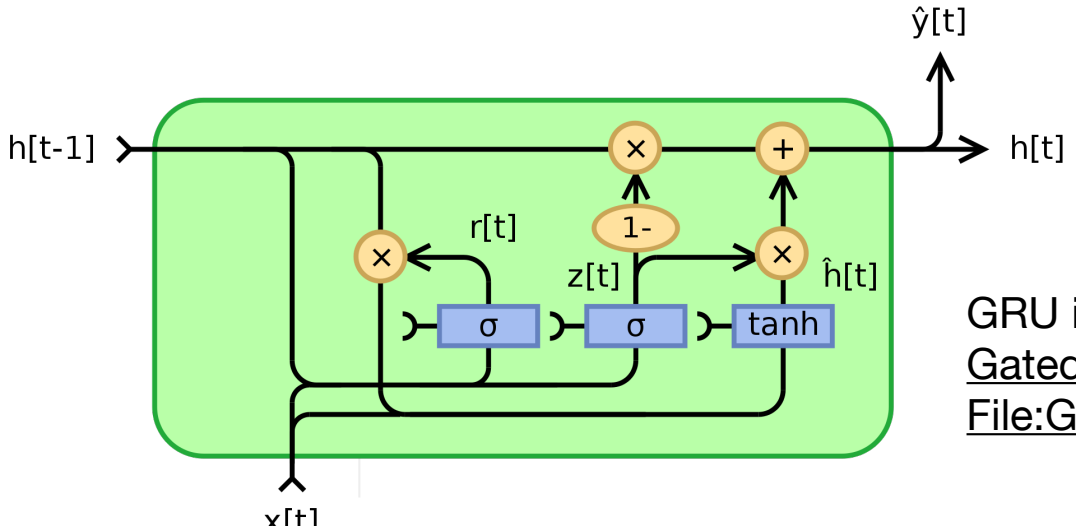


$$\mathbf{h}^{\langle t \rangle} = \mathbf{o}_t \odot \tanh\left(\mathbf{C}^{\langle t \rangle}\right)$$



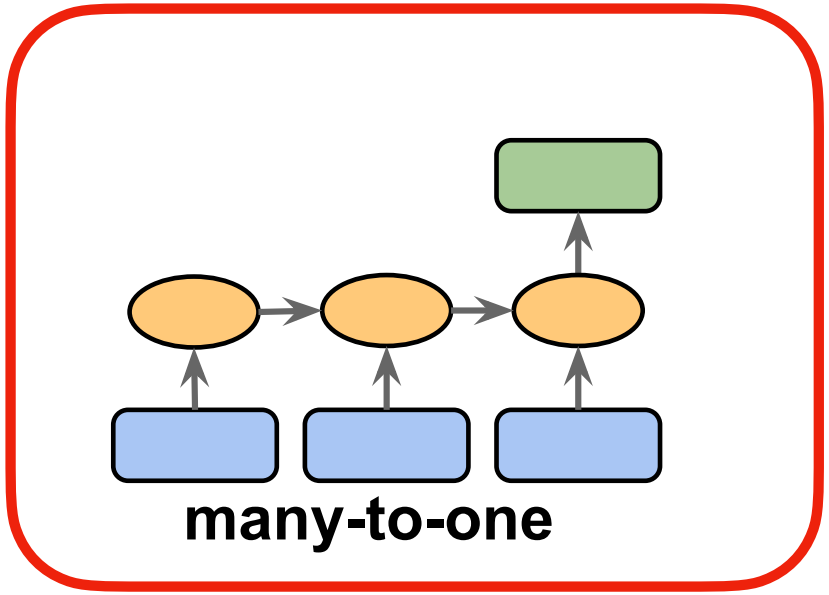
Long-short term memory (LSTM)

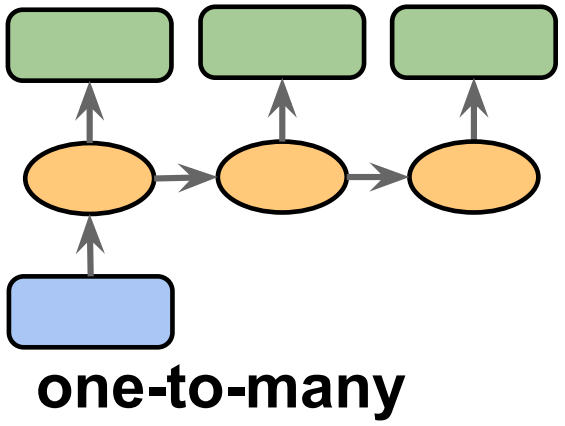
- Still popular and widely used today
- A recent, related approach is the Gated Recurrent Unit (GRU)
  Cho, Kyunghyun, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. "Learning phrase representations using RNN encoder-decoder for statistical machine translation." *arXiv preprint arXiv:1406.1078* (2014).
- Nice article exploring LSTMs and comparing them to GRUs
  Jozefowicz, Rafal, Wojciech Zaremba, and Ilya Sutskever. "An empirical exploration of recurrent network architectures." In *International Conference on Machine Learning*, pp. 2342-2350. 2015.

