STAT 992: Foundation Models for Biomedical Data

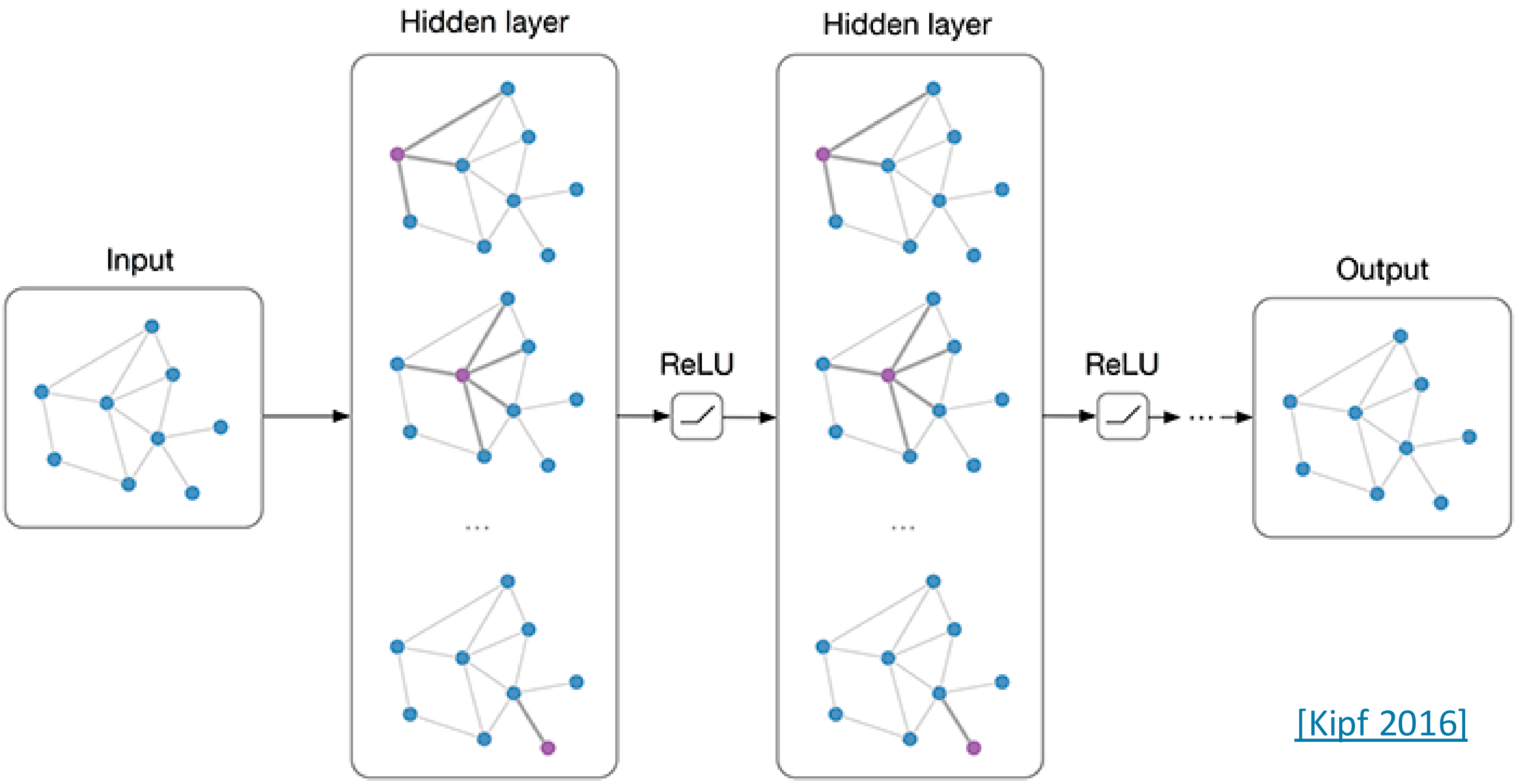
Ben Lengerich

Lecture 05: Graphs, RNNs

Feb 09, 2026

Convolutions on non-image data?

Graph Convolutional Networks



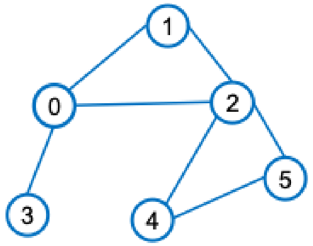
[Kipf 2016]

Graph Convolutional Networks

Graph

- $G = (V, E)$

$$\textbf{Adjacency matrix } A = \begin{bmatrix} 0 & 1 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 0 \end{bmatrix}$$

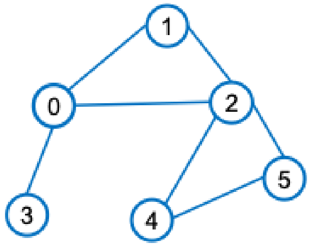


[Slides courtesy of Zhewei Wei]

Graph Convolutional Networks

Graph

- $G = (V, E)$



Degree matrix $\boldsymbol{D} = \begin{bmatrix} 3 & 0 & 0 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 4 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 0 & 0 & 2 \end{bmatrix}$

Slides courtesy of Zhewei Wei

Graph Convolutional Networks

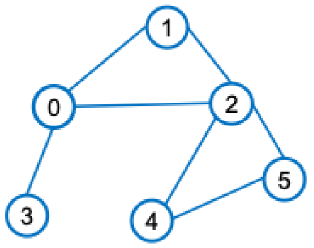
Graph

- $G = (V, E)$

$$P = D^{-1/2} A D^{-1/2}$$

Normalized adjacency matrix

$$P = \begin{bmatrix} 0 & \frac{1}{\sqrt{3}\cdot\sqrt{2}} & \frac{1}{\sqrt{3}\cdot\sqrt{4}} & \frac{1}{\sqrt{3}\cdot\sqrt{1}} & 0 & 0 \\ \frac{1}{\sqrt{3}\cdot\sqrt{2}} & 0 & \frac{1}{\sqrt{2}\cdot\sqrt{4}} & 0 & 0 & 0 \\ \frac{1}{\sqrt{3}\cdot\sqrt{4}} & \frac{1}{\sqrt{2}\cdot\sqrt{4}} & 0 & 0 & \frac{1}{\sqrt{4}\cdot\sqrt{2}} & \frac{1}{\sqrt{4}\cdot\sqrt{2}} \\ \frac{1}{\sqrt{3}\cdot\sqrt{1}} & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{1}{\sqrt{4}\cdot\sqrt{2}} & 0 & 0 & \frac{1}{\sqrt{2}\cdot\sqrt{2}} \\ 0 & 0 & \frac{1}{\sqrt{4}\cdot\sqrt{2}} & 0 & \frac{1}{\sqrt{2}\cdot\sqrt{2}} & 0 \end{bmatrix}$$



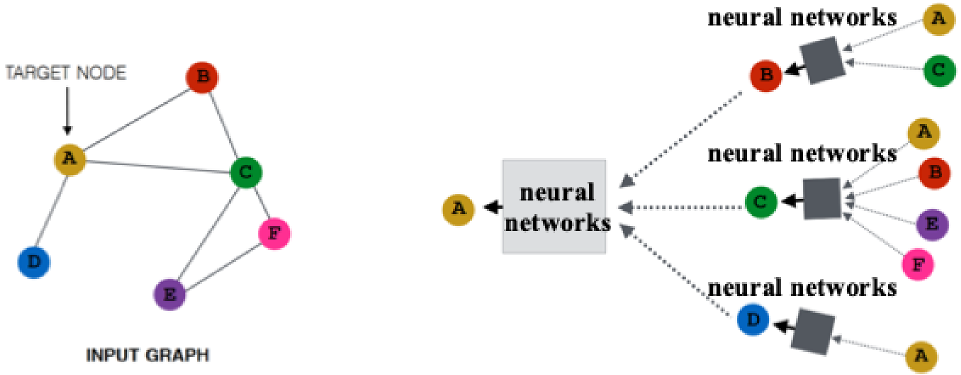
Slides courtesy of Zhewei Wei

Graph Convolutional Networks

Graph Neural Network

- Graph Convolution Neural Network(GCN) [Kipf et al.,2017]
  - Aggregating the neighbors' node features,
  - Training the weights with Message-Passing Scheme
  - Architecture:

$$H^{(\ell+1)} = \sigma\left(\widetilde{P}H^{(\ell)}W^{(\ell)}\right)$$

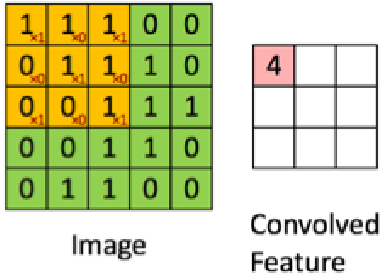


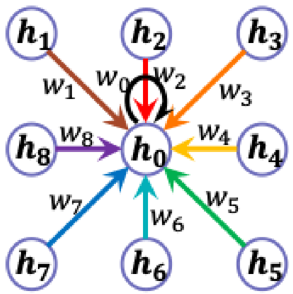
[Slides courtesy of Zhewei Wei]

Graph Convolutional Networks

GCN and CNN

- CNN is also a (Message-Passing) GNN
  - Aggregating the eight neighbors‘ and its own features





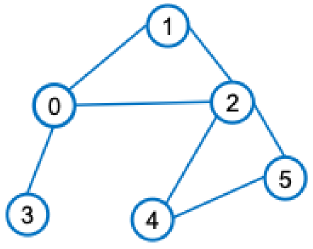
Slides courtesy of Zhewei Wei

Graph Convolutional Networks

Graph

- $G = (V, E)$

$$L = I - D^{-1/2} A D^{-1/2}$$



Normalized Laplacian matrix

$$L = \begin{bmatrix} 1 & \frac{-1}{\sqrt{3}\cdot\sqrt{2}} & \frac{-1}{\sqrt{3}\cdot\sqrt{4}} & \frac{-1}{\sqrt{3}\cdot\sqrt{1}} & 0 & 0 \\ \frac{-1}{\sqrt{3}\cdot\sqrt{2}} & 1 & \frac{-1}{\sqrt{2}\cdot\sqrt{4}} & 0 & 0 & 0 \\ \frac{-1}{\sqrt{3}\cdot\sqrt{4}} & \frac{-1}{\sqrt{2}\cdot\sqrt{4}} & 1 & 0 & \frac{-1}{\sqrt{4}\cdot\sqrt{2}} & \frac{-1}{\sqrt{4}\cdot\sqrt{2}} \\ \frac{-1}{\sqrt{3}\cdot\sqrt{1}} & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & \frac{-1}{\sqrt{4}\cdot\sqrt{2}} & 0 & 1 & \frac{-1}{\sqrt{2}\cdot\sqrt{2}} \\ 0 & 0 & \frac{-1}{\sqrt{4}\cdot\sqrt{2}} & 0 & \frac{-1}{\sqrt{2}\cdot\sqrt{2}} & 1 \end{bmatrix}$$

Slides courtesy of Zhewei Wei

Graph Convolutional Networks

Graph Fourier Transform

- The eigendecomposition of Laplacian matrix

$$\boldsymbol{L} = \boldsymbol{U}\boldsymbol{\Lambda}\boldsymbol{U}^T = \boldsymbol{U}\begin{pmatrix} \lambda_1 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & \lambda_n \end{pmatrix}\boldsymbol{U}^T,$$

where $\boldsymbol{U} = [\boldsymbol{u}_1, \dots, \boldsymbol{u}_n]$, $\boldsymbol{\Lambda} = \mathrm{diag}([\lambda_1, \dots, \lambda_n])$, $\boldsymbol{u}_i$ and λ_i for $i \in \{1,2,\dots,n\}$ denote the eigenvectors and eigenvalues, respectively, and $\lambda_i \in [0,2]$.

