STAT 453: Introduction to Deep Learning and Generative Models

Ben Lengerich

Lecture 19: Recurrent Neural Networks

November 10, 2025

Reading: See course homepage

Our semester

Week	Lecture Dates	Topic
		Module 1: Introduction and Foundations
1	9/3	Course Introduction
2	9/8, 9/10	A Brief History of DL, Statistics / linear algebra / calculus review
3	9/15, 9/17	Single-layer networks Parameter Optimization and Gradient Descent
4	9/22, 9/24	Automatic differentiation with PyTorch, Cluster and cloud computing resources
		Module 2: Neural Networks
5	9/29, 10/1	Multinomial logistic regression, Multi-layer perceptrons and backpropagation
6	10/6, 10/8	Regularization Normalization / Initialization
7	10/13, 10/15	Optimization, Learning Rates CNNs
8	10/20, 10/22	Review, **Midterm Exam**

		Module 3: Intro to Generative Models	
9	10/27, 10/29	A Linear Intro to Generative Models, Factor Analysis, Autoencoders, VAEs	
10	11/3, 11/5	Generative Adversarial Networks, Diffusion Models	Project Midway Report
		Module 4: Large Language Models	
11	11/10, 11/12	Sequence Learning with RNNs Attention, Transformers	HW4
12	11/17, 11/19	GPT Architectures, Unsupervised Training of LLMs	
13	11/24, 11/26	Supervised Fine-tuning of LLMs, Prompts and In-context learning	HW5
14	12/1, 12/3	Foundation models, alignment, explainability Open directions in LLM research	
15	12/8, 12/10	**Project Presentations**	Project Final Report
16	12/17	**Final Exam**	Final Exam

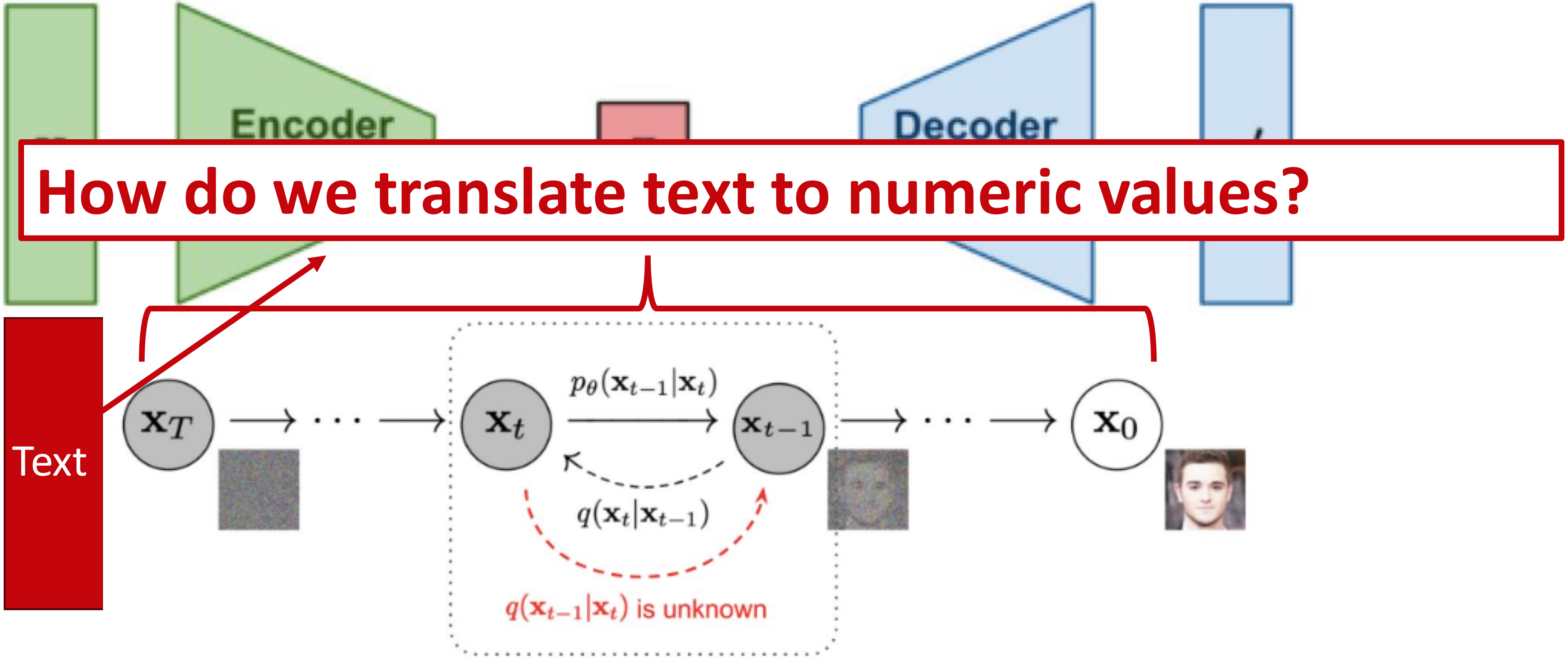
A quick vote…

Week	Lecture Dates	Topic	
		Module 1: Introduction and Foundations	
1	9/3	Course Introduction	
2	9/8, 9/10	A Brief History of DL, Statistics / linear algebra / calculus review	
3	9/15, 9/17	Single-layer networks Parameter Optimization and Gradient Descent	
4	9/22, 9/24	Automatic differentiation with PyTorch, Cluster and cloud computing resources	
		Module 2: Neural Networks	
5	9/29, 10/1	Multinomial logistic regression, Multi-layer perceptrons and backpropagation	
6	10/6, 10/8	Regularization Normalization / Initialization	
7	10/13, 10/15	Optimization, Learning Rates CNNs	
8	10/20, 10/22	Review, **Midterm Exam**	
		Module 3: Intro to Generative Models	
9	10/27, 10/29	A Linear Intro to Generative Models, Factor Analysis, Autoencoders, VAEs	
10	11/3, 11/5	Generative Adversarial Networks, Diffusion Models	Project Midway Report
		Module 4: Large Language Models	
11	11/10, 11/12	Sequence Learning with RNNs Attention, Transformers	HW4
[illegible]	11/17, 11/19	GPT Architectures, Unsupervised Training of LLMs	
13	[illegible]24, 11/26	Supervised Fine-tuning of LLMs, Prompts and In-context learning	HW5
14	12/1, [illegible]	Foundation models, alignment, explainability Open directions in LLM research	
15	12/8, 12/10	**Project Presentations**	Project Final Report
16	12/17	**Final Exam**	Final Exam

HW4

- Released on [the website]
- Due next Friday
- Auto-encoder (4 parts) + bonus GAN
- We recommend Colab

Recall "Conditioning on Text"…



Challenges with text

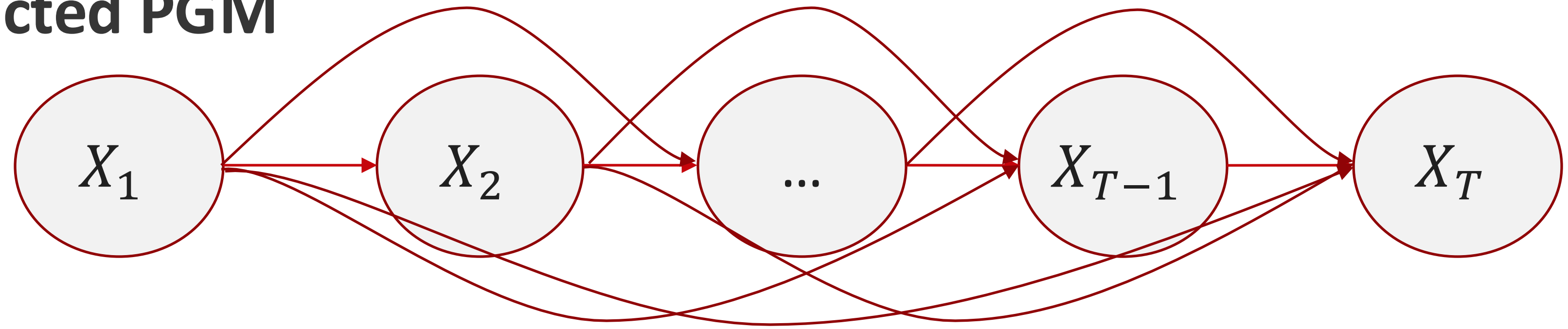
- Variable length input
- Long-range dependencies
- Want a generative model
- Scale
- Emergent properties of large language models

Module 4: Large Language Models			
11	11/10, 11/12	Sequence Learning with RNNs Attention, Transformers	HW4
12	11/17, 11/19	GPT Architectures, Unsupervised Training of LLMs	
13	11/24, 11/26	Supervised Fine-tuning of LLMs, Prompts and In-context learning	HW5
14	12/1, 12/3	Foundation models, alignment, explainability Open directions in LLM research	
15	12/8, 12/10	**Project Presentations**	Project Final Report
16	12/17	**Final Exam**	Final Exam

Where we're going

GPT = Auto-regressive probabilistic model

- **Directed PGM**



$$P_\theta(X) = \prod_i \prod_t P_\theta(X_{i,t} \mid X_{i,<t})$$

- **Probabilistic objective:** Max log-likelihood of observed seqs

$$\max_\theta \sum_i \sum_t \log P_\theta\left(X_{i,t} \mid X_{i,<t} \right)$$

[Radford et al., Improving Language Understanding by Generative Pre-Training]

Today

- **Different Ways to Model Text**
- Sequence Modeling with RNNs
- Different Types of Sequence Modeling Tasks
- Backpropagation Through Time
- Long-Short Term Memory (LSTM)
- Many-to-one Word RNNs

A classic approach: Bag-of-words

"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

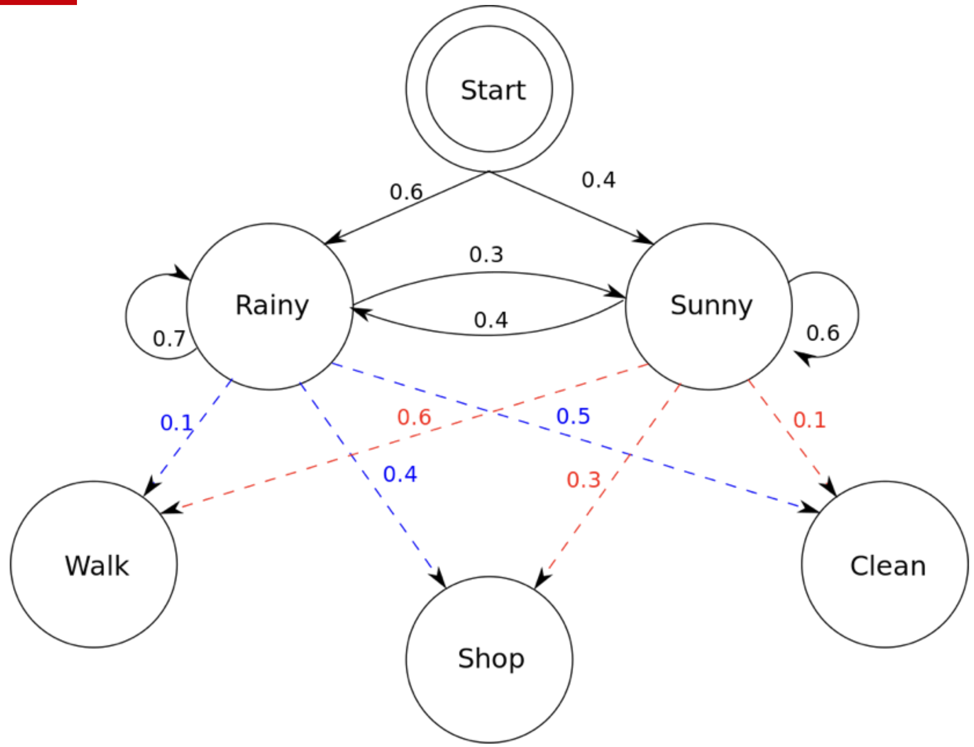
training

Classifier

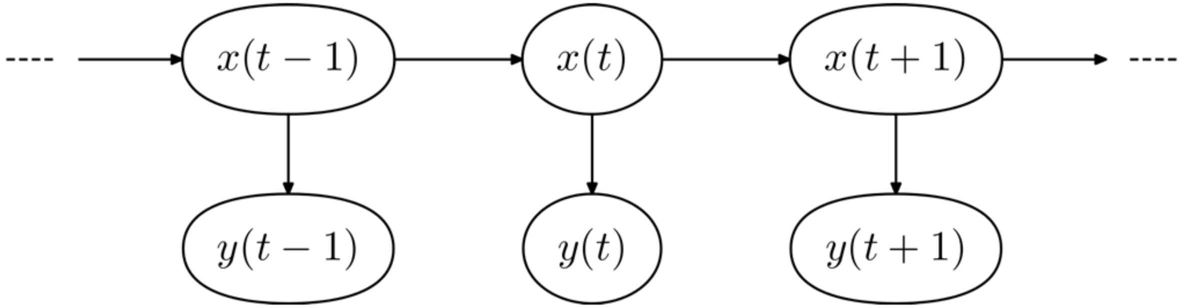
(e.g., logistic regression, MLP, ...)