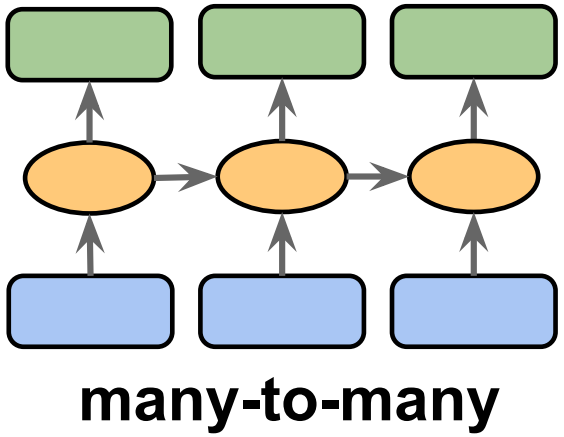


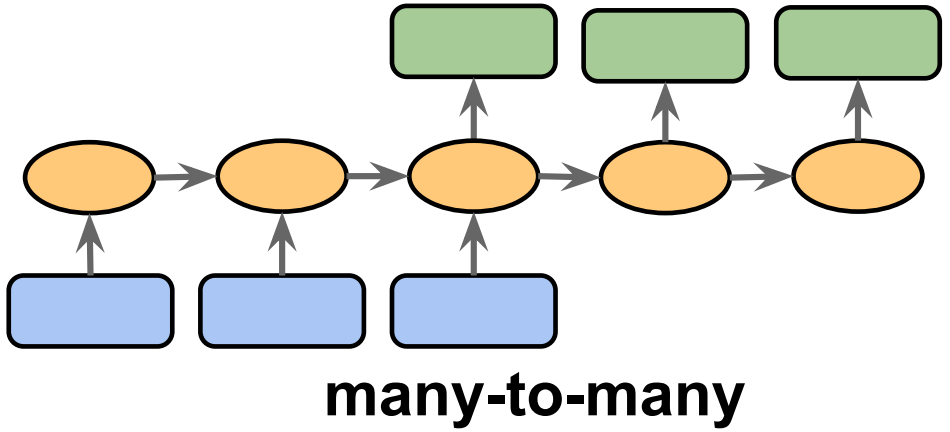
GRU image source: https://en.wikipedia.org/wiki/Gated_recurrent_unit#/media/File:Gated_Recurrent_Unit,_base_type.svg

Different Types of Sequence Modeling Tasks









A Classic Approach for Text Classification: Bag-of-Words Model

"Raw" training dataset

$\mathbf{x}^{[1]}$ = "The sun is shining"

$\mathbf{x}^{[2]}$ = "The weather is sweet"

$\mathbf{x}^{[3]}$ = "The sun is shining, the weather is sweet, and one and one is two"

$$\mathbf{y} = [0, 1, 0]$$

class labels

```
vocabulary = {
    'and': 0,
    'is': 1
    'one': 2,
    'shining': 3,
    'sun': 4,
    'sweet': 5,
    'the': 6,
    'two': 7,
    'weather': 8,
}
```

Training set as design matrix

$$\mathbf{X} = \begin{bmatrix} 0 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 1 \\ 2 & 3 & 2 & 1 & 1 & 1 & 2 & 1 & 1 \end{bmatrix}$$

$$\mathbf{y} = [0, 1, 0]$$

class labels

training

Classifier

(e.g., logistic regression, MLP, ...)