- ☐ Orthonormal basis: $\boldsymbol{U} \cdot \boldsymbol{U}^T = \boldsymbol{I}$,

- Graph Fourier Transform of a signal: $\hat{\boldsymbol{x}} = \boldsymbol{U}^T \boldsymbol{x}$
- Inverse Graph Fourier Transform of a signal: $\boldsymbol{x} = \boldsymbol{U}\hat{\boldsymbol{x}}$

Slides courtesy of Zhewei Wei

Recurrent Neural Networks

Sequence data: order matters

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

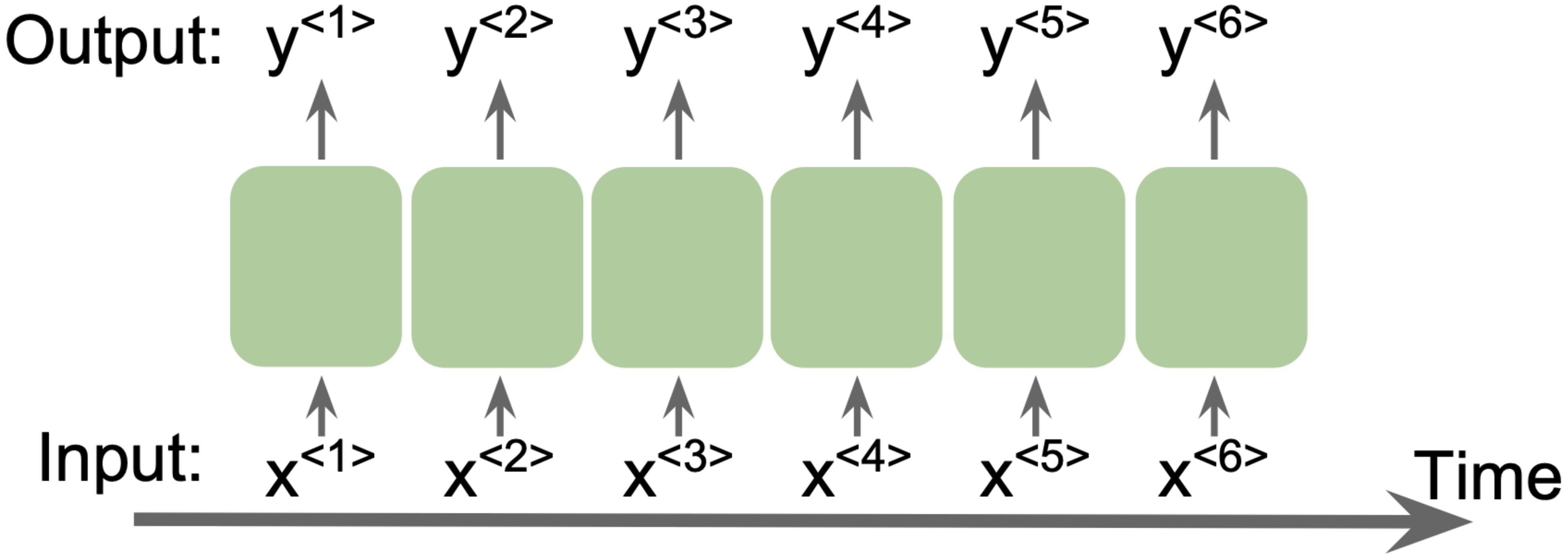
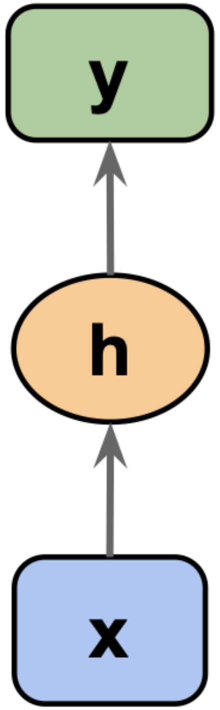


Image source: Sebastian Raschka, Vahid Mirjalili. Python Machine Learning. 3rd Edition. Packt, 2019

Recurrent Neural Networks (RNNs)

Networks we used previously: also called feedforward neural networks



Recurrent Neural Network (RNN)

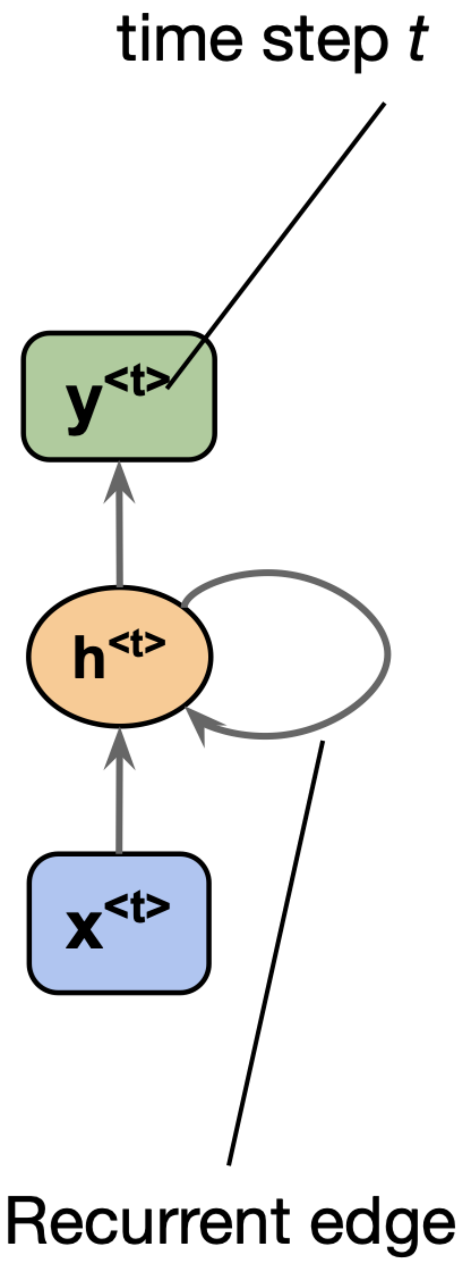


Image source: Sebastian Raschka, Vahid Mirjalili. Python Machine Learning. 3rd Edition. Packt, 2019

Recurrent Neural Networks (RNNs)

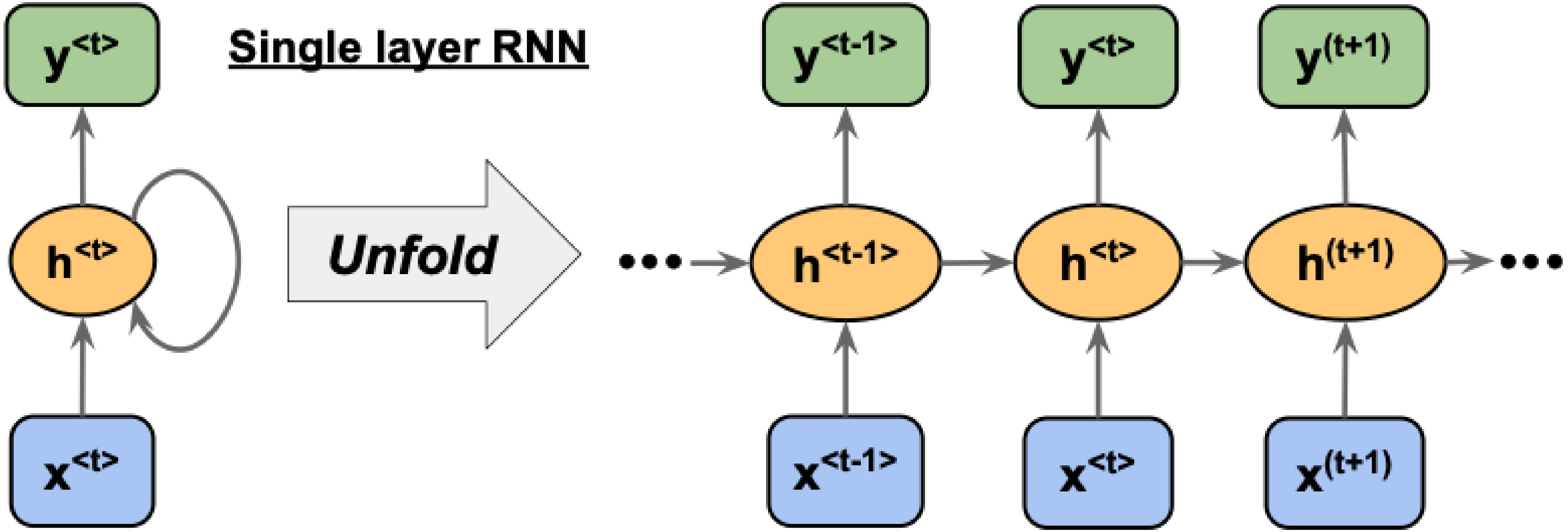
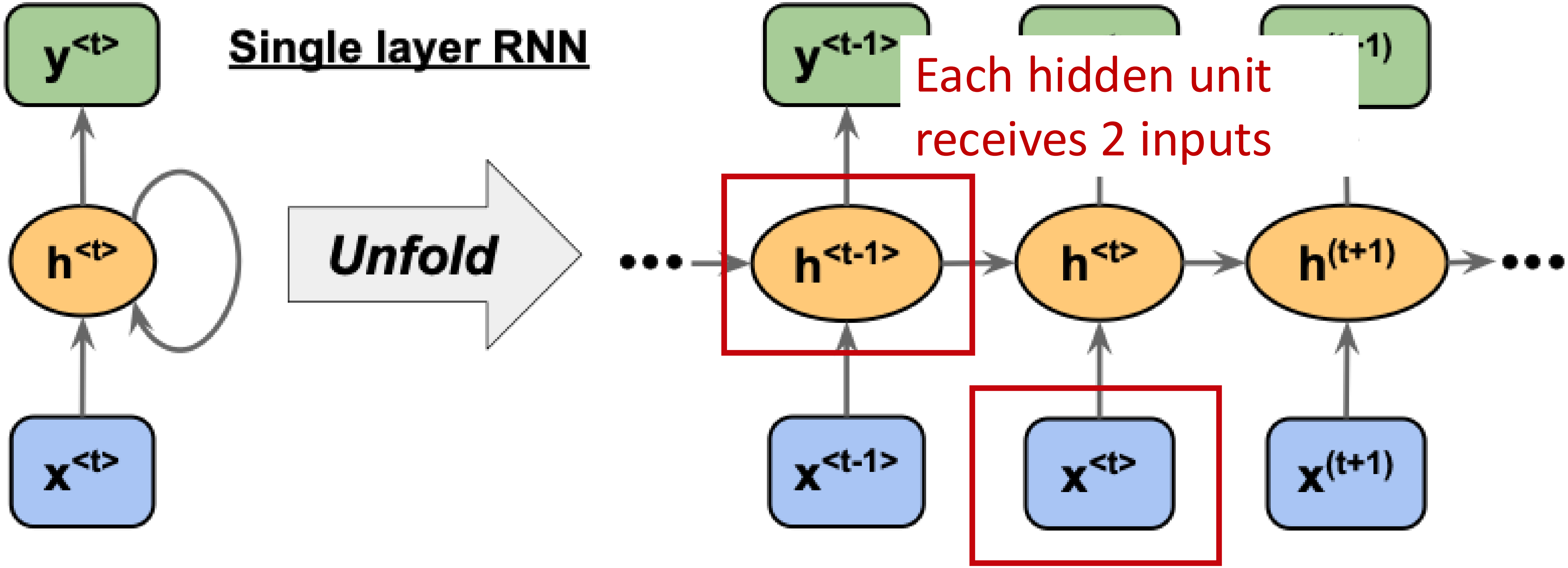


Image source: Sebastian Raschka, Vahid Mirjalili. Python Machine Learning. 3rd Edition. Packt, 2019

Recurrent Neural Networks (RNNs)