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

Another classic approach: Hidden Markov Model



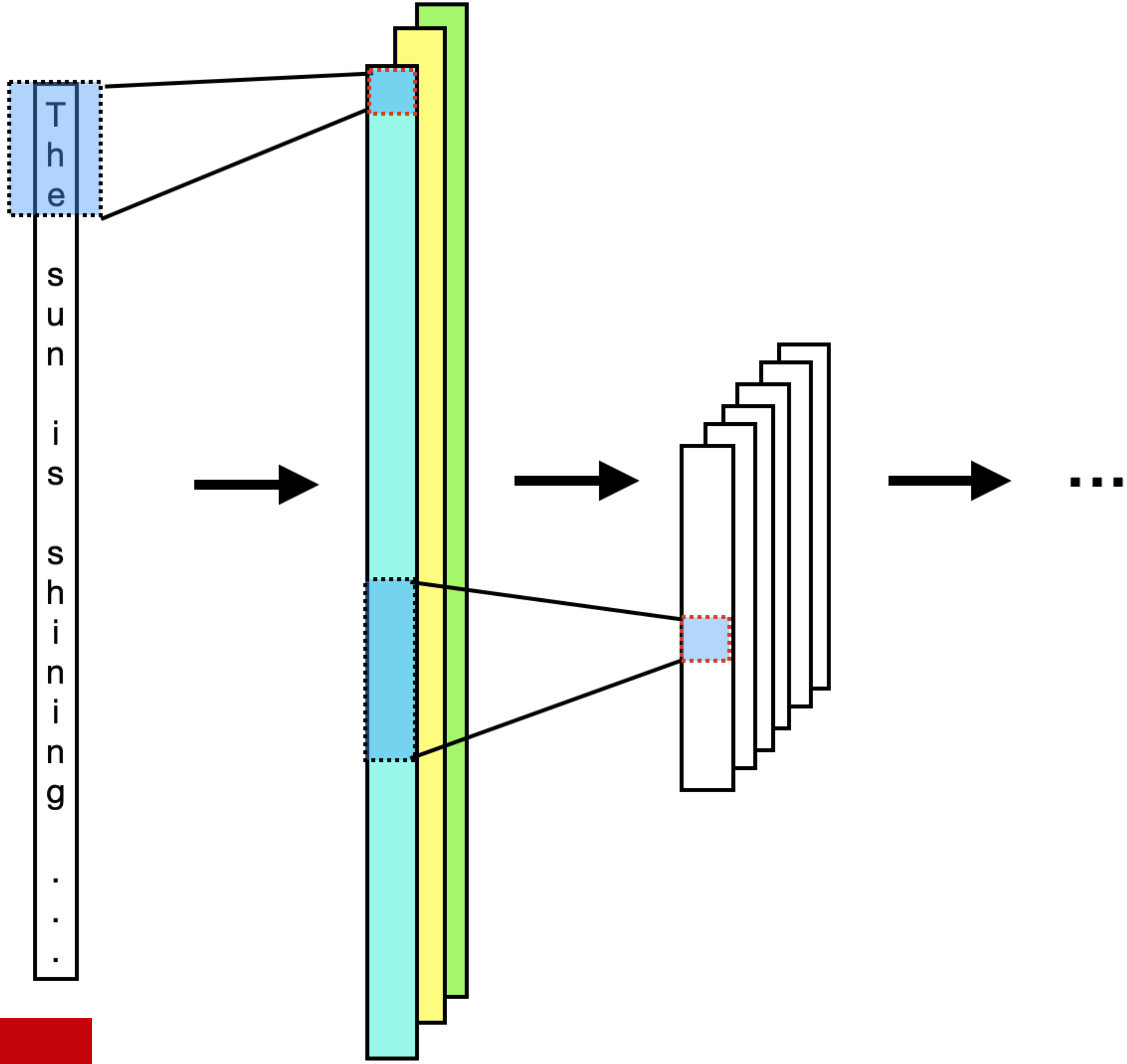
Wikipedia example: each day, weather follows a Markov chain, and activities are observables



$$\mathbb{P}(Y_n = y \mid X_1 = x_1, \ldots, X_n = x_n) = \mathbb{P}(Y_n = y \mid X_n = x_n)$$

Another approach: CNNs

Can’t handle variable length input,
→ need padding to max input length



Today

- Different Ways to Model Text
- **Sequence Modeling with RNNs**
- Different Types of Sequence Modeling Tasks
- Backpropagation Through Time
- Long-Short Term Memory (LSTM)
- Many-to-one Word RNNs

Sequence data: order matters

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

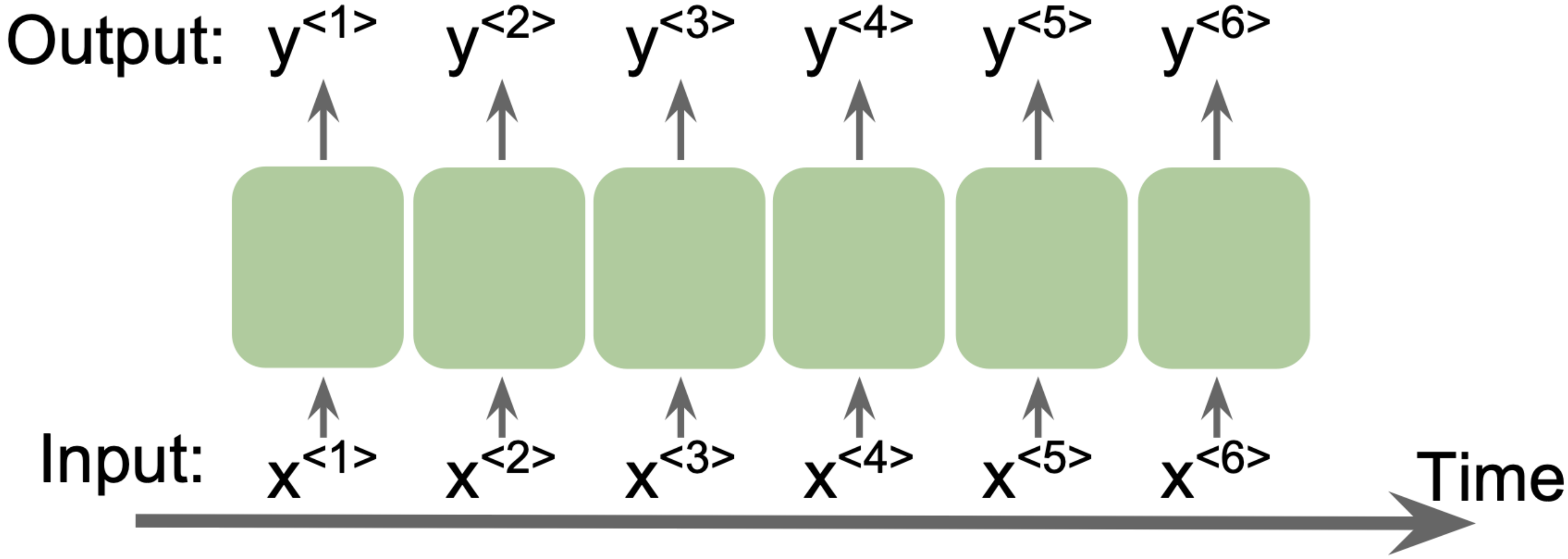
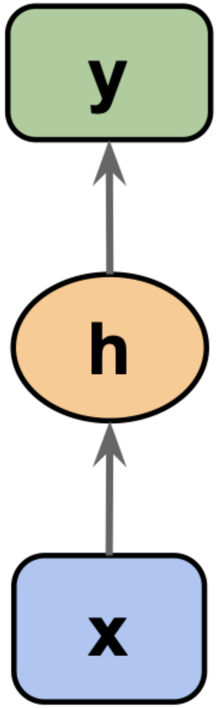


Image source: Sebastian Raschka, Vahid Mirjalili. Python Machine Learning. 3rd Edition. Packt, 2019

Recurrent Neural Networks (RNNs)

Networks we used previously: also called feedforward neural networks



Recurrent Neural Network (RNN)

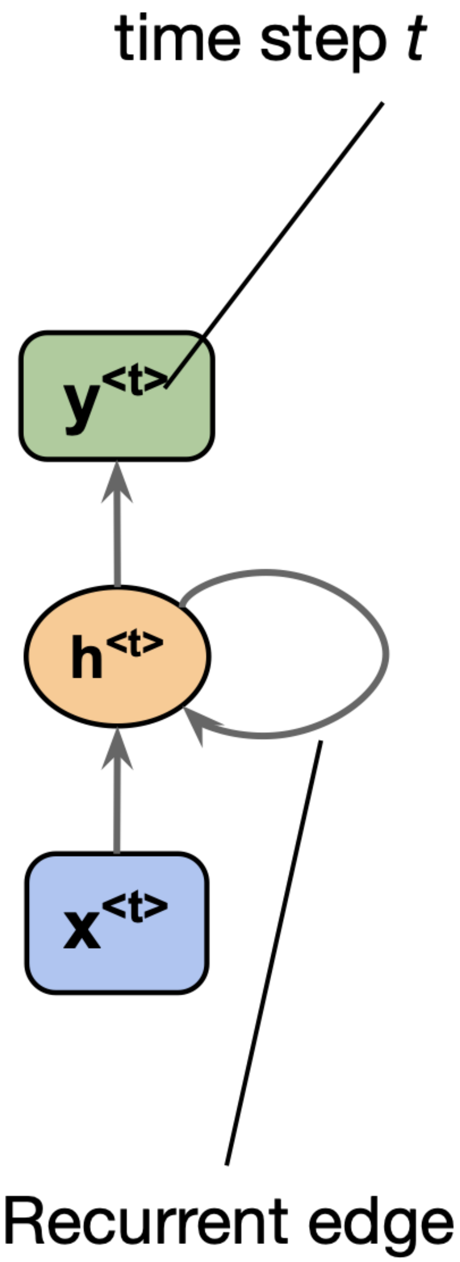


Image source: Sebastian Raschka, Vahid Mirjalili. Python Machine Learning. 3rd Edition. Packt, 2019

Recurrent Neural Networks (RNNs)

