Advanced machine learning and data analysis for the physical sciences

Morten Hjorth-Jensen[1]

Department of Physics and Center for Computing in Science Education,
University of Oslo, Norway[1]

March 6

Plans for the week of March 3-7

1. Reminder on basics of recurrent neural networks (RNNs)
2. Mathematics of RNNs
3. Writing our own codes for RNNs
4. Video of lecture
5. "Whiteboard notes":"https://github.com/CompPhysics/AdvancedMachineLearning
6. Reading recommendations:
   - 6.1 Goodfellow, Bengio and Courville's chapter 10 from Deep Learning
   - 6.2 Sebastian Rashcka et al, chapter 15, Machine learning with Sickit-Learn and PyTorch
   - 6.3 David Foster, Generative Deep Learning with TensorFlow, see chapter 5

The last two books have codes for RNNs in PyTorch and TensorFlow/Keras.

What is a recurrent NN?

A recurrent neural network (RNN), as opposed to a regular fully connected neural network (FCNN) or just neural network (NN), has layers that are connected to themselves.
In an FCNN there are no connections between nodes in a single layer. For instance, $(h_1^1$ is not connected to $(h_2^1$. In addition, the input and output are always of a fixed length.
In an RNN, however, this is no longer the case. Nodes in the hidden layers are connected to themselves.

Why RNNs?

Recurrent neural networks work very well when working with sequential data, that is data where the order matters. In a regular fully connected network, the order of input doesn't really matter. Another property of RNNs is that they can handle variable input and output. Consider again the simplified breast cancer dataset. If you have trained a regular FCNN on the dataset with the two features, it makes no sense to suddenly add a third feature. The network would not know what to do with it, and would reject such inputs with three features (or any other number of features that isn't two, for that matter).

Basic layout, Figures from Sebastian Rashcka et al, Machine learning with Sickit-Learn and PyTorch