Each hidden unit receives 2 inputs

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

Multilayer RNNs

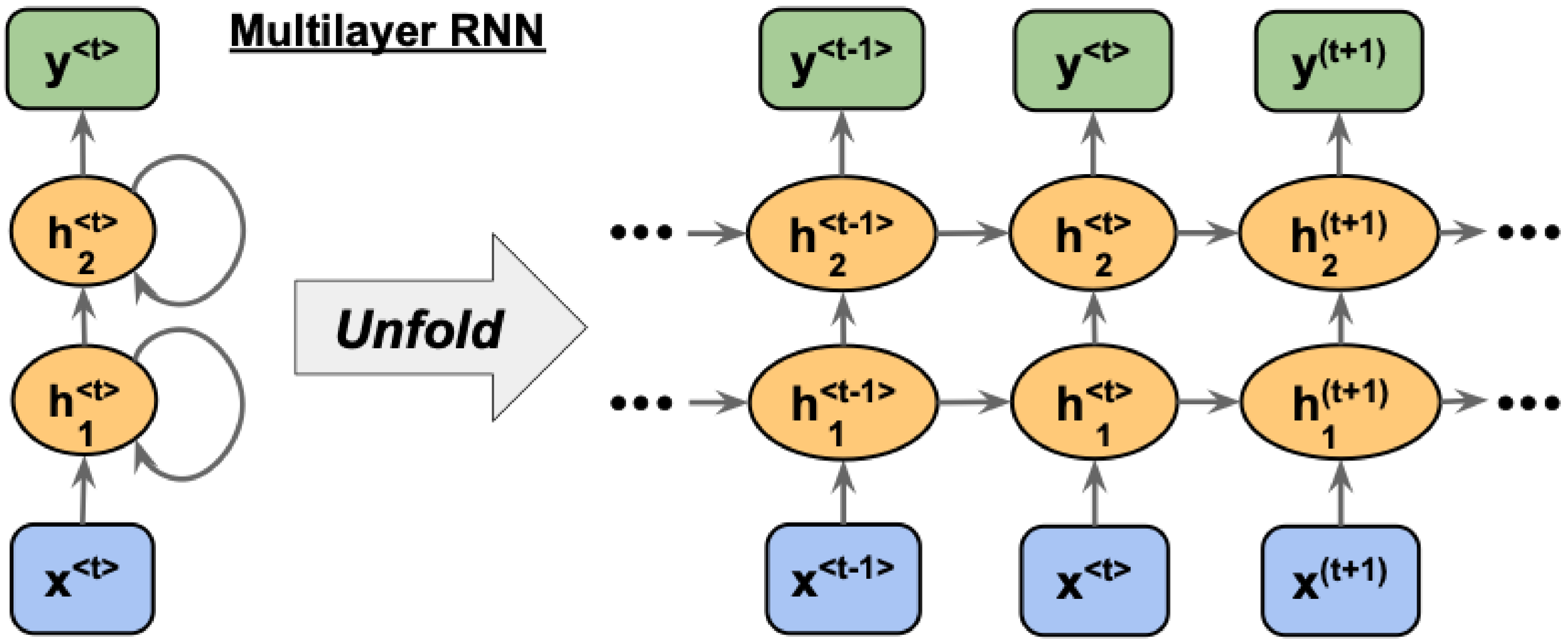


Image source: Sebastian Raschka, Vahid Mirjalili. Python Machine Learning. 3rd Edition. Packt, 2019

Recurrence unlocks many types of sequence tasks

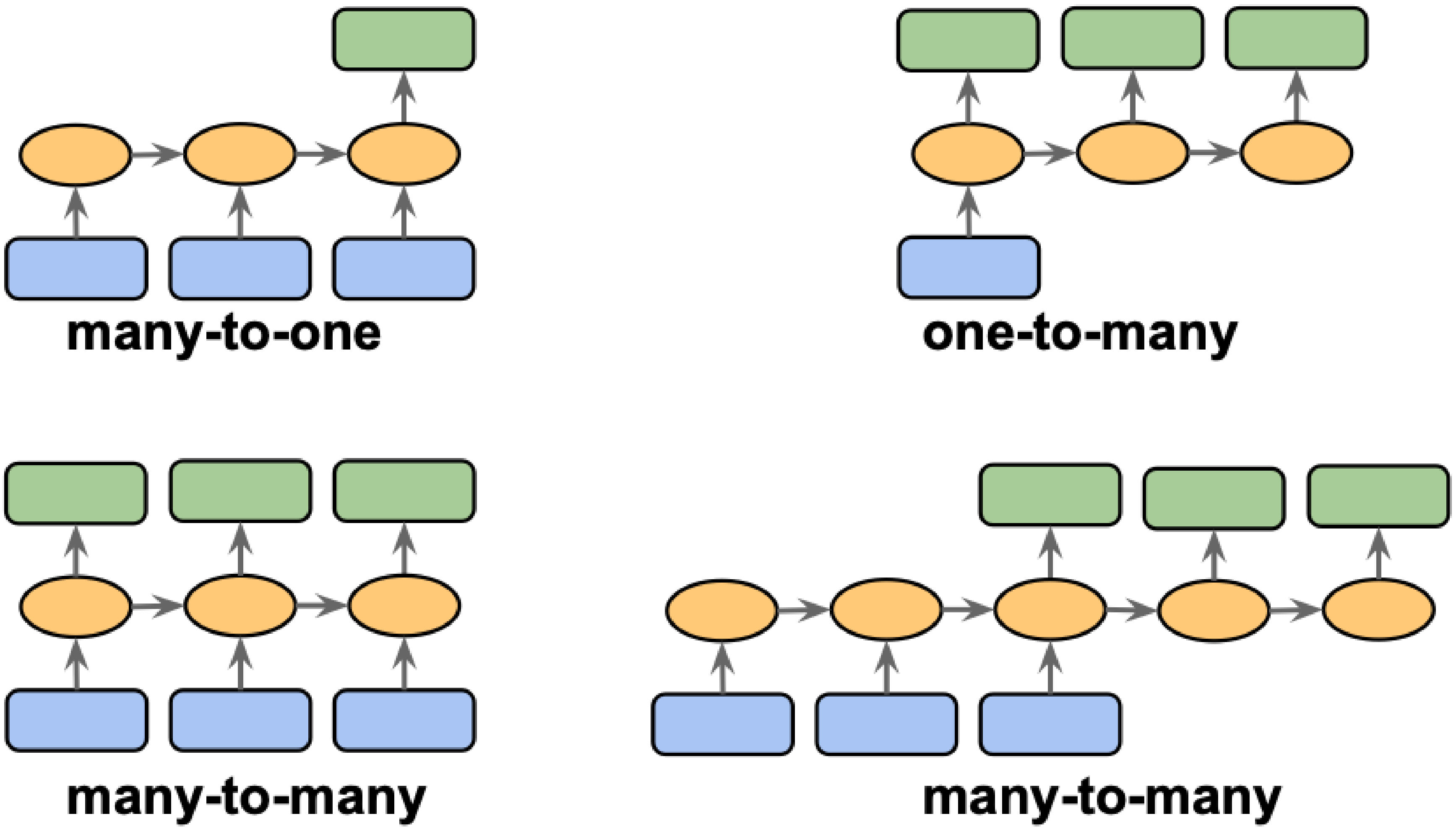
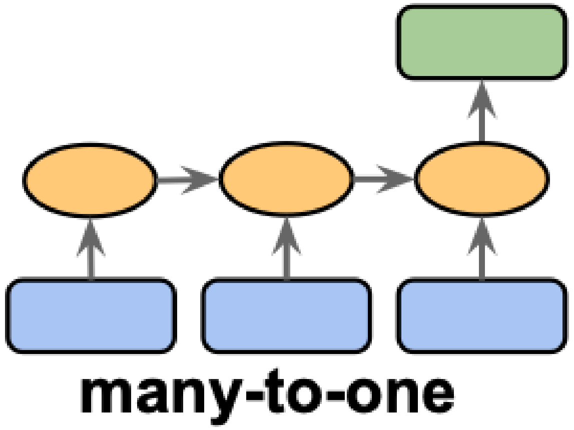


Image source: Sebastian Raschka, Vahid Mirjalili. Python Machine Learning. 3rd Edition. Packt, 2019

Recurrence unlocks many types of sequence tasks

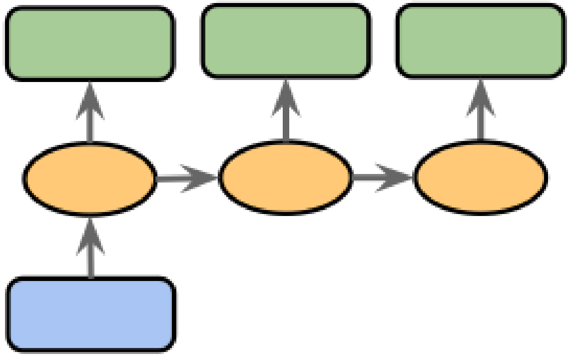


Many-to-one: The input data is a sequence, but the output is a fixed-size vector, not a sequence.

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

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

Recurrence unlocks many types of sequence tasks



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

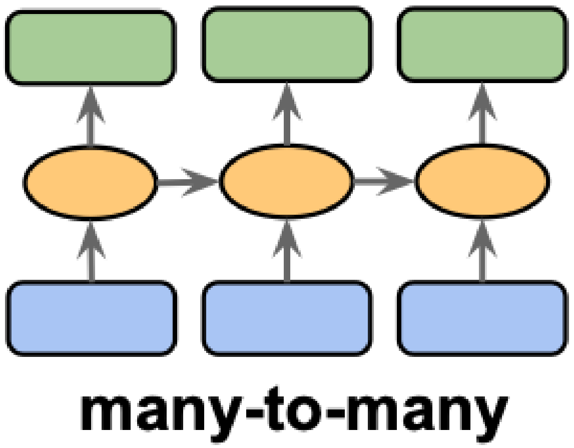
Example: Image captioning, where the input is an image, the output is a text description of that image

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

Recurrence unlocks many types of sequence tasks

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

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



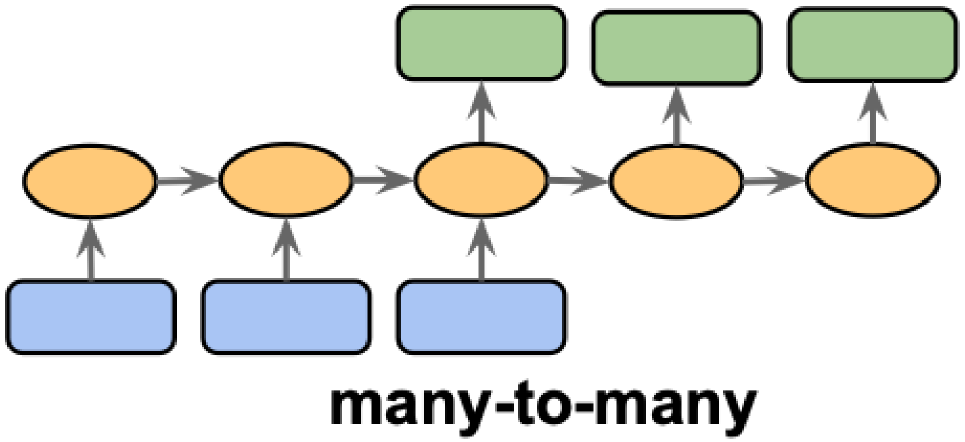


Image source: Sebastian Raschka, Vahid Mirjalili. Python Machine Learning. 3rd Edition. Packt, 2019