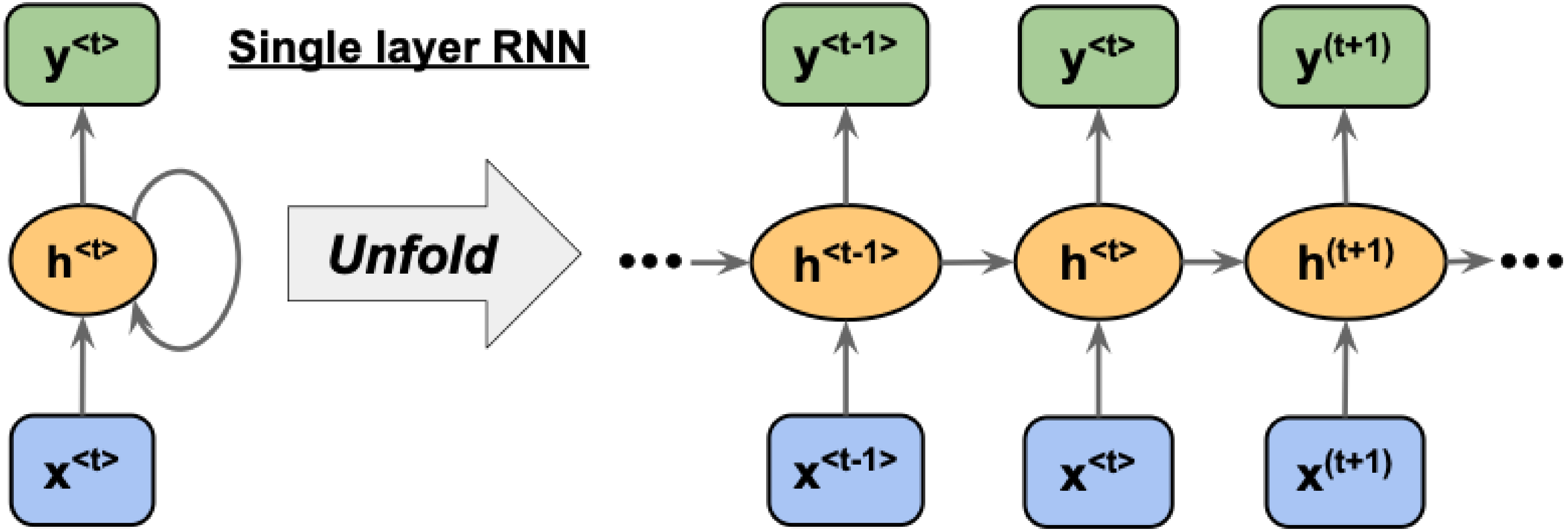
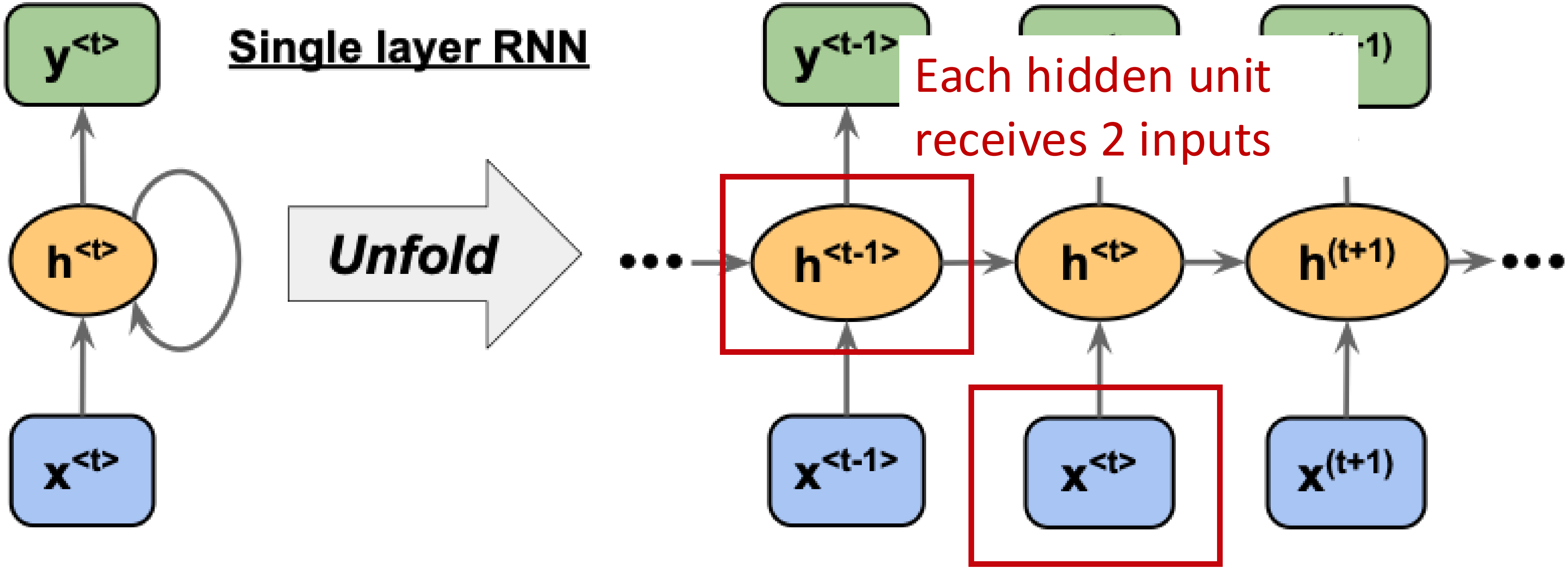


Image source: Sebastian Raschka, Vahid Mirjalili. Python Machine Learning. 3rd Edition. Packt, 2019

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

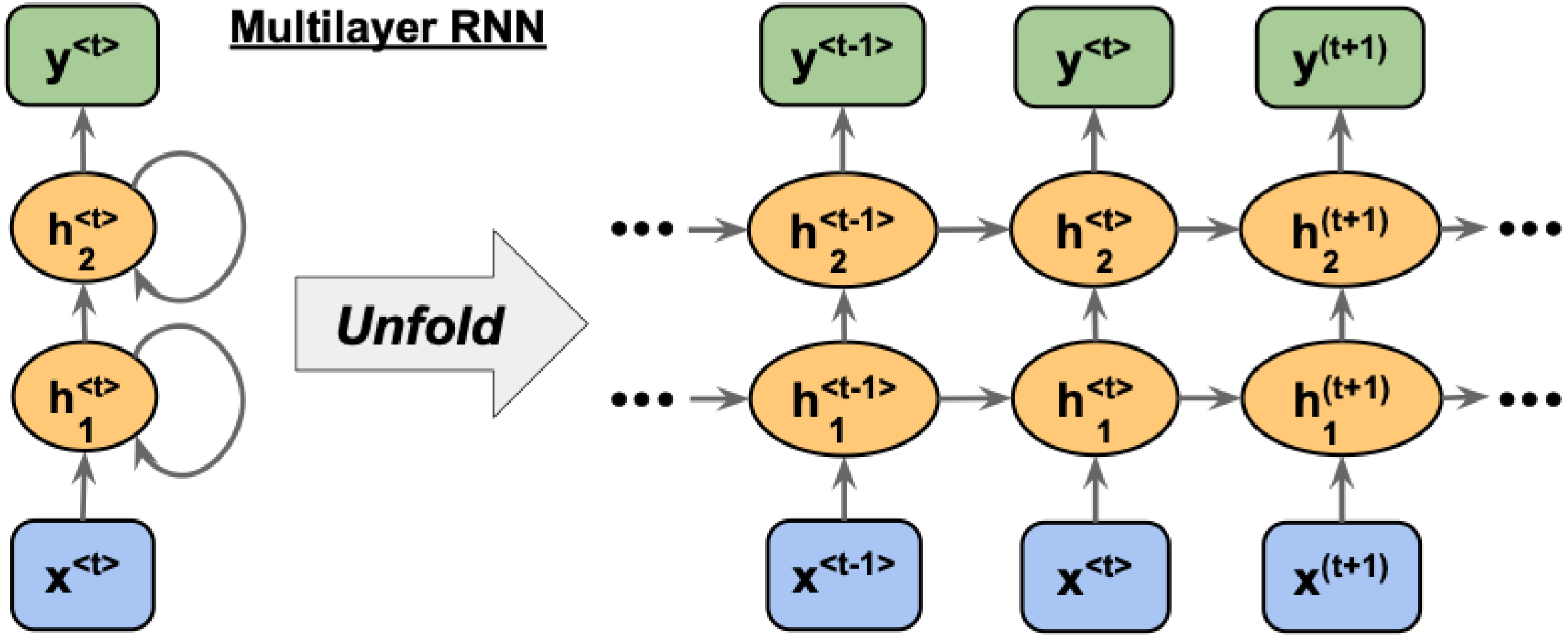
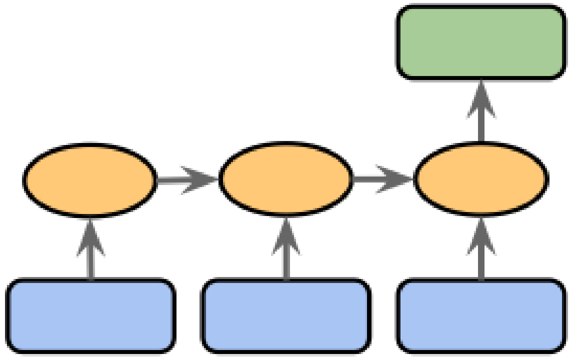
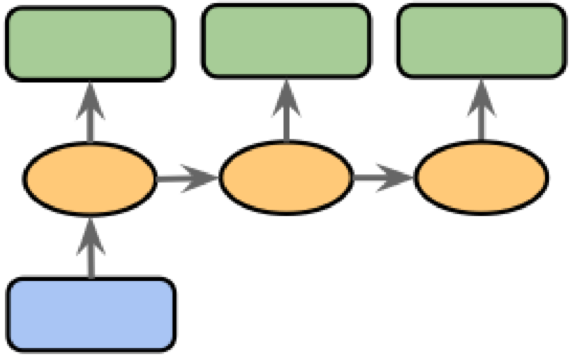


Image source: Sebastian Raschka, Vahid Mirjalili. Python Machine Learning. 3rd Edition. Packt, 2019

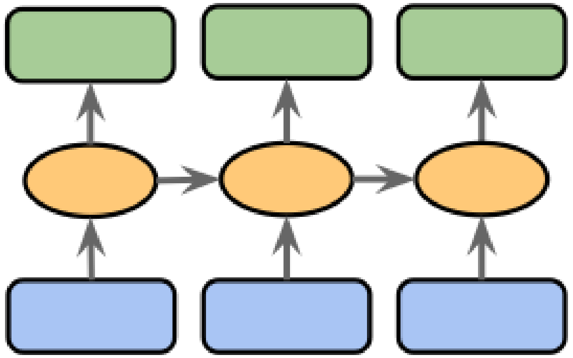
Recurrence unlocks many types of sequence tasks



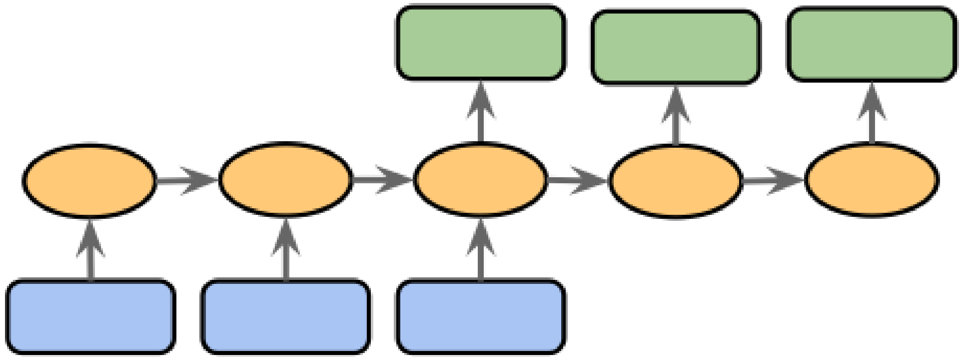
many-to-one



one-to-many



many-to-many



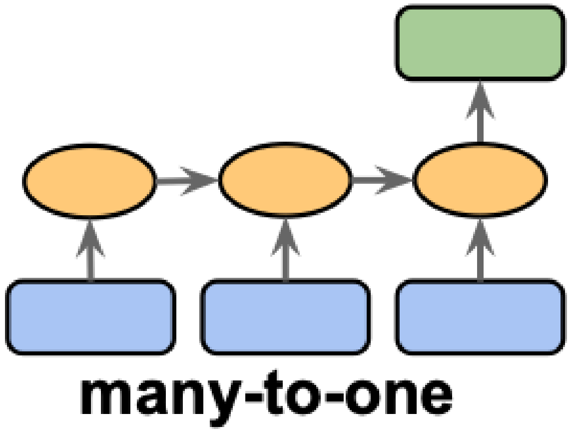
many-to-many

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

Today

- Different Ways to Model Text
- Sequence Modeling with RNNs
- **Different Types of Sequence Modeling Tasks**
- Backpropagation Through Time
- Long-Short Term Memory (LSTM)
- Many-to-one Word RNNs

Recurrence unlocks many types of sequence tasks

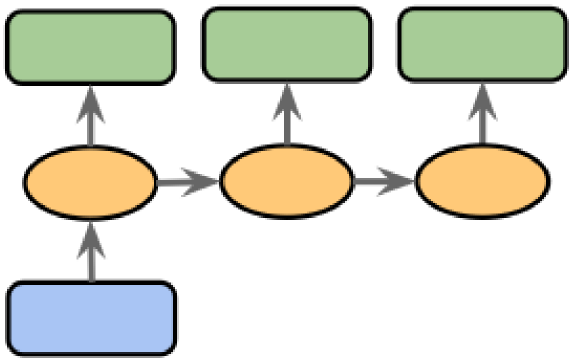


Many-to-one: The input data is a sequence, but the output is a fixed-size vector, not a sequence.

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.

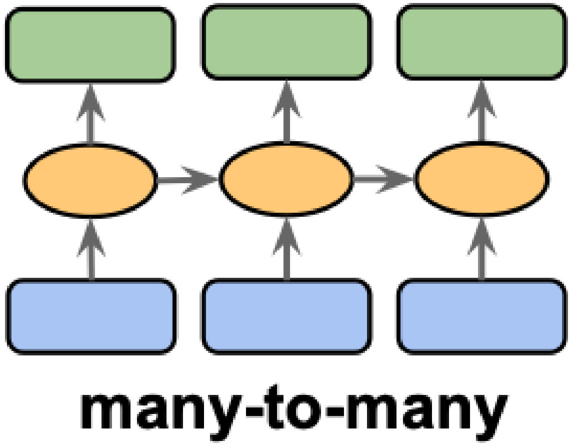
Example: Image captioning, where the input is an image, the output is a text description of that image

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.