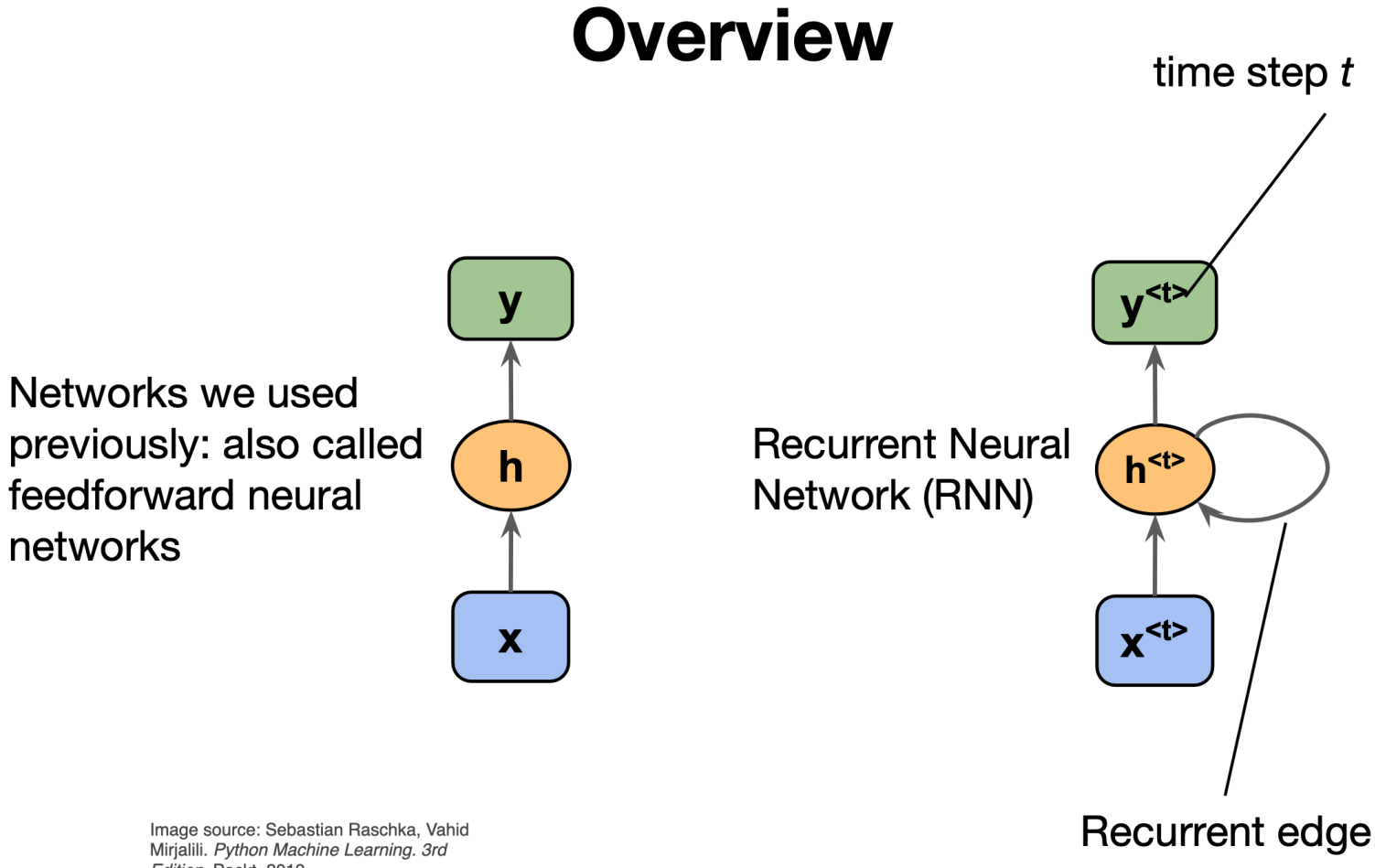


Image source: Sebastian Raschka, Vahid Mirjalili. *Python Machine Learning. 3rd Edition.* Packt, 2019