Under the hood: weight matrices in an RNN

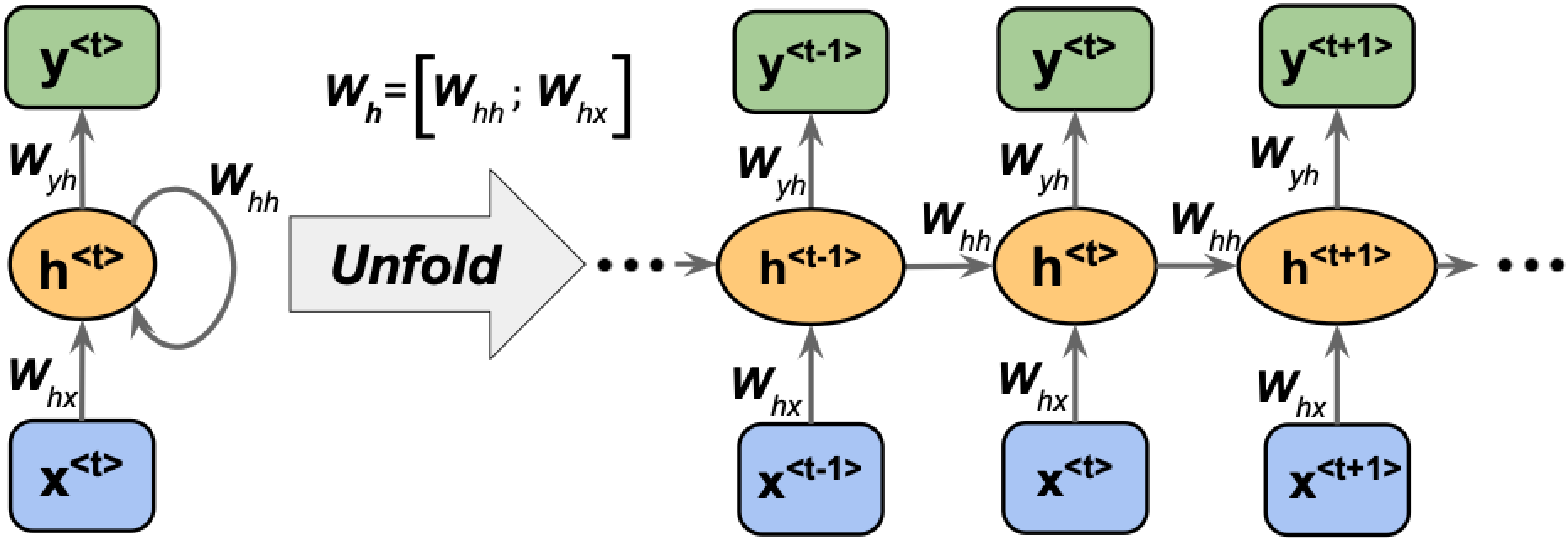
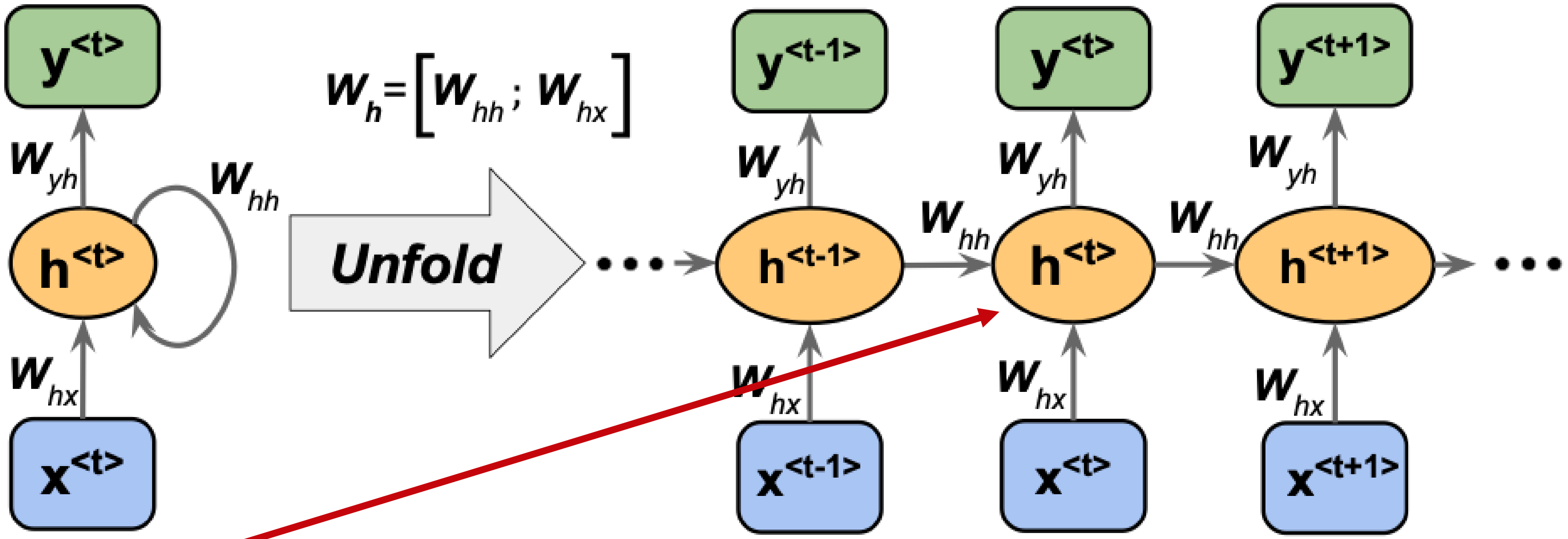


Image source: Sebastian Raschka, Vahid Mirjalili. Python Machine Learning. 3rd Edition. Packt, 2019

Under the hood: weight matrices in an RNN

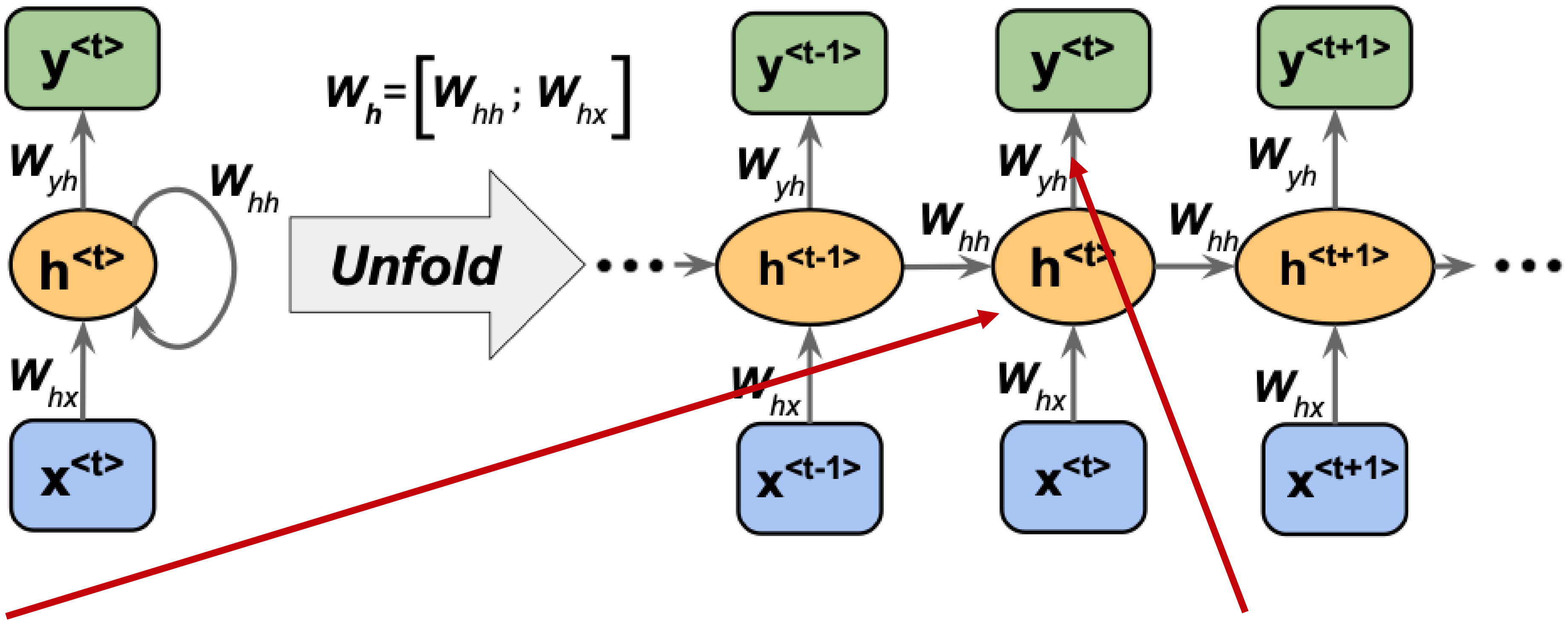


Net input: $$\mathbf{z}_h^{\langle t \rangle} = \mathbf{W}_{hx}\mathbf{x}^{\langle t \rangle} + \mathbf{W}_{hh}\mathbf{h}^{\langle t-1 \rangle} + \mathbf{b}_h$$

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

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

Under the hood: weight matrices in an RNN



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

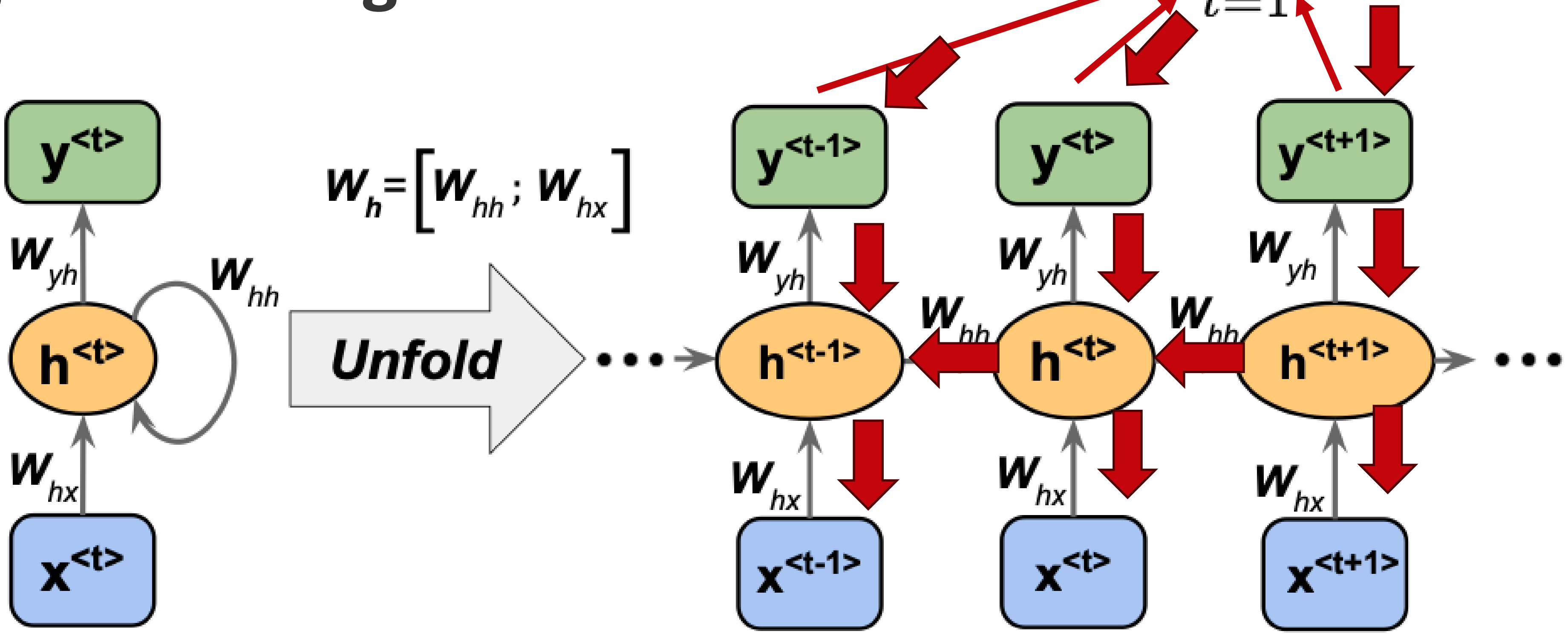
Net input: $\mathbf{z}_y^{\langle t \rangle} = \mathbf{W}_{yh}\mathbf{h}^{\langle t \rangle} + \mathbf{b}_y$