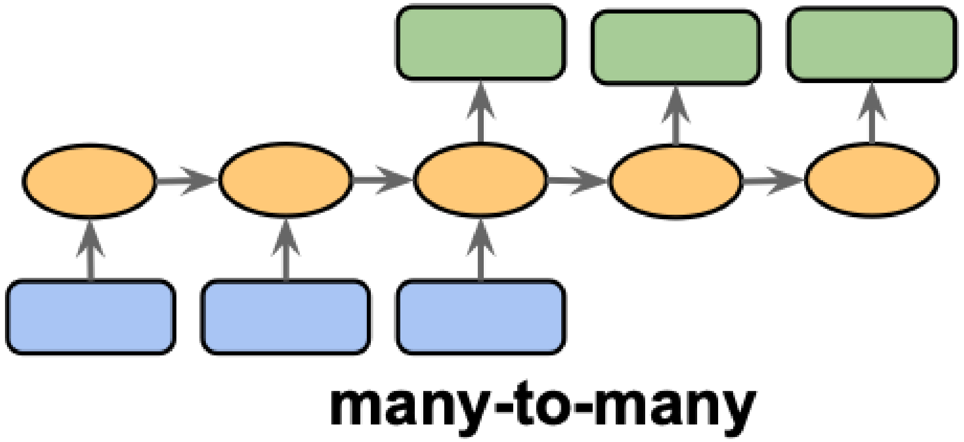


Image source: Sebastian Raschka, Vahid Mirjalili. Python Machine Learning. 3rd Edition. Packt, 2019

Today

- Different Ways to Model Text
- Sequence Modeling with RNNs
- Different Types of Sequence Modeling Tasks
- **Backpropagation Through Time**
- Long-Short Term Memory (LSTM)
- Many-to-one Word RNNs

Under the hood: weight matrices in an RNN

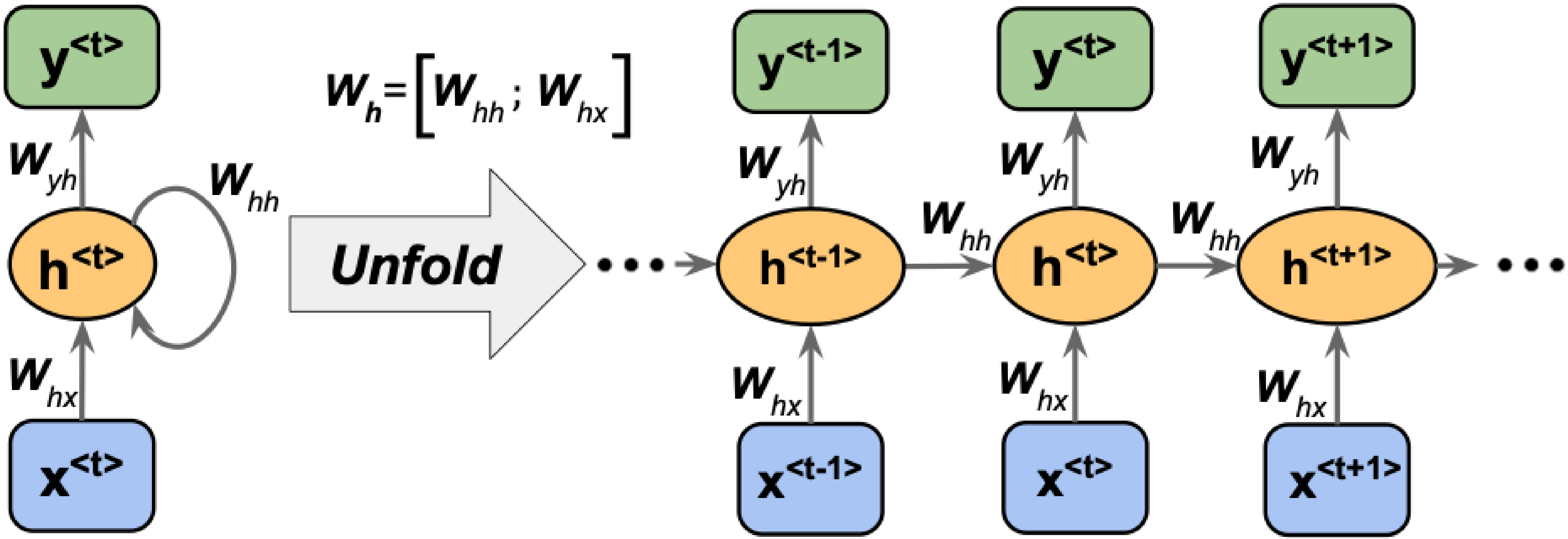
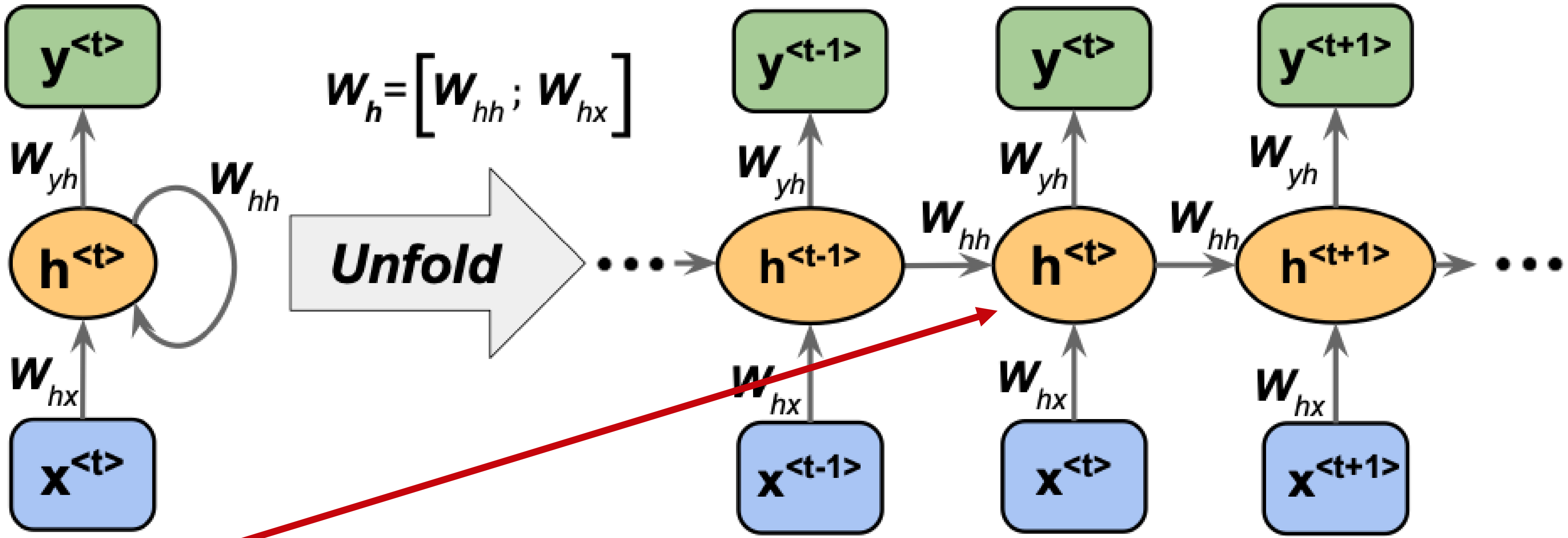


Image source: Sebastian Raschka, Vahid Mirjalili. Python Machine Learning. 3rd Edition. Packt, 2019

Under the hood: weight matrices in an RNN

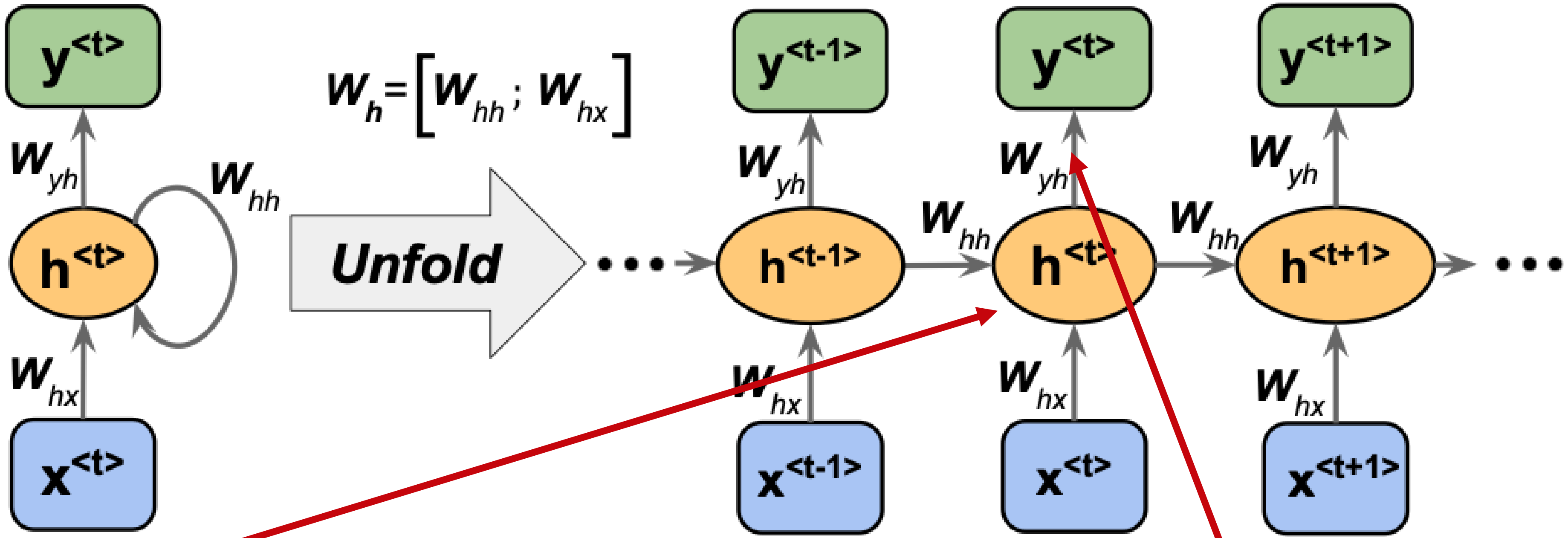


Net input: $$\mathbf{z}_h^{\langle t \rangle} = \mathbf{W}_{hx}\mathbf{x}^{\langle t \rangle} + \mathbf{W}_{hh}\mathbf{h}^{\langle t-1 \rangle} + \mathbf{b}_h$$

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

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

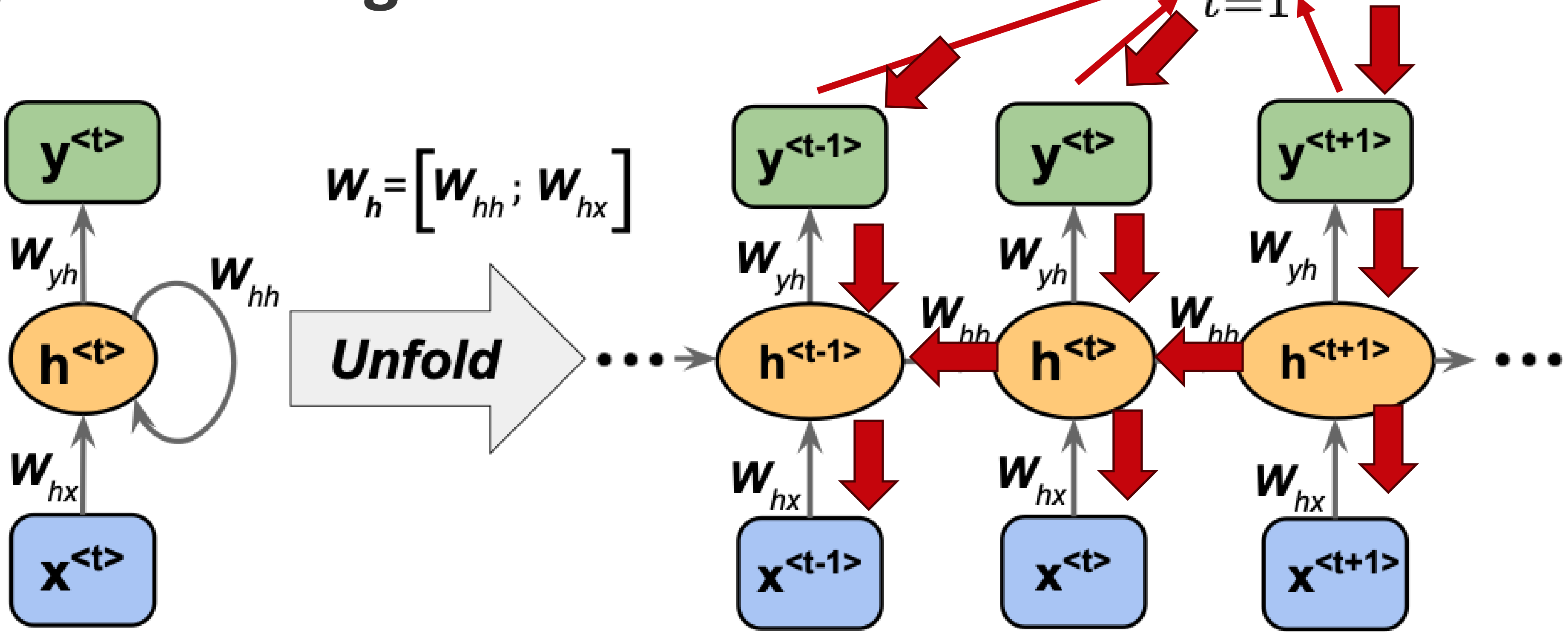
Activation: $\mathbf{h}^{\langle t \rangle} = \sigma_h\left(\mathbf{z}_h^{\langle t \rangle}\right)$