RNNs in more detail

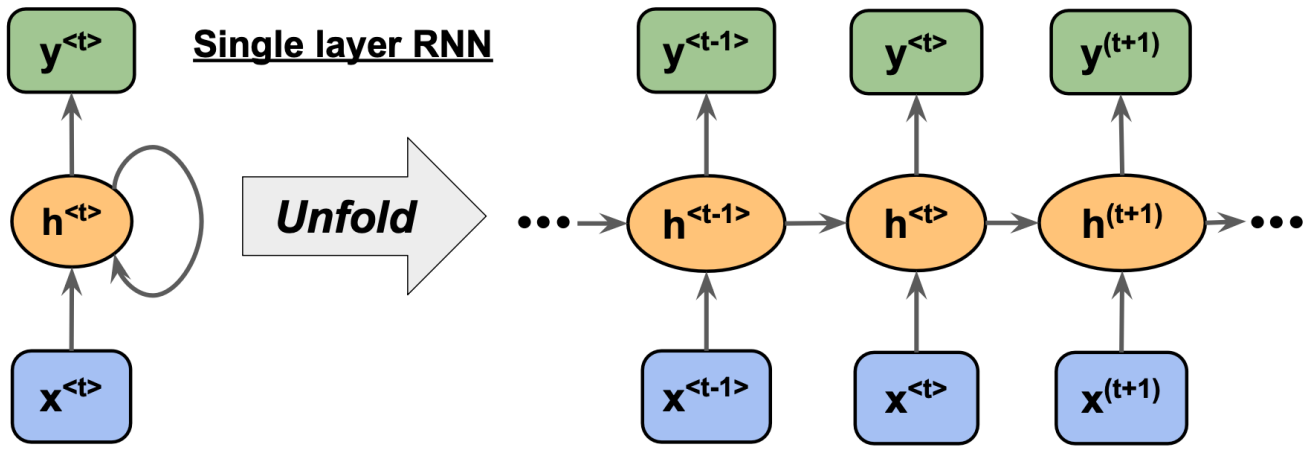