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

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

Backpropagation through time

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

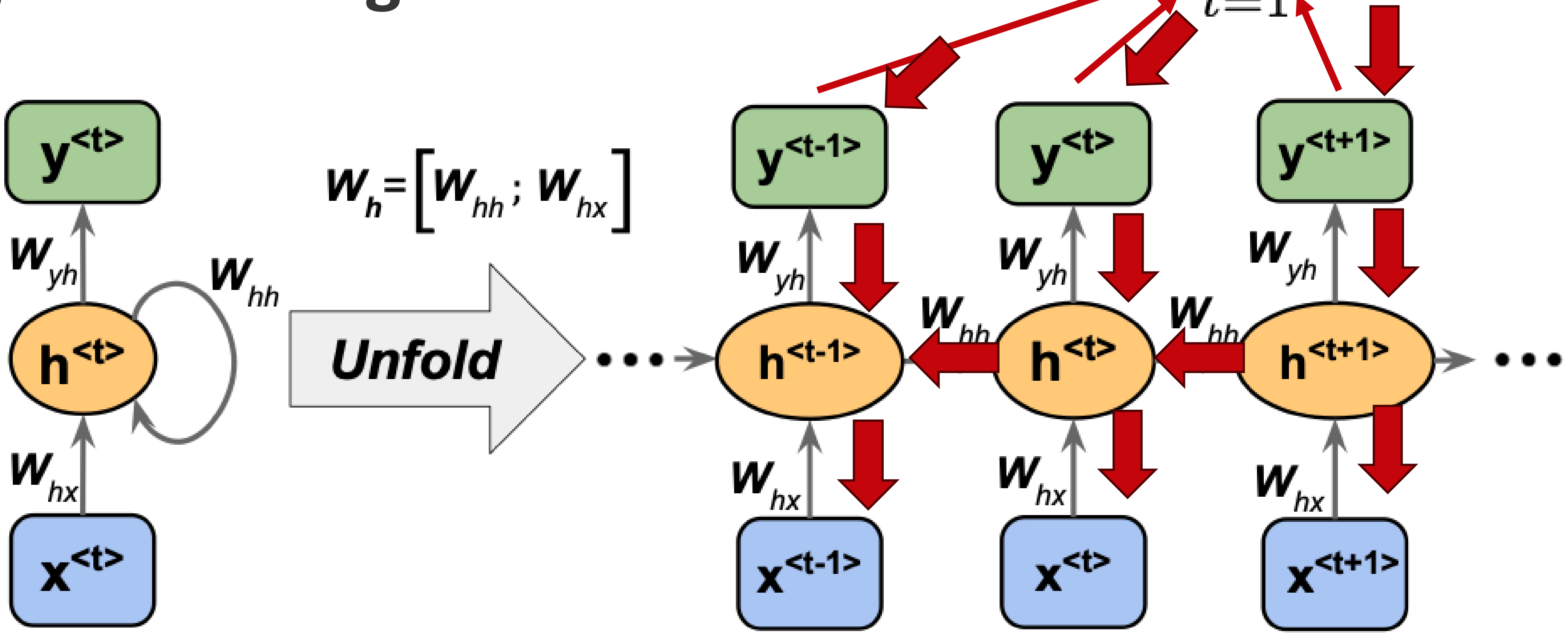


The overall loss can be computed as the sum over all time steps

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

Backpropagation through time

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

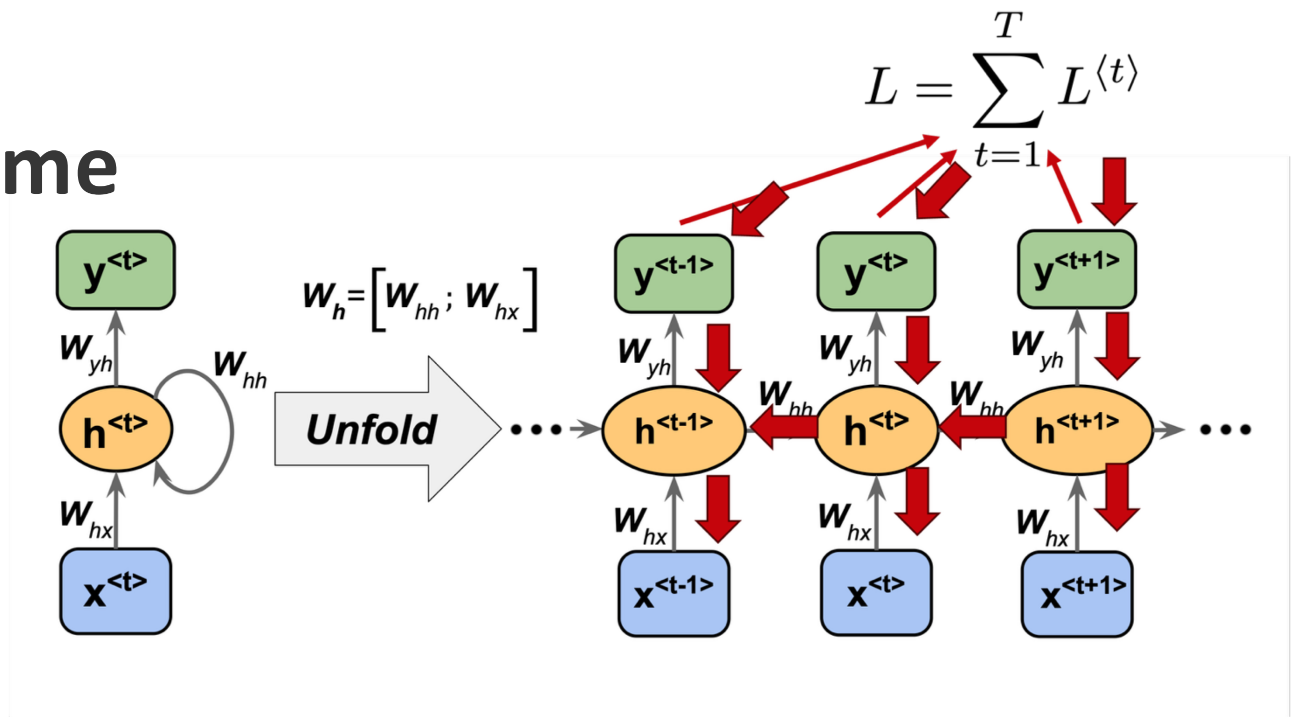


$$L = \sum_{t=1}^{T} L^{(t)}$$

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

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

Backpropagation through time



Computed as a multiplication of adjacent time steps:

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

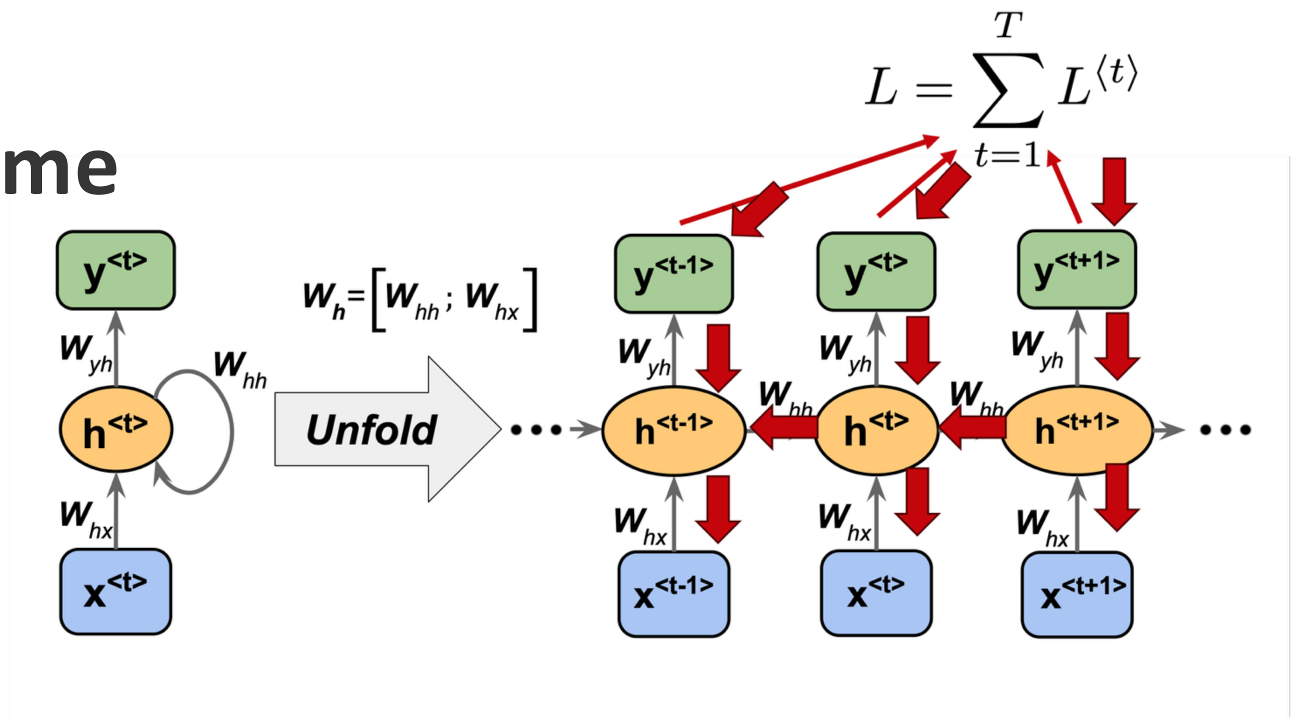
$$\frac{\partial L^{(t)}}{\partial \mathbf{W}_{hh}} = \frac{\partial L^{(t)}}{\partial y^{(t)}} \cdot \frac{\partial y^{(t)}}{\partial \mathbf{h}^{(t)}} \cdot \left(\sum_{k=1}^{t} \frac{\partial \mathbf{h}^{(t)}}{\partial \mathbf{h}^{(k)}} \cdot \frac{\partial \mathbf{h}^{(k)}}{\partial \mathbf{W}_{hh}}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

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

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

Backpropagation through time

Straightforward, but problematic: vanishing / exploding gradients!