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

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

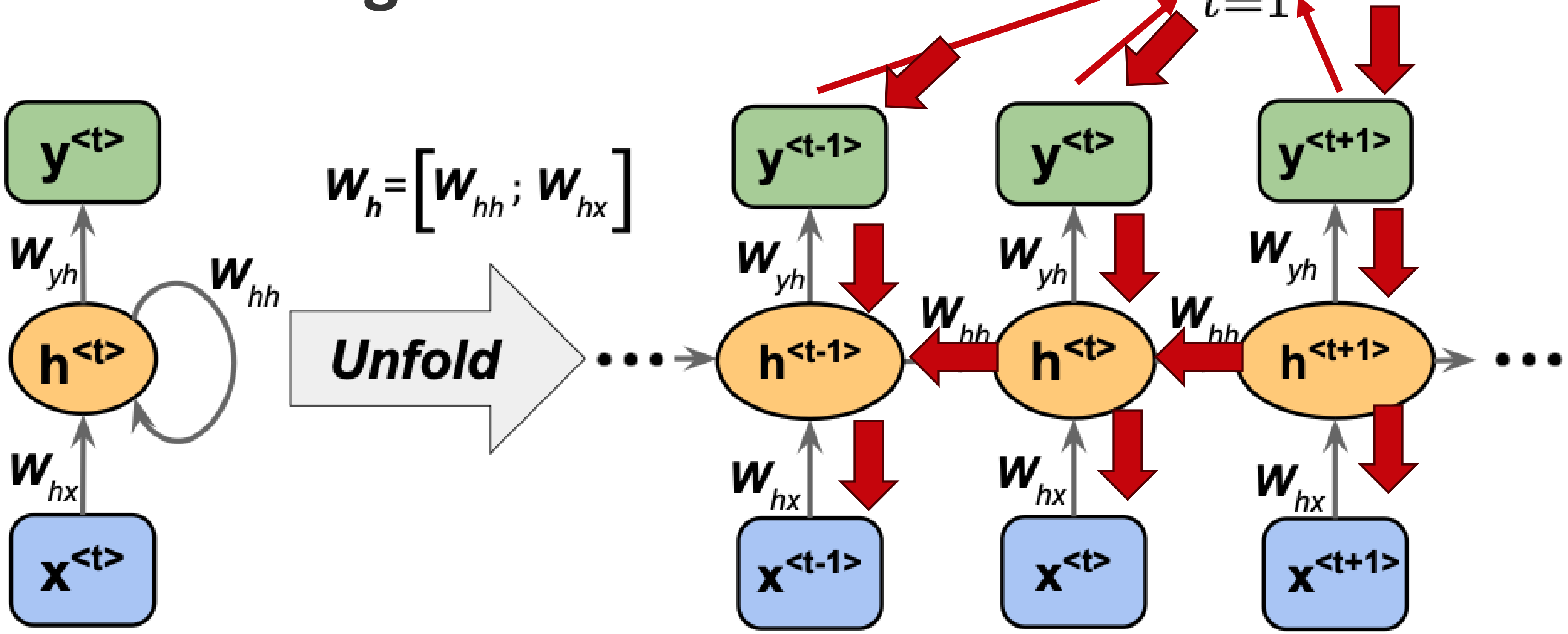


The overall loss can be computed as the sum over all time steps

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

Backpropagation through time

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

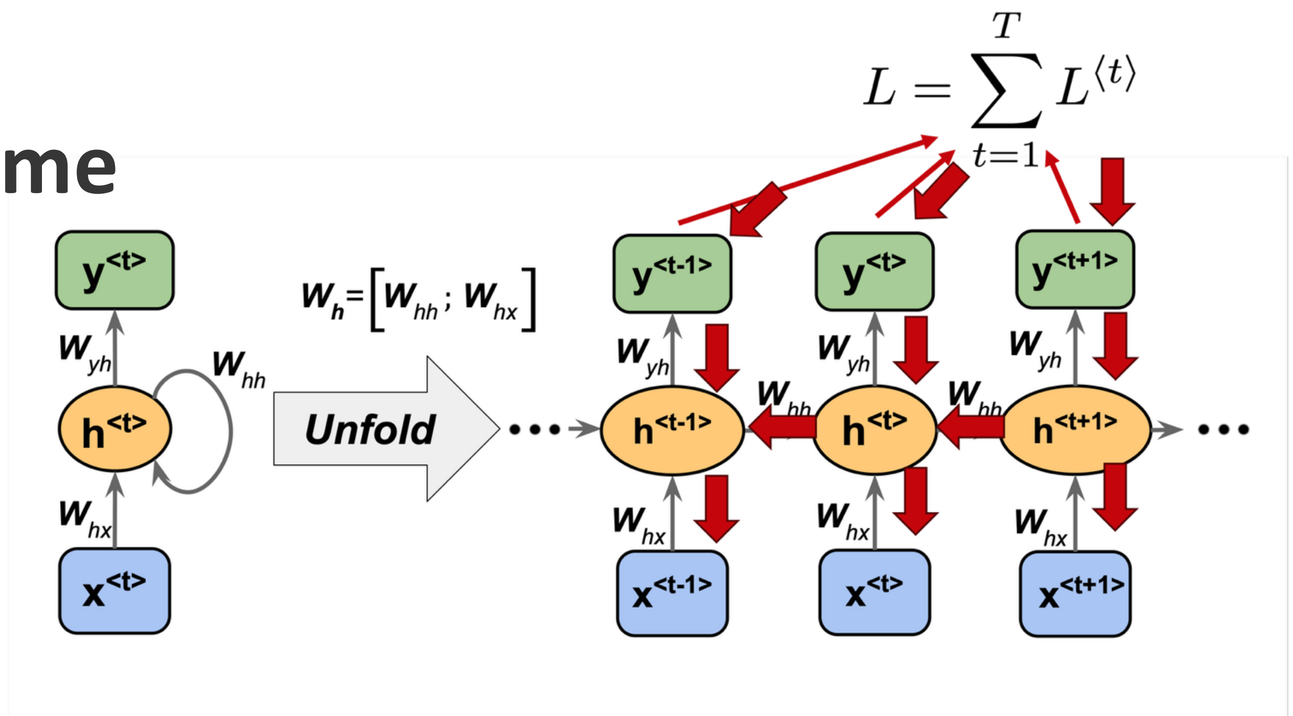


$$L = \sum_{t=1}^{T} L^{(t)}$$

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

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

Backpropagation through time



Computed as a multiplication of adjacent time steps:

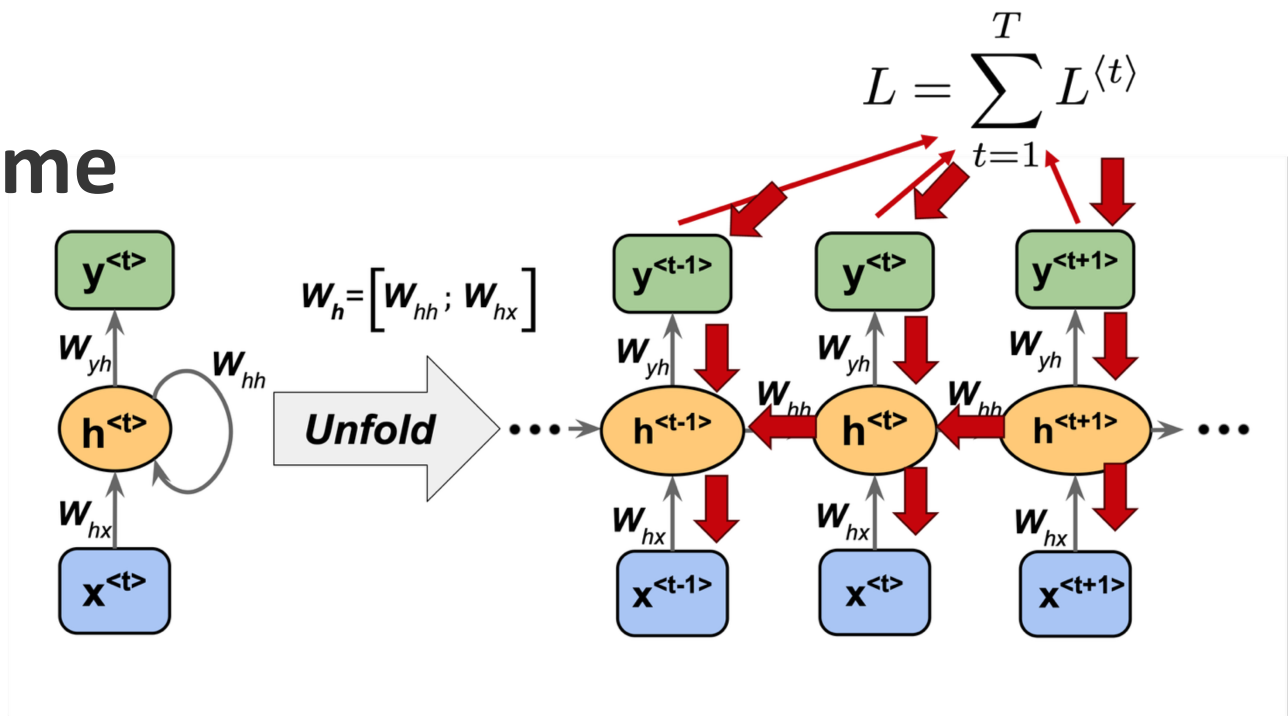
$$L = \sum_{t=1}^{T} L^{(t)}$$

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

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

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

Backpropagation through time



Straightforward, but problematic: vanishing / exploding gradients!

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

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

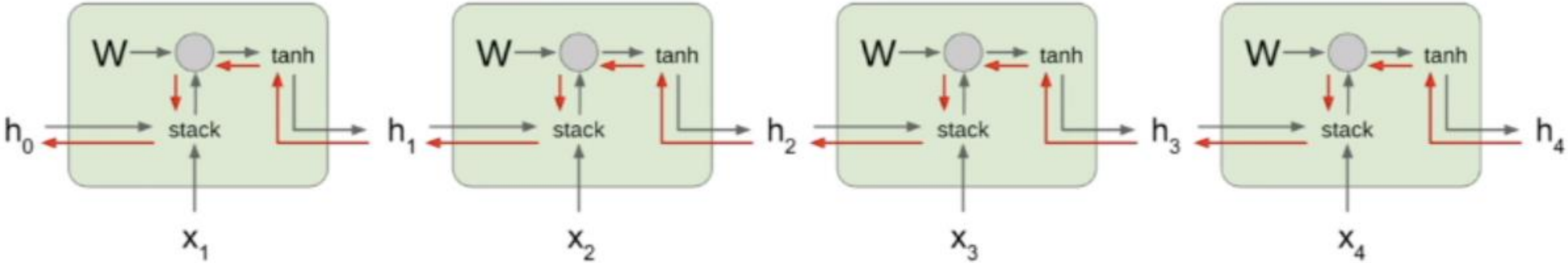
Computed as a multiplication of adjacent time steps:

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

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"