Raschka & Mirjalili. *Python Machine Learning* 3rd Ed.
https://github.com/rasbt/python-machine-learning-book-3rd-edition/blob/master/ch08/ch08.ipynb

RNN Step 1): Building the Vocabulary

"Raw" training dataset

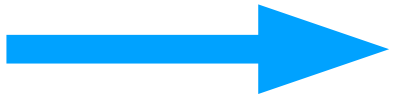
$\mathbf{x}^{[1]} =$ "The sun is shining"

$\mathbf{x}^{[2]} =$ "The weather is sweet"

$\mathbf{x}^{[3]} =$ "The sun is shining,
the weather is sweet, and
one and one is two"

$\mathbf{y} = [0, 1, 0]$

class labels



```
vocabulary = {
    '<unk>': 0,
    'and': 1,
    'is': 2
    'one': 3,
    'shining': 4,
    'sun': 5,
    'sweet': 6,
    'the': 7,
    'two': 8,
    'weather': 9,
    '<pad>': 10
}
```

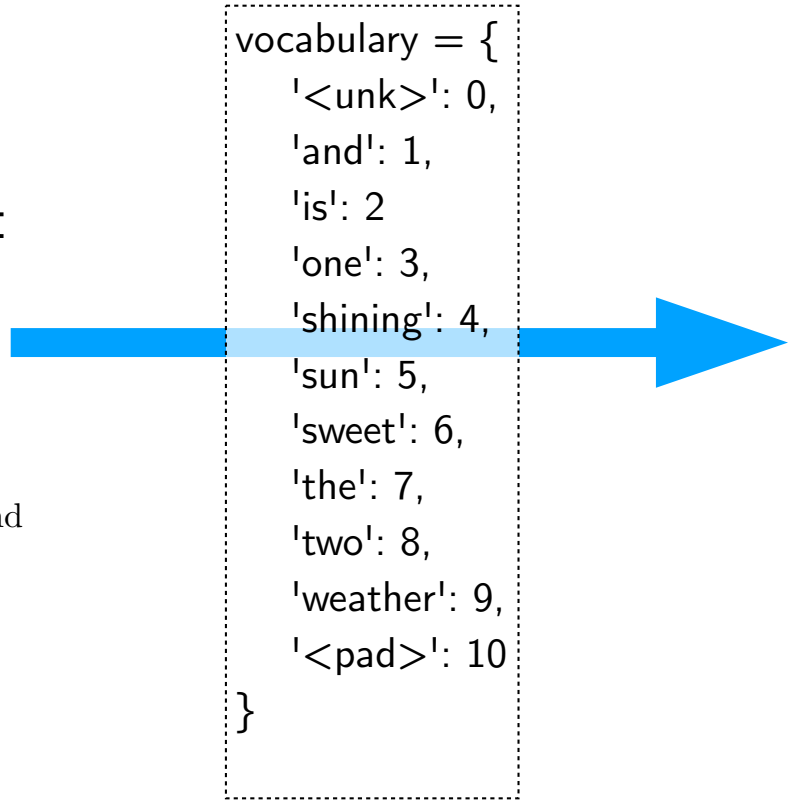
RNN Step 2): Training Example Texts to Indices

"Raw" training dataset

$\mathbf{x}^{[1]}$ = "The sun is shining"

$\mathbf{x}^{[2]}$ = "The weather is sweet"

$\mathbf{x}^{[3]}$ = "The sun is shining,
the weather is sweet, and
one and one is two"



$\mathbf{x}^{[1]}$ = "The sun is shining"

[7 5 2 4 ... 10 10 10]