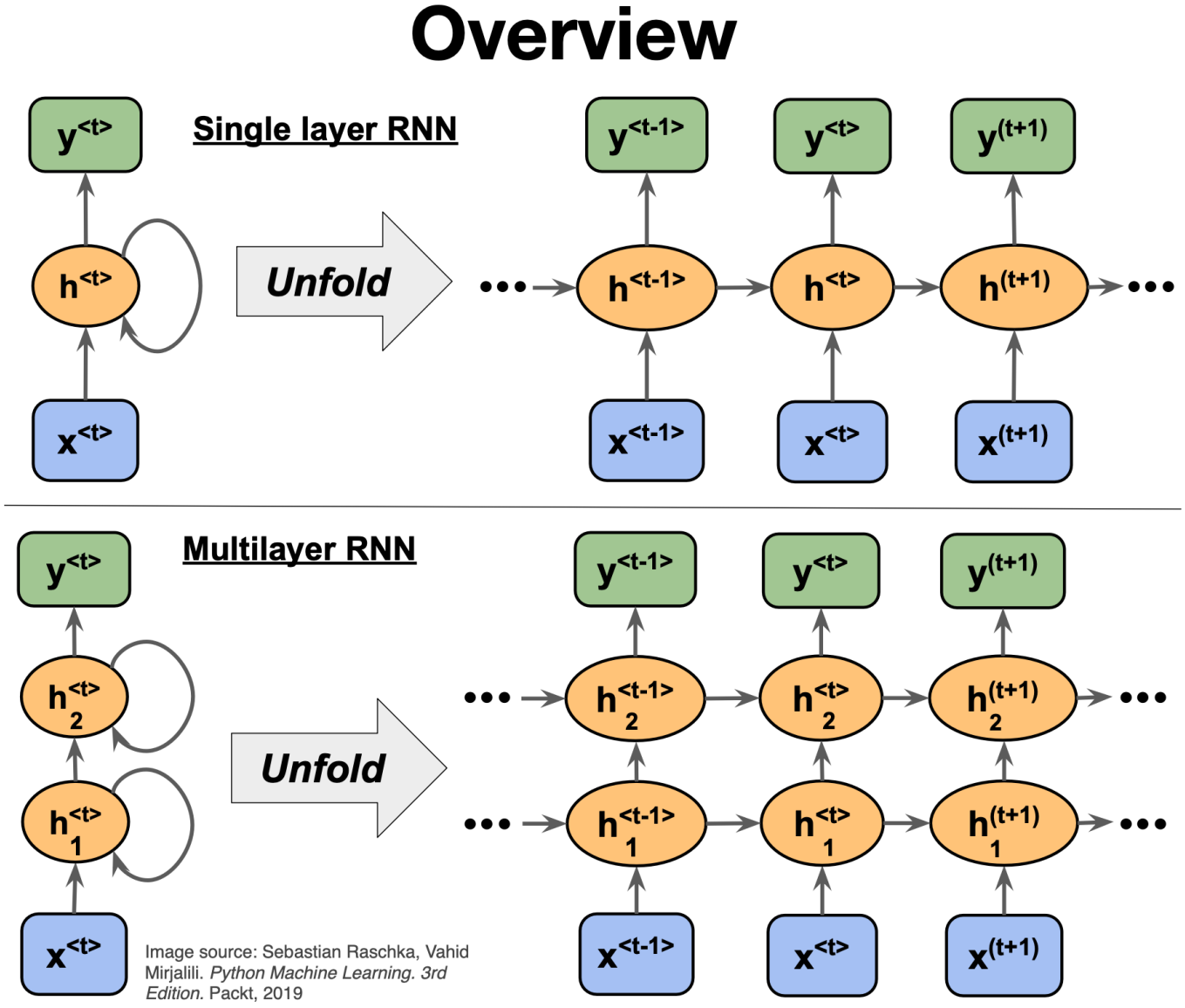
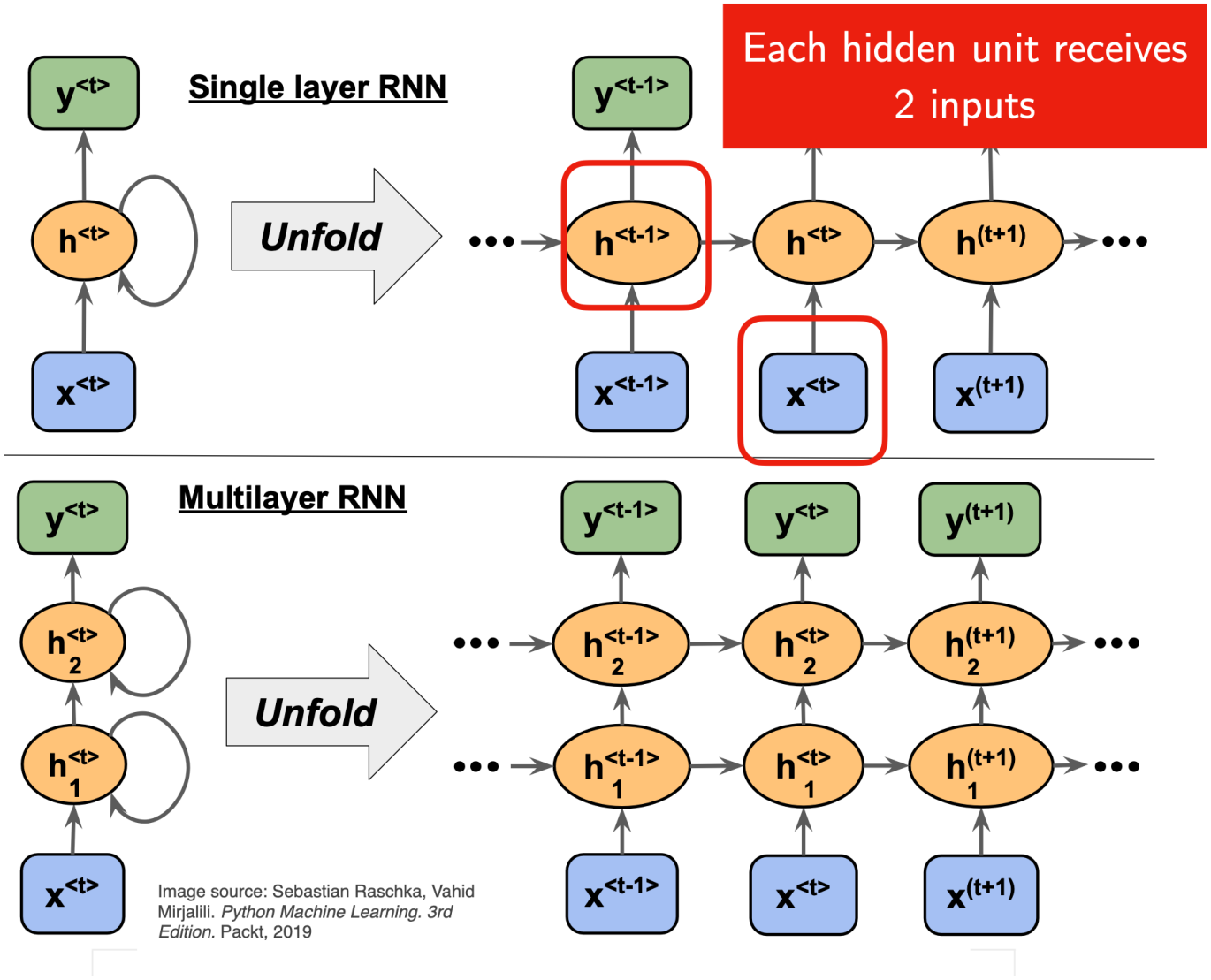