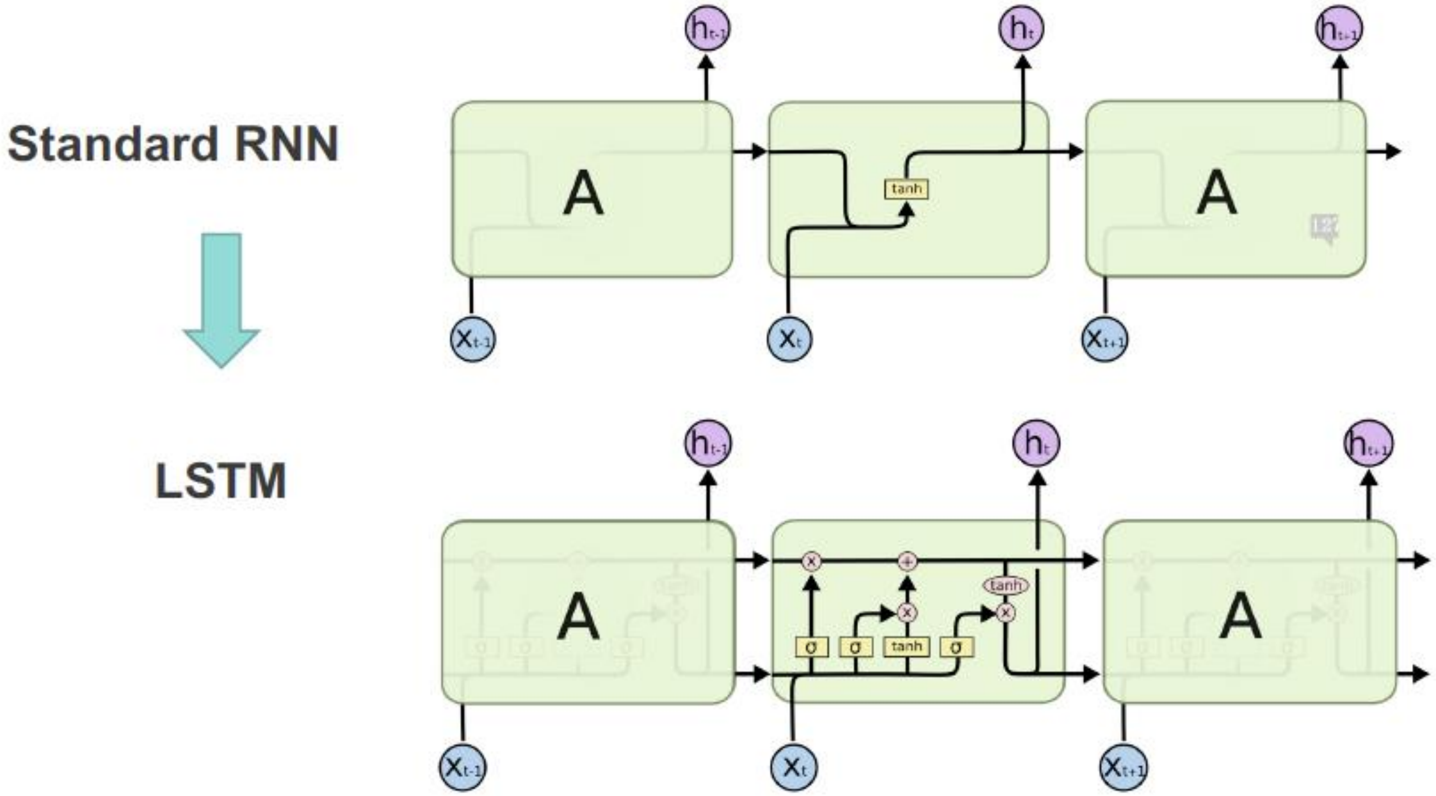
Solutions to Vanishing / Exploding Gradients

- **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

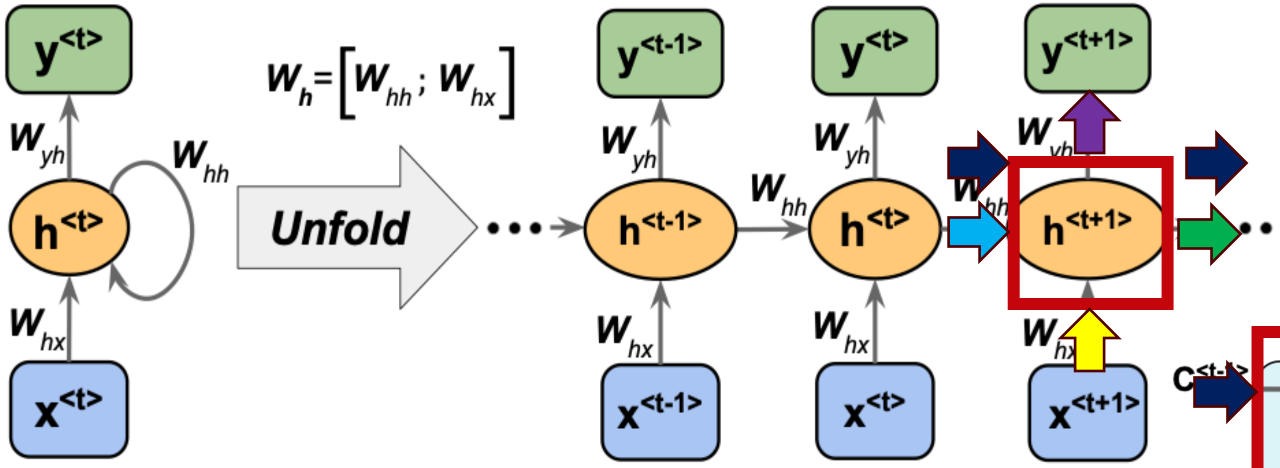


Today

- Different Ways to Model Text
- Sequence Modeling with RNNs
- Different Types of Sequence Modeling Tasks
- Backpropagation Through Time
- **Long-Short Term Memory (LSTM)**
- Many-to-one Word RNNs

Long-short term memory (LSTM)

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



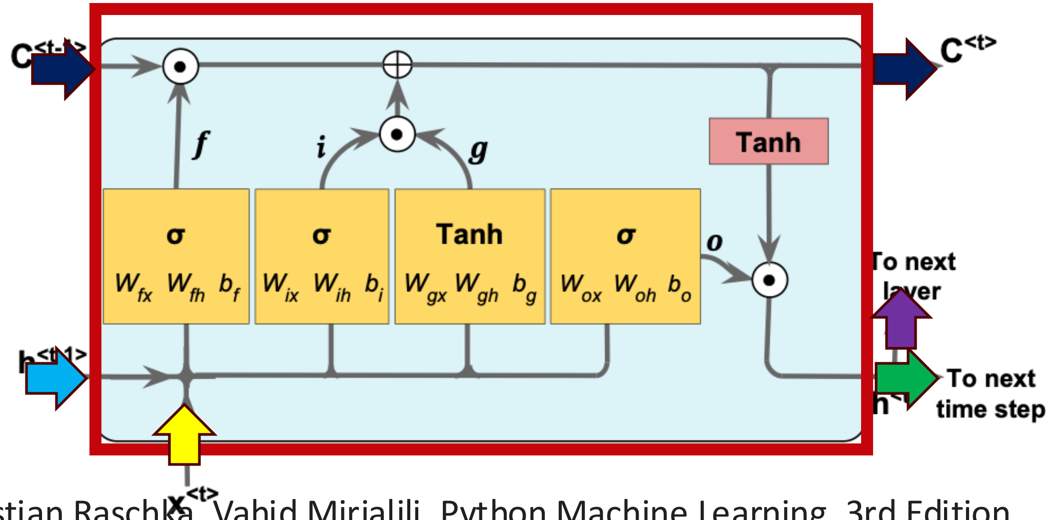


Figure: Sebastian Raschka, Vahid Mirjalili. Python Machine Learning. 3rd Edition. Birmingham, UK: Packt Publishing, 2019

Long-short term memory (LSTM)

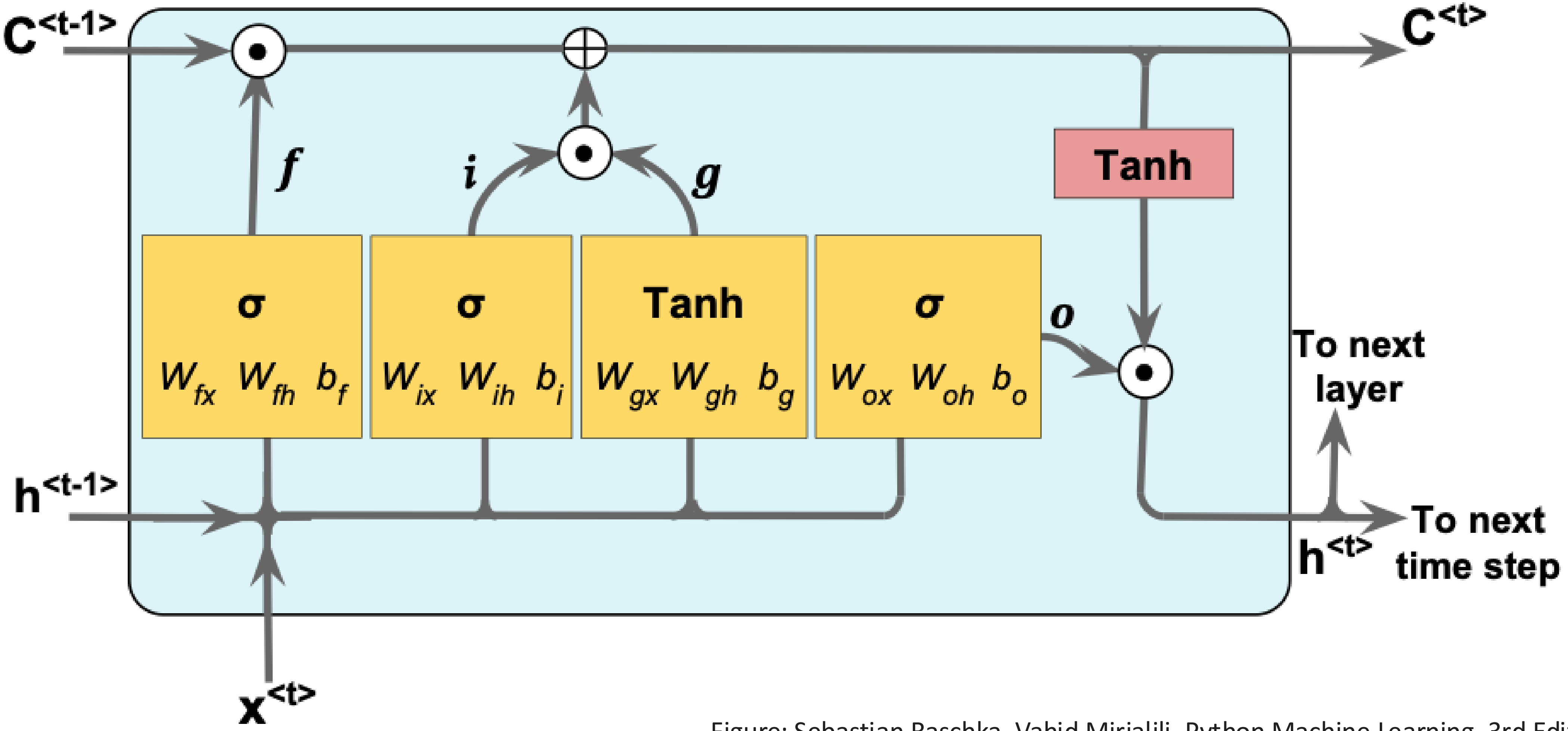


Figure: Sebastian Raschka, Vahid Mirjalili. Python Machine Learning. 3rd Edition. Birmingham, UK: Packt Publishing, 2019