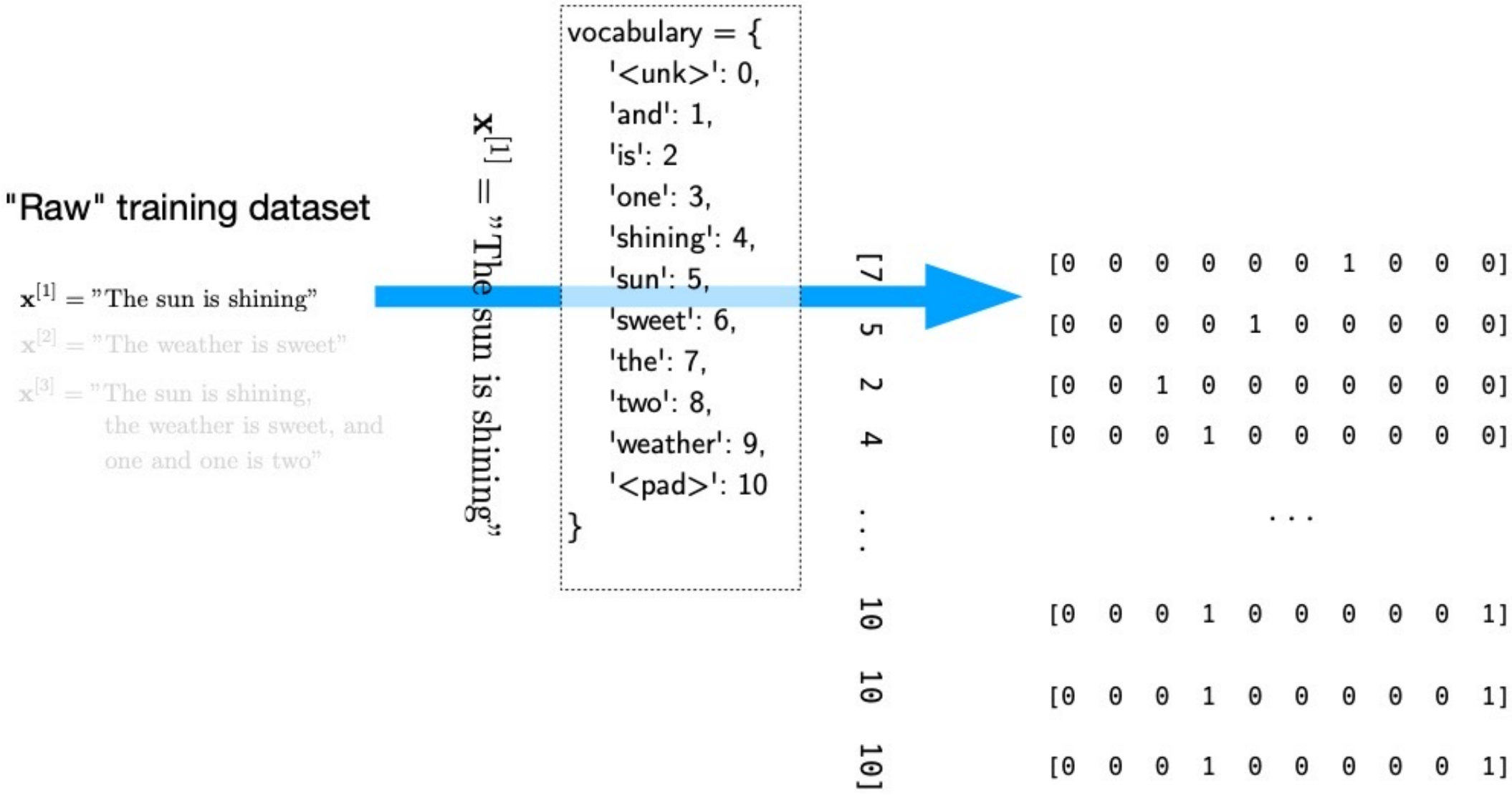
$\mathbf{x}^{[2]}$ = "The weather is sweet"

[7 9 2 6 ... 10 10 10]

$\mathbf{x}^{[3]}$ = "The sun is shining,
the weather is sweet, and
one and one is two"

[7 5 2 4 ... 3 2 8]

RNN Step 3): Indices to One-Hot Representation



RNN Step 4): One-Hot to Real via Embedding Matrix

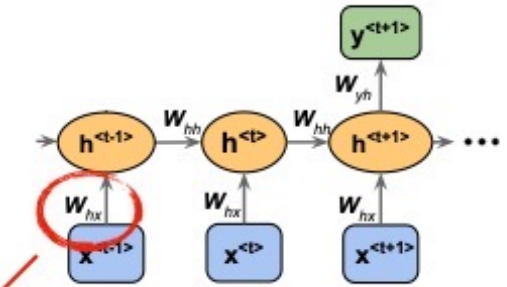
$\mathbf{x}^{[1]}$ = "The sun is shining"

```
vocabulary = {
    '<unk>': 0,
    'and': 1,
    'is': 2
    'one': 3,
    'shining': 4,
    'sun': 5,
    'sweet': 6,
    'the': 7,
    'two': 8,
    'weather': 9,
    '<pad>': 10
}
```

[7 5 2 4 ... 10 10 10]

[0 0 0 0 0 0 1 0 0 0]