Computed as a multiplication of adjacent time steps:

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

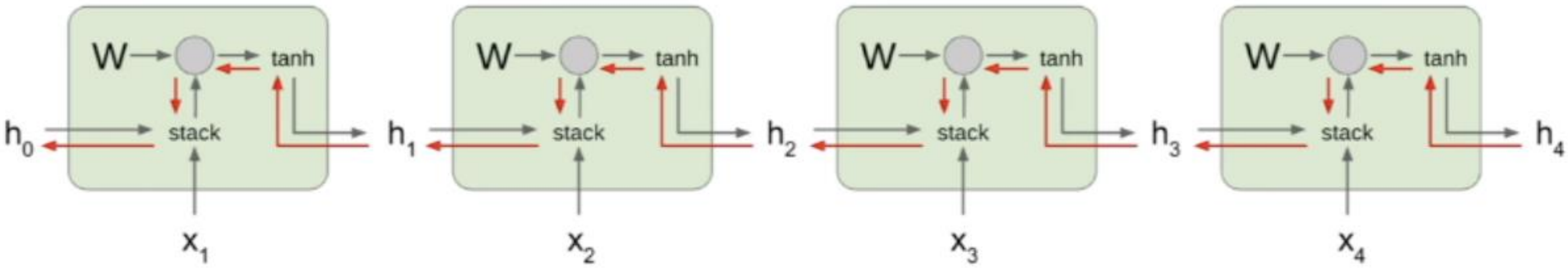
$$\frac{\partial L^{(t)}}{\partial \mathbf{W}_{hh}} = \frac{\partial L^{(t)}}{\partial y^{(t)}} \cdot \frac{\partial y^{(t)}}{\partial \mathbf{h}^{(t)}} \cdot \left(\sum_{k=1}^{t} \frac{\partial \mathbf{h}^{(t)}}{\partial \mathbf{h}^{(k)}} \cdot \frac{\partial \mathbf{h}^{(k)}}{\partial \mathbf{W}_{hh}}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

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

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

A challenge: Vanishing / exploding gradients

$$\boldsymbol{h}_t = tanh(W^{hh}\boldsymbol{h}_{t-1} + W^{hx}\boldsymbol{x}_t)$$



Computing gradient of h_0 involves many factors of W (and repeated tanh)

Largest singular value > 1:
Exploding gradients

→ **Gradient clipping**: Scale gradient if its norm is too big

```
grad_norm = np.sum(grad * grad)
if grad_norm > threshold:
  grad *= (threshold / grad_norm)
```

Largest singular value < 1:
Vanishing gradients

Bengio et al., 1994 "Learning long-term dependencies with gradient descent is difficult"
Pascanu et al., 2013 "On the difficulty of training recurrent neural neworks"

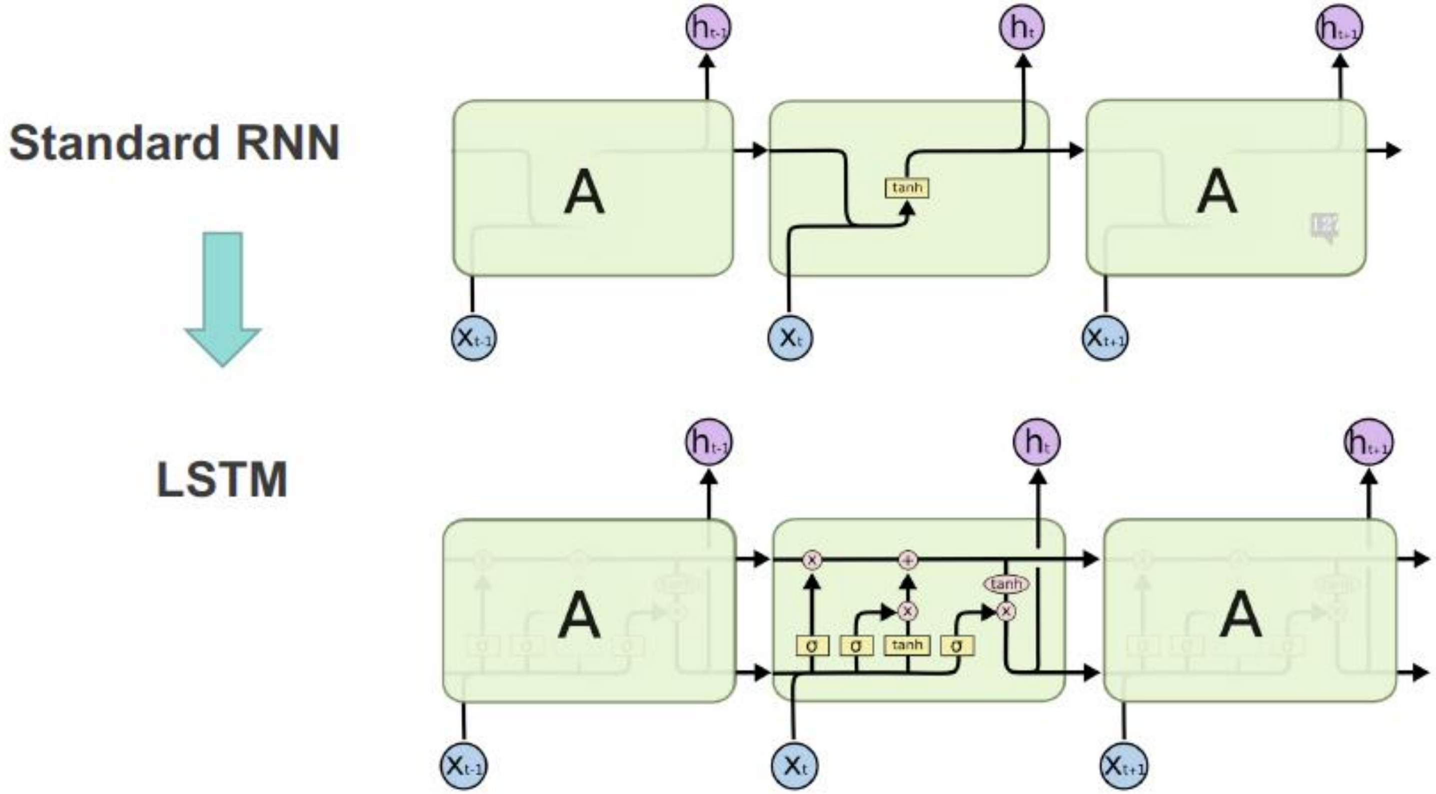
Solutions to Vanishing / Exploding Gradients

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

- **Truncated backpropagation through time (TBPTT):** limit 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.

Solutions to Vanishing / Exploding Gradients

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



Long-short term memory (LSTM)

- Not an oxymoron: **2 paths** of memory

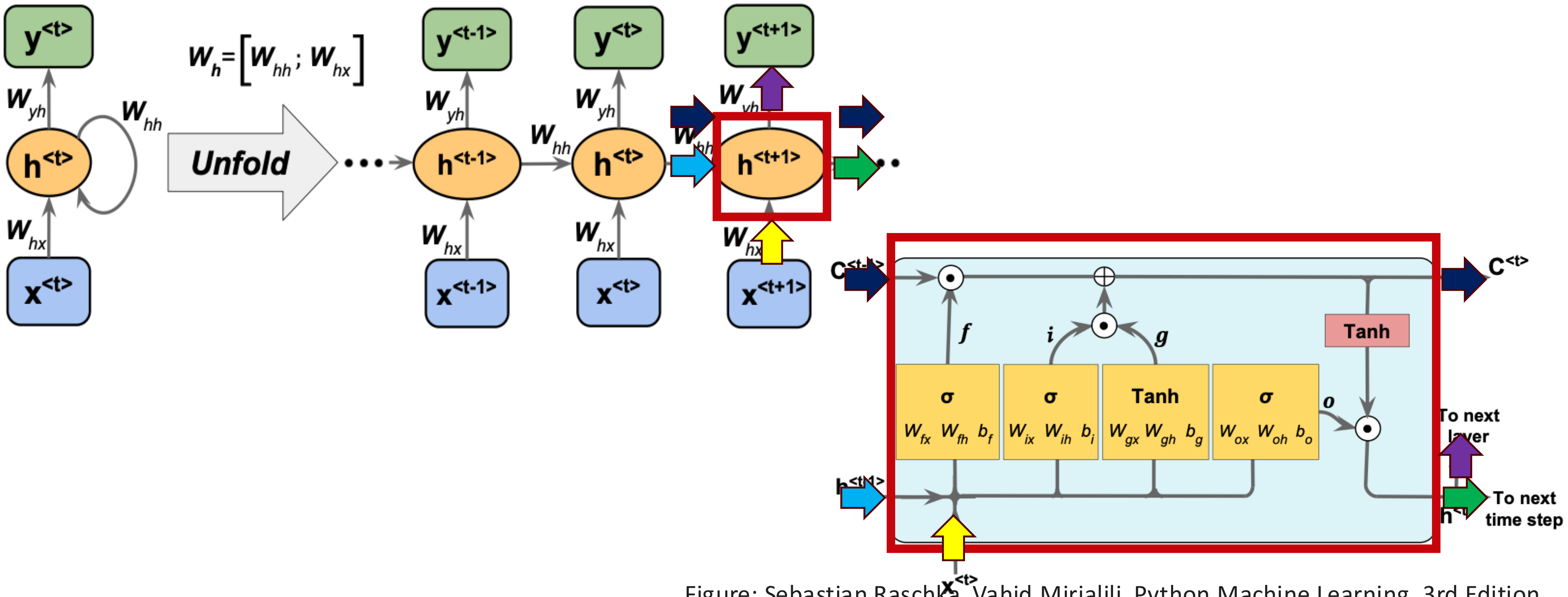


Figure: Sebastian Raschka, Vahid Mirjalili. Python Machine Learning. 3rd Edition. Birmingham, UK: Packt Publishing, 2019

Long-short term memory (LSTM)

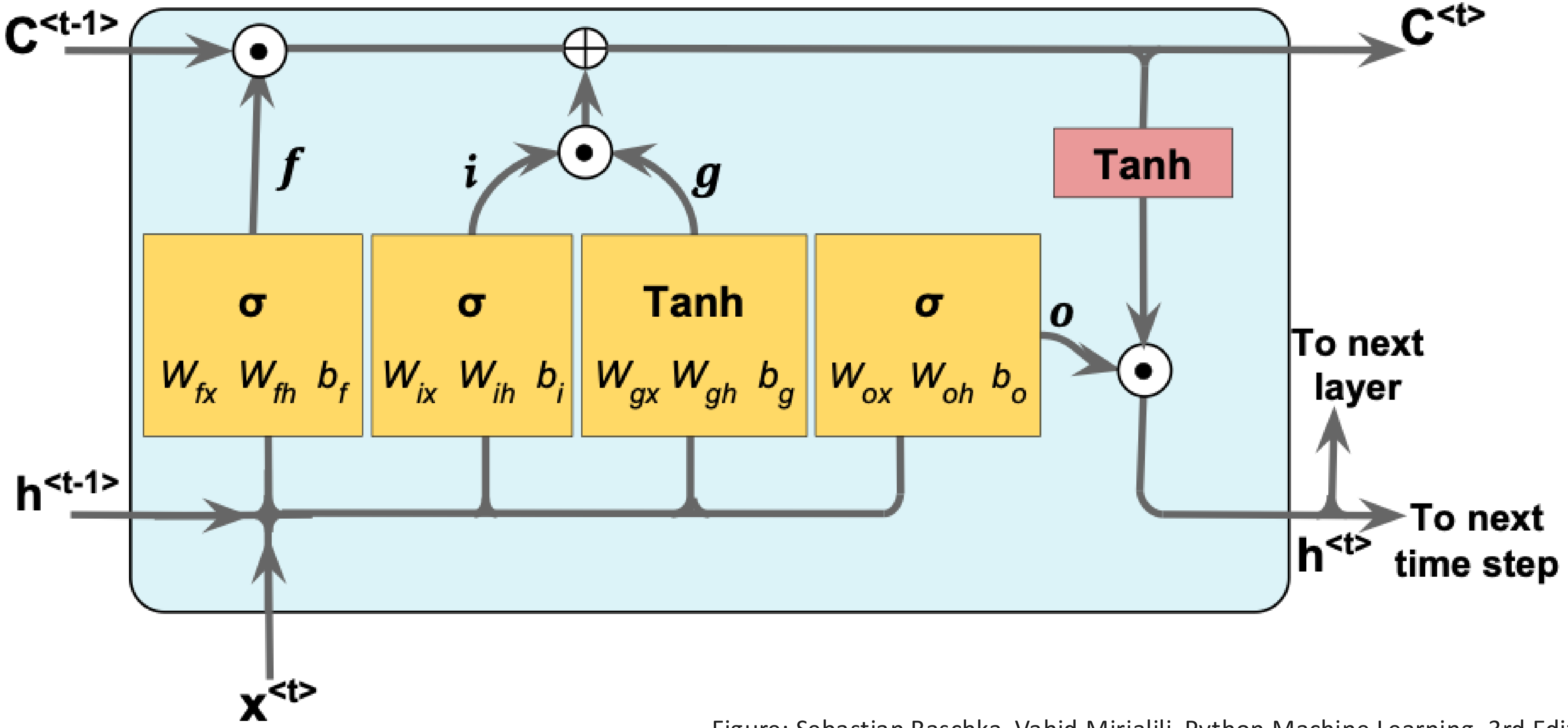


Figure: Sebastian Raschka, Vahid Mirjalili. Python Machine Learning. 3rd Edition. Birmingham, UK: Packt Publishing, 2019