Image source: Sebastian Raschka, Vahid Mirjalili. *Python Machine Learning. 3rd Edition.* Packt, 2019

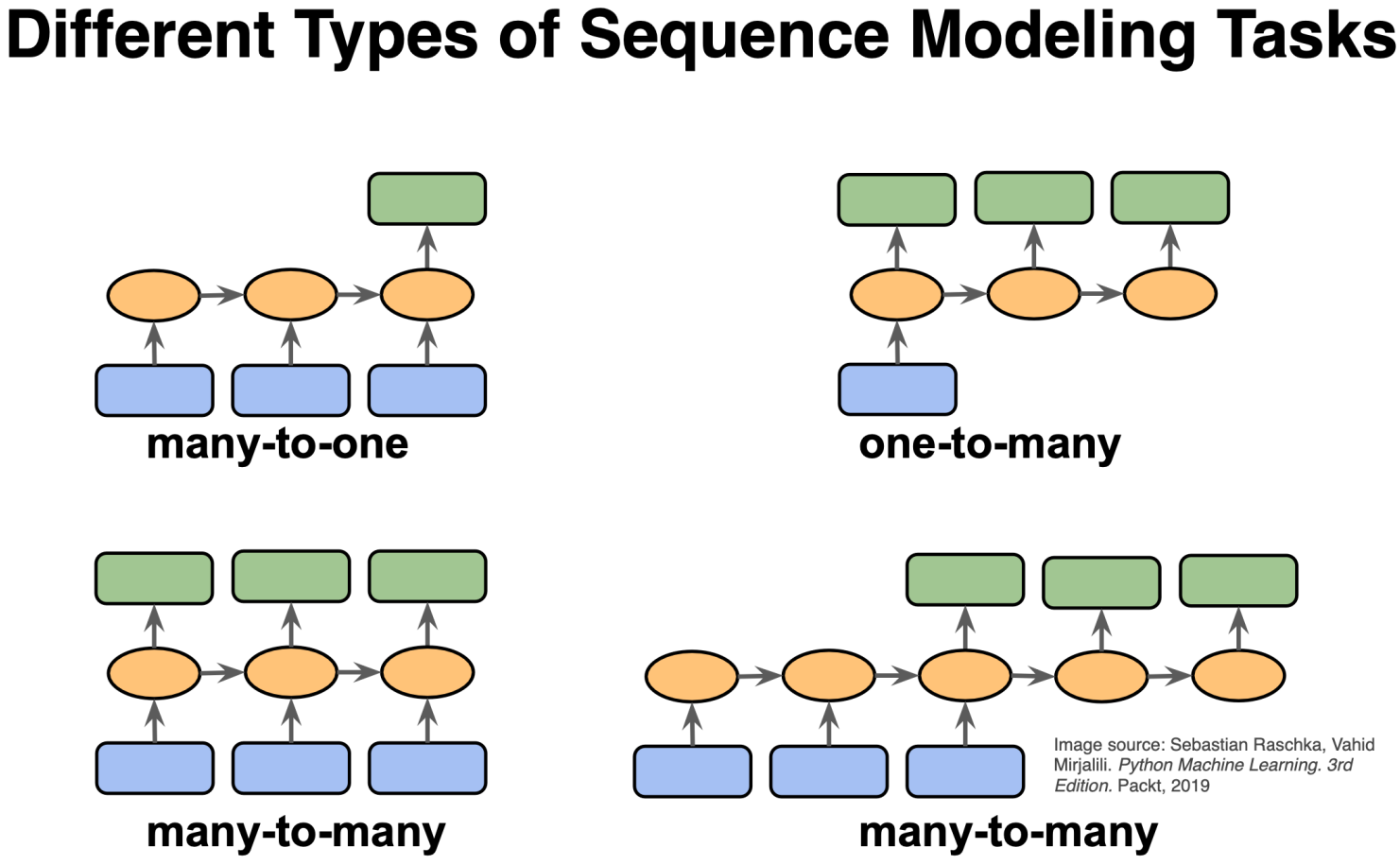
RNNs in more detail, part 2



RNNs in more detail, part 3



RNNs in more detail, part 4



RNNs in more detail, part 5

Weight matrices in a single-hidden layer RNN

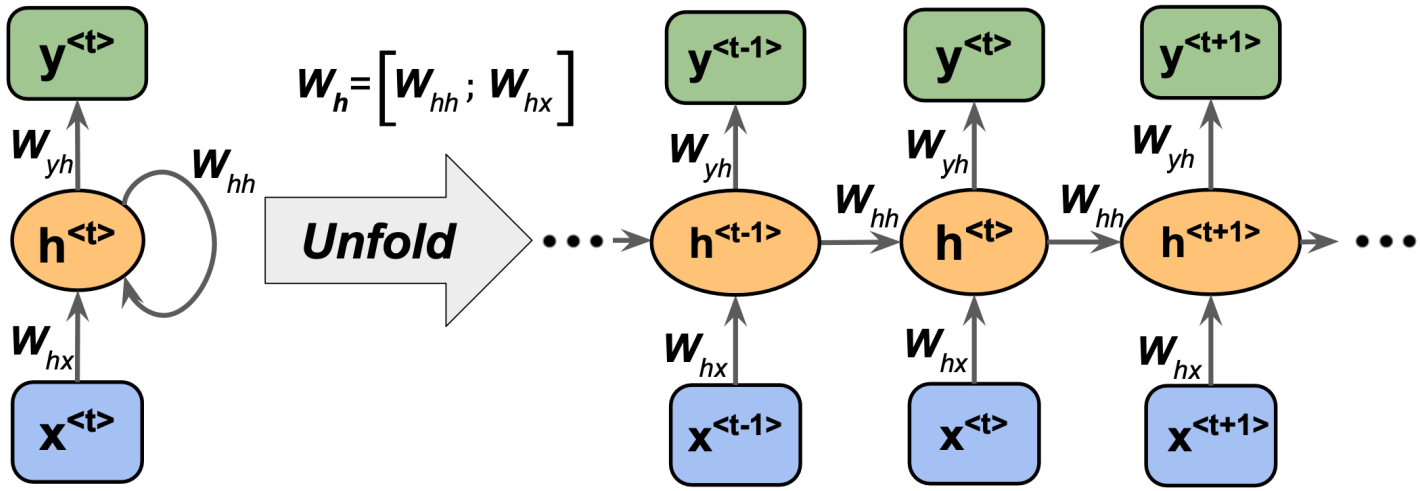


Image source: Sebastian Raschka, Vahid Mirjalili. *Python Machine Learning. 3rd Edition.* Packt, 2019