Embedding is a linear layer



Embedding matrix

One-hot vector

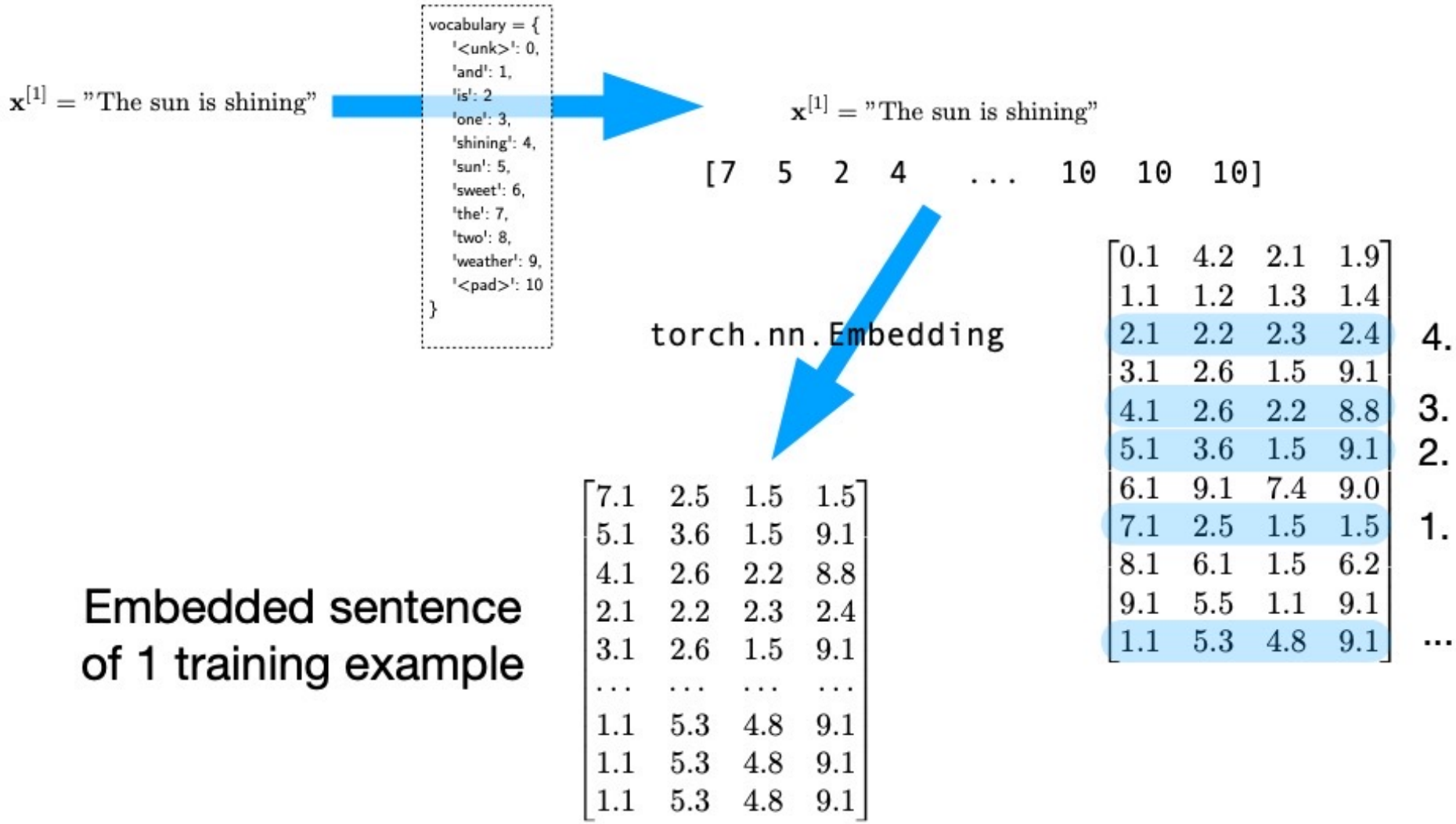
[0 0 0 0 0 0 1 0 0 0]

$$\times \begin{bmatrix} 0.1 & 4.2 & 2.1 & 1.9 \\ 1.1 & 1.2 & 1.3 & 1.4 \\ 2.1 & 2.2 & 2.3 & 2.4 \\ 3.1 & 2.6 & 1.5 & 9.1 \\ 4.1 & 2.6 & 2.2 & 8.8 \\ 5.1 & 3.6 & 1.5 & 9.1 \\ 6.1 & 9.1 & 7.4 & 9.0 \\ 7.1 & 2.5 & 1.5 & 1.5 \\ 8.1 & 6.1 & 1.5 & 6.2 \\ 9.1 & 5.5 & 1.1 & 9.1 \\ 1.1 & 5.3 & 4.8 & 9.1 \end{bmatrix} = \begin{bmatrix} 7.1 & 2.5 & 1.5 & 1.5 \end{bmatrix}$$

Hidden layer output

In Practice, Skip Steps 3 & 4 And ...

use a lookup function (`torch.nn.Embedding`)



IMDB Dataset

- Dataset for binary sentiment classification
- 25,000 highly polar movie reviews for training, and 25,000 for testing

https://ai.stanford.edu/~amaas/data/sentiment/

	review	sentiment
0	In 1974, the teenager Martha Moxley (Maggie Gr...	1
1	OK... so... I really like Kris Kristofferson a...	0
2	***SPOILER*** Do not read this, if you think a...	0
3	hi for all the people who have seen this wonde...	1
4	I recently bought the DVD, forgetting just how...	0