Inside LSTM

Cell state at time t-1

Cell state at time t

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

Inside LSTM

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

Inside LSTM

"Forget gate": controls which information is remembered and which is forgotten

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

Element-wise multiplication

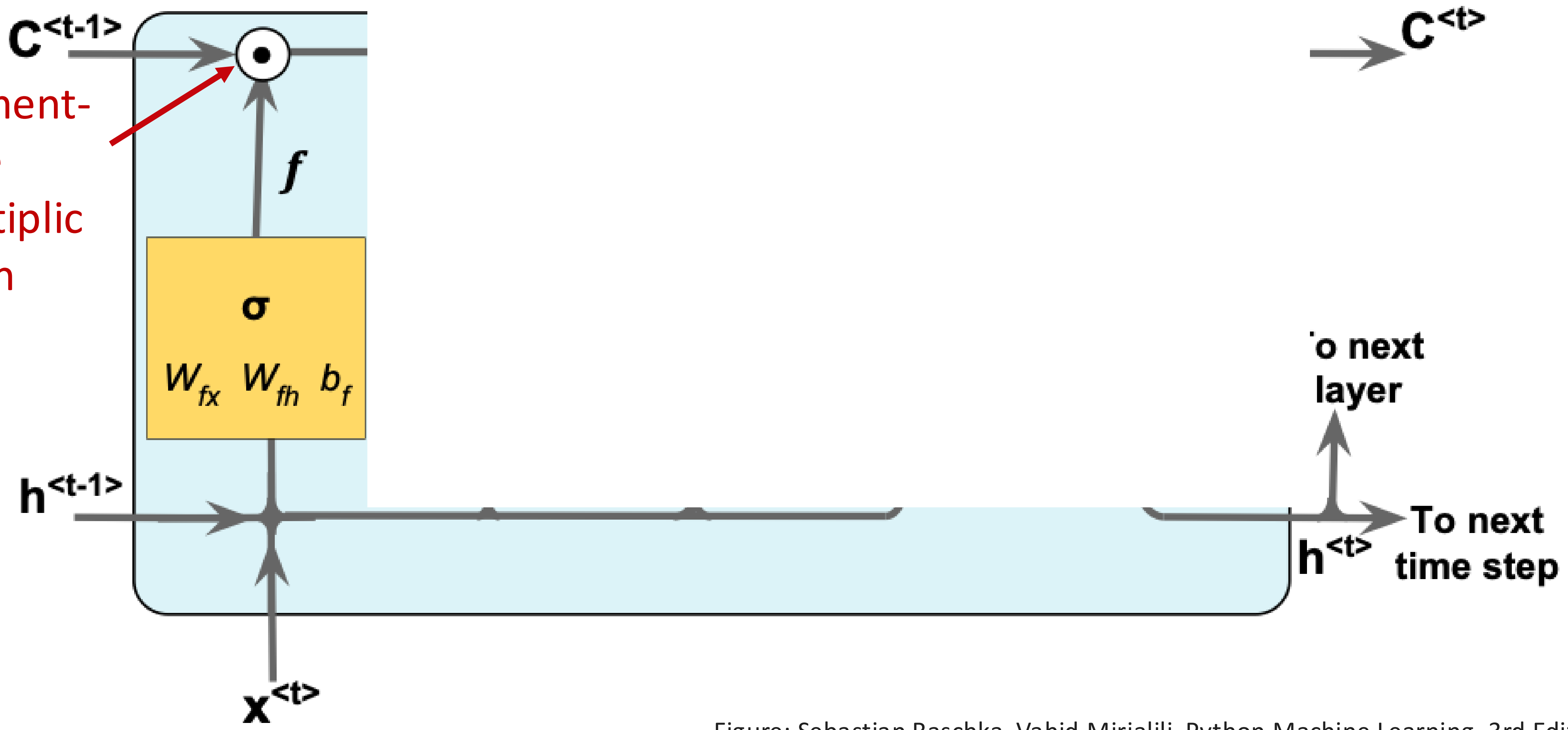


Figure: Sebastian Raschka, Vahid Mirjalili. Python Machine Learning. 3rd Edition. Birmingham, UK: Packt Publishing, 2019

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

Element-wise addition

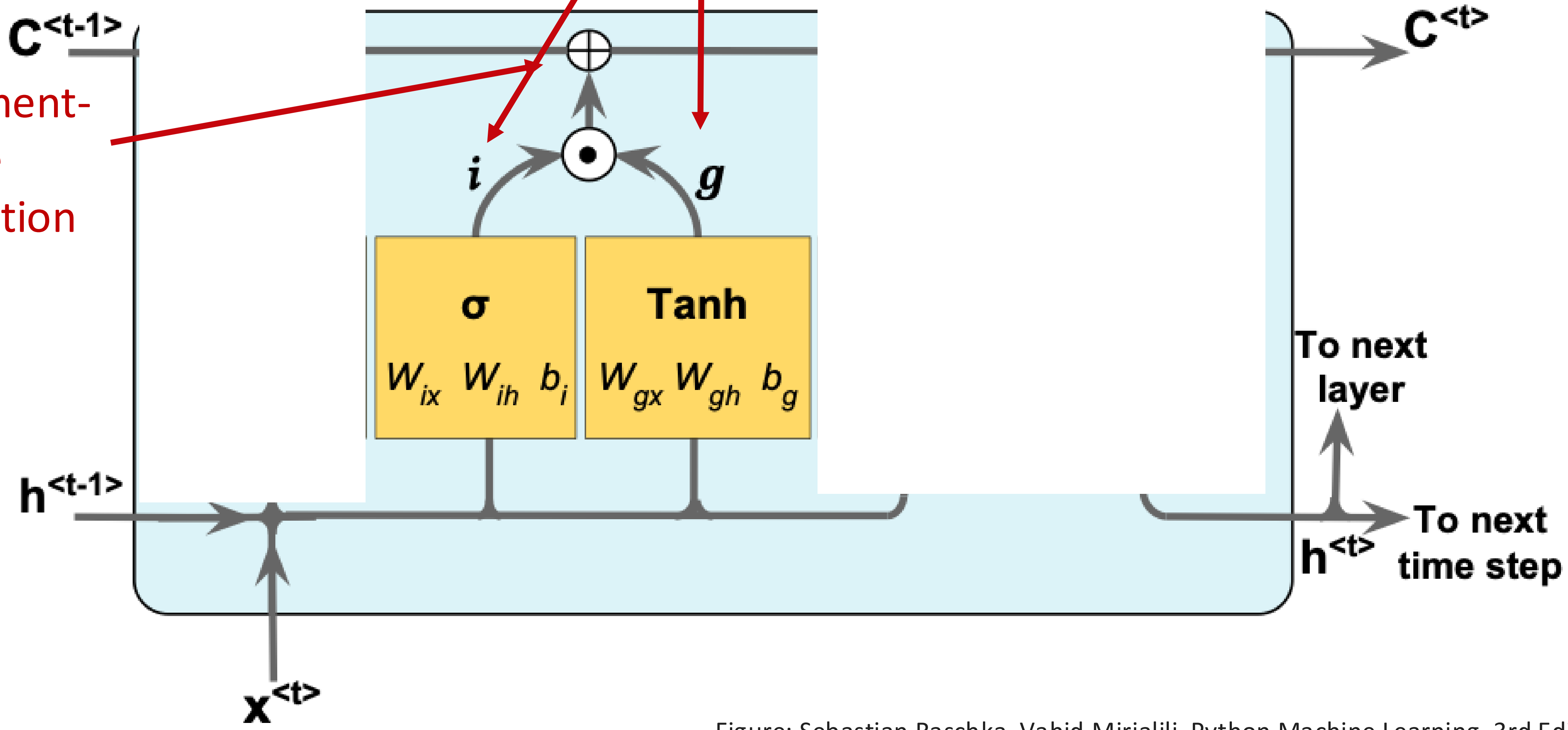


Figure: Sebastian Raschka, Vahid Mirjalili. Python Machine Learning. 3rd Edition. Birmingham, UK: Packt Publishing, 2019