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

Activation from time t-1

Activation at time t

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

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

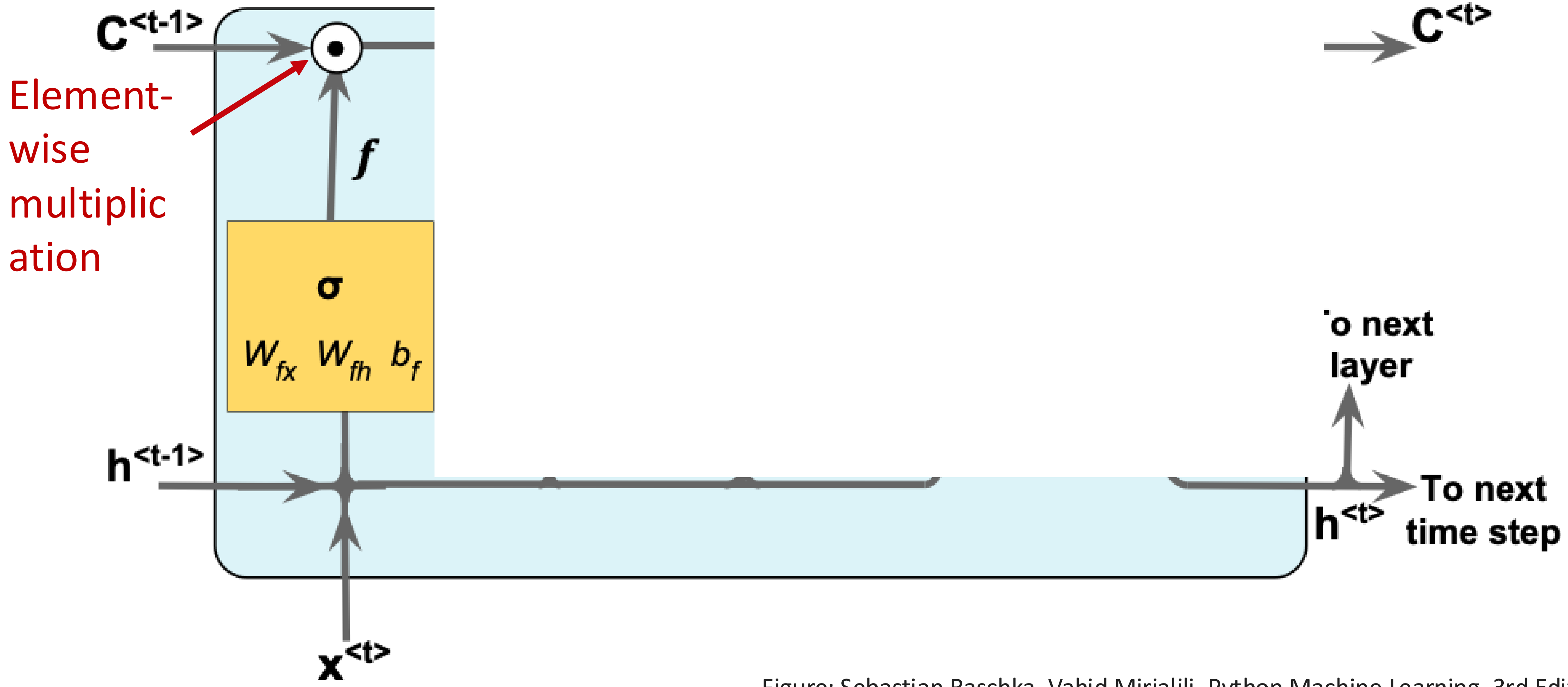


Figure: Sebastian Raschka, Vahid Mirjalili. Python Machine Learning. 3rd Edition. Birmingham, UK: Packt Publishing, 2019

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

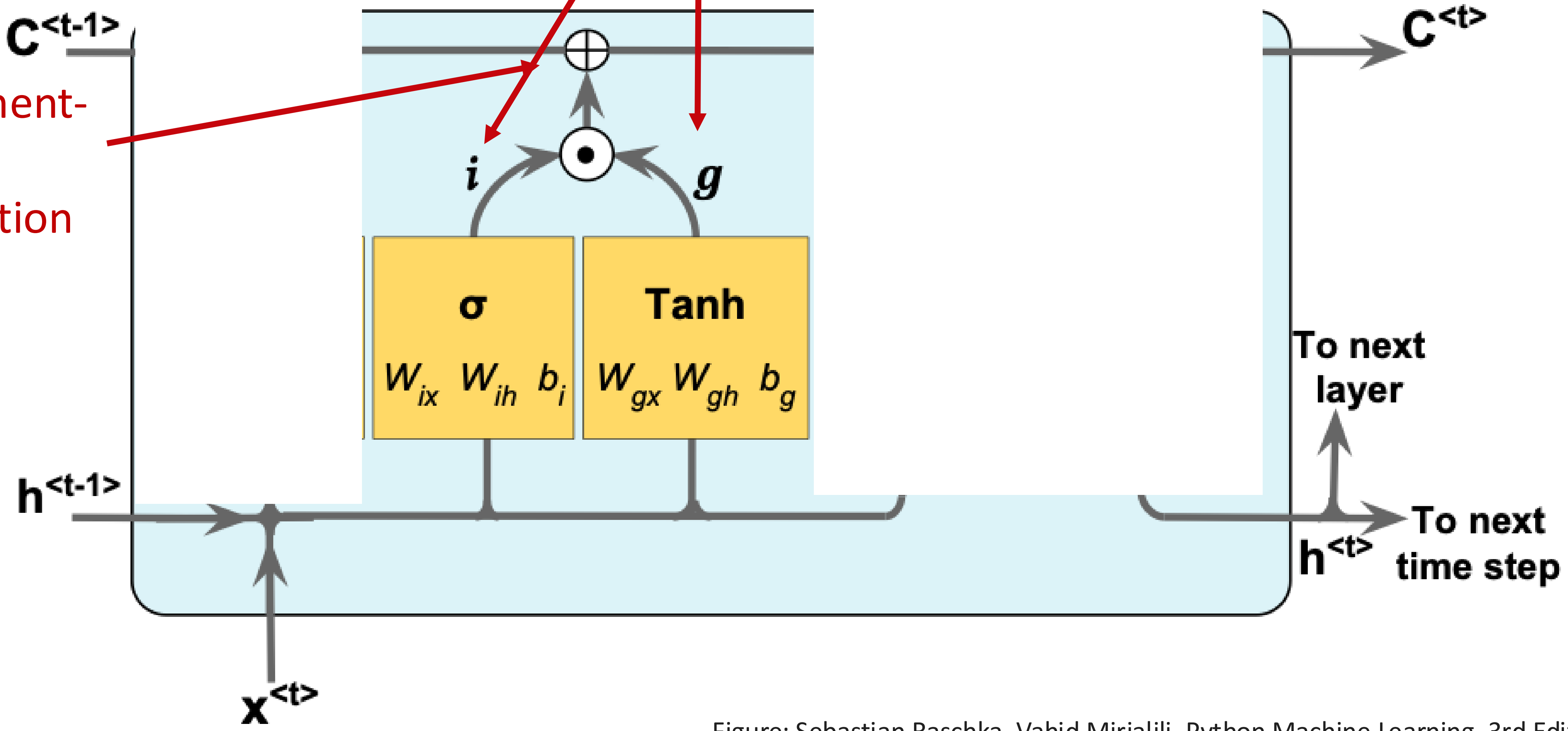


Figure: Sebastian Raschka, Vahid Mirjalili. Python Machine Learning. 3rd Edition. Birmingham, UK: Packt Publishing, 2019

Inside LSTM

Forget Gate Input Node Input Gate

$$C^{\langle t \rangle} = \left(C^{\langle t-1 \rangle} \odot f_t\right) \oplus \underbrace{(i_t \odot g_t)}_{\text{For updating the cell state}}$$

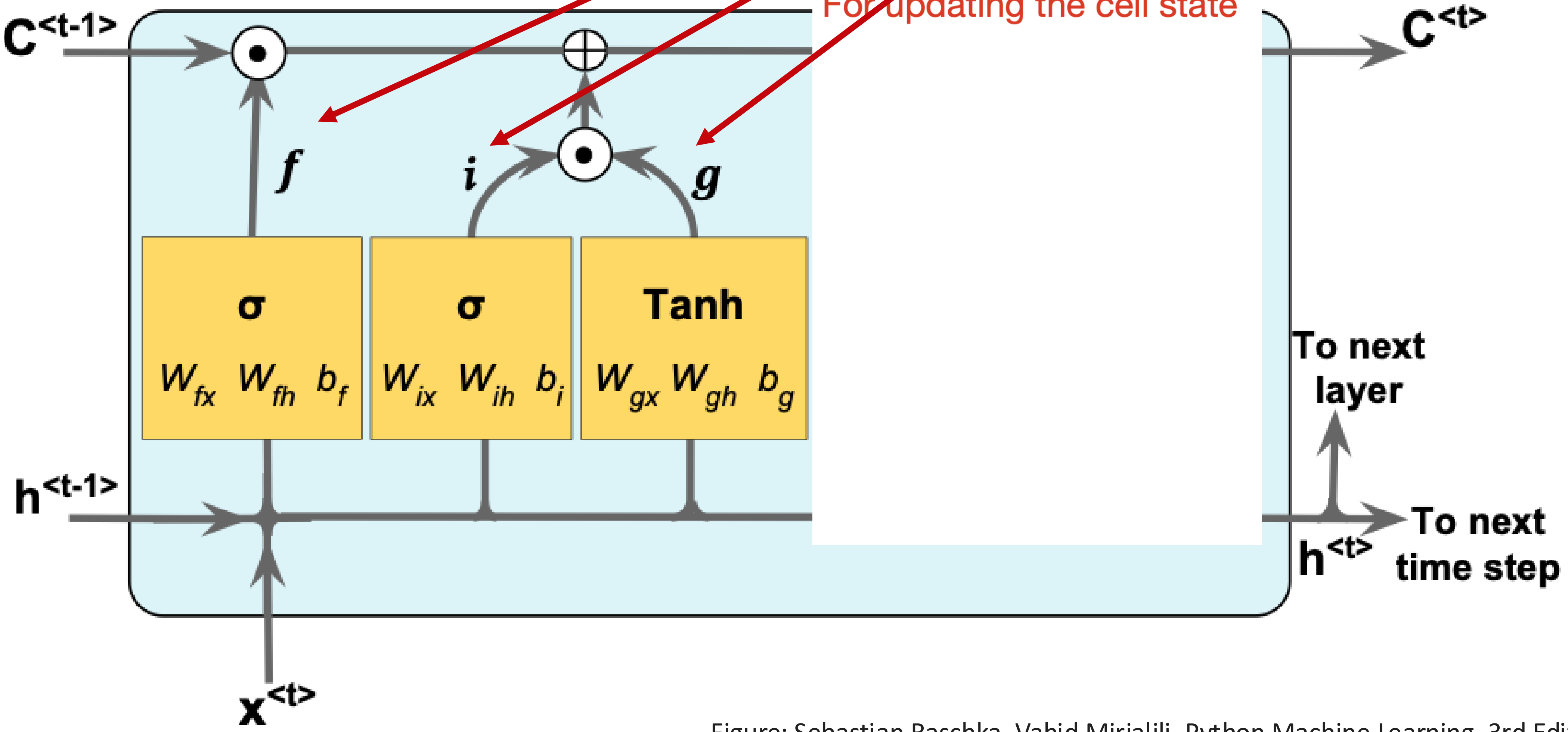
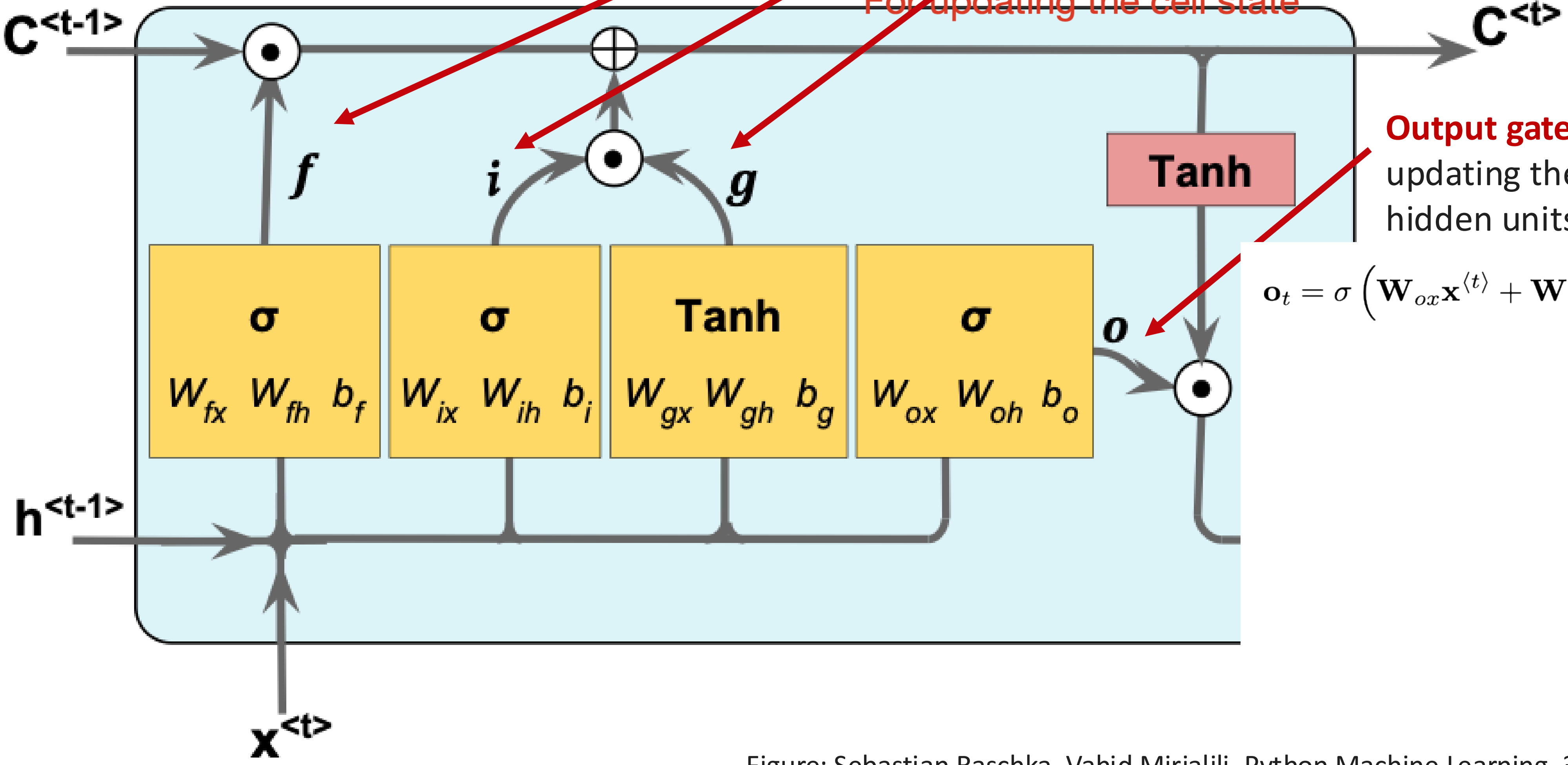