RNNs in more detail, part 6

Weight matrices in a single-hidden layer RNN

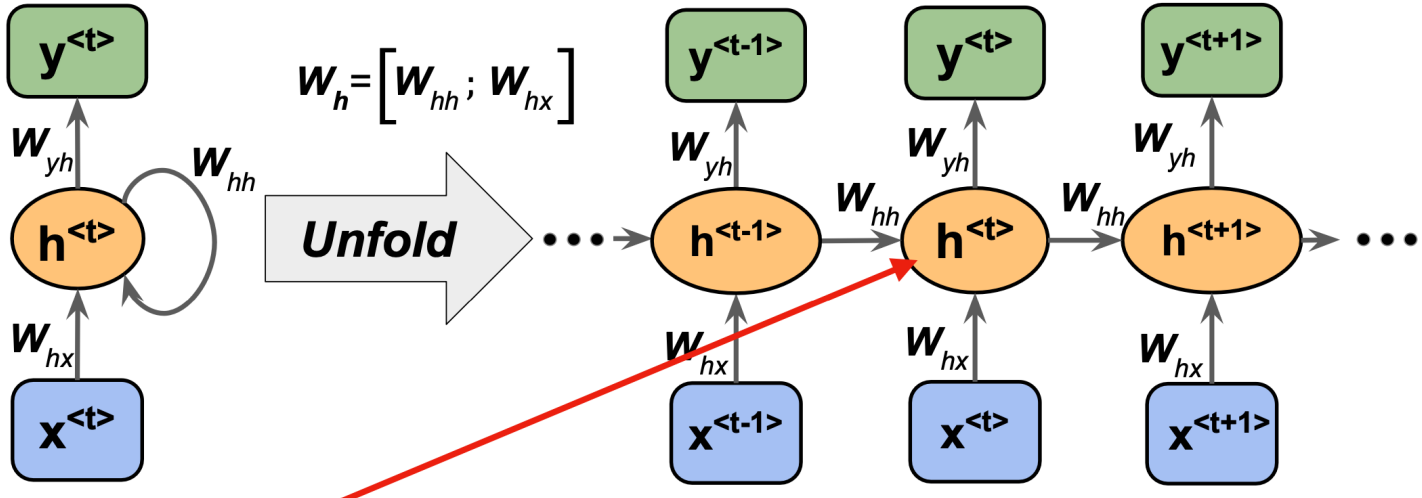


Image source: Sebastian Raschka, Vahid Mirjalili. *Python Machine Learning. 3rd Edition.* Packt, 2019

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:

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

RNNs in more detail, part 7

Weight matrices in a single-hidden layer RNN

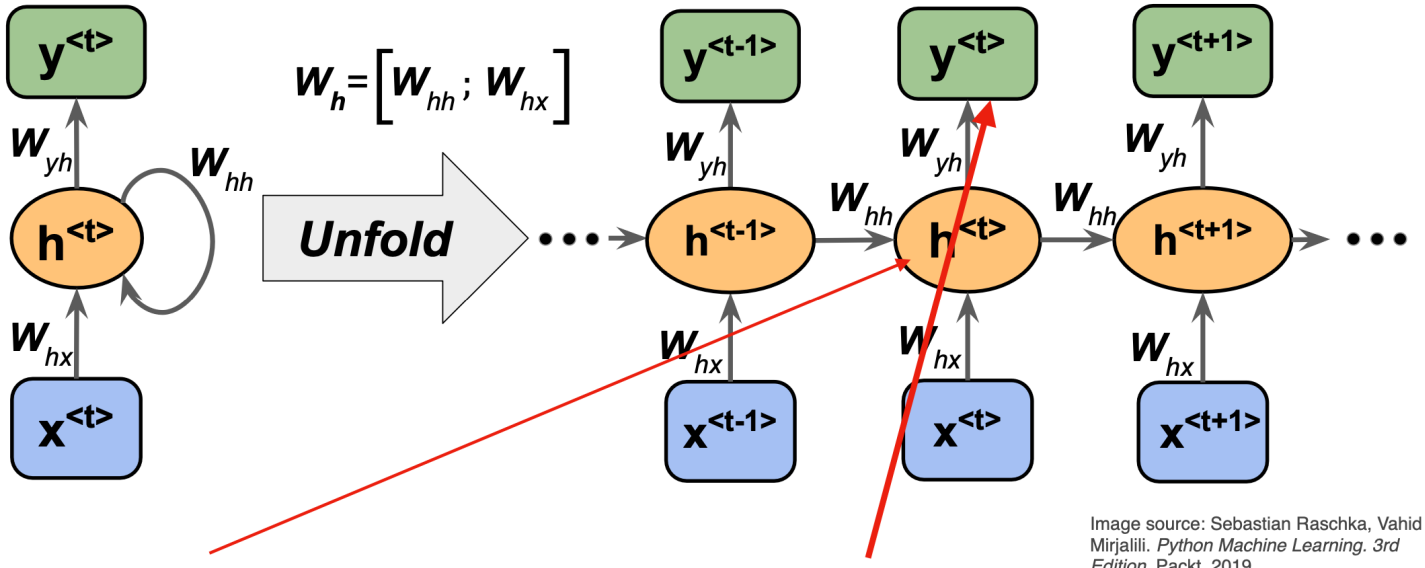


Image source: Sebastian Raschka, Vahid Mirjalili. *Python Machine Learning. 3rd Edition.* Packt, 2019

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

We can think of the recurrent net as a layered, feed-forward net with shared weights and then train the feed-forward net with weight constraints.

We can also think of this training algorithm in the time domain:

1. The forward pass builds up a stack of the activities of all the units at each time step.
2. The backward pass peels activities off the stack to compute the error derivatives at each time step.
3. After the backward pass we add together the derivatives at all the different times for each weight.

The backward pass is linear

1. There is a big difference between the forward and backward passes.
2. In the forward pass we use squashing functions (like the logistic) to prevent the activity vectors from exploding.
3. The backward pass, is completely linear. If you double the error derivatives at the final layer, all the error derivatives will double.

The forward pass determines the slope of the linear function used for backpropagating through each neuron

The problem of exploding or vanishing gradients

- What happens to the magnitude of the gradients as we backpropagate through many layers?
  1. If the weights are small, the gradients shrink exponentially.
  2. If the weights are big the gradients grow exponentially.
- Typical feed-forward neural nets can cope with these exponential effects because they only have a few hidden layers.
- In an RNN trained on long sequences (e.g. 100 time steps) the gradients can easily explode or vanish.
  1. We can avoid this by initializing the weights very carefully.
- Even with good initial weights, its very hard to detect that the current target output depends on an input from many time-steps ago.

RNNs have difficulty dealing with long-range dependencies.

The mathematics of RNNs, the basic architecture

See notebook at https://github.com/CompPhysics/AdvancedMachineLearning/blob/main/doc/pub/week7/ipynb/rnnmath.ipynb

Four effective ways to learn an RNN and preparing for next week

1. Long Short Term Memory Make the RNN out of little modules that are designed to remember values for a long time.
2. Hessian Free Optimization: Deal with the vanishing gradients problem by using a fancy optimizer that can detect directions with a tiny gradient but even smaller curvature.
3. Echo State Networks (ESN): Initialize the input a hidden and hidden-hidden and output-hidden connections very carefully so that the hidden state has a huge reservoir of weakly coupled oscillators which can be selectively driven by the input. ESNs only need to learn the hidden-output connections.
4. Good initialization with momentum Initialize like in Echo State Networks, but then learn all of the connections using momentum

Long Short Term Memory (LSTM)

LSTM uses a memory cell for modeling long-range dependencies and avoid vanishing gradient problems.

1. Introduced by Hochreiter and Schmidhuber (1997) who solved the problem of getting an RNN to remember things for a long time (like hundreds of time steps).
2. They designed a memory cell using logistic and linear units with multiplicative interactions.
3. Information gets into the cell whenever its "write" gate is on.
4. The information stays in the cell so long as its **keep** gate is on.
5. Information can be read from the cell by turning on its **read** gate.