Inside LSTM

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

Forget Gate, Input Node, Input Gate

For updating the cell state

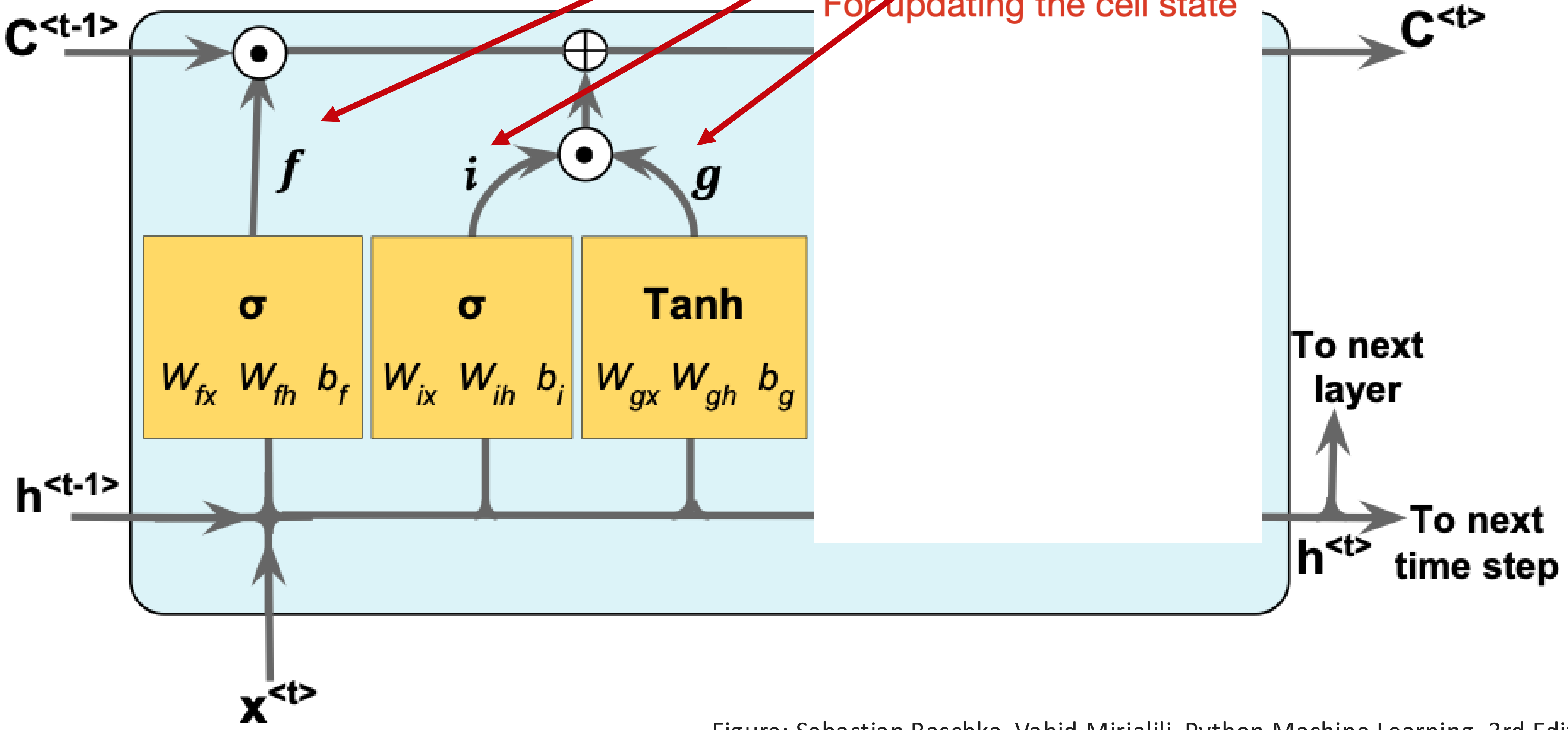


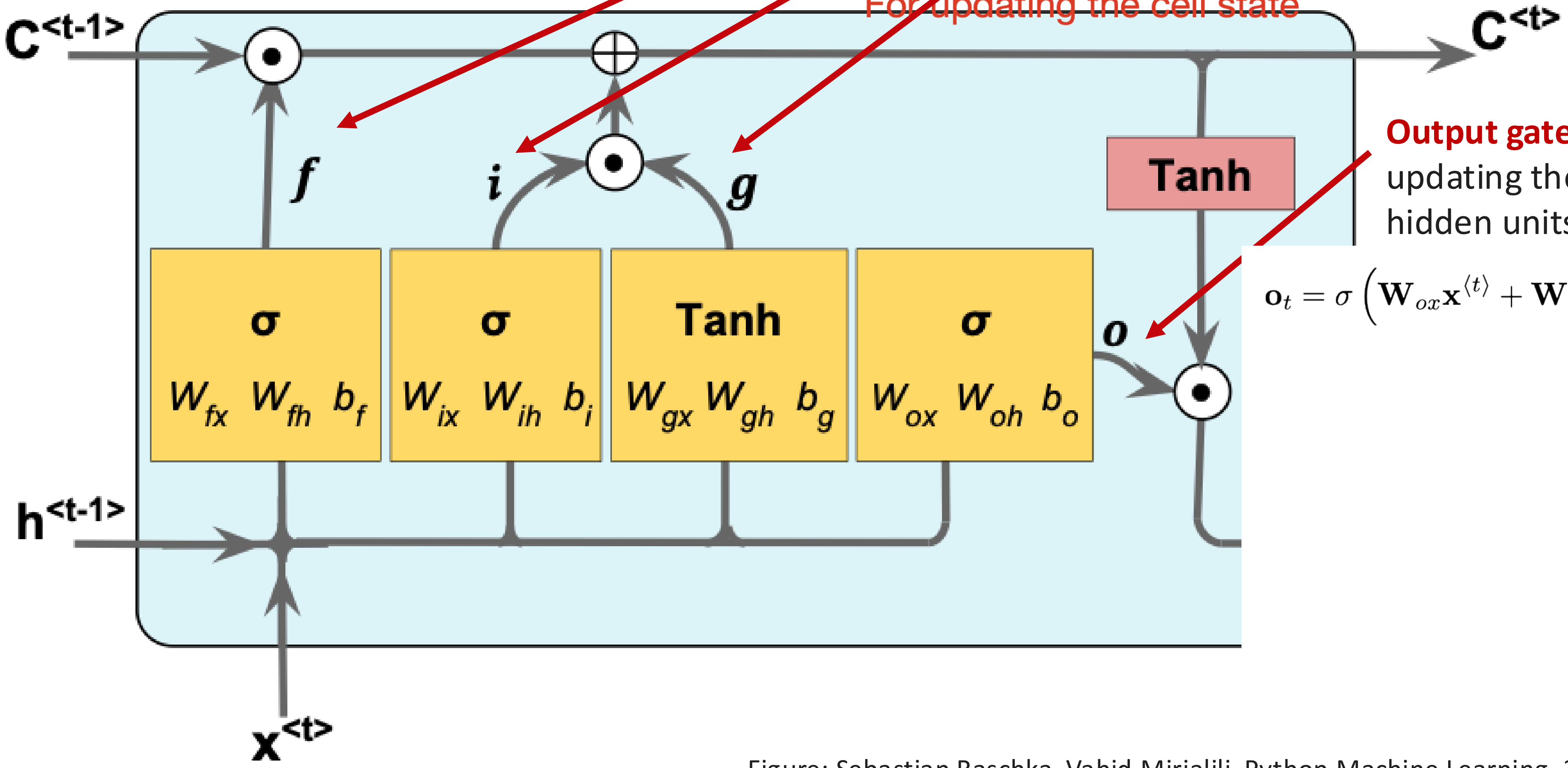
Figure: Sebastian Raschka, Vahid Mirjalili. Python Machine Learning. 3rd Edition. Birmingham, UK: Packt Publishing, 2019

Inside LSTM

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



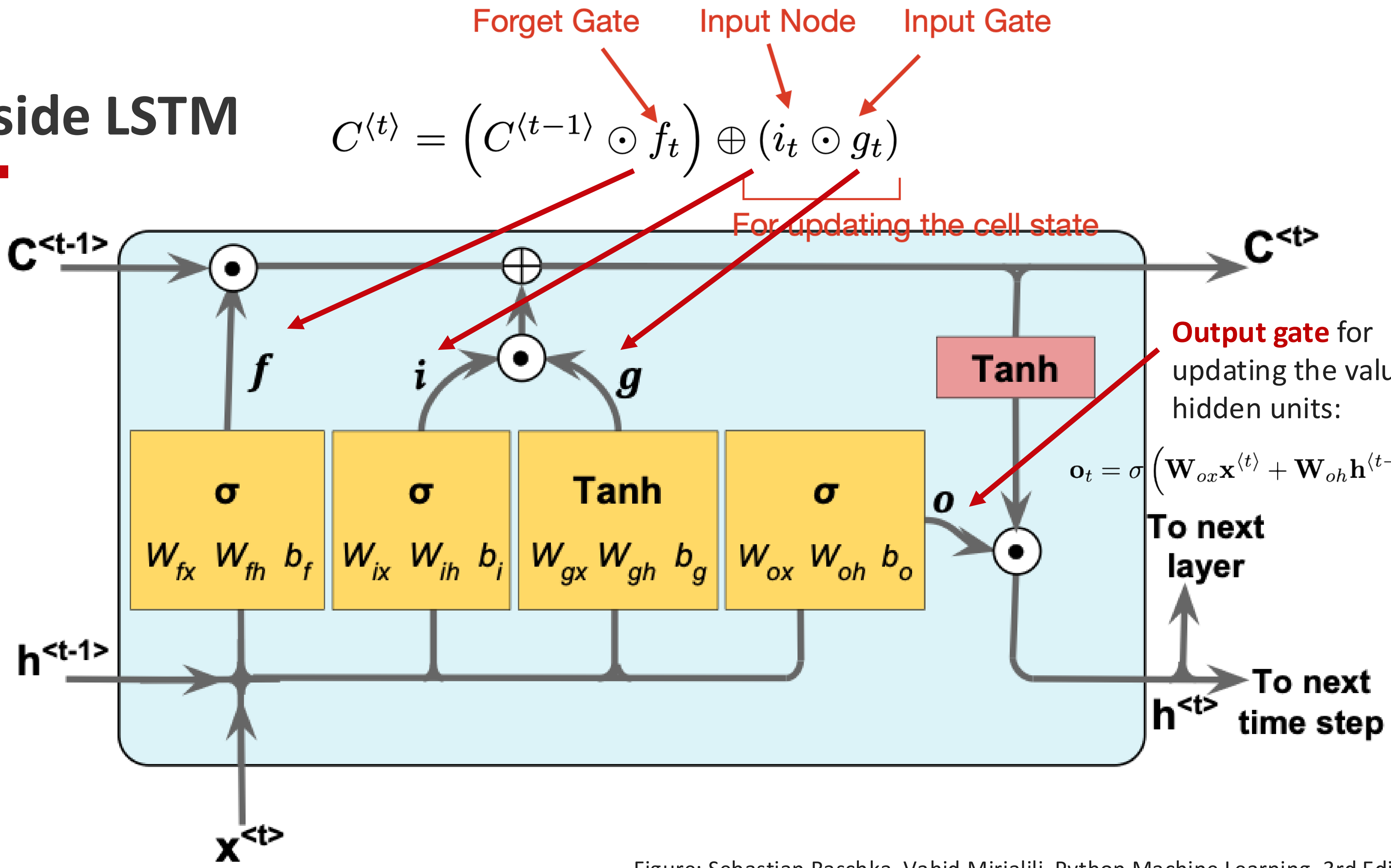
Output gate for updating the values of hidden units:

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

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

Inside LSTM

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



Output gate for updating the values of hidden units:

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

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

LSTM Back Together

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

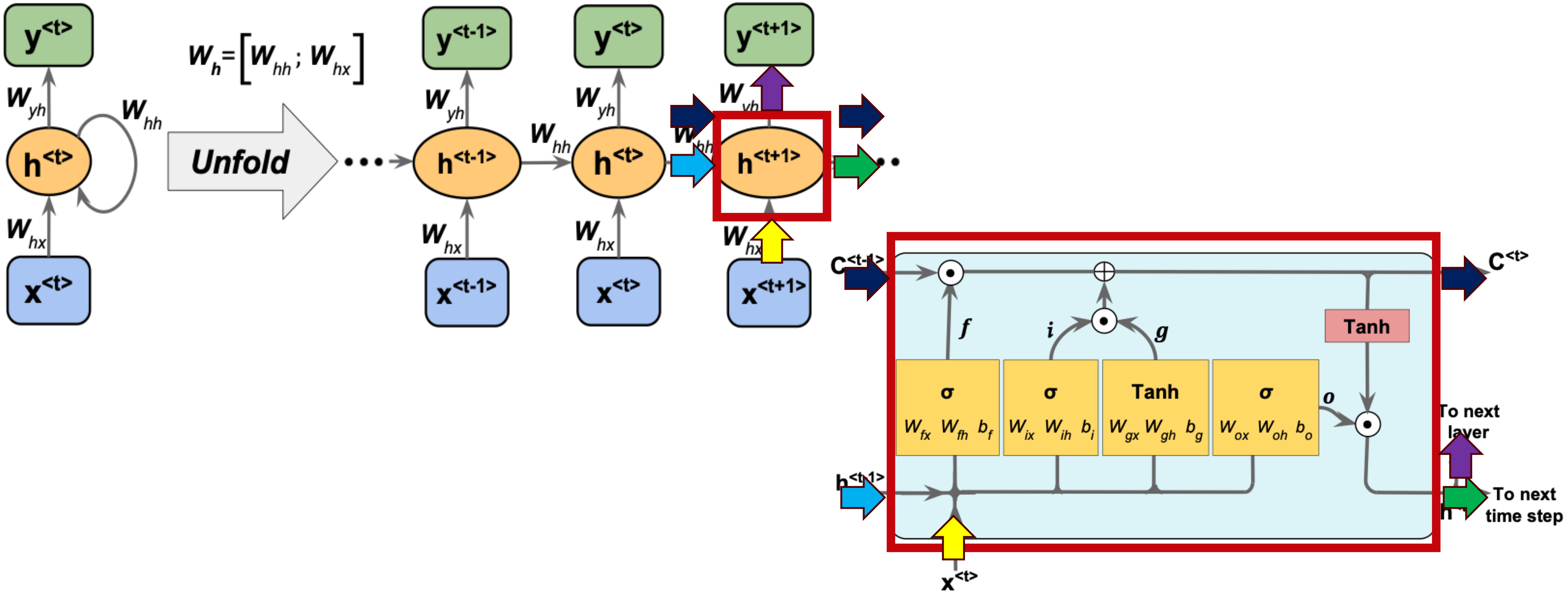


Figure: Sebastian Raschka, Vahid Mirjalili. Python Machine Learning. 3rd Edition. Birmingham, UK: Packt Publishing, 2019

Good reading

- The Unreasonable Effectiveness of Recurrent Neural Networks by Andrej Karpathy
- On the difficulty of training recurrent neural networks by Razvan Pascanu, Tomas Mikolov, Yoshua Bengio

Questions?

WISCONSIN
UNIVERSITY OF WISCONSIN–MADISON