Figure: Sebastian Raschka, Vahid Mirjalili. Python Machine Learning. 3rd Edition. Birmingham, UK: Packt Publishing, 2019

Inside LSTM

$$C^{\langle t \rangle} = \left(C^{\langle t-1 \rangle} \odot f_t\right) \oplus \underbrace{(i_t \odot g_t)}_{\text{For updating the cell state}}$$

Forget Gate (f_t), Input Node (i_t), Input Gate (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

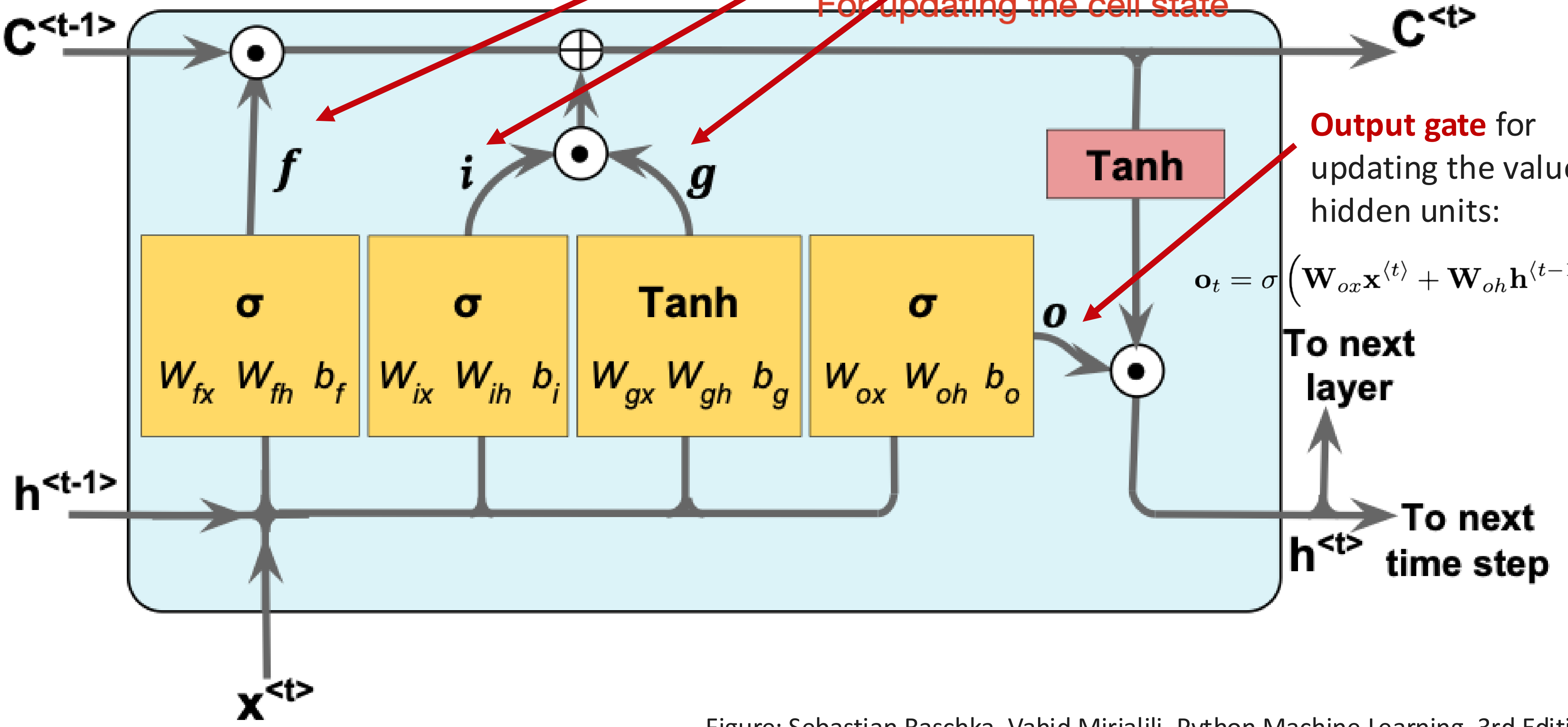
Inside LSTM

Forget Gate

Input Node

Input Gate

$$C^{\langle t \rangle} = \left(C^{\langle t-1 \rangle} \odot f_t\right) \oplus \underbrace{(i_t \odot g_t)}_{\text{For updating the cell state}}$$



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

LSTM Back Together

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

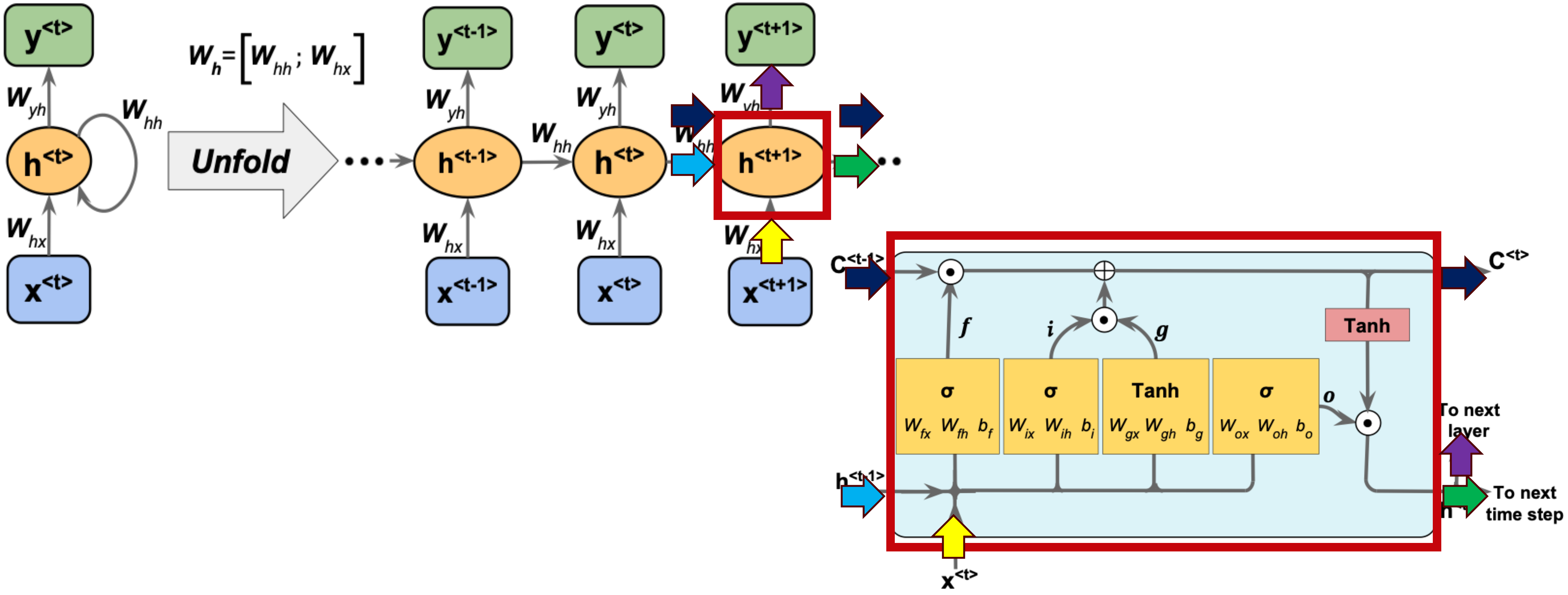


Figure: Sebastian Raschka, Vahid Mirjalili. Python Machine Learning. 3rd Edition. Birmingham, UK: Packt Publishing, 2019

Today

- Different Ways to Model Text
- Sequence Modeling with RNNs
- Different Types of Sequence Modeling Tasks
- Backpropagation Through Time
- Long-Short Term Memory (LSTM)
- **Many-to-one Word RNNs**

RNN Step 1: Build Vocabulary

"Raw" training dataset

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

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

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

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

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: Convert text 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"

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

$\mathbf{x}^{[1]} =$ "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: Convert indices to one-hot representation

"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"

```
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]
[0  0  0  0  1  0  0  0  0  0]
[0  0  1  0  0  0  0  0  0  0]
[0  0  0  1  0  0  0  0  0  0]
              ...
[0  0  0  1  0  0  0  0  0  1]
[0  0  0  1  0  0  0  0  0  1]
[0  0  0  1  0  0  0  0  0  1]
```

RNN Step 4: Convert one-hot to embeddings

One-hot vector

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

Embedding matrix

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

Hidden layer output

PyTorch: Skip steps 3 and 4. Instead…

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

$\mathbf{x}^{[1]} =$ "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
}
```

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

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

`torch.nn.Embedding`

$$\begin{array}{cc} \begin{bmatrix} 0.1 & 4.2 & 2.1 & 1.9 \\ 1.1 & 1.2 & 1.3 & 1.4 \\ 2.1 & 2.2 & 2.3 & 2.4 \\ 3.1 & 2.6 & 1.5 & 9.1 \\ 4.1 & 2.6 & 2.2 & 8.8 \\ 5.1 & 3.6 & 1.5 & 9.1 \\ 6.1 & 9.1 & 7.4 & 9.0 \\ 7.1 & 2.5 & 1.5 & 1.5 \\ 8.1 & 6.1 & 1.5 & 6.2 \\ 9.1 & 5.5 & 1.1 & 9.1 \\ 1.1 & 5.3 & 4.8 & 9.1 \end{bmatrix} & \begin{array}{l} \\ \\ 4. \\ \\ 3. \\ 2. \\ \\ 1. \\ \\ \\ \ldots \end{array} \end{array}$$

Embedded sentence of 1 training example

$$\begin{bmatrix} 7.1 & 2.5 & 1.5 & 1.5 \\ 5.1 & 3.6 & 1.5 & 9.1 \\ 4.1 & 2.6 & 2.2 & 8.8 \\ 2.1 & 2.2 & 2.3 & 2.4 \\ 3.1 & 2.6 & 1.5 & 9.1 \\ \cdots & \cdots & \cdots & \cdots \\ 1.1 & 5.3 & 4.8 & 9.1 \\ 1.1 & 5.3 & 4.8 & 9.1 \\ 1.1 & 5.3 & 4.8 & 9.1 \end{bmatrix}$$

LSTMs in PyTorch

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

Parameters

- **input_size** – The number of expected features in the input *x*
- **hidden_size** – The number of features in the hidden state *h*
- **num_layers** – Number of recurrent layers. E.g., setting `num_layers=2` would mean stacking two LSTMs together to form a *stacked LSTM*, with the second LSTM taking in outputs of the first LSTM and computing the final results. Default: 1
- **bias** – If `False`, then the layer does not use bias weights *b_ih* and *b_hh*. Default: `True`
- **batch_first** – If `True`, then the input and output tensors are provided as (batch, seq, feature). Default: `False`
- **dropout** – If non-zero, introduces a *Dropout* layer on the outputs of each LSTM layer except the last layer, with dropout probability equal to `dropout`. Default: 0
- **bidirectional** – If `True`, becomes a bidirectional LSTM. Default: `False`
- **proj_size** – If `> 0`, will use LSTM with projections of corresponding size. Default: 0

Examples:

```
>>> rnn = nn.LSTM(10, 20, 2)
>>> input = torch.randn(5, 3, 10)
>>> h0 = torch.randn(2, 3, 20)
>>> c0 = torch.randn(2, 3, 20)
>>> output, (hn, cn) = rnn(input, (h0, c0))
